

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Системное и прикладное программное обеспечение»
для студентов специальности 10.05.01 Компьютерная безопасность
специализации «Информационно-аналитическая и техническая экспертиза
компьютерных систем»

Ставрополь
2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
Лабораторная работа №1	5
Лабораторная работа №2	11
Лабораторная работа №3	18
Лабораторная работа №4	23
Лабораторная работа №5	29
Лабораторная работа №6	33
Лабораторная работа №7	36
Лабораторная работа №8	38
Лабораторная работа №9	40
Список литературы.....	45

ВВЕДЕНИЕ

Цели и задачи освоения дисциплины

Целью изучения дисциплины «Системное и прикладное программное обеспечение» является формирование набора общепрофессиональных компетенций студента специальности 10.05.01 – Компьютерная безопасность (специализация «Информационно-аналитическая и техническая экспертиза компьютерных систем»).

Задачи освоения дисциплины «Системное и прикладное программное обеспечение»:

- изучить способы классификации программного обеспечения, виды программного обеспечения с точки зрения назначения, внутреннего устройства, целевого оборудования, принципов разработки, лицензирования и распространения;
- изучить принципы построения, разработки и использования системного и прикладного программного обеспечения;
- сформировать базовые теоретические знания и практические навыки, необходимые для обеспечения безопасности программного обеспечения и обрабатываемой с его помощью информации.

Для освоения дисциплины, обучающиеся должны владеть следующими знаниями и компетенциями:

владеть навыками разработки программного обеспечения, приобретенными в ходе освоения дисциплин «Методы программирования», «Языки программирования»;

владеть знаниями об устройстве, назначении и принципах функционирования операционных систем, приобретенными в ходе освоения дисциплины «Операционные системы»;

знать материалы курса информатики.

Компетенции обучающегося, формируемые в результате изучения дисциплины:

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ОПК-2 Способен применять программные средства системного и прикладного назначения, в том числе отечественного производства, для решения задач профессиональной деятельности	ИД-2 ОПК-2 применяет программные средства системного и прикладного назначения, в том числе отечественного производства, при решении задач профессиональной деятельности	Способен применять системное и прикладное программное обеспечение для решения рабочих задач. Способен выбирать наиболее подходящие по совокупности различных критериев программные продукты для использования в профессиональной деятельности
ОПК-12 Способен администрировать операционные системы и выполнять работы по восстановлению работоспособности прикладного и системного программного обеспечения	ИД-1 ОПК-12 использует системные приложения для управления настройками операционных систем	Владеет навыками применения системных и прикладных приложений для решения задач администрирования операционных систем и обеспечения корректной работы программного и аппаратного обеспечения ЭВМ

НАИМЕНОВАНИЕ ЛАБОРАТОРНЫХ РАБОТ

№	Наименование тем дисциплины, их краткое содержание	Объем часов	Из них практическая подготовка, часов
7 семестр			
1. Системное программное обеспечение			
1.	Создание дистрибутива программы	3	
2.	Создание приложения с защитой от несанкционированного копирования	3	
3.	Приложение с обновлением данных через интернет	3	
4.	Создание программ с функцией автоматического обновления	3	
5.	Использование компонентов для работы с графикой	3	
6.	Создание приложения для работы с базами данных	3	
7.	Использование диаграмм и бизнес-графики	3	
8.	Использование динамических библиотек	3	
9.	Создание многопоточных приложений	3	
	Итого за 7 семестр	27	

СТРУКТУРА И СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа №1 Создание дистрибутива программы

Цель и содержание: изучить назначение и особенности дистрибутивов программ. Получить навыки создания дистрибутива с помощью программы InnoSetup

Теоретические сведения:

Определение: Дистрибутив (ПО) – это комплект (как правило, набор файлов), приспособленный для распространения ПО. Может включать вспомогательные инструменты для автоматической или автоматизированной начальной настройки ПО (установщик).

Примеры:

- Архив (.zip, .rar, .tar.gz и др.) – неавтоматизированный дистрибутив, как правило, содержит инструкцию по установке (наиболее часто встречаются INSTALL, README, install.html).
- Исполняемый файл – дистрибутив с автоматизированным установщиком, позволяет пользователю указать необходимые параметры при установке. По завершении процесса установки ПО, как правило, готово к работе.
- Пакет (.msi, .rpm, .deb и др.) – файл, имеющий специальный формат. Обрабатывается менеджером пакетов операционной системы. В основном, имеет те же пользовательские характеристики, что и дистрибутив в исполняемом файле.
- Комплект на CD/DVD – такой дистрибутив, как правило, состоит из нескольких файлов и сопровождается автоматизированным установщиком. Используется для крупных пакетов ПО и системного программного обеспечения (дистрибутивы ОС Windows, различные дистрибутивы Linux).
- Сетевой репозиторий – сетевой ресурс, содержащий компоненты дистрибутива. Дистрибутив содержит автоматизированный установщик, который умеет получать выбранные пользователем компоненты ПО из сетевого репозитория. Как правило, репозиторий содержит пакеты (см. выше), а установщиком является менеджер пакетов, позволяющий также автоматически устанавливать обновления, как только они появляются в репозитории. Большинство дистрибутивов Linux, различных веток BSD и др. используют этот вид дистрибутивов в комбинации с комплектами на CD/DVD для начальной установки.

Inno Setup – система создания инсталляторов для Windows программ. Программа обладает бесплатной лицензией. Впервые выпущенный в 1997 году, Inno Setup сегодня конкурирует и даже превосходит многие коммерческие установщики по функциональности и стабильности.

Ключевые особенности:

- Поддержка всех версий Windows (Windows 95, 98, 2000, 2003, XP, Me, NT 4.0), в т.ч. и 64-разрядных Windows XP и Windows Server 2003 (поддерживаются как 64-разрядные, так и Itanium платформы).
- Поддержка создания упакованного однофайлового инсталлятора, поддержка разбиения инсталлятора на диски.
- Использования стандартного стиля Windows 2000/XP в диалогах и окнах установки.
- Создание нескольких типов установок в одном инсталляторе, например, Полная установка, Минимальная и Выборочная.
- Полноценный деинсталлятор.

- Поддержка следующих операций с файлами: извлечение Bzip2 и 7-Zip lzma архивов; сравнение версий; замещение используемых файлов; учёт общих файлов при множественных установках/удалениях/обновлениях; регистрация и deregистрация DLL/OCX; установка шрифтов.

- Создание ярлыков в любом месте диска, включая Меню Пуск и Рабочий стол.
- Создание и удаление записей реестра и значений ini-файлов.
- Интегрированный движок Pascal-скриптов.
- Поддержка мультязычных установок.
- Поддержка шифрованных инсталляторов и паролей.
- Тихий режим установки и удаления программ (silentmode).
- Открытый исходный код.

Преимущества и недостатки

Рассмотрим сильные и слабые стороны данного инсталлятора в приложении к автоустановке по сравнению с другими вариантами: WinRAR и 7zSFX, простыми cmd-файлами и wsh-скриптами.

К главному преимуществу Inno Setup как и любого инсталлятора относится прежде всего создание деинсталлятора.

Другими плюсами являются:

- большие возможности и простота, по сравнению с другими freeware инсталляторами,
- поддержка языка Pascal, и соответственно Pascal-скриптов, которые значительно расширяют стандартные возможности программы,
- структурированность кода скрипта, в связи с этим он интуитивно понятен, разобраться в нем очень просто.

Способы создания и редактирования скриптов

Скрипт можно создавать двумя способами: вручную или с помощью специализированных инструментов типа ISTools (также достаточно широко применяется InnoSetup Script Generator), которые сильно облегчают рутинные действия, кроме того обладают очень полезной функцией – возможностью импорта файлов реестра.

Пример создания простого скрипта

В качестве примера создадим скрипт установки программы ATNotes, который будет устанавливать приложение в выбранную пользователем папку (или папку, заданную по умолчанию), создавать ярлыки на *Рабочем столе*, в *меню Пуск* и в панели *Быстрого запуска*.

Прежде чем начать написание скрипта, нужно подготовить необходимые для установщика файлы. В нашем случае это непосредственно сами файлы программы. Для удобства перепишем их в папку «files» (пустой файл скрипта [назовём его atnotes.iss] и папку files предварительно нужно создать). Файл скрипта, как и говорилось ранее, можно создавать с помощью мастера в программе ISTools, заполнив необходимые поля или вручную. При написании скрипта вручную он получается чуть более коротким, т. к. мастер автоматически проставляет все параметры, которые часто необязательны.

```
[Setup]
AppName=ATnotes
AppVerName=ATnotes 9.5
DefaultDirName={pf}\ATnotes
DefaultGroupName=ATnotes
[Files]
Source: files\*; DestDir: {app}; Flags: ignoreversion
Source: atnotes.iss; Flags: dontcopy
[Icons]
Name: {group}\ATnotes; Filename: {app}\ATnotes.exe; WorkingDir: {app}
```

Name: {group}\Справка; Filename: {app}\ATnotes.chm; WorkingDir: {app}
Name: {group}\Удалить ATnotes; Filename: {uninstallexe}; WorkingDir: {app};
IconFilename: {sys}\shell32.dll; IconIndex: 31
Name: {userstartup}\ATnotes; Filename: {app}\ATnotes.exe; WorkingDir: {app}

Скрипт состоит из секций [Setup], [Files] и [Icons], из них первая секция является обязательной, без неё невозможно скомпилировать (создать) дистрибутив. Секций [Files] и [Icons] может и не быть, если они не нужны. Всего же может быть 18 различных секций с разным назначением и параметрами.

Рассмотрим подробнее каждую из наших секций и параметров.

[Setup] – основная секция скрипта, которая определяет, как будет называться дистрибутив, куда он по умолчанию будет установлен, какая Группа по умолчанию будет создана в меню Пуск в Программах, будет ли иметь дистрибутив свою иконку и своё описание, владельца, также в ней определяются такие параметры как сжатие, отображение или скрытие определённых страниц при установке уже готовой программы, поведение установщика и деинсталлятора.

AppName – определяет название дистрибутива и заголовок окна дистрибутива.

AppVerName – название, которое попадает в меню «Установка и удаление программ».

DefaultDirName – путь по умолчанию, куда установится программа, значение этого параметра присваивается переменной {app}, которую потом очень удобно использовать.

DefaultGroupName – группа по умолчанию в меню Пуск – Программы, значение этого параметра присваивается переменной {group}, которую потом удобно использовать при создании иконок.

[Files] – секция, в которой указывается, где лежат нужные для дистрибутива файлы (т. е. где компилятор будет искать файлы, которые и запакует в дистрибутив) и куда они будут скопированы при установке. Кроме того, каждая из строк в этой секции может содержать несколько флагов, которые укажут компилятору как себя вести при распаковке данных файлов. В нашем случае использован флаг «ignoreversion», указывающий установщику перезаписывать файлы независимо от их версий (даже если существующий файл новее) и «dontcopy» означающий, что файл не будет распакован в папку назначения при установке (удобно хранить исходный файл скрипта в самом дистрибутиве). Рассмотрим наши строки: Source: files*; DestDir: {app}; Flags: ignoreversion.

Source – это папка или файл источник, т. е. место, откуда будет взят файл (или файлы с подкаталогами). Путь может быть относительным или абсолютным, например: C:\ATnotes* (Относительный путь определяется от файла скрипта). При указании исходных файлов, можно использовать маски: * – все файлы (тоже что и *.*), *.exe, file.* и т. п., чтобы включить также все подпапки необходимо использовать флаг: «recursesubdirs».

DestDir – папка назначения, т.е. куда будут установлены файлы. Здесь, также как и в папке *source*, можно использовать константы путей (аналог системных переменных окружения), в нашем случае константа {app} равна {pf}\ATnotes, а константа {pf} равна соответственно полному пути до папки Program Files (Например C:\Program Files). Всего в InnoSetup более 30 различных констант, определяющих путь до наиболее часто используемых системных и пользовательских папок.

Flags – флаги применяемые к данному файлу (-ам).

[Icons] – секция, содержащая информацию о ярлыках приложения. Константа {uninstallexe} включает полный путь к деинсталлятору (uninsXXX.exe). Ни в коем случае не прописывается путь к деинсталлятору как {app}\unins000.exe, так как деинсталлятор может называться unins001.exe или даже unins002.exe.

Секция [Setup]

Параметры секции Setup (дополнительные):

AlwaysRestart=yes Перезагрузка компьютера после установки.
UninstallRestartComputer=yes Перезагрузка компьютера после удаления.
SetupIconFile=compiler:uninst.ico Сделать значком Setup файл uninst.ico из папки с Inno Setup. Или так: *SetupIconFile=C:\MyIcon.ico* Задать значок Setup файл MyIcon.ico .

UninstallDisplayIcon={app}\uninst.ico Задать значок в Установке и удалении программ. Или так: *UninstallDisplayIcon={app}\MyProg.exe* Или так:

UninstallDisplayIcon={app}\MyProg.exe,1 Здесь 1 – индекс иконки, начиная с нуля. Смотреть индексы можно программой ResHacker.

Языки

Языками ведает секция [Languages]. Для русификации скрипта она должна иметь вид:

```
[Languages]      Name: "russian";      MessagesFile: "compiler:Languages\Russian.isl"
```

Естественно, файл Russian.isl должен реально присутствовать в папке Languages, которая лежит в папке с Inno Setup.

Для двуязычных программ:

```
[Languages]      Name: "english";      MessagesFile: "compiler:Default.isl"      Name: "russian";  
MessagesFile: "compiler:Languages\Russian.isl"
```

При запуске будет выведено предложение выбрать язык – английский или русский.

Реестр

Запись значений в системный реестр Windows возможна с помощью секции [Registry].

Пример:

```
[Registry]  
Root: HKLM; Subkey: "SOFTWARE\Crytek"; Flags: uninsdeletekeyifempty  
Root: HKLM; Subkey: "SOFTWARE\Crytek\Crysis"; Flags: uninsdeletekey      Root:  
HKLM; Subkey: "SOFTWARE\Crytek\Crysis"; ValueName: "GUID"; ValueType: String;  
ValueData: "{014ED150-1F18-4E85-BECE-  
E4B4C1E3DB29}";  
Root: HKLM; Subkey: "SOFTWARE\Crytek\Crysis"; ValueName: "InstallDir";  
ValueType: String; ValueData: "{app}";  
Root: HKLM; Subkey: "SOFTWARE\Electronic Arts"; Flags: uninsdeletekeyifempty  
Root: HKLM; Subkey: "SOFTWARE\Electronic Arts\Electronic Arts"; Flags:  
uninsdeletekeyifempty  
Root: HKLM; Subkey: "SOFTWARE\Electronic Arts\Electronic Arts\Crysis";  
Flags: uninsdeletekey  
Root: HKLM; Subkey: "SOFTWARE\Electronic Arts\Electronic Arts\Crysis\ergs";  
ValueType: String; ValueData:  
"RM88SDWAU3A3L1O31911";
```

Пример кода

Часто бывает, что в папке, куда устанавливается приложение, остаются файлы, которые не удаляются при деинсталляции. В этом поможет команда DelTree – удаление каталога. (Следует помнить, что если программа будет установлена скажем в C:\Program Files или в C:\WINDOWS, то эта папка будет удалена. `procedure CurUninstallStepChanged (CurUninstallStep: TUninstallStep); begin`

```
    case CurUninstallStep of      usPostUninstall:
```

```
        DelTree(ExpandConstant('{app}'),true,true,true);      end; end;
```

InnoUnp. Утилита декомпиляции дистрибутивов Inno Setup

Утилита InnoUnp позволяет распаковывать установщики, созданные с помощью Inno Setup. После распаковки получите скрипт install_script.iss и папки {app}, {win}, embedded (возможно, не все). Недостатками утилиты являются ошибки в секции [Files],

такие как добавление в нее деинсталлятора и, что важнее, невозможность извлечения секции [Code] в неизменном виде.

Программа создает файл CompiledCode.bin, но привести его содержание в читаемый вид часто невозможно.

Параметры командной строки (ключи автоматической установки)

Программа-инсталлятор поддерживает приведенные ниже параметры командной строки. Они пригодятся системным администраторам и другим программам, вызывающим инсталлятор.

/SP – отключает страницу «Эта программа установит... Вы хотите продолжить?» инсталлятора. Выполнится только если DisableStartupPrompt секции [Setup] равна yes.

/SILENT, /VERYSILENT – установка в ускоренном или сверхускоренном режиме. При установке в ускоренном режиме окно мастера и фоновое окно мастера не отображаются, отображается только окно прогресса.

При сверхускоренной установке окно прогресса тоже не отображается. Остальная установка проходит как обычно – отображаются сообщения об ошибках в ходе установки, страница приветствия (если ее отображение не отменено директивой DisableStartupPrompt или параметром '/SP-' командной строки).

Если после установки необходима перезагрузка, команды '/NORESTART' не дано и установка проходит в ускоренном режиме, появится сообщение «Перезагрузить компьютер? ». Если режим установки сверхускоренный, система перезагрузится, не спрашивая.

/LOG – указывает инсталлятору создать log-файл в папке TEMP для подробного отчета о действиях и событиях в секции [Run], предпринятых в ходе установки. Применяется при отслеживании ошибок. Например, если вы подозреваете, что файл не был удален, хотя должен был, log-файл сообщит, что файл был действительно пропущен и почему.

Log-файлу присваивается уникальное имя на базе текущей даты. (Существующие файлы не перезаписываются и не дополняются). Информация, содержащаяся в log-файле, техническая и поэтому не может быть понята пользователем.

/LOG="filename" – параметр похож на /LOG, но в отличие от него позволяет указать фиксированный путь/имя файла для log-файла. Если файл с указанным именем уже существует, он будет заменен. Если файл не может быть создан, установка прервется и выдаст сообщение об ошибке.

/NOCANCEL – не дает пользователю прервать установку, отключая кнопку Отменить и игнорируя клики на кнопке Закрыть. Используется в комбинации с '/SILENT' или '/VERYSILENT'.

/NORESTART – не перезагружать компьютер, даже если это необходимо.

/RESTARTEXITCODE=exit code – задает код, который инсталлятор должен возвратить, если необходима перезагрузка. Используется с '/NORESTART'.

/LOADINF="filename" – загрузить параметры установки из указанного файла после проверки командной строки. Этот файл может быть подготовлен с помощью '/SAVEINF=' command. Не забудьте ставить кавычки при использовании имен, содержащих пробелы.

/SAVEINF="filename" – сохранять параметры установки в указанный файл. Не забудьте ставить кавычки при использовании имен, содержащих пробелы.

/LANG=language – задает используемый язык. Language задает имя языка, заданного в параметре секции [Languages]. При использовании параметра /LANG появляется диалог «Выбор языка установки».

/DIR="x:\dirname" – меняет имя папки, заданное по умолчанию, на странице Выбор папки назначения. Должен быть задан полный путь.

/GROUP="folder name" – меняет имя папки на странице Выбор папки меню Пуск. Если директива DisableProgramGroupPage секции [Setup] равна yes, этот параметр командной строки игнорируется.

/NOICONS – проверять, выбрана ли команда «Не создавать значков на странице Выбор папки меню Пуск».

/COMPONENTS="comma separated list of component names" – меняет выбранные по умолчанию компоненты. В этом случае параметр командной строки указывает инсталлятору менять тип установки на «Выборочный».

/PASSWORD=password – задает пароль. Если не указана директива Password секции [Setup], этот параметр игнорируется.

Если введен неправильный пароль, параметр также игнорируется.

Содержание задания:

Создать и протестировать дистрибутив программы с помощью freeware программы InnoSetup. В качестве устанавливаемой программы можно использовать notepad.exe из системного каталога Windows, или любую пользовательскую программу. Дистрибутив должен предлагать каталог для установки по умолчанию, возможность выбора каталога, обеспечивать вывод информации об устанавливаемой программе, должен присутствовать деинсталлятор.

Варианты заданий:

1. Установка по заданному паролю.
2. Два языковых пакета при установке.
3. Записать ключ в системный реестр.
4. Перезагрузить компьютер после установки.
5. Произвести установку в ускоренном режиме.

Порядок выполнения работы:

1. Изучить назначение и характер работы программ-дистрибутивов.
2. Изучить прикладную программу для создания установочных приложений на примере InnoSetup.
3. Изучить возможности и базовые понятия скриптовых языков на примере встроенного языка InnoSetup.
4. Ответить на контрольные вопросы.
5. Получить у преподавателя номер варианта задания.
6. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Для чего предназначаются дистрибутивы программ?
2. Отметьте основные возможности и варианты, предлагаемые пользователю при установке программ с помощью дистрибутива.
3. Для чего предназначен встроенный скриптовый язык прикладных программ для создания дистрибутивов?

Лабораторная работа №2

Создание приложения с защитой от несанкционированного копирования

Цель и содержание: изучить способы защиты программных продуктов от несанкционированного копирования. Получить навыки создания программ с защитой от копирования.

Теоретические сведения:

Виды защиты программ от копирования.

Защита от несанкционированного копирования – система мер, направленных на противодействие несанкционированному копированию информации, как правило, представленной в электронном виде (данных или программного обеспечения). При защите от копирования используются различные меры: организационные, юридические, программные и программноаппаратные.

Преимуществом технических мер защиты, то есть программных и программноаппаратных, является возможность обеспечения противодействия непосредственно в момент попытки эксплуатации нелегальной копии программ или данных.

Хотя законодательством, как правило, разрешено копирование в ряде случаев (например, резервное), невозможно только техническими средствами надежно определить «законность» копирования. Поэтому технические средства защиты авторских прав зачастую запрещают любое копирование, создавая неудобства пользователям, за что подвергаются критике со стороны правозащитников.

Организационные меры защиты

Основная идея организационных мер защиты заключается в том, что полноценное использование программного продукта невозможно без соответствующей поддержки со стороны производителя: подробной пользовательской документации, «горячей линии», системы обучения пользователей, обновления версий со скидкой и т. п.

Организационные меры защиты применяются, как правило, крупными разработчиками к достаточно большим и сложным программным продуктам.

Правовые меры защиты

Предусмотрена ответственность, в соответствии с действующим законодательством, как за использование контрафактных экземпляров программ для ЭВМ и баз данных, так и за преодоление применяемых технических средств защиты.

Защита при помощи ключевой дискеты

Была распространена во времена MS-DOS, сейчас в силу устаревания технологии FDD практически не применяется.

Есть несколько основных способов создания не копируемых меток на дискетах:

Считывание конкретного сектора дискеты (возможно, пустого или сбойного). Это самый простой способ защиты, и при копировании «дорожка в дорожку» дискета копируется.

Запоминание сбойных секторов дискеты. Перед тем, как записать на дискету информацию, её царапают (или прожигают лазером), после этого записывают номера сбойных секторов. Для проверки подлинности дискеты программа пытается записать в эти сектора информацию, затем считать её.

Нестандартное форматирование дискеты. Известна программа FDA (Floppy Disk Analyzer), которая могла проводить исследование и копирование таких дискет.

«Плавающий бит». Один бит записывается так, что в некоторых случаях он читается как «0», в некоторых как «1». Проводится многократное считывание дискеты; среди результатов считывания должны быть и нули, и единицы.

Защита при помощи компьютерных компакт-дисков

Как правило, этот способ применяется для защиты программ, записанных на компакт-диске, являющимся одновременно ключевым. Для защиты от копирования используется:

- запись информации в неиспользуемых секторах;
- проверка расположения и содержимого «сбойных» секторов;
- проверка скорости чтения отдельных секторов.

Первые два метода бесполезны при снятии полного образа с диска. Третий метод более надёжный. Он используется, например, в защите StarForce. В этой защите также делается попытка проверить возможность записи на вставленный диск. Если запись возможна, то диск считается нелегальным. Но существуют программы, которые могут эмулировать диски с учётом геометрии расположения данных, тем самым обходя эту защиту, и, к тому же, возможно записать диск CD-R с её учётом, и он будет признаваться легальным. Также возможно скрыть тип диска, чтобы CD-R или CD-RW был виден как обычный CD-ROM. Но и системы защиты тоже (используя специальный драйвер) борются с ними, пытаясь обнаружить наличие эмуляции. В настоящее время наибольшую известность в мире имеют системы защиты от копирования SecuROM, StarForce, SafeDisc и Tages.

Защита программ, установленных на жёстком диске

Для программ, установленных на жёстком диске, могут применяться такие меры защиты:

Программа может требовать вставленную дискету или компактдиск. В частности, это широко применяется в играх. Но для многих программ такие меры недоступны (например, shareware-программы или программы повседневного пользования).

Электронный ключ (донгл), вставленный в один из портов компьютера (обычно LPT или USB, редко – COM). Достоинство ключа в том, что его можно вставлять в любой компьютер, на котором намереваетесь запустить программу. Кроме того, электронный ключ быстр и не занимает дисковод. Но электронные ключи дороги (5–15 долларов), и применяются лишь в дорогостоящем ПО. Также теоретически возможны конфликты периферийных устройств с ключом. Типичный пример электронного ключа – HASP.

Привязка к серийным номерам компонентов компьютера. Её достоинство в том, что не требуется никакого специфического аппаратного обеспечения, и программу можно распространять посредством цифровой дистрибуции. Если пользователь производит модернизацию компьютера, защита отказывает. Авторы многих программ, защищённых привязкой, в подобных случаях готовы дать новый регистрационный код. Например, Microsoft в Windows XP разрешает раз в 120 дней генерировать новый регистрационный код (но в исключительных случаях, позвонив в службу активации, можно получить новый код и по прошествии этого срока). В качестве привязки используются серийный номер BIOS материнской платы, серийный номер винчестера.

Сканирование сети. Это исключает одновременный запуск двух программ с одним регистрационным ключом на двух компьютерах в пределах одной локальной сети. Локальный брандмауэр можно настроить так, чтобы он не пропускал пакеты, принадлежащие защищённой программе. Правда, настройка брандмауэра требует некоторых пользовательских навыков. Кроме того, в большинстве реальных сетей «все друг другу доверяют» (это ускоряет доступ к ресурсам других компьютеров и сетевую игру), и брандмауэр безусловно пропускает локальный трафик.

Если программа работает с каким-то централизованным сервером и без него бесполезна (например, онлайн-игра, антивирус), она может **передавать серверу свой серийный номер**; если номер неправильный, сервер отказывает в услуге. Это единственный стопроцентный способ защиты от копирования. Впрочем, существует возможность создать сервер, который не делает такой проверки. Например, существовал сервер battle.da, который по функциям был аналогичен Battle.net, но пускал пользователей неавторизованных копий игр. Сейчас этот сервер закрыт, но существует немалое количество PvPGN-серверов, которые также не проверяют регистрационные номера.

Помодульная продажа ПО за счет использования возможностей системы удаленной активации/деактивации отдельных модулей позволяет продавать программный продукт в различных конфигурациях и дистанционно изменять конфигурацию у клиента.

Пароль на программу. В этом случае, конечно, программу можно скопировать, но работать она будет лишь у того пользователя, который знает пароль. Понятно, что такой метод имеет смысл лишь тогда, когда пользователь программы не заинтересован в ее тиражировании. В связи с этим, данный метод применяется ограниченно. Чаще всего – при изготовлении некоей специализированной программы по индивидуальному заказу.

Разновидность данного метода часто используется в дистрибутивах продукции «Microsoft» и других западных производителей. Для инсталляции программы необходимо ввести индивидуальный код продукта (пользователя). Естественно, к нелегальным (контрафактным) версиям продукта прилагается такой код, взятый с того экземпляра, который служил оригиналом. Часто можно ввести и какой-либо другой взятый наугад номер. Не с первой, так со второй-третьей попытки удаётся найти приемлемый.

Метод преодоления

Проще всего узнать пароль. Если нет такой возможности, то надо прибегнуть к методам взлома. Обычно проверка пароля реализована в виде подпрограммы (процедуры, функции), которая возвращает значение типа

«верный-неверный». Следует найти вызов этой процедуры и отключить его, т. е. вместо команды перехода по условию вставить безусловный переход. Иногда проверка пароля несколько более сложная, чем просто переход по условию. В таком случае взлом может не дать результата.

Такой метод – нахождение процедуры и отмена перехода по условию – применим и к другим методам защиты.

Разработчики, зная это, иногда пытаются застраховаться от подобного взлома. Например, вставляют проверку пароля в нескольких местах, с помощью разных процедур, которые трудно разыскать. Есть и более действенный метод защиты от обхода проверки пароля. Процедура проверки пароля возвращает не бинарный результат проверки («верный-неверный»), а некое число, показывающее правильность пароля. При верном пароле оно равно величине, известной разработчикам, но не известной хакерам. Далее это число не проверяется на соответствие контрольному значению, а используется в работе программы, например, в команде перехода по адресу или в вычислительной процедуре. При неверном пароле получается неверное значение этого числа. Программа при этом будет работать, но с непредсказуемым результатом – может и зависнуть, и испортить жёсткий диск. Если вы сами разрабатываете защиту программного продукта, возьмите на вооружение этот метод. Только не забудьте предусмотреть страховку от ошибочного ввода пароля.

Это можно сделать, например, при помощи хэш-функции.

Другой метод преодоления данного вида защиты – попробовать подобрать пароль, используя методы криптоанализа. Однако самое простое всё же – узнать пароль у законного пользователя программы.

Ограничение срока действия программы. Используется, как правило, в демо-версиях программ, но встречается и в лицензионных копиях, когда срок лицензии ограничен. С некоторой натяжкой данный метод можно отнести к методам защиты от копирования.

Суть метода понятна из названия. Реализуется это ограничение, как правило, через проверку текущей даты по таймеру компьютера. Иногда правильность показаний таймера может контролироваться другими способами.

Бывает, что ограничивается не календарный срок действия, а количество запусков программы. Тогда где-то в программе есть соответствующий счётчик.

Метод преодоления

Простейший метод преодоления ясен. Надо перевести часы назад.

В программе могут быть и дополнительные проверки. Как известно, во всех файловых системах имеется запись о дате создания файла. Программа может проверить заголовки некоторых файлов. И если найдёт среди них даты позже текущей, то заподозрит обман. Как можно справиться с такой проверкой, подумайте сами.

Здесь также применимы методы взлома. Можно найти в программе место проверки даты и поставить обход. Можно найти контрольную дату – дату окончания срока действия и изменить её. Но такие усилия редко оправданы. Проще менять дату или по истечении срока действия установить программу заново. В последнем случае вы можете наткнуться на ещё один метод защиты, описанный ниже.

Ограничение функций программы. Демо-версии программ очень широко используются в настоящее время. Такая версия предоставляется пользователю бесплатно с тем, чтобы он, испробовав её возможности, приобрёл затем полную версию.

Демо-версия отличается от полной либо сроком действия (эта ситуация рассматривалась выше), либо набором функций. Набор выполняемых функций демо-версии, её возможностей, объём доступных данных, число карт, сценариев и т.д. искусственно ограничены по сравнению с полной версией.

Разработка отдельной демо-версии – это дополнительные затраты. Поэтому обычно демо-версия делается из стандартной версии путём добавления нескольких команд в код программы, которые и реализуют ограничения. В таком случае возможен и обратный процесс – сделать из демо-версии полнофункциональную версию продукта.

Устаревшие методы защиты программного обеспечения

В прошлом применялись и другие методы защиты ПО от копирования.

Некоторые старые программы для DOS создавали не копируемые метки на жёстком диске. Например, файл длиной 1 байт занимает на диске один кластер (не менее 512 байт), и в оставшиеся 511 байт можно записать некоторую информацию. Сейчас эта практика не используется, так как она чревата потерями данных.

Привязка к руководству пользователя. Например, программа выводит: «Введите 5-е слово на 12-й сверху строке 26-й страницы». Более изощрённый способ защиты – в руководстве находится важная информация, без которой невозможно пройти игру, например, этим известна серия Space Quest. Распространение сканеров и многозадачных операционных систем положило конец этой практике.

Привязка к некоторому механическому устройству. Игра Another World поставлялась с «кодовым колесом». В системе защиты от копирования Lenslok,

применявшейся в играх для ZX Spectrum, надо было, посмотрев на картинку через систему призм, увидеть двухбуквенный код.

Внешний модуль против API

Некоторые защиты присоединяют к программе защитный модуль (как это делают вирусы). Преимущество этой защиты в том, что её можно «надеть» на любую программу. Недостаток в том, что защита, как правило, легко отключается.

Более прочные защиты имеют свой API, с помощью которого можно сделать весьма сложную логику защиты. Весь защитный код оказывается равномерно «размазанным» по программе, и взлом оказывается трудным или невозможным.

Методы взлома/обхода технических мер защиты

Для взлома защищённого программного обеспечения применяются такие методы:

- Копирование защищённого диска специальными программами.
- Эмуляция диска – специальный драйвер делает логический диск, который программа принимает за лицензионный. Во многих играх применяется вариант этого метода под названием «Mini Image», когда подставной диск имеет маленький размер (несколько мегабайт), тем не менее, игра признаёт его лицензионным.
- Эмуляция электронного ключа.
- Генерация регистрационного кода по серийным номерам аппаратного обеспечения.
- «Отключение» защиты взломом программного кода.

Методы противодействия взлому

Почти во всех защитах есть методы противодействия взлому: дестабилизация отладчика; шифрование кода, исключающее работу дизассемблером; «ложные ветви», сбивающие хакера с толку; проверка целостности файла, не дающая накладывать патчи; виртуальная машина с собственной системой команд. Эти методы не дают узнать, как защита работает (или хотя бы оттягивают этот момент настолько, насколько можно).

Замусоривание кода программы. Способ, при котором в программу вносятся специальные функции и вызовы, которые выполняют сложные действия, обращаются к накопителям, но по факту ничего не делают. Типичный способ обмана. Хакера нужно отвлечь, создав ответвление, которое и будет привлекать внимание сложными вызовами и содержать в себе сложные и большие вычисления. Хакер рано или поздно поймет, что его обманывают, но время будет потеряно.

Использование мультипоточности. Тоже эффективный способ защиты, использующий возможности Windows по параллельному исполнению функций. Любое приложение может идти как линейно, то есть инструкция за инструкцией, и легко читаться отладчиком, а может разбивать на несколько потоков, исполняемых одновременно, естественно, в этом случае, нет никакого разговора о линейности кода, а раз нет линейности, то анализ здесь трудноосуществим. Как правило, создание 5-6 и более потоков существенно усложняет жизнь хакеру. А если потоки еще и шифруются, то хакер надолго завязнет, пытаясь вскрыть приложение.

Подавление изменения операционной среды – программа сама несколько раз перенастраивает среду окружения, либо вообще отказывается работать в измененной среде. Не все отладчики способны на 100 % имитировать среду системы, и если «подопытное» приложение будет менять настройки среды, то рано или поздно «неправильный» отладчик может дать сбой.

Противодействие постановке контрольных точек. Специальный механизм, поддерживаемый микропроцессором, при помощи которого можно исследовать не всю программу, начиная с начала, а, например, только начиная с середины. Для этого в

середине программы ставят специальный вызов (точка вызова – BreakPoint), который передает управление отладчику. Недостаток способа кроется в том, что для осуществления прерывания в код исследуемого приложения надо внести изменение. А если приложение время от времени проверяет себя на наличие контрольных точек, то сделать подобное будет весьма и весьма непросто.

Изменений определенных регистров процессора, на которые отладчики неадекватно реагируют. Также как и со средой. Отладчик – тоже программа и тоже пользуется и операционной системой, и процессором, который один на всех. Так если менять определенные регистры микропроцессора, которые отладчик не может эмулировать, то можно существенно «подорвать» его здоровье.

Шифрование. Самый простой и эффективный способ противодействия. Подразумевает, что определенная часть кода никогда не появляется в свободном виде. Код дешифруется только перед передачей ему управления. То есть вся программа или ее часть находится в зашифрованном виде, а расшифровывается только перед тем как исполниться. Соответственно, чтобы проанализировать ее код, надо воспользоваться отладчиком, а его работу можно очень и очень усложнить.

Шифрование и дешифрование (динамическое изменение кода). Более продвинутый способ шифрования, который не просто дешифрует часть кода при исполнении, но и шифрует его обратно, как только он был исполнен. При такой защите хакеру придется проводить все время с отладчиком, и взлом защиты затянется на очень и очень долгое время.

Использование виртуальных машин. Еще одна модернизация шифрования. Способ заключается в том, чтобы не просто шифровать и дешифровать целые фрагменты кода целиком, а делать это покомандно, подобно тому, как действует отладчик или виртуальная машина: взять код, преобразовать в машинный и передать на исполнение, и так пока весь модуль не будет исполнен. Этот способ гораздо эффективнее предыдущих, так как функции приложения вообще никогда не бывают открытыми для хакера. Естественно, что его трудно реализовать, но реализовав, можно оградить себя от посягательств любых хакеров. Но в этом способе кроется недостаток – производительность, ведь на подобное транслирование требуется много времени, и, соответственно, способ хорош для защиты только для критических участков кода.

Дополнительные способы противодействия – использовать системные ресурсы Windows для хранения данных защиты. Дополнительную память, выделяемую для параметров окон, и локальные хранилища потоков.

Суть способа состоит в нестандартном использовании стандартных областей, скажем, хранить ключи, пароли и т.п., совсем не там, где их будут искать при взломе в первую очередь.

Использовать операции сравнения нестандартными способами, во избежание их явного присутствия. Для сравнения есть определенные инструкции микропроцессора, о которых знают и разработчики и хакеры. А если попытаться использовать нестандартные виды сравнения, то можно слегка запутать хакера, ожидающего стандартного ответа.

Избегать обращений к переменным, относящимся к защите напрямую. То есть использовать любые косвенные способы доступа к специальным областям.

Использовать метод «зеркалирования» событий, то есть применять нестандартные действия на стандартные вызовы. Об этом говорилось выше.

Использовать для шифрования надежные, проверенные временем алгоритмы и т. д.

Содержание задания:

Создать и протестировать программу, содержащую, по крайней мере, два алгоритма защиты от несанкционированного копирования. Алгоритмы выбираются из вариантов заданий. Для создания программы используется среда программирования Turbo Explorer либо Turbo C++.

Варианты заданий:

1. Пароль на запуск программы.
2. Ключ, генерируемый по аппаратным характеристикам компьютера пользователя (код жесткого диска, процессора и т.п.).
3. Ключ, генерируемый по программным характеристикам компьютера (метка жесткого диска, имя компьютера и т.п.).
4. Ограничение количества запусков программы.
5. Ограничение запуска программы до определенной даты.
6. Ввод имени пользователя и текста об ограничении прав запуска и шифрование кода программы.
7. Электронный ключ (вместо реального ключа – программа должна требовать наличие специального файла на USB-диске).

Порядок выполнения работы:

1. Изучить виды защиты программ от копирования.
2. Изучить способы обхода стандартных мер защиты от копирования.
3. Ответить на контрольные вопросы.
4. Получить у преподавателя номер варианта задания.
5. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Для чего необходимо защищать программу от несанкционированного копирования?
2. Перечислить меры защиты от копирования.
3. Перечислить программные способы защиты от копирования.
4. Перечислить программно-аппаратные способы защиты от копирования.
5. Перечислить методы взлома систем защиты от копирования, рассказать, как можно противодействовать этим методам?

Лабораторная работа №3

Приложение с обновлением данных через интернет

Цель и содержание: изучить способы защиты программных продуктов от несанкционированного копирования. Получить навыки создания программ с защитой от копирования.

Теоретические сведения:

Интернет – глобальная телекоммуникационная сеть информационных и вычислительных ресурсов. Интернет состоит из многих тысяч корпоративных, научных, правительственных и домашних компьютерных сетей. Объединение сетей разной архитектуры и топологии стало возможно благодаря протоколу IP (англ. *Internet Protocol*) и принципу маршрутизации пакетов данных.

Протокол IP был специально создан агностическим в отношении физических каналов связи. То есть любая система (сеть) передачи цифровых данных, проводная или беспроводная, для которой существует стандарт инкапсуляции в неё IP-пакетов, может передавать и трафик интернета. Агностицизм протокола IP, в частности, означает, что компьютер или маршрутизатор должен знать тип сетей, к которым он непосредственно присоединён, и уметь работать с этими сетями; но не обязан (и в большинстве случаев не может) знать, какие сети находятся за маршрутизаторами.

Протокол – это, образно говоря, «язык», используемый компьютерами для обмена данными при работе в сети. Чтобы различные компьютеры сети могли взаимодействовать, они должны «разговаривать» на одном «языке», то есть использовать один и тот же протокол. Проще говоря, протокол – это правила передачи данных между узлами компьютерной сети. Систему протоколов интернет называют «стеком протоколов TCP/IP».

Протокол TCP/IP – это два протокола нижнего уровня, являющиеся основой связи в сети интернет. Протокол TCP (Transmission Control Protocol) разбивает передаваемую информацию на порции и нумерует все порции. С помощью протокола IP (Internet Protocol) все части передаются получателю. Далее с помощью протокола TCP проверяется, все ли части получены. При получении всех порций TCP располагает их в нужном порядке и собирает в единое целое.

Наиболее известные протоколы, используемые в сети интернет:

- HTTP (Hyper Text Transfer Protocol) – это протокол передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц с одного компьютера на другой.

- FTP (File Transfer Protocol) – это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером, пользователь может скопировать файл с удаленного компьютера на свой или скопировать файл со своего компьютера на удаленный.

- POP (Post Office Protocol) – это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

- SMTP (Simple Mail Transfer Protocol) – протокол, который задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

- TELNET – это протокол удаленного доступа. TELNET дает возможность абоненту работать на любой ЭВМ сети интернет, как на своей собственной, то есть запускать программы, менять режим работы и т. д.

На практике возможности лимитируются тем уровнем доступа, который задан администратором удаленной машины.

Рассмотрим подробнее протокол HTTP.

HTTP (англ. *HyperText Transfer Protocol* – «протокол передачи гипертекста») – протокол прикладного уровня передачи данных (изначально – в виде гипертекстовых документов). Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом. HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с web-сайтов.

Основным объектом манипуляции в HTTP является *ресурс*, на который указывает URI (англ. *Uniform Resource Identifier*) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

HTTP – протокол прикладного уровня, аналогичными ему являются FTP и SMTP. Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами. Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

Довольно часто современному программисту приходится решать вопросы, которые связаны с отображением или обработкой информации, представленной в виде html-ресурсов. Например, на некотором сайте приводятся ежедневные котировки акций и Вам нужно собирать и обрабатывать статистику за определенный период времени.

Можно, используя WinInet.dll, получить доступ к требуемому webресурсу, а затем с помощью многочисленных строковых функций получить интересующую нас ее часть. Эта технология вполне работает, но довольно трудоемкая и, в большинстве случаев далека от элегантности. Другой путь – использование функциональности Microsoft Internet Explorer.

Internet Explorer – это не одна отдельная программа, а целая коллекция компонент, которые можно использовать при разработке своих приложений. Наиболее интересными с точки зрения прикладного программиста являются компоненты из **shdocvw.dll** и **mshtml.dll**. Первая DLL содержит **WebBrowser** – Microsoft ActiveX control, используемый для визуального просмотра web-страниц (рабочая область окна Internet Explorer и есть тот самый компонент WebBrowser). Вторая DLL содержит синтаксический анализатор HTML кода, а также средства взаимодействия с отдельными элементами этого кода.

Рассмотрим пример работы с WebBrowser на примере среды разработки программ Turbo Explorer.

Для любого зарегистрированного в палитре ActivX-контроля Turbo Explorer при импорте создает класс-оболочку, которая наследуется от **TOleControl**. Для **WebBrowser** от IE такой оболочкой есть **TWebBrowser**.

Создать web -браузер с использованием TWebBrowser довольно легко, и займет не более одной минуты. Открываем новый проект, центрируем форму, в нижней части формы создаем панель, на которой располагаем ComboBox для ввода URL, слева от нее соответствующий Label, справа кнопку «Go». Разместим также главное действующее лицо TWebBrowser над панелью. Дадим имена нашим компонентам (например, ComboBox можно назвать «selUrl»), поставим соответствующие опции выравнивания. Создадим обработчик нажатия на кнопку «Go»:

```
procedure TFormSimpleWB.btGoClick(Sender: TObject); var
    _URL, Flags, TargetFrameName, PostData, Headers: OleVariant; begin
    _URL := selUrl.Text;
    Flags := 0;
    TargetFrameName := 0;
    Postdata := 0;
    Headers := 0;
    WebBrowser1.Navigate2(_URL,    Flags, TargetFrameName,    PostData,
    Headers); end;
```

Метод **Navigate**. Этот метод дает команду WebBrowser начать навигацию к указанному ресурсу. Синтаксис этого метода следующий: **procedure** Navigate(**const** URL: WideString; **var** Flags: OleVariant; **var** TargetFrameName: OleVariant; **var** PostData: OleVariant; **var** Headers: OleVariant); overload;

Здесь **URL** – адрес навигации.

Для значения **Flags** определены такие константы:

navOpenInNewWindow (1) – открывает URL в новом окне браузера по умолчанию. Как открыть URL в новом экземпляре нашего браузера будет описано ниже при описании обработчика OnNewWindow2.

navNoHistory (2) – не заносит адрес в список History.

navNoReadFromCache (4) – не использует сохраненную в кеше страницу, а загружает с сервера.

navNoWriteToCache (8) – не записывает страницу в дисковый кеш.

navAllowAutosearch (16) – если браузер не может найти указанный домен, он передает его в поисковый механизм.

TargetFrameName – определяет целевой фрейм по имени. Если присвоить system.NULL (или имя несуществующего фрейма) страница загрузится в текущий браузер.

PostData – определяет данные для передачи на сервер. **Headers** – определяет HTTP-хедер для передачи на сервер.

property Busy – если это свойство активно (равно True), то это свидетельствует о том, что браузер еще не закончил загрузки страницы или выполняет некоторую команду. И он проигнорирует новую команду. Так что в этом случае лучше подождать, когда это

свойство станет равным false (или, когда идет загрузка, то остановить ее можно с помощью метода Stop).

Теперь несколько слов о событиях, которые сопровождают процесс загрузки. Они, как отмечалось выше, есть своего рода «продолжение» соответствующих методов **DWebBrowserEvents2**. Наиболее существенными из них являются события, возникающие для каждого фрейма:

OnBeforeNavigate2

Возникает при попытке начать навигацию. Из параметров наиболее существенным является **pDisp: IDispatch**. Этот параметр определяет браузер, который начинает навигацию. Для многофреймового документа этот браузер может не соответствовать браузеру верхнего уровня. К сожалению, этот обработчик не запрашивается при вызове метода Refresh.

OnNavigateComplete2

Возникает, когда попытка навигации была успешной. Наблюдения за WebBrowser позволяют сделать предположение, что это событие возникает после того, как с сервера придет первая порция данных. Документ еще продолжает загружаться.

OnDocumentComplete

Возникает при окончании загрузки в независимости от того, был ли документ загружен полностью или нет. К сожалению, нет простого критерия для определения того, была ли страница загружена полностью или нет.

OnNewWindow2

Возникает при попытке открыть документ в новом окне. Если Вы хотите, чтобы документ был открыт в Вашем экземпляре браузера, то Вам нужно создать свой экземпляр браузера и параметру **ppDisp** присвоить интерфейсную ссылку на этот экземпляр.

Ниже приведены примеры использования компонента для доступа к информации, размещенной на web-странице.

Как получить текст HTML документа из TWebBrowser? uses

```
..., MSHTML;  
procedure TForm1.FormCreate(Sender: TObject); begin  
  WebBrowser1.Navigate('http://decoding.narod.ru'); end;  
procedure TForm1.Button1Click(Sender: TObject); var  
  Document: IHTMLDocument2; begin  
  Document := WebBrowser1.Document as IHTMLDocument2;  
  if Document <> nil then  
    Memo1.Text :=  
    (Document.all.Item(NULL, 0) as IHTMLDocument2).OuterHTML; end;
```

Как получить текст HTML документа из TWebBrowser без тегов? uses

```
..., MSHTML, ActiveX;  
procedure TForm1.FormCreate(Sender: TObject); begin  
  WebBrowser1.Navigate('http://decoding.narod.ru'); end;  
procedure GetHtmlCode(WebBrowser: TWebBrowser; FileName: string); var  
  htmlDoc: IHTMLDocument2;  
  PersistFile: IPersistFile; begin  
  htmlDoc := WebBrowser.document as IHTMLDocument2;  
  PersistFile := HTMLDoc as IPersistFile;  
  PersistFile.Save(StringToOleStr(FileName), true); end;
```

```
procedure TForm1.Button1Click(Sender: TObject); begin  
    GetHtmlCode(WebBrowser1, 'C:\HTMLDoc.txt'); end;
```

Содержание задания:

Создать и протестировать программу, реализующую вывод информации о курсе валюты на выбранную пользователем дату. Для получения информации о курсе валюты на любую дату использовать любой доступный web-ресурс, например, <http://conv.rbc.ru/convert.shtm>. Вид валюты выбирается из вариантов заданий. Для создания программы используется среда программирования Turbo Explorer либо Turbo C++.

Варианты заданий:

1. Доллар США.
2. ЕВРО.
3. Австралийский доллар.
4. Английский фунт стерлингов.
5. Японская йена.
6. Китайский юань.
7. Украинская гривна.

Порядок выполнения работы:

1. Изучить компоненты и методы для программного доступа в интернет.
2. Изучить процедуры и функции для доступа в интернет.
3. Ответить на контрольные вопросы.
4. Получить у преподавателя номер варианта задания.
5. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Для чего программам нужен доступ в интернет?
2. Перечислить компоненты и методы, используемые для доступа в интернет в выбранной вами среде разработки.
3. Перечислить возможные ошибки, которые могут появиться при попытке получения информации из интернета.
4. Перечислить протоколы, используемые в интернете. Каким протоколом вы пользовались в программе для получения необходимой информации?

Лабораторная работа №4

Создание программ с функцией автоматического обновления

Цель и содержание: изучить способы защиты программных продуктов от несанкционированного копирования. Получить навыки создания программ с защитой от копирования.

Теоретические сведения:

Зачем нужно производить обновления?

Программы постоянно дорабатываются и совершенствуются в соответствии с пожеланиями пользователей. С каждой новой версией в них добавляются новые возможности, устраняются недочеты, повышается уровень совместимости с другим программным обеспечением. Поэтому каждая следующая версия программы устраняет возможные ошибки в работе и повышает её эффективность.

Программы можно обновлять путем деинсталлирования и, затем, полной установки новой версии. Либо скачав на диск или USB-драйв набор файлов, называемый «пакетом обновления».

Наиболее удобным для пользователя способом является автоматическая проверка программой при запуске наличия своей свежей версии в интернет. При этом в случае наличия обновления пользователю предлагается установить его.

Обычно, в таком случае, обновления представляют собой exe-файл программы в архиве и файлы библиотек – dll. Они скачиваются во временную папку. После этого обновляемая программа выгружается из памяти. Стартует простая служебная утилита, которая заменяет файлы ПО на новые.

И, затем, запускает обновленную программу.

Информация о новой версии программы хранится на сервере, и сравнивается со служебной информацией, записанной в реестре Windows в разделе программы, либо в ini-файле.

Что такое Реестр?

Реестр – это иерархически построенная, централизованная база данных в составе операционных систем Microsoft Windows 9x/2000/XP/2003/Vista/7, содержащая сведения, которые используются операционной системой для работы с пользователями, программными продуктами и устройствами.

В реестре хранятся данные, которые необходимы для правильного функционирования Windows. К ним относятся профили всех пользователей, сведения об установленном программном обеспечении и типах файлов, которые могут быть созданы каждой программой, информация о свойствах папок и значках приложений, а также установленном оборудовании и используемых портах.

Системный реестр заменяет собой большинство текстовых ini-файлов, которые использовались в Windows 3.x, а также файлы конфигурации MS-DOS (например, Autoexec.bat и Config.sys).

Версии реестра для разных версий операционных систем семейства Windows имеют определенные различия.

Во время запуска операционной системы происходит до тысячи обращений к *Реестру*, а во время работы на ПК в течение одного сеанса работы – до 10 тысяч!

Отдельные компоненты *Реестра* хранятся в оперативной памяти ПК в течение всего сеанса работы.

Запись (считывание) информации в *Реестр* (из *Реестра*) происходит постоянно: например, если мы устанавливаем какую-нибудь программу, вся информация, необходимая для запуска и работы этой программы, записывается в *Реестр*. Если мы устанавливаем новое устройство, в *Реестре* будет отмечено, где находится его драйвер и т.д. Если же мы запускаем какую-либо программу или устройство, то из *Реестра* считывается вся необходимая для запуска программы (устройства) информация.

Значение Реестра

Значение *Реестра* для Windows трудно переоценить – это «святое святых» операционной системы, ее сердце. От корректности данных *Реестра* зависит эффективность работы как программного обеспечения (операционной системы и прикладных программ), так и аппаратной части ПК. С помощью *Реестра* можно заставить ПК или работать с максимально возможным быстродействием, или «тормозить».

Появление всевозможных неисправностей в работе ОС говорит о том, что какие-то настройки *Реестра* стали некорректными. При серьезном повреждении *Реестра* операционную систему загрузить невозможно. Поэтому вирусы зачастую стараются испортить *Реестр* (если их цель – «завалить» ОС) или заблокировать доступ к *Реестру* пользователя (чтобы затруднить обнаружение и удаление вируса).

Что представляет собой Реестр и где он хранится?

Реестр Windows состоит из пяти ветвей:

1) HKEY_CLASSES_ROOT (HKCR) – в этой ветви содержатся сведения о расширениях всех зарегистрированных в системе типов файлов (хранящиеся здесь сведения отвечают за запуск необходимой программы при открытии файла с помощью *Проводника Windows*);

2) HKEY_CURRENT_USER (HKCU) – в этой ветви содержится информация о пользователе, вошедшем в систему в настоящий момент (здесь хранятся папки пользователя, цвета экрана и параметры панели управления); 3) HKEY_LOCAL_MACHINE (HKLM) – в этой ветви содержится информация об аппаратной части ПК, о драйверах устройств, сведения о загрузке Windows;

4) HKEY_USERS (HKU) – в этой ветви содержится информация о всех активных загруженных профилях пользователей данного ПК;

5) HKEY_CURRENT_CONFIG (HKCC) – в этой ветви содержится информация о профиле оборудования, используемом локальным компьютером при запуске системы.

Реестр Windows хранится в папке **\Windows\System32\config** в двоичных файлах.

Работа с реестром в Turbo Explorer

В Turbo Explorer есть объект TRegistry, при помощи которого можно работать с реестром. Реестр предназначен для хранения системных переменных и позволяет регистрировать файлы программы, что обеспечивает их показ в проводнике с соответствующей иконкой, вызов программы при щелчке на этом файле, добавление ряда команд в меню, вызываемое при нажатии правой кнопки мыши над файлом. Кроме того, в реестр можно внести некую свою информацию (переменные, константы, данные об установленной программе). Программу можно добавить в список деинсталляции, что позволит удалить ее из менеджера «Установка/Удаление программ» панели управления.

Для работы с реестром применяется ряд функций API:

RegCreateKey (Key: HKey; SubKey: PChar; var Result: HKey): Longint; Создать подраздел в реестре. Key указывает на «корневой» раздел реестра, в SubKey – имя раздела – строится по принципу пути к файлу в DOS (пример subkey1\subkey2\ ...). Если такой раздел уже существует, то он открывается (в любом случае при успешном вызове Result

содержит Handle на раздел). Об успешности вызова судят по возвращаемому значению, если ERROR_SUCCESS, то успешно, если иное – ошибка.

RegOpenKey(Key: HKey; SubKey: PChar; var Result: HKey): Longint;

Открыть подраздел Key\SubKey и возвращает Handle на него в переменной Result. Если раздела с таким именем нет, то он не создается. Возврат – код ошибки или ERROR_SUCCESS, если успешно.

RegCloseKey(Key: HKey): Longint;

Закрывает раздел, на который ссылается Key. Возврат – код ошибки или ERROR_SUCCESS, если успешно.

RegDeleteKey(Key: HKey; SubKey: PChar): Longint;

Удалить подраздел Key\SubKey. Возврат – код ошибки или ERROR_SUCCESS, если нет ошибок.

RegEnumKey(Key: HKey; index: Longint; Buffer: PChar; cb: Longint): Longint;

Получить имена всех подразделов раздела Key, где Key – Handle на открытый или созданный раздел (см. RegCreateKey и RegOpenKey), Buffer – указатель на буфер, cb – размер буфера, index – индекс, должен быть равен 0 при первом вызове RegEnumKey. Типичное использование – в цикле While, где index увеличивается до тех пор, пока очередной вызов RegEnumKey не завершится ошибкой (см. пример).

RegQueryValue(Key: HKey; SubKey: PChar; Value: PChar; var cb: Longint): Longint;

Возвращает текстовую строку, связанную с ключом Key\SubKey. Value – буфер для строки; cb – размер, на входе – размер буфера, на выходе – длина возвращаемой строки. Возврат – код ошибки.

RegSetValue(Key: HKey; SubKey: PChar; ValType: Longint; Value: PChar; cb: Longint): Longint;

Задать новое значение ключу Key\SubKey, ValType – тип задаваемой переменной, Value – буфер для переменной, cb – размер буфера. В Windows

3.1 допустимо только Value=REG_SZ. Возврат – код ошибки или ERROR_SUCCESS, если нет ошибок.

Примеры:

```
{Создаем список всех подразделов указанного раздела} procedure
TForm1.Button1Click(Sender: TObject); var
  MyKey: HKey; {Handle для работы с разделом}
  Buffer: array[0..1000] of char; {Буфер} Err, {Код ошибки}
  index: longint; {Индекс подраздела} begin
  Err:=RegOpenKey(HKEY_CLASSES_ROOT,'DelphiUnit',MyKey); {Открыли раздел}
  if Err<> ERROR_SUCCESS then begin
    MessageDlg('Нет такого раздела !!',mtError,[mbOk],0); exit; end; index:=0;
    {Определили имя первого подраздела}
    Err:=RegEnumKey(MyKey,index,Buffer,Sizeof(Buffer)); while err=ERROR_SUCCESS
  do {Цикл, пока есть подразделы} begin
    memo1.lines.add(StrPas(Buffer)); {Добавим имя подраздела в список} inc(index);
    {Увеличим номер подраздела}
    Err:=RegEnumKey(MyKey,index,Buffer,Sizeof(Buffer)); {Запрос} end;
    RegCloseKey(MyKey); {Закрыли подраздел} end;
Объект INIFILES – работа с INI файлами.
```

Почему иногда лучше использовать ini-файлы, а не реестр?

1. Ini-файлы можно просмотреть и отредактировать в обычном блокноте.
2. Если ini-файл хранить в папке с программой, то при переносе папки на другой компьютер настройки сохраняются.
3. Новичку в реестре можно запросто запутаться или испортить полезную информацию.

Для хранения параметров настройки программы удобно использовать стандартные ini-файлы Windows. Работа с ini-файлами ведется при помощи объекта TIniFiles модуля IniFiles. Краткое описание методов объекта TIniFiles дано ниже.

Constructor Create('d:\test.INI');

Создать экземпляр объекта и связать его с файлом. Если такого файла нет, то он создается, но только тогда, когда произведете в него запись информации.

WriteBool(const Section, Ident: string; Value: Boolean);

Присвоить элементу с именем Ident раздела Section значение типа boolean.

WriteInteger(const Section, Ident: string; Value: Longint);

Присвоить элементу с именем Ident раздела Section значение типа Longint.

WriteString(const Section, Ident, Value: string);

Присвоить элементу с именем Ident раздела Section значение типа String.

ReadSection (const Section: string; Strings: TStrings);

Прочитать имена всех корректно описанных переменных раздела Section (некорректно описанные опускаются).

ReadSectionValues(const Section: string; Strings: TStrings);

Прочитать имена и значения всех корректно описанных переменных раздела Section. Формат : имя_переменной = значение.

EraseSection(const Section: string);

Удалить раздел Section со всем содержимым.

ReadBool(const Section, Ident: string; Default: Boolean): Boolean;

Прочитать значение переменной типа Boolean раздела Section с именем Ident, и если его нет, то вместо него подставить значение Default.

ReadInteger(const Section, Ident: string; Default: Longint): Longint; Прочитать значение переменной типа Longint раздела Section с именем Ident, и если его нет, то вместо него подставить значение Default.

ReadString(const Section, Ident, Default: string): string;

Прочитать значение переменной типа String раздела Section с именем Ident, и если его нет, то вместо него подставить значение Default.

Free;

Закрыть и освободить ресурс. Необходимо вызвать при завершении работы с ini-файлом.

Property Values[const Name: string]: string;

Доступ к существующему параметру по имени Name.

Пример:

```
Procedure TForm1.FormClose(Sender: TObject); var
```

```
IniFile:TIniFile; begin
```

```
IniFile := TIniFile.Create('d:\test.INI'); {Создали экземпляр объекта}
```

```
IniFile.WriteBool('Options', 'Sound', True); {Секция Options: Sound:=true}
```

```
IniFile.WriteInteger('Options', 'Level', 3); {Секция Options: Level:=3}
```

```
IniFile.WriteString('Options', 'Secret password', Pass);
```

```
{Секция Options: в Secret password записать значение переменной Pass}
```

```

IniFile.ReadSection('Options ', memo1.lines); {Читаем имена переменных}
IniFile.ReadSectionValues('Options ', memo2.lines); {Читаем имена и значения}
IniFile.Free; {Закрыли файл, уничтожили объект и освободили память} end;

```

Загрузка файлов из интернет в Turbo Explorer. Пример функции, реализующей скачивание файла.

Turbo Explorer предоставляет полный доступ к WinInet API (wininet.pas), который можно использовать для соединения и получения файлов с web-сайта, который использует либо Hypertext Transfer Protocol (HTTP), либо File Transfer Protocol (FTP).

```

uses Wininet;
function GetInetFile(const fileURL, FileName: string): boolean; const
  BufferSize = 1024; var
  hSession, hURL: HInternet;
  Buffer: array[1..BufferSize] of Byte; BufferLen: DWORD;
  f: file;
  sAppName: string; begin
  Result := False;
  sAppName := ExtractFileName(Application.ExeName); hSession :=
InternetOpen(PChar(sAppName), INTERNET_OPEN_TYPE_PRECONFIG, nil, nil, 0); try
  hURL := InternetOpenURL(hSession, PChar(fileURL), nil, 0, 0, 0); try
  AssignFile(f, FileName); Rewrite(f,1); repeat
  InternetReadFile(hURL, @Buffer, SizeOf(Buffer), BufferLen);
  BlockWrite(f, Buffer, BufferLen); until
  BufferLen = 0;
  CloseFile(f);
  Result := True;
  finally InternetCloseHandle(hURL); end; finally
  InternetCloseHandle(hSession); end; end;

```

Как перед скачиванием узнать размер файла?

```

GetUrlInfo(HTTP_QUERY_CONTENT_LENGTH, 'http://in.com/in.zip'); function
GetUrlInfo(const dwInfoLevel: DWORD; const FileURL: string): string; var
  hSession, hFile: HInternet; dwBuffer: Pointer; dwBufferLen, dwIndex: DWORD;
begin
  Result := "";
  hSession := InternetOpen('STEROID Download',
    INTERNET_OPEN_TYPE_PRECONFIG, nil, nil, 0); if
Assigned(hSession) then begin
  hFile := InternetOpenURL(hSession, PChar(FileURL), nil, 0,
INTERNET_FLAG_RELOAD, 0); dwIndex := 0; dwBufferLen := 20;
  if HttpQueryInfo(hFile, dwInfoLevel, @dwBuffer, dwBufferLen, dwIndex) then
  Result := PChar(@dwBuffer);
  if Assigned(hFile) then InternetCloseHandle(hFile); InternetCloseHandle(hsession);
end; end;

```

Варианты заданий:

1. Программа обновляется через интернет, данные о версии хранятся в ini-файле.
2. Программа обновляется через интернет, данные о версии хранятся в реестре.

3. Программа обновляется через интернет и через диалог выбора файла, данные о версии хранятся любым способом, на ваш выбор.

4. Программа обновляется через интернет, перед обновлением выдается информация о размере файла новой версии. Данные о версии хранятся любым способом, на ваш выбор.

Порядок выполнения работы:

1. Изучить методы работы с файлами.
2. Изучить методы хранения информации в реестре и ini-файлах.
3. Изучить процедуры и функции для передачи файлов через интернет.
4. Ответить на контрольные вопросы.
5. Получить у преподавателя номер варианта задания.
6. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Что такое версия и номер сборки программы?
2. Зачем необходимо обновлять программное обеспечение.
3. Перечислить способы обновления программного обеспечения.
4. Какие протоколы можно использовать для передачи файлов в интернет? Каким протоколом для передачи файлов вы пользовались в программе?

Лабораторная работа №5

Использование компонентов для работы с графикой

Цель и содержание: изучить компоненты для работы с графикой. Изучить методы формирования графических примитивов. Получить навыки создания программ, использующих графические компоненты.

Теоретические сведения:

Для рисования 2D графики Windows использует GDI (Graphics Device Interface). GDI обычно не используют для создания особых графических эффектов, для этого есть DirectX, OpenGL, или любые графические библиотеки (такие как: DelphiX, FastLib, DIBUltra, Graphics32). Однако для создания простых эффектов с минимальными усилиями, GDI вполне подходит. Рассмотрим работу с GDI на примере Turbo Explorer.

С GDI тесно связана ещё одна аббревиатура – DC («Device Context» – контекст устройства). Это то, на чём мы рисуем. В Turbo Explorer контекст устройства представлен как TCanvas. Идея контекста устройства заключается в том, что это универсальное устройство вывода, поэтому можно использовать одинаковые функции как для экрана, так и для принтера. Все графические функции в Turbo Explorer являются надстройками над стандартными GDI функциями Windows.

Классы GDI:

ИМЯ	ОПИСАНИЕ
Pen	Используется для рисования простых линий. Обычно применяется для функции LineTo или при рисовании рамки для определённой фигуры (например, для функции Rectangle).
Brush	Кисть используется для заполнения области определённым цветом. Применяется в функциях Rectangle, FillRect или FloodFill.
Font	Используется для задания шрифта, которым будет напечатан текст. Можно указать название шрифта, размер и т.д.
Region	Позволяет задать регион (замкнутое пространство). Регионом может быть круг, квадрат или произвольная фигура. Позволяет так же делать дырки в фигурах.

Рисование линий

Координата (0,0) – это верхний левый угол экрана. То есть значения по оси y увеличиваются вниз экрана. Соответственно, координата (0, 50) означает, что мы отступили на 50 пикселей от верха экрана.

Перо (Pen) используется при рисовании линий или рамок, а кисть (Brush) – для заполнения фигуры.

Ниже приведены две функции, которые используются для рисования линий и обе принадлежат TCanvas:

ИМЯ	ОПИСАНИЕ	ПРИМЕР
MoveTo	Перемещает точку начала рисования линии в указанные координаты x и y	Canvas.MoveTo(50, 100);
LineTo	Рисует линию, начиная с текущей позиции (см. MoveTo) до указанных координат x и y.	Canvas.LineTo(50, 100);

Эффект перемещения точки начала рисования линии так же достигается при помощи установки свойства PenPos на Canvas. Например, «Canvas.PenPos.x:=20;», «Canvas.PenPos.y:= 50», или «Canvas.PenPos:= Point(20,50);».

По умолчанию, точка начала рисования установлена в (0,0), то есть, если сразу вызвать «Canvas.LineTo(100,100);» то будет нарисована линия из точки (0,0) в точку (100, 100). Точка начала рисования автоматически переместится в (100, 100), то есть, если выполнить команду «Canvas.LineTo(200, 100);», то следующая линия будет нарисована из точки (100, 100) в (200, 100). Поэтому, если мы хотим рисовать линии несоединённые друг с другом, то придётся воспользоваться методом MoveTo.

Линия, нарисованная при помощи LineTo, использует текущее перо канваса (типа TPen). Основные свойства пера, это ширина – «Canvas.Pen.Width := 4;» (при помощи которого можно задавать различную ширину линий), и цвет «Canvas.Pen.Color := clLime;».

Рисование фигур

Для рисования фигур в TCanvas предусмотрены следующие функции:

ИМЯ	ОПИСАНИЕ	ПРИМЕР
Ellipse	Рисует эллипс, вписанный в невидимый квадрат с координатами верхнего левого угла и правого нижнего. Если координаты x и y углов будут совпадать, то получится круг.	Canvas.Ellipse(0,0,50,50);
FillRect	Заполняет прямоугольник цветом текущей кисти (brush), но никак не за пределами него.	Canvas.FillRect(Bounds(0,0,100,100));
FloodFill	Заполняет данную область цветом текущей кисти до тех пор, пока не будет достигнут край.	Canvas.FloodFill(10, 10, clBlack, fsBorder);
Rectangle	Рисует прямоугольник (или квадрат), заполненный цветом текущей кисти и обрамлённый цветом текущего пера	Canvas.Rectangle(Bounds(20, 20, 50, 50));

RoundRect	Тоже, что и Rectangle, но с закруглёнными углами.	Canvas.RoundRect(20, 20, 50, 50, 3, 3);
TextOut	Рисует данную строку на Canvas, начиная с координат (x,y) – фон текста заполняется текущим цветом кисти.	Canvas.TextOut(10, 10, 'Some text');

Функция позволяет рисовать текст, не заполняя его фон. Если Вам необходимо изменить шрифт, используемый в TextOut, то необходимо изменить свойство Font (это свойство имеет тип TFont) – например «Canvas.Font.Name:='Verdana';», «Canvas.Font.Size:=24;» или «Canvas.Font.Color:=clRed;».

Перерисовка окна

Рисунок сохраняется до тех пор, пока окно (форма) не было обновлено.

Перерисовка несколько отличается от понятия "рисование". Когда окну необходимо перерисоваться, то Windows посылает определённое сообщение. Это сообщение поступает в обработчик события "OnPaint". Любой код, который поместить в обработчик OnPaint будет вызван каждый раз, когда форме необходимо обновиться.

Что такое Bitmap?

Bitmap – это графический объект, который содержит заголовок, необходимую информацию о картинке (такую как высота, ширина, цвета и т.д.) и, собственно, само изображение (большой массив, содержащий цвет каждой точки). В Turbo Explorer для этой цели уже предусмотрен класс TBitmap.

Bitmap можно рисовать не только на форме, но и по всему экрану. Может это и может показаться немного странным, но иногда это бывает полезно, особенно при создании скринсейвера. Однако сначала нам необходимо разобраться с тем, как работать с bitmap. Вот небольшой пример:

```
procedure Form1.DrawBitmap(const Filename: string; const x, y:
Integer);
var
  Bmp: TBitmap; begin
  if not FileExists(Filename) then begin
    ShowMessage('The bitmap ' + Filename + ' was not found!');
    Exit; end;
  Bmp := TBitmap.Create;
  try
    Bmp.LoadFromFile(Filename); Canvas.Draw(x, y, Bmp); finally
    Bmp.Free; end; end;
```

Эта функция пытается загрузить и показать картинку (с именем Filename, например 'myBitmap.bmp'), начиная с точки (x,y).

Функции рисования в GDI

TCanvas имеет несколько полезных функций, которые работают с типом TGraphic. Тип TGraphic является базовым классом для графических объектов в Turbo Explorer, таких как: битмапы (TBitmap), иконки (TIcon), метафайлы (TMetafile) и JPEG-и (TJPEGImage). Все они используют одни и те же функции, которые приведены в таблице:

ИМЯ	ОПИСАНИЕ	ПРИМЕР ИСПОЛЬЗОВАНИЯ
Draw	Рисует TGraphic на канвасе так как он есть, не растягивая.	Canvas.Draw(5,10,MyGraphic);
StretchDraw	Рисует TGraphic на Canvas, подгоняя (растягивая) его под заданную область.	Canvas.StretchDraw(Bounds (0,0,32,32), MyGraphic);
CopyRect	Копирует часть Canvas в другую, при необходимости растягивая его.	Canvas.CopyRect(Bounds (0,0,32,32), MyBmp.Canvas, Bounds(0, 0, 640, 480));

Все эти функции являются методами TCanvas.

Описанный выше способ позволяет рисовать bitmap в run-time. Также можно поместить на форму компонент TImage и установить в нем картинку.

Пример использования:

```
procedure TForm1.Button1Click(Sender: TObject); begin
  if opendialog1.execute then
    image1.Picture.LoadFromFile(opendialog1.filename); end;
```

Для изменения графической информации, например, для создания игры, потребуется несколько раз в секунду перерисовывать объекты на Canvas или перемещать Image. Для отслеживания интервалов и запуска при этом пользовательских процедур, предназначен компонент TTimer.

TTimer создает событие OnTimer, интервал которых задан в свойстве Interval. Этот интервал задается в миллисекундах и работает относительно точно. Тем не менее не рекомендуется его применение при точных вычислениях. С помощью свойства Enabled счетчик можно запустить (true) или остановить (false).

Варианты заданий:

1. Разработать программу, реализующую игру «Тетрис».
2. Разработать программу, реализующую игру «Ксоникс».
3. Разработать программу, реализующую игру «Арканоид».

Порядок выполнения работы:

1. Изучить компоненты для работы с графикой.
2. Изучить методы формирования графических примитивов.
3. Изучить процедуры и функции для изменения графической информации.
4. Изучить работу с таймером (TTimer).
5. Ответить на контрольные вопросы.
6. Получить у преподавателя номер варианта задания.
7. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Перечислите компоненты, используемые для работы с графикой.
2. Что такое графические примитивы?
3. Какие функции реализуют работу с графическими примитивами?
4. Какие графические форматы файлов можно использовать в программе для вывода заранее сформированных и сохраненных изображений?
5. Для чего нужен в программе компонент «таймер»?

Лабораторная работа №6

Создание приложения для работы с базами данных

Цель и содержание: изучить компоненты для работы с базами данных. Изучить методы открытия баз данных, выполнения запросов. Получить навыки создания программ для управления базами данных.

Теоретические сведения:

База данных – некоторый набор постоянно хранимых данных, используемых прикладными программными системами.

Система управления базами данных (СУБД) – специализированная программа (чаще комплекс программ), предназначенная для организации и ведения базы данных.

Для создания, модификации и управления данными в реляционных базах данных **применяется SQL** (*Structured Query Language* – язык структурированных запросов) – универсальный компьютерный язык.

С помощью запроса можно выполнить следующие виды обработки данных:

- включить в результирующую таблицу запроса заданные пользователем поля;
- выбрать записи, удовлетворяющие условиям отбора;
- произвести вычисления в каждой из полученных записей;
- сгруппировать записи, которые имеют одинаковые значения в одном или нескольких полях, в одну запись и одновременно для других полей образовавшихся групп выполнить одну из статистических функций;
- произвести обновление полей в выбранном подмножестве записей;
- создать новую таблицу базы данных, используя данные из существующих таблиц;
- удалить выбранное подмножество записей из таблицы базы данных;
- добавить выбранное подмножество записей в другую таблицу.

Рассмотрим компоненты для создания доступа к базе данных и работы с ней, используя интерфейс ADO и язык программирования Turbo Explorer. **ADO** (*ActiveX Data Objects* – «объекты данных ActiveX») – интерфейс программирования приложений для доступа к данным, разработанный компанией Microsoft (MS Access, MS SQL Server) и основанный на технологии компонентов ActiveX. ADO позволяет представлять данные из разнообразных источников (баз данных) в объектно-ориентированном виде. Существуют версии указанных программ с бесплатной лицензией – Microsoft Access RunTime и MS SQL Server Express Edition.

Для работы с ADO на вкладке компонентов ADO есть шесть компонентов: TADODConnection, TADODCommand, TADODDataSet, TADODTable, TADODQuery, TADODStoredProc.

TADODConnection – для указания базы данных и работы с транзакциями.

TADOTable – таблица доступная через ADO.

TADOQuery – запрос к базе данных на языке SQL. Это может быть как запрос, в результате которого возвращаются данные и базы (например, SELECT), так и запрос, не возвращающий данных (например, INSERT).

TADOStoredProc – вызов хранимой процедуры. Хранимые процедуры в ADO могут возвращать набор данных.

TADOCommand и TADODataSet являются наиболее общими компонентами для работы с ADO, но и наиболее сложными в работе. Оба компонента позволяют выполнять команды на языке провайдера данных (так в ADO называется драйвер базы данных).

Разница между ними в том, что команда, исполняемая через TADODataSet, должна возвращать набор данных и этот компонент позволяет работать с ними средствами Turbo Explorer (например, привязать компонент типа TDataSource). А компонент TADOCommand позволяет исполнять команды, не возвращающие набор данных, но не имеет штатных средств для последующего использования возвращенного набора данных.

Очевидно, что все компоненты должны связываться с базой данных. Делается это двумя способами либо через компонент TADOConnection либо прямым указанием базы данных в остальных компонентах. Остальные компоненты к TADOConnection привязываются с помощью свойства Connection, или к базе данных напрямую – через свойствоConnectionString.

База данных может быть указана двумя способами: через файл настроек соединения к данным (файл в формате Microsoft Data Link, расширение UDL) либо прямым заданием параметров соединения.

Значения свойств всех ConnectionString этих компонент могут быть введены напрямую в текстовой форме, но куда проще вызвать редактор свойства, нажав на кнопку «...» в конце поля ввода.

Для отображения информации, полученной с помощью запросов и таблиц, используются компоненты DBGrid, DBEdit, DBMemo и т.д.

Для вывода на экран либо принтер таблицы или результатов запроса используется отчет. Его вид определяется генератором отчетов. На данный момент существует множество генераторов отчетов – RaveReports, FastReports, CrystalReports, QReports и т.д. Как правило, одна из версий генератора отчетов поставляется вместе с системой разработки приложений. Например, в комплекте с Turbo Explorer может идти RaveReports. Как правило, отчет позволяет располагать данные на виртуальном листе бумаги, с любой сортировкой, группировкой, различными вариантами оформления. Также доступен экспорт отчетов в файлы различных форматов.

Варианты заданий:

1. Домашняя бухгалтерия, группировка данных по дням, отчеты по категориям расходов и доходов, прогноз доходов и расходов на текущий месяц.
2. Ежедневник-планировщик.
3. Домашняя библиотека.
4. Домашняя видеотека.
5. База данных – календарь праздников и именин.

Порядок выполнения работы:

1. Изучить компоненты для работы с базами данных.
2. Изучить методы работы с запросами.

3. Изучить методы работы с отчетами.
4. Ответить на контрольные вопросы.
5. Получить у преподавателя номер варианта задания.
6. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Перечислите компоненты, используемые для работы с базами данных.
2. Что такое запрос к базе данных?
3. Перечислите виды запросов к базе данных?
4. Что такое отчет, для каких целей используется в программе?
5. Предложите варианты развития вашей программы, какие функции в неё можно добавить?

Лабораторная работа №7

Использование диаграмм и бизнес-графики

Цель и содержание: изучить компоненты для работы с диаграммами. Изучить способы отображения данных, виды диаграмм. Получить навыки создания программ для презентаций и демонстраций данных.

Теоретические сведения:

Деловая графика

Возможности деловой графики преследуют единую цель: улучшить восприятие информации человеком, сделать ее более наглядной и выразительной. Например, если на персональном компьютере установлен цветной дисплей, можно все картинки выдавать в цвете, причем цвета вы выбираете сами. Картинки можно запоминать в памяти компьютера и затем в нужный момент вызывать на экран, или сделать их копию на листе бумаги, воспользовавшись принтером.

Основной целью деловой графики, как уже было сказано выше, является оперативная выдача информации в графической форме в процессе анализа задачи при ее решении средствами электронной таблицы. Такая постановка задачи не предъявляет слишком высоких требований к качеству выдаваемых на экран картинок. Главным критерием здесь является быстрота подготовки и отображения графических образов, соответствующих оперативно изменяющейся числовой информации.

Диаграмма – это графическое представление числовых данных, позволяющее быстро оценить соотношение нескольких величин. Представляет собой двухмерное геометрическое символьное изображение информации, с применением различных приёмов техники визуализации.

Иногда, для оформления диаграмм используется трёхмерная визуализация, спроецированная на плоскость, что придаёт диаграмме отличительные черты или позволяет иметь общее представление области, в которой она применяется. Например: финансовая диаграмма, связанная с денежными суммами, может представлять собой количество купюр в пачке или монет в стопке; диаграмма сравнения количества подвижного состава – различную длину изображённых поездов и т. д.

В различных процессорах графопостроения (графических редакторах и электронных таблицах, при изменении данных, на основе которых построена диаграмма, она будет автоматически перестроена с учётом внесённых изменений в таблицу исходных данных диаграммы. Это позволяет быстро сравнивать различные показатели, статистические данные и т. д. – можно вводить новые данные и сразу видеть изменения диаграммы.

Преимущества диаграмм

Преимущество диаграмм перед другими типами наглядной статистической информации заключается в том, что они позволяют быстро произвести логический вывод из большого количества полученных данных. Результаты расчётов, выполненных с помощью систем статистических вычислений, заносятся в таблицы. Они являются основой для последующего анализа или для подготовки статистического отчёта.

Сами по себе цифры в этих таблицах не являются достаточно наглядными, а если их много, не производят достаточного впечатления. Кроме того, графическое изображение позволяет осуществить контроль достоверности полученных данных, так как на графике достаточно ярко проявляются возможные неточности, которые могут быть связаны с ошибками на каком-либо этапе проведения исследования. В основном, все статистические

пакеты, позволяют графически предоставить полученную числовую информацию в виде различных диаграмм, а затем, если это необходимо, перенести их в текстовый редактор для сборки окончательного варианта статистического отчёта.

Основные типы диаграмм

Диаграммы в основном состоят из геометрических объектов (точек, линий, фигур различной формы и различных цветов) и вспомогательных элементов (осей координат, условных обозначений, заголовков и т. п.). Также диаграммы делятся на плоскостные или двухмерные, и пространственные (трёхмерные или объёмные). Сравнение и сопоставление геометрических объектов на диаграммах может происходить по различным измерениям: по площади фигуры или её высоте, по местонахождению точек, по их густоте, по интенсивности цвета и т. д. Кроме того, данные могут быть представлены в прямоугольной или полярной системе координат.

Пример использования в Turbo Explorer

Для отображения диаграмм предназначены компоненты TChart (данные – Tseries или TLineSeries) и TDBChart (используют данные из таблицы или запроса базы данных). Данные в series добавляются с помощью процедуры AddXY(x,y).

Варианты заданий:

1. Демонстрация изменения температуры воздуха.
2. График биржевой активности.
3. Демонстрация изменения активности пользователя (использование клавиатуры).
4. График изменения количества покупателей в супермаркете.
5. Демонстрация изменения активности пользователя (использование мыши).

Порядок выполнения работы:

1. Изучить компоненты для работы с диаграммами.
2. Изучить виды диаграмм.
3. Ответить на контрольные вопросы.
4. Получить у преподавателя номер варианта задания.
5. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Перечислите компоненты, используемые для работы с диаграммами.
2. Перечислите основные виды диаграмм.
3. Как в вашей программе обеспечивается динамическое изменение диаграмм при изменении исходных данных?

Лабораторная работа №8

Использование динамических библиотек

Цель и содержание: изучить назначение и методы использования динамических библиотек. Получить навыки создания и использования динамических библиотек.

Теоретические сведения:

DLL (Dynamic-link library – динамически подключаемая библиотека) – понятие операционной систем Microsoft Windows; динамическая библиотека, позволяющая многократное применение различными программными приложениями. К DLL относятся также элементы управления ActiveX и драйверы. В мире UNIX аналогичные функции выполняют так называемые shared objects («разделяемые объекты»).

Формат файлов DLL придерживается тех же соглашений, что и формат исполняемых файлов, включая в себя код, таблицы и ресурсы.

Цели введения DLL

Первоначально предполагалось, что введение DLL позволит эффективно организовать память и дисковое пространство, используя только один экземпляр библиотечного модуля для различных приложений. Это было особенно важно для ранних версий Microsoft Windows с жёсткими ограничениями по памяти.

Далее, предполагалось улучшить эффективность разработок и использования системных средств за счёт модульности. Замена DLL-программ с одной версии на другую должна была позволить независимо наращивать систему, не затрагивая приложений. Кроме того, библиотеки DLL могли использоваться разнотипными приложениями – например, Microsoft Office,

Microsoft Visual Studio и т. п.

Фактически, полных преимуществ от внедрения DLL получить не удалось по причине явления, называемого DLL hell («ад DLL»). DLL hell возникает, когда несколько приложений требуют одновременно различные, не полностью совместимые, версии DLL-библиотек, что приводит к сбоям в этих приложениях. Когда система выросла до определённых размеров, количество DLL стало превышать многие тысячи, не все из них обладали полной надёжностью и совместимостью, и конфликты типа DLL hell стали возникать очень часто, резко понижая общую надёжность системы. Поздние версии Microsoft Windows стали разрешать параллельное использование разных версий DLL, что свело на нет преимущества изначального принципа модульности.

Примеры использования DLL в Turbo Explorer Экспорт в DLL library Example;

```
{Производим некие действия и возвращаем результат} function AddNumbers(a, b: Double): Double; cdecl; begin
```

```
    AddNumbers := a + b end;
```

```
{Эта функция экспортируется} exports
```

```
    AddNumbers;
```

```
{Инициализатор DLL, аналог функции main() в языках C и C++} begin end.
```

Импорт из DLL program Example;

```
{$APPTYPE CONSOLE}
```

```
{Указываем, что функция находится в DLL}
```

```
function AddNumbers(a, b: Double): Double; cdecl; external 'Example.dll';
```

```
{Используем функцию как обычно}
```

```
var result: Double; begin
```

```
    result := AddNumbers(1, 2); Writeln('The result was: ', result) end.
```

```

Подгрузка и импорт из DLL во время работы program Example;
{$APPTYPE CONSOLE}
var result: Double;
    LibHandle: Hmodule;
    AddNumbers: function(a, b: Double): Double; begin
    LibHandle := LoadLibrary('Example.dll'); // загружаем DLL
if LibHandle <> 0 then begin
    @AddNumbers := GetProcAddress(LibHandle, 'AddNumbers');
    // запоминаем адрес необходимой функции
    if @AddNumbers <> nil then
        begin
            result := AddNumbers(1, 2);      // используем функцию
result was: ', result);
        end;
    FreeLibrary(LibHandle);      // выгружаем DLL
end; end.
Writeln('The

```

Варианты заданий:

1. Динамическая графическая заставка на весь экран.
2. Диаграмма по передаваемому набору значений.
3. Обновление указанного файла с сайта в интернет.
4. Помещение передаваемых данных в базу данных.
5. Проверка имени компьютера.
6. Проверка номера жесткого диска.
7. Вывод текстовой информации по запросу кодового слова.

Порядок выполнения работы:

1. Изучить методы создания динамических библиотек.
2. Изучить методы использования функций из состава динамических библиотек.
3. Ответить на контрольные вопросы.
4. Получить у преподавателя номер варианта задания.
5. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Что такое динамическая библиотека?
2. Перечислите цели создания и использования динамических библиотек.
3. Какие сложности могут возникнуть при использовании нескольких однотипных библиотек разных версий?
4. Как в вашей программе обеспечивается использование функций из состава динамической библиотеки?

Лабораторная работа №9

Создание многопоточных приложений

Цель и содержание: изучить назначение и особенности многопоточных приложений. Получить навыки создания и использования программ для многопоточной обработки данных.

Теоретические сведения:

Многопоточность – свойство платформы (например, операционной системы, JVM и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Такие потоки называют также потоками выполнения (от англ. thread of execution); иногда называют «нитеями» (буквальный перевод англ. thread) или неформально «тредами».

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

К достоинствам многопоточности в программировании можно отнести следующее:

1. Упрощение программы в некоторых случаях, за счет использования общего адресного пространства.
2. Меньшие относительно процесса временные затраты на создание потока.
3. Повышение производительности процесса за счет распараллеливания процессорных вычислений и операций ввода/вывода.

Типы реализации потоков

Поток в пространстве пользователя. Каждый процесс имеет таблицу потоков, аналогичную таблице процессов ядра.

Достоинства и недостатки этого типа следующие:

Достоинства:

1. Возможность реализации на ядре, не поддерживающем многопоточность.
2. Более быстрое переключение, создание и завершение потоков.
3. Процесс может иметь собственный алгоритм планирования.

Недостатки:

1. Отсутствие прерывания по таймеру внутри одного процесса.
2. При использовании блокирующего системного запроса для процесса все его потоки блокируются.
3. Сложность реализации.

Поток в пространстве ядра. Наряду с таблицей процессов в пространстве ядра имеется таблица потоков.

«Волокна» (fibers). Несколько потоков режима пользователя, исполняющихся в одном потоке режима ядра. Поток пространства ядра потребляет заметные ресурсы, в первую очередь физическую память и диапазон адресов режима ядра для стека режима ядра. Поэтому было введено понятие «волокон» – облегчённого потока, выполняемого исключительно в режиме пользователя. У каждого потока может быть несколько «волокон».

Взаимодействие потоков

В многопоточной среде часто возникают проблемы, связанные с использованием параллельно исполняемыми потоками одних и тех же данных или устройств. Для решения подобных проблем используются такие методы взаимодействия потоков, как взаимное исключение (мьютексы), семафоры, критические секции и события.

Взаимоисключения (mutex, мьютекс) – это объект синхронизации, который устанавливается в особое сигнальное состояние, когда не занят каким-либо потоком. Только один поток владеет этим объектом в любой момент времени, отсюда и название таких объектов (от английского mutually exclusive access – взаимно исключающий доступ) – одновременный доступ к общему ресурсу исключается. После всех необходимых действий мьютекс освобождается, предоставляя другим потокам доступ к общему ресурсу.

Семафоры представляют собой доступные ресурсы, которые могут быть приобретены несколькими потоками в одно и то же время, пока пул ресурсов не опустеет. Тогда дополнительные потоки должны ждать, пока требуемое количество ресурсов не будет снова доступно. Семафоры очень эффективны, поскольку они позволяют одновременный доступ к ресурсам.

События. Объект, хранящий в себе 1 бит информации «просигнализован или нет», над которым определены операции «просигнализировать», «сбросить в непросигнализованное состояние» и «ожидать». Ожидание на просигнализованном событии есть отсутствие операции с немедленным продолжением исполнения потока. Ожидание на непросигнализованном событии приводит к приостановке исполнения потока до тех пор, пока другой поток (или же вторая фаза обработчика прерывания в ядре ОС) не просигнализирует событие. Возможно ожидание нескольких событий в режимах «любого» или «всех». Возможно также создания события, автоматически сбрасываемого в непросигнализованное состояние после пробуждения первого же – и единственного – ожидающего потока (такой объект используется как основа для реализации объекта «критическая секция»). Активно используются в MS Windows, как в режиме пользователя, так и в режиме ядра.

Критические секции обеспечивают синхронизацию подобно мьютексам за исключением того, что объекты, представляющие критические секции, доступны в пределах одного процесса. События, мьютексы и семафоры также можно использовать в однопроцессном приложении, однако, реализации критических секций в некоторых ОС (например, Windows NT), обеспечивают более быстрый и более эффективный механизм взаимно-исключающей синхронизации – операции «получить» и «освободить» на критической секции оптимизированы для случая единственного потока (отсутствия конкуренции) с целью избежать любых ведущих в ядро ОС системных вызовов. Подобно мьютексам объект, представляющий критическую секцию, может использоваться только одним потоком в данный момент времени, что делает их крайне полезными при разграничении доступа к общим ресурсам.

Условные переменные (condvars). Сходны с событиями, но не являются объектами, занимающими память, – используется только адрес переменной, понятие

«содержимое переменной» не существует, в качестве условной переменной может использоваться адрес произвольного объекта. В отличие от событий, установка условной переменной в просигнализированное состояние не влечет за собой никаких последствий в случае, если на данный момент нет потоков, ожидающих на переменной. Установка события в аналогичном случае влечет за собой запоминание состояния «просигнализировано» внутри самого события, после чего следующие потоки, желающие ожидать события, продолжают исполнение немедленно без остановки. Для полноценного использования такого объекта необходима также операция «освободить mutex и ожидать условную переменную атомарно». Активно используются в UNIX-подобных ОС. Дискуссии о преимуществах и недостатках событий и условных переменных являются заметной частью дискуссий о преимуществах и недостатках Windows и UNIX.

Порт завершения ввода-вывода (IO completion port, IOCP). Реализованный в ядре ОС и доступный через системные вызовы объект «очередь» с операциями «поместить структуру в хвост очереди» и «взять следующую структуру с головы очереди» – последний вызов приостанавливает исполнение потока в случае, если очередь пуста, и до тех пор, пока другой поток не осуществит вызов «поместить». Главнейшей особенностью IOCP является то, что структуры в него могут помещаться не только явным системным вызовом из режима пользователя, но и неявно внутри ядра ОС как результат завершения асинхронной операции ввода-вывода на одной из дескрипторов файлов. Для достижения такого эффекта необходимо использовать системный вызов «связать дескриптор файла с IOCP». В этом случае помещенная в очередь структура содержит в себе код ошибки операции ввода-вывода, а также, для случая успеха этой операции – число реально введенных или выведенных байт. Реализация порта завершения также ограничивает число потоков, исполняющихся на одном процессоре/ядре после получения структуры из очереди. Объект специфичен для MS Windows и позволяет осуществлять обработку входящих запросов соединения и порций данных в серверном программном обеспечении в архитектуре, где число потоков может быть меньше числа клиентов (нет требования создавать отдельный поток с расходами ресурсов на него для каждого нового клиента).

Пример работы с TThread в Turbo Explorer:

```

файл unit1.pas unit Unit1; interface uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Unit2, StdCtrls; type
TForm1 = class(TForm)   Button1: TButton;
  procedure Button1Click(Sender: TObject);   procedure FormCreate(Sender: TObject);
private
  {Private declarations}   protected
    FMyThread: TMyThread;   public
    {Public declarations}   end; var
Form1: TForm1;
StartPosX,
StartPosY: Integer; implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject); begin
  FMyThread := TMyThread.Create(False);
  // запускает новый поток при нажатии на кнопку end;
procedure TForm1.FormCreate(Sender: TObject); begin
  StartPosX := 50; // стартовая позиция надписи   StartPosY := 50; end; end.

```

файл unit2.pas unit Unit2; interface uses

Classes, StdCtrls; type

TMyThread = class(TThread) private

{Private declarations} protected

procedure Execute; override; procedure UpdateCaption; end;

implementation uses Unit1;

{Important: Methods and properties of objects in VCL can only be used in a method called using Synchronize, for example, Synchronize(UpdateCaption); and UpdateCaption could look like,

```
    procedure TMyThread.UpdateCaption; begin
        Form1.Caption := 'Updated in a thread'; end;}
{TMyThread} procedure TMyThread.Execute; begin
    {Place thread code here}
    Synchronize(UpdateCaption); // действие потока end;
procedure TMyThread.UpdateCaption; var lblThread: TLabel; begin
    if Form1.Caption <> 'Main program window' then StartPosY := StartPosY + 20; if
StartPosY > 220 then begin
        StartPosX := StartPosX + 120; StartPosY := 50 end;
        Form1.Caption := 'New thread started'; lblThread := TLabel.Create(Form1); with
lblThread do begin Left := StartPosX;
            Top := StartPosY;
            Parent := Form1;
            Visible := True;
            Enabled := True;
            Caption := 'Created by Thread'; end end; end.
```

Варианты заданий:

1. Фигуры, перемещающиеся в окне приложения.
2. Многопоточный поиск файлов на дисках.
3. Подбор пароля к файлу *.rar.
4. Подбор пароля к файлу *.zip.

Порядок выполнения работы:

1. Изучить назначение и особенности многопоточных приложений.
2. Изучить методы использования потоков при создании программ.
3. Ответить на контрольные вопросы.
4. Получить у преподавателя номер варианта задания.
5. Разработать и отладить программу.

Содержание отчета:

1. Распечатка исходного текста программы с комментариями.
2. Демонстрация работы программы.

Контрольные вопросы:

1. Что такое многопоточное приложение?
2. Перечислите плюсы использования многопоточности в программе.

3. Как взаимодействуют потоки в программе?
4. Каким образом в вашей программе обеспечивается использование многопоточности?

Список литературы

Основная литература:

1. Смирнов А.А. Прикладное программное обеспечение: учебно-практическое пособие / А.А. Смирнов. – М.: Изд. Центр ЕАОИ, 2011. – 384 с.
2. Привалов И. М. Основы аппаратного и программного обеспечения: учебное пособие / И. М. Привалов. – Ставрополь: Северо-Кавказский федеральный университет, 2015. – 145 с.

Дополнительная литература:

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. – СПб.: Питер, 2015. – 1120 с.: ил. – (Серия «Классика computer science»).
2. Приходченко О.В. Прикладное программное обеспечение профессиональной деятельности: учебное пособие / О.В. Приходченко, А.А. Просолович. – Комсомольск-на-Амуре: ФГБОУ ВПО «КНАГТУ», 2012. – 158 с.
3. А.Ю. Щербаков. Современная компьютерная безопасность. Теоретические основы, практические аспекты. Учебное пособие. – М.: Книжный мир, 2009. – 352 с.
4. Пальмов, С. В. Методы и средства моделирования программного обеспечения: конспект лекций / С. В. Пальмов. – Самара: Поволжский государственный университет телекоммуникаций и информатики, 2016. – 105 с.

Методическая литература:

1. Влацкая, И. В. Проектирование и реализация прикладного программного обеспечения: учебное пособие / И. В. Влацкая, Н. А. Заельская, Н. С. Надточий. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2015. – 119 с.
2. Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие / Т. М. Зубкова. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017. – 469 с.

Интернет-ресурсы:

1. База знаний и набор вычислительных алгоритмов WolframAlpha <https://www.wolframalpha.com>
2. Лекториум – Онлайн-курсы и медиатека видеолекций на русском языке <https://lektorium.tv/>
3. Электронно-библиотечная система IPRbooks <https://iprbooks.ru/>
4. Википедия – Свободная энциклопедия <https://ru.wikipedia.org/>

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по проведению практических занятий
по дисциплине «Системное и прикладное программное обеспечение»
для студентов специальности 10.05.01 Компьютерная безопасность
специализация «Информационно-аналитическая и техническая экспертиза
компьютерных систем»

Ставрополь
2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	49
Практическое занятие №1.....	53
Практическое занятие №2.....	56
Практическое занятие №3.....	58
Практическое занятие №4.....	60
Практическое занятие №5.....	72
Практическое занятие №6.....	73
Практическое занятие №7.....	75
Практическое занятие №8.....	90
Практическое занятие №9.....	92
Практическое занятие №10.....	96
Практическое занятие №11.....	104
Практическое занятие №12.....	107
Практическое занятие №13.....	109
Практическое занятие №14.....	110
Практическое занятие №15.....	112
Практическое занятие №16.....	115
Практическое занятие №17.....	118
Практическое занятие №18.....	120
Практическое занятие №19.....	122
Практическое занятие № 20.....	124
Практическое занятие № 21.....	133
Практическое занятие № 22.....	136
Практическое занятие № 23.....	142
Практическое занятие № 24.....	149
Практическое занятие №25.....	161
Практическое занятие №26.....	172
Практическое занятие №27.....	174
Практическое занятие №28.....	177
Практическое занятие №29.....	181
Практическое занятие №30.....	185
Практическое занятие №31.....	187
Практическое занятие № 32.....	191
Практическое занятие № 33.....	199
Практическое занятие №34.....	201
Практическое занятие №35.....	210

Практическое занятие №36.....	215
Практическое занятие №37.....	216
Практическое занятие №38.....	218
Практическое занятие №39.....	220
Практическое занятие №40.....	228
Практическое занятие №41.....	238
Практическое занятие №42.....	250
Список литературы.....	265

ВВЕДЕНИЕ

Цели и задачи освоения дисциплины

Целью изучения дисциплины «Системное и прикладное программное обеспечение» является формирование набора общепрофессиональных компетенций студента специальности 10.05.01 – Компьютерная безопасность (специализация «Информационно-аналитическая и техническая экспертиза компьютерных систем»).

Задачи освоения дисциплины «Системное и прикладное программное обеспечение»:

- изучить способы классификации программного обеспечения, виды программного обеспечения с точки зрения назначения, внутреннего устройства, целевого оборудования, принципов разработки, лицензирования и распространения;
- изучить принципы построения, разработки и использования системного и прикладного программного обеспечения;
- сформировать базовые теоретические знания и практические навыки, необходимые для обеспечения безопасности программного обеспечения и обрабатываемой с его помощью информации.

Для освоения дисциплины, обучающиеся должны владеть следующими знаниями и компетенциями:

владеть навыками разработки программного обеспечения, приобретенными в ходе освоения дисциплин «Методы программирования», «Языки программирования»;

владеть знаниями об устройстве, назначении и принципах функционирования операционных систем, приобретенными в ходе освоения дисциплины «Операционные системы»;

знать материалы курса информатики.

Компетенции обучающегося, формируемые в результате изучения дисциплины:

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ОПК-2 Способен применять программные средства системного и прикладного назначения, в том числе отечественного производства, для решения задач профессиональной деятельности	ИД-2 ОПК-2 применяет программные средства системного и прикладного назначения, в том числе отечественного производства, при решении задач профессиональной деятельности	Способен применять системное и прикладное программное обеспечение для решения рабочих задач. Способен выбирать наиболее подходящие по совокупности различных критериев программные продукты для использования в профессиональной деятельности
ОПК-12 Способен администрировать операционные системы и выполнять работы по восстановлению работоспособности прикладного и системного программного обеспечения	ИД-1 ОПК-12 использует системные приложения для управления настройками операционных систем	Владеет навыками применения системных и прикладных приложений для решения задач администрирования операционных систем и обеспечения корректной работы программного и аппаратного обеспечения ЭВМ

НАИМЕНОВАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

№	Наименование тем дисциплины, их краткое содержание	Объем часов	Из них практическая подготовка, часов
7 семестр			
1. Системное программное обеспечение			
1.	Настройка BIOS / UEFI персонального компьютера	1,5	
2.	Загрузчики операционных систем	1,5	
3.	Управление модулями ядра Linux	1,5	
4.	Дисковые утилиты GNU/Linux и Windows	1,5	
5.	Установка и настройка систем программирования	1,5	
6.	Введение в программирование с использованием WinAPI	1,5	
7.	Работа с файлами	1,5	
8.	Файловые системы FAT	1,5	
9.	Сложные обработчики и взаимодействие резидентных программ	1,5	
10.	Приложения Windows с использованием WinAPI	1,5	
11.	Создание и использование элементов управления	1,5	
12.	Графический ввод-вывод в оконном приложении	1,5	
13.	Динамическое отображение данных на окне	1,5	
14.	Обмен сообщениями между окнами, обработка сообщений ввода-вывода	1,5	
15.	Использование потоков	1,5	
16.	Синхронизация доступа к ресурсам	1,5	
17.	Приоритеты процессов и потоков	1,5	
18.	Реестр Windows	1,5	
	Итого за 7 семестр	27	
8 семестр			
2. Прикладное программное обеспечение			
19.	Прикладное программное обеспечение, реализующее типовые процедуры обработки экономической информации	1,5	
20.	Управление задачами в ОС Windows	1,5	
21.	Исследование блоков управления памятью	1,5	
22.	Диагностика IP-протокола	1,5	
23.	Управление устройствами ввода-вывода и файловыми системами в ОС Windows	1,5	
24.	Файлы пакетной обработки	1,5	
25.	Программирование на языке SHELL в ОС Unix	1,5	
26.	Процессы в ОС QNX	1,5	
27.	Потоки в ОС QNX	1,5	
28.	Обмен сообщениями	1,5	
29.	Таймер и периодические уведомления	1,5	
30.	Среда визуальной разработки программ PHOTON APPLICATION BUILDER - PHAB	1,5	
31.	Улучшение навыков программирования	1,5	
32.	Изучение структуры программы на ассемблере	1,5	
33.	Изучение функций ввода/вывода	1,5	

34.	Изучение арифметических и логических команд	1,5	
35.	Модульное программирование	1,5	
36.	Совершенствование навыков работы на языке ассемблера	1,5	
37.	Интерфейс с языками высокого уровня и обработка массивов	1,5	
38.	Использование цепочечных команд	1,5	
39.	Основы программирования VBA	1,5	
40.	Циклы и ветвления в языке VBA	1,5	
41.	Работа с символьными данными на языке VBA	1,5	
42.	Создание форм VBA	1,5	
	Итого за 8 семестр	36	
	Итого	63	

СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Практическое занятие №1 Настройка BIOS / UEFI персонального компьютера

Цель и содержание: изучение настроек BIOS персонального компьютера. Определение набора установленного оборудования и характеристик ПК.

Теоретические сведения

Утилита (англ. utility или tool) – вспомогательная компьютерная программа в составе общего программного обеспечения для выполнения специализированных типовых задач, связанных с работой оборудования и операционной системы (ОС).

Утилиты предоставляют доступ к возможностям (параметрам, настройкам, установкам), недоступным без их применения, либо делают процесс изменения некоторых параметров проще (автоматизируют его).

Утилиты могут входить в состав операционных систем, идти в комплекте со специализированным оборудованием или распространяться отдельно. BIOS

BIOS (Basic Input-Output System – базовая система ввода-вывода) – небольшая программа, находящаяся в постоянном запоминающем устройстве (ПЗУ) и отвечающая за самые базовые функции интерфейса и настройки оборудования, на котором она установлена. Другими словами, можно сказать, что BIOS – основа работы вычислительной системы, так как отвечает за самые базовые функции компьютера (аналогично системе рефлексов у человека).

Наиболее широко среди пользователей компьютеров известна BIOS материнской платы, но BIOS присутствуют почти у всех компонентов компьютера: у видеоадаптеров, сетевых адаптеров, модемов, дисковых контроллеров, принтеров. BIOS материнской платы отвечает за инициализацию (подготовку к работе), тестирование и запуск всех ее компонентов.

Операционная система и прикладные программы работают с аппаратным обеспечением компьютера посредством BIOS, которая переводит понятные пользователю команды операционной системы на язык, понятный компьютеру.

BIOS материнской платы

Физически BIOS – это набор микросхем постоянной памяти (ROM, Read Only Memory – только для чтения), расположенных на материнской плате. Поэтому микросхему иногда называют ROM BIOS.

Если заглянуть под крышку системного блока, то на материнской плате можно обнаружить микросхему с голографической наклейкой с надписью и логотипом, означающим производителя BIOS. Рядом обязательно будет круглый аккумулятор, питающий микросхему CMOS (Complementary Metal Oxide Semiconductor – энергозависимая память, применяемая для хранения установок BIOS). BIOS Setup Utility

Среди программ, содержащихся в BIOS, имеется программа настройки параметров BIOS Setup Utility, которая позволяет изменять данные, хранящиеся в памяти CMOS, с помощью системы меню.

Для обеспечения правильной работы операционной системы и прикладных программ с помощью BIOS Setup Utility вводятся параметры всех компонентов компьютера, начиная от оперативной памяти и рабочей частоты процессора и заканчивая режимом работы принтера и других периферийных устройств. Правильно настроив содержимое BIOS вашего компьютера, можно увеличить производительность его работы до 30%.

Замечание: неосторожные действия пользователя, как правило, не могут привести к физическому повреждению компьютера – он может лишь перестать загружаться. Это легко исправить. Современные BIOS имеют довольно обширные средства

автоконфигурирования, поэтому роль пользователя в установке «правильных» параметров можно свести к минимуму. В последнее время в программе установки параметров появился пункт «Загрузить оптимизированные параметры». Выбор этого пункта позволяет пользователю установить параметры «параметры по умолчанию» для имеющегося оборудования.

Как войти в BIOS Setup Utility

Программа установки параметров BIOS Setup Utility недоступна пользователю во время работы компьютера. Вход в BIOS Setup Utility обычно выполняется путем нажатия клавиши [Del] во время загрузки компьютера. Так же встречаются версии BIOS, вход в настройки которой выполняется с использованием других клавиш или их сочетаний.

В данной лабораторной работе для входа в BIOS будет использован наиболее распространенный вариант (клавиша [Del]).

Задания к работе

Задание 1. Узнать тип и версию BIOS/ UEFI.

Задание 2. Узнать дату создания BIOS /Uefi.

Задание 3. Установленный и максимально поддерживаемый размер памяти.

Задание 4. Определить параметры накопителей, подключенных к каналам стандартного IDE/SATA-контроллера.

Задание 5. Определить текущий порядок опроса накопителей при загрузке.

Задание 6. Изменить порядок опроса накопителей при загрузке так, чтобы в первую очередь опрашивался CDROM, затем жесткий диск. Остальные носители не опрашиваются.

Задание 7. На персональном компьютере запустить программу для определения характеристик устройств компьютера (например, программа CPU-Z) и найти следующие сведения:

- Производитель материнской платы
- Наименование материнской платы
- Форм-фактор
- Процессорный интерфейс
- Северный мост
- Южный мост
- Частота системной шины
- Тип оперативной памяти
- Количество слотов для ОП
- Максимальная пропускная способность ОП
- Максимальный объем оперативной памяти
- Количество слотов PCI
- Пропускная способность IDE
- Название поддерживаемого протокола для IDE
- Количество разъёмов USB
- Пропускная способность USB
- Наличие встроенной звуковой карты
- Наличие встроенной видеокарты
- Наличие встроенной сетевой карты
- Количество портов LPT
- Количество портов COM
- Количество портов PS/2
- Количество игровых портов
- Количество аудиоразъёмов

Порядок выполнения работы:

Изучить теоретическую часть.

Выполнить практические задания. Оформить отчет следующего содержания:

1. Тема.
2. Цель.
3. Практическое задание.

Вопросы для самоконтроля

1. Перечислите программы для тестирования материнской платы.
2. Какие основные производители чипсетов вам известны?
3. Дайте определение термину утилита.

Практическое занятие №2

Загрузчики операционных систем

Цель и содержание: изучение загрузчиков операционных систем, их назначения и функциональных возможностей.

Теоретические сведения

Загрузчик операционной системы – это программа, которая обеспечивает загрузку этой самой системы непосредственно после включения компьютера. Основная задача загрузчика – подготовить компьютер для загрузки ядра операционной системы (если, конечно, в такой предварительной подготовке есть какая-то необходимость), ну и, собственно, загрузить само ядро. В случае, если есть несколько ядер, доступных для загрузки, загрузчик нередко выполняет ещё и функцию средства диалога с пользователем, позволяя тому выбрать одно из доступных ядер.

Загрузчик операционной системы – системное программное обеспечение, обеспечивающее загрузку операционной системы непосредственно после включения компьютера.

Загрузчик операционной системы:

- 1) обеспечивает необходимые средства для диалога с пользователем компьютера (например, загрузчик позволяет выбрать ядро операционной системы для загрузки);
- 2) приводит аппаратуру компьютера в состояние, необходимое для старта ядра операционной системы (например, на не-x86 архитектурах перед запуском ядра загрузчик должен правильно настроить виртуальную память);
- 3) загружает ядро операционной системы в ОЗУ. Загрузка ядра операционной системы не обязательно происходит с НЖМД. Загрузчик может получать ядро по сети. Ядро может храниться в ПЗУ или загружаться через последовательные интерфейсы (это может пригодиться на ранней стадии отладки создаваемой компьютерной системы);
- 4) формирует параметры, передаваемые ядру операционной системы (например, ядру Linux передаются параметры, указывающие способ подключения корневой файловой системы);
- 5) передаёт управление ядру операционной системы.

На компьютерах архитектуры IBM PC запуск загрузчика осуществляется программным обеспечением BIOS, записанной в ПЗУ компьютера, после успешного окончания процедуры POST. Опишем процедуру, с помощью которой происходит загрузка с НЖМД IBM PC: BIOS производит чтение 512 байт первого сектора НЖМД (MBR) в ОЗУ по адресу 0x00007C00 (0x07C0:0x0000 в формате реального режима), затем прочитанному коду передаётся управление. Этот код читает и анализирует таблицу разделов жёсткого диска, а затем, в зависимости от вида загрузчика, либо передаёт управление загрузочному коду активного раздела жёсткого диска, либо самостоятельно загружает ядро с диска в оперативную память и передаёт ему управление. Первоначально загрузчик работает в режиме реальной адресации при отключенной адресной линии A20, что создает определенные трудности при написании загрузчиков. Распространённые загрузчики:

- NTLDR – загрузчик ядра Windows NT;
- Windows Boot Manager (bootmgr.exe, winload.exe) – загрузчик ядра Windows, начиная с Vista;
- LILO (Linux LOader) – загрузчик, в основном применяемый для загрузки ядра Linux;
- GRUB (Grand Unified Bootloader) – применяется для загрузки ядра Linux и Hurd (StartUp Manager);
- OS/2 BootManager – загрузчик ядра OS/2;

- RedBoot – загрузчик для встраиваемых систем;
- SILO (SPARC Improved bootLOader) – загрузчик Linux и Solaris для машин с архитектурой SPARC. Домашняя страница SILO;
- Loadlin – загружает Linux из под DOS или Windows;
- Syslinux – загружает Linux из под DOS или Windows;
- BOOTP – применяется для загрузки по сети;
- Yaboot – загрузчик Linux на PowerPC;
- BootX – загрузчик Mac OS X;
- Das U-Boot – универсальный загрузчик, применяется в основном во встраиваемых (embedded) системах;
- SyMon – универсальный загрузчик, независим от типа операционной системы.
- Acronis OS Selector – утилита прилагаемая к Acronis Disk Director Suite, поддерживает Windows и Linux, появляется перед каждой загрузкой системы, умеет копировать системы/

Существует также ряд достаточно известных в узких кругах загрузчиков для встраиваемых операционных систем – Das U-Boot, uMon, RedBoot, SyMon.

Задания к работе

Задание 1. Перечислить основные загрузчики Windows.

Задание 2. Перечислить основные загрузчики Linux и универсальные загрузчики.

Задание 3. Дать определение понятию «загрузчик операционной системы».

Порядок выполнения работы:

Изучить теоретическую часть.

Выполнить практические задания. Оформить отчет следующего содержания:

- Тема.
- Цель.
- Практическое задание.

Вопросы для самоконтроля

1. Назовите основные функции загрузчика ОС.
2. Опишите достоинства и недостатки 3-х загрузчиков на выбор.

Практическое занятие №3

Управление модулями ядра Linux

Цель и содержание: изучение модулей ядра Linux, принципов их подключения, отключения и настройки

Теоретические сведения

Модули ядра Linux собираются только под определенную версию ядра, есть способ запуска модуля независимо от версии ядра, если они совместимы с помощью dkms, но об этом мы поговорим позже.

Находятся все модули в папке /lib/modules/. Учитывая, что модули рассчитаны только для определенной версии ядра, то в этой папке создается отдельная подпапка, для каждой установленной в системе версии ядра.

Системы на ядре Linux обладают как свойственными для всех Unix-систем характеристиками, так и полностью уникальными особенностями.

Многозадачность предполагает, что в ядро встроена функция разделения процессорного времени, которая в порядке очередности выделяет временной отрезок для выполнения каждой конкретной задачи. Это дает возможность процессам выполняться независимо друг от друга.

Многопользовательский доступ заключается в том, что в системах Linux одновременно могут работать несколько пользователей. Для предоставления системных ресурсов в этом случае используются удаленные терминалы. Здесь подобно другим ОС каждый пользователь относится к какой-то группе, определяющей для него права доступа к файлам (чтение, запись, исполнение).

Использование страниц памяти – системная память в ОС Linux организована по страничному принципу. Выделяются страницы объемом по 4К, и как только начинается нехватка оперативной памяти, система начинает искать свободные страницы для их перемещения на жесткий диск. Эти перемещенные участки затем по мере необходимости восстанавливаются.

Использование модулей ядра по требованию – в оперативную память загружаются только необходимые в данный момент компоненты ядра. Неиспользуемые модули хранятся на основном носителе.

Динамическое кэширование - память может кэшироваться на диск. В случае необходимости в освобождении места на носителе эта память динамически уменьшается.

Работа с программами, разработанными под другие ОС - имеется возможность запускать в Linux программы, созданные для Windows различных версий. Для этого используются соответствующие эмуляторы.

Соответствие стандарту POSIX 1003.1 - также частично поддерживаются стандарты BSD и SystemV.

Поддержка нескольких файловых систем - Linux может работать с MINIX, Xenix и SystemV. Нативные файловые системы Linux поддерживают работу с файлами объемом до 4 ТБ, имеющими имена длиной до 255 символов.

Поддержка разделов OS/2 FAT - В Linux можно получить доступ к файловым системам DOS и VFAT.

Для начала изучим главные преимущества операционной системы Linux.

- Каждый дистрибутив Linux имеет свой собственный репозиторий, представляющий собой хранилище всего доступного ПО.
- Правильно настроенную систему можно эксплуатировать годами без перезагрузок. При этом она будет работать стабильно и не зависнет.
- Linux славится своей безопасностью в силу полной защищенности от вирусов. Любое важное и потенциально опасное действие должно быть подтверждено

администратором. Тогда как для Windows каждый год разрабатываются новые, все более разрушительные вирусы, что вынуждает пользователя запускать тяжелые антивирусные системы и постоянно сканировать жесткий диск.

- Открытый исходный код предоставляет пользователю поистине неограниченные возможности по тонкой настройке системы под собственные нужды.

- В работающей системе происходит гораздо меньше сбоев, если сравнивать с ОС Windows.

- Linux также отличается меньшим потреблением ресурсов компьютера по сравнению с другими системами.

- ОС на базе Linux можно запускать прямо с флэш-накопителя без предварительной установки на жесткий диск. Это позволяет использовать систему, например, для устранения критических ошибок на имеющемся носителе.

- В Линуксе отсутствует привычный пользователям Windows реестр. Соответственно, отпадает надобность его чистить. Здесь все конфигурационные файлы располагаются в каталоге /etc.

- Подавляющее большинство дистрибутивов Линукс не «следит» за пользователями (не собирает их данные), в отличие от той же Windows.

- Для системы энтузиастами написано огромное количество бесплатных программ. Впрочем, платные продукты также имеются.

Поиск необходимого модуля

Модули хранятся в каталоге `"/lib/modules/<версия ядра>"` в виде файлов с расширением «ko». Для получения списка всех модулей из дерева можно выполнить команду поиска всех файлов с расширением «ko» в каталоге с модулями текущего ядра:

```
find /lib/modules/`uname -r` -name '*.ko'
```

Полученный список даст некоторое представление о доступных модулях, их назначении и именах. Например, путь «`kernel/drivers/net/wireless/rt2x00/rt73usb.ko`» явно указывает на то, что этот модуль – драйвер устройства беспроводной связи на базе чипа `rt73`.

Задания к работе

Задание 1. Загрузить модуль ядра на выбор

Задание 2. Удалить добавленный модуль ядра

Задание 3. Вызвать перечень всех модулей

Порядок выполнения работы:

Изучить теоретическую часть.

Выполнить практические задания. Оформить отчет следующего содержания:

- Тема.
- Цель.
- Практическое задание.

Вопросы для самоконтроля

1. Дайте определение понятию «модуль ядра»
2. Назовите виды модулей ядра.
3. Опишите функциональное назначение модулей ядра.

Практическое занятие №4

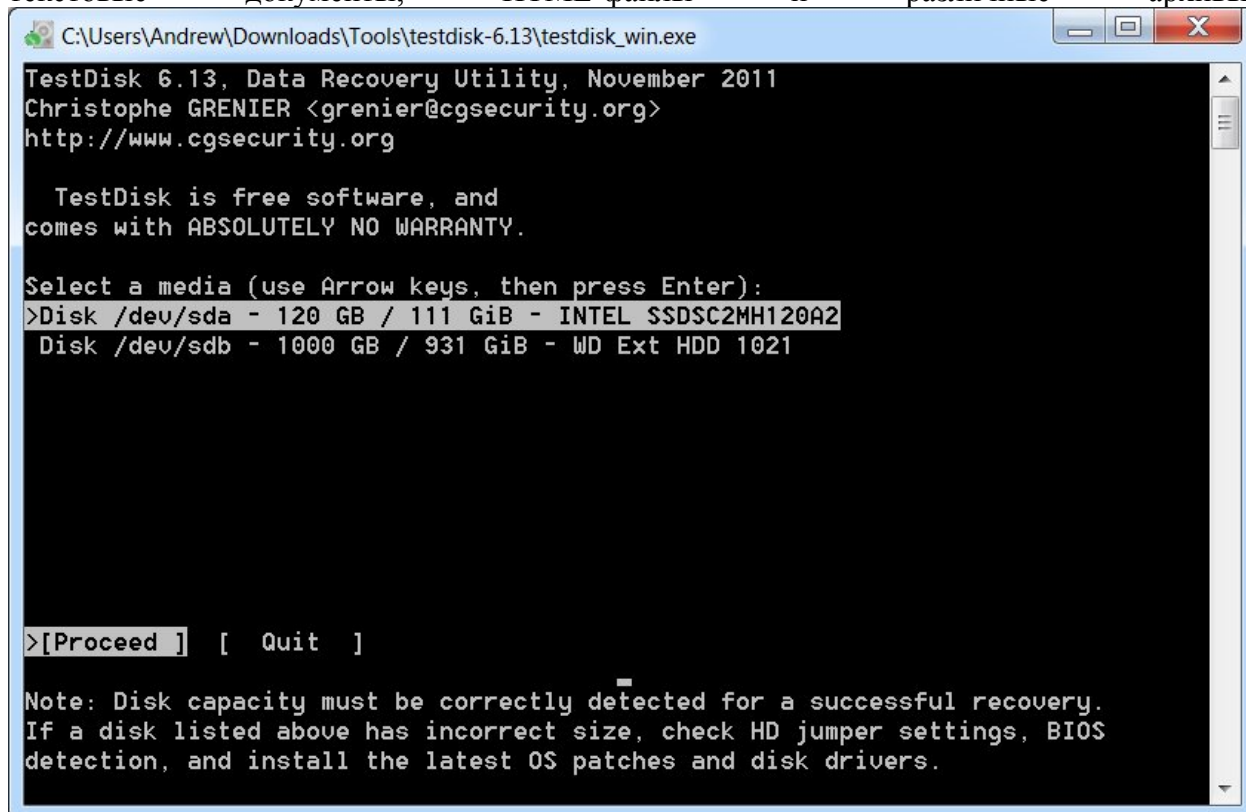
Дисковые утилиты GNU/Linux и Windows

Цель и содержание: изучение утилит, позволяющих управлять устройствами хранения данных, в том числе осуществлять их обслуживание.

Теоретические сведения

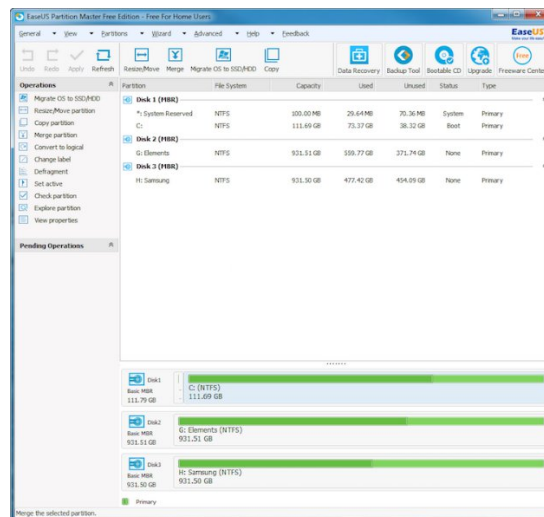
1. **TestDisk** позволяет восстанавливать загрузочные разделы, удаленные разделы, фиксировать поврежденные таблицы разделов и восстанавливать данные, а также создавать копии файлов с удаленных/недоступных разделов.

Примечание: PhotoRec это связанное с TestDisk приложением. С его помощью возможно восстановить данные в памяти цифровой камеры на жестких дисках и компакт-дисках. Кроме того, можно восстановить основные форматы изображений, аудиофайлы, текстовые документы, HTML-файлы и различные архивы.



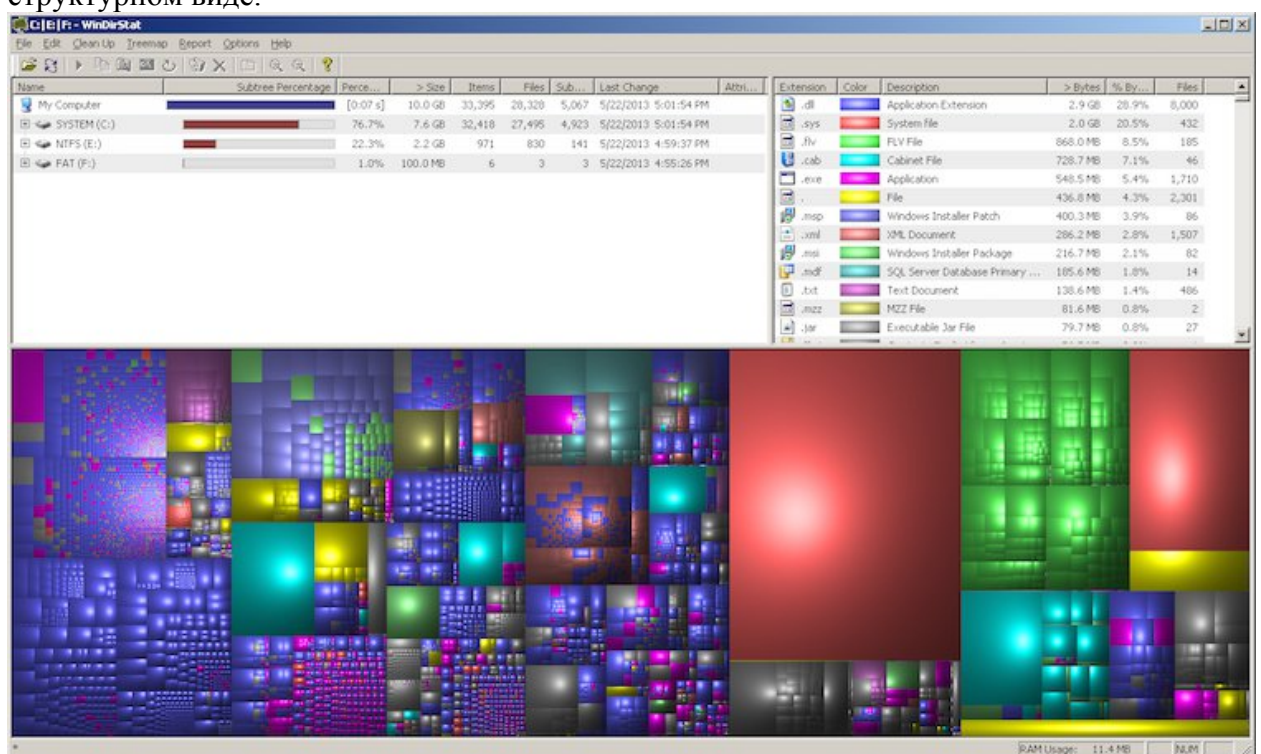
При запуске TestDisk предоставляется список разделов жесткого диска, с которыми можно работать. Выбор доступных действий, осуществляемых в разделах, включает: анализ для корректировки структуры (и последующее восстановление, в случае обнаружения проблемы); изменение дисковой геометрии; удаление всех данных в таблице разделов; восстановление загрузочного раздела; перечисление и копирование файлов; восстановление удаленных файлов; создание снимка раздела.

2. **EaseUS Partition Master** – инструмент для работы с разделами жесткого диска. Он позволяет создавать, перемещать, объединять, разделять, форматировать, изменяя их размер и расположение без потери данных. Также помогает восстанавливать удаленные или потерянные данные, проверять разделы, перемещать ОС на другой HDD/SSD и т.д.



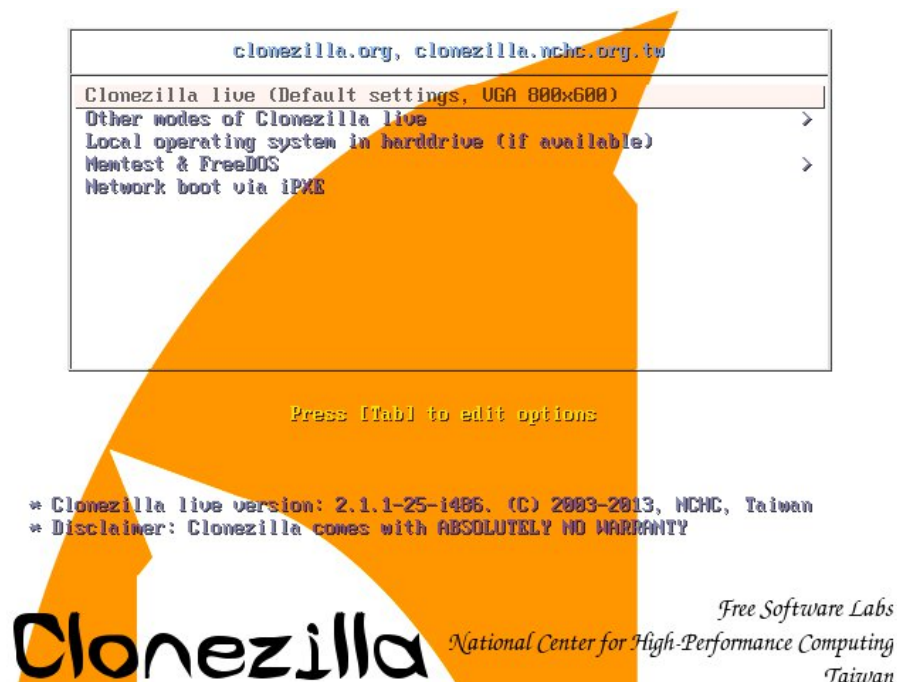
Слева представлен перечень операций, которые можно выполнить с выбранным разделом.

3. WinDirStat. Бесплатная программа WinDirStat проводит анализ использованного места на диске. Демонстрирует, как данные распределяются и какие из них занимают больше места. Клик по полю в диаграмме выведет на экран рассматриваемый файл в структурном виде.



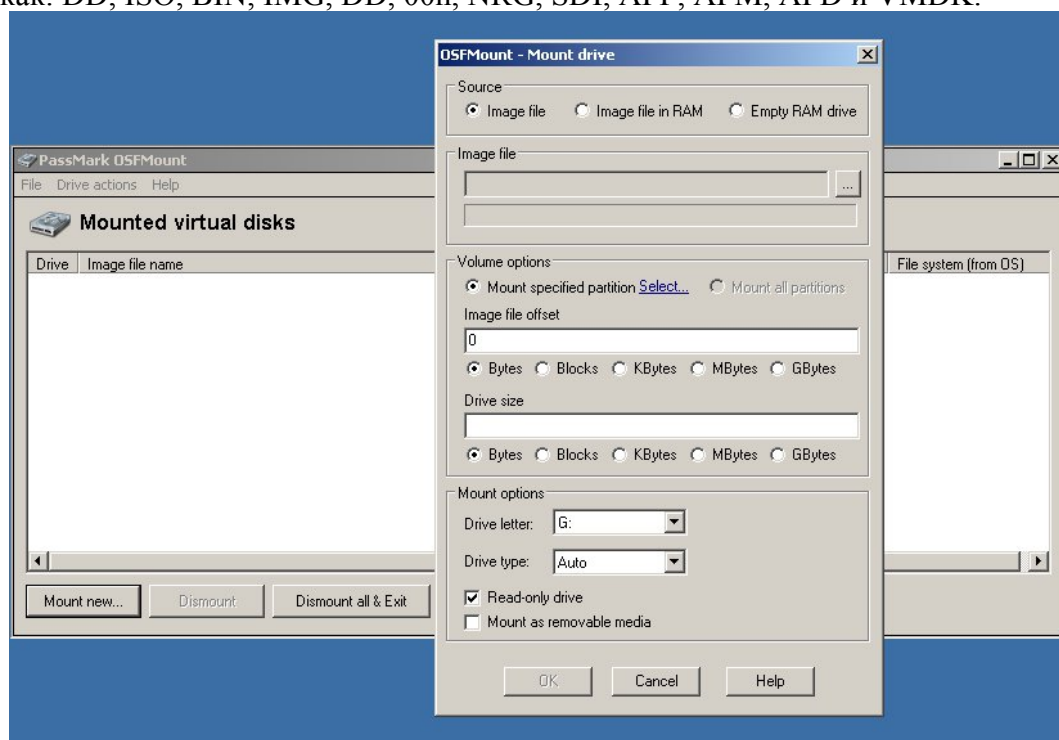
После загрузки WinDirStat и выбора дисков для анализа, программа сканирует дерево каталогов и предоставляет статистику в таких вариантах: список каталогов; карта каталогов; список расширений.

4. Clonezilla создает образ диска с инструментом клонирования, который также упакован с Parted Magic и первоначально доступен, как автономный инструмент. Представлен в двух версиях: Clonezilla Live и Clonezilla SE (Server Edition).



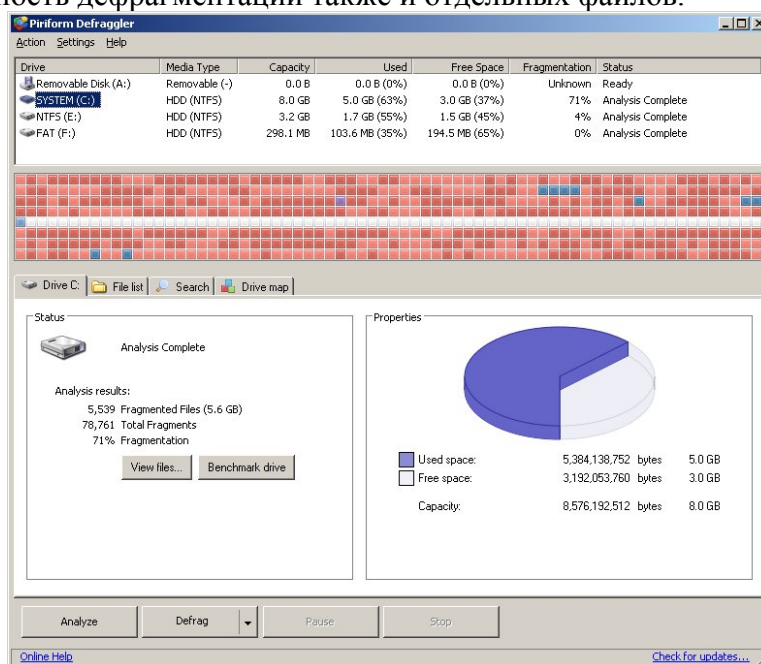
Clonezilla Live является загрузочным дистрибутивом Linux, позволяющим клонировать отдельные устройства. Clonezilla SE – это пакет, который устанавливается на дистрибутиве Linux. Он используется для одновременного клонирования множества компьютеров по сети.

5. **OSFMount.** Использование данной утилиты дает возможность монтировать ранее сделанные образы дисков и представлять их в виде виртуальных приводов, непосредственно просматривая сами данные. OSFMount поддерживает файлы образов, такие как: DD, ISO, BIN, IMG, DD, 00n, NRG, SDI, AFF, AFM, AFD и VMDK.



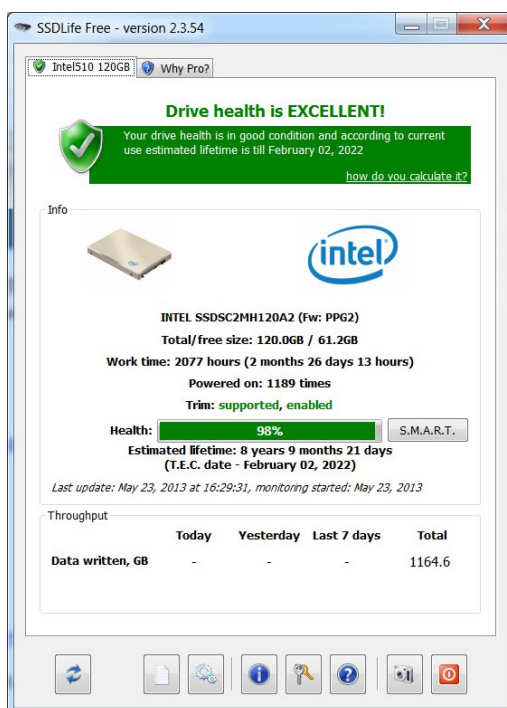
Дополнительная функция OSFMount – создание RAM-дисков, находящихся в оперативной памяти компьютера, что существенно ускоряет работу с ними. Для запуска процесса нужно перейти в File > Mount new virtual disk.

6. **Defraggler** – бесплатная программа для дефрагментации жесткого диска, которая способствует увеличению его скорости и срока службы. Особенностью программы является возможность дефрагментации также и отдельных файлов.



Defraggler проводит анализ содержимого на диске и отображает список всех фрагментированных файлов. Во время процесса дефрагментации отображается передвижение данных по диску. Выделенные желтым – данные, которые считываются, а зеленым – те, что записываются. По завершению Defraggler выдает соответствующее сообщение. Поддерживает файловые системы NTFS, FAT32 и exFAT.

7. **SSDLife** – проводит диагностику твердотельного диска, выводит на экран информацию о его состоянии и оценивает предполагаемый срок службы. Поддерживает удаленный мониторинг, управляет уровнем производительности на некоторых моделях жестких дисков.



Благодаря контролю износа SSD можно повысить уровень безопасности данных, вовремя выявлять проблемы. На основе анализа программа делает вывод насколько часто используется твердотельный диск.

8. Darik's Boot And Nuke (DBAN) Довольно популярная бесплатная утилита DBAN, применяется для очистки жестких дисков.

```
Darik's Boot and Nuke

Warning: This software irrecoverably destroys data.

This software is provided without any warranty; without even the implied
warranty of merchantability or fitness for a particular purpose. In no event
shall the software authors or contributors be liable for any damages arising
from the use of this software. This software is provided "as is".

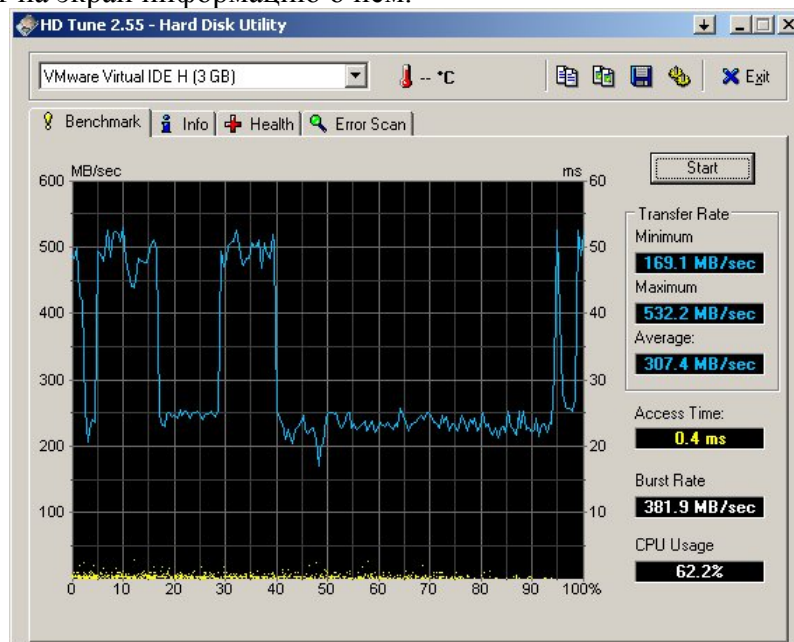
http://www.dban.org/

* Press the F2 key to learn about DBAN.
* Press the F3 key for a list of quick commands.
* Press the F4 key to read the RAID disclaimer.
* Press the ENTER key to start DBAN in interactive mode.
* Enter autonuke at this prompt to start DBAN in automatic mode.

boot: _
```

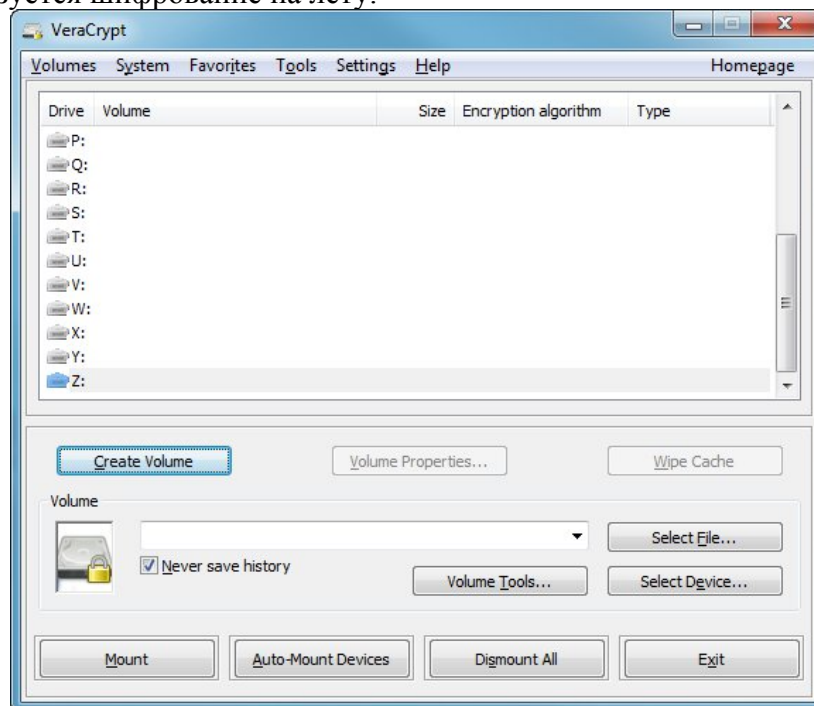
В DBAN два основных режима: интерактивный (interactive mode) и автоматический (automatic mode). Интерактивный режим позволяет подготовить диск к удалению данных и выбирать необходимые опции стирания. Автоматический режим очищает все обнаруженные диски.

9. Утилита **HD Tune** предназначена для работы с жестким диском и SSD. Измеряет уровень чтения-записи HDD/SSD, сканирует ошибки, проверяет состояние диска и выводит на экран информацию о нем.



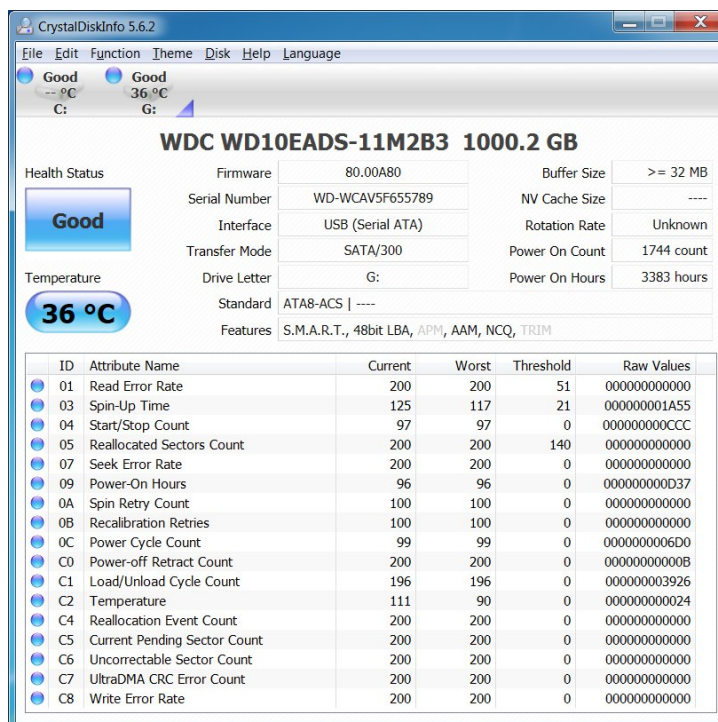
При запуске приложения, нужно выбрать диск из выпадающего списка и перейти к надлежущей вкладке, чтобы просмотреть информацию.

10. **VeraCrypt** – бесплатное приложение для шифрования с открытым исходным кодом. Используется шифрование на лету.



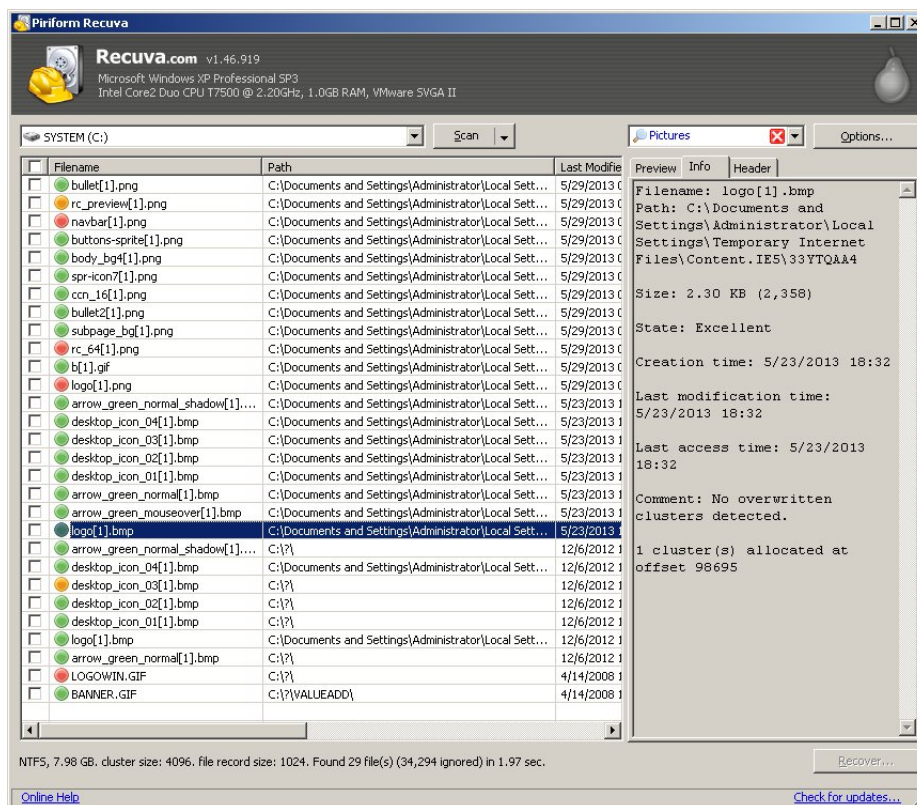
Проект VeraCrypt создан на основе TrueCrypt с целью усиления методов защиты ключей шифрования.

11. **CrystalDiskInfo** отображает состояние жестких дисков, поддерживающих технологию S.M.A.R.T. Утилита проводит мониторинг, оценивает общее состояние и отображает детальную информацию о жестких дисках (версия прошивки, серийный номер, стандарт, интерфейс, общее время работы и т. д.). У CrystalDiskInfo есть поддержка внешних жестких дисков.



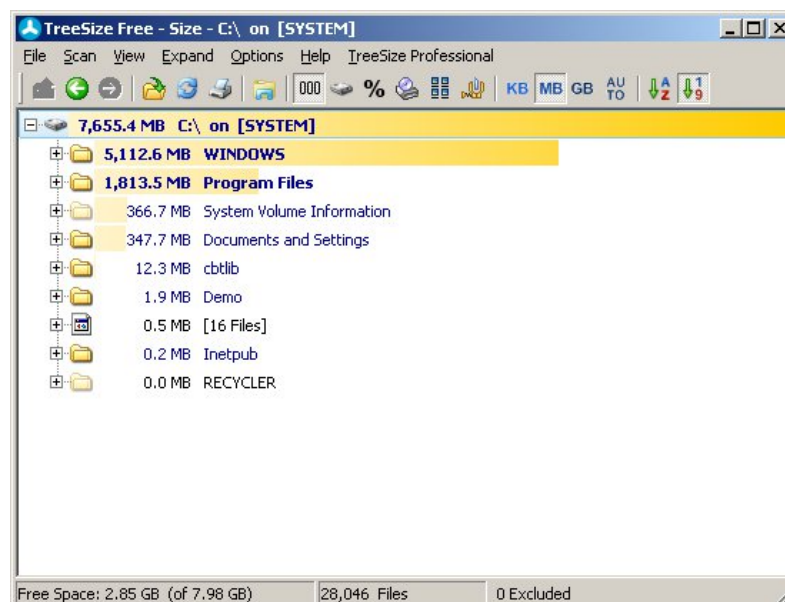
В верхней панели на экране отображаются все активные жесткие диски. Щелчок по каждому из них показывает информацию. Иконки Health Status и Temperature меняют цвет в зависимости от значения.

12. Утилита **Recuva** служит для восстановления случайно удаленных или потерянных файлов. Она сканирует нужный носитель информации, после чего выводит на экран список удаленных файлов. Каждый файл имеет свои параметры (имя, тип, путь, вероятность восстановления, состояние).



Необходимые файлы определяются с помощью функции предпросмотра и отмечаются флажками. Результат поиска можно отсортировать по типу (графика, музыка, документы, видео, архивы) и сразу просмотреть содержимое.

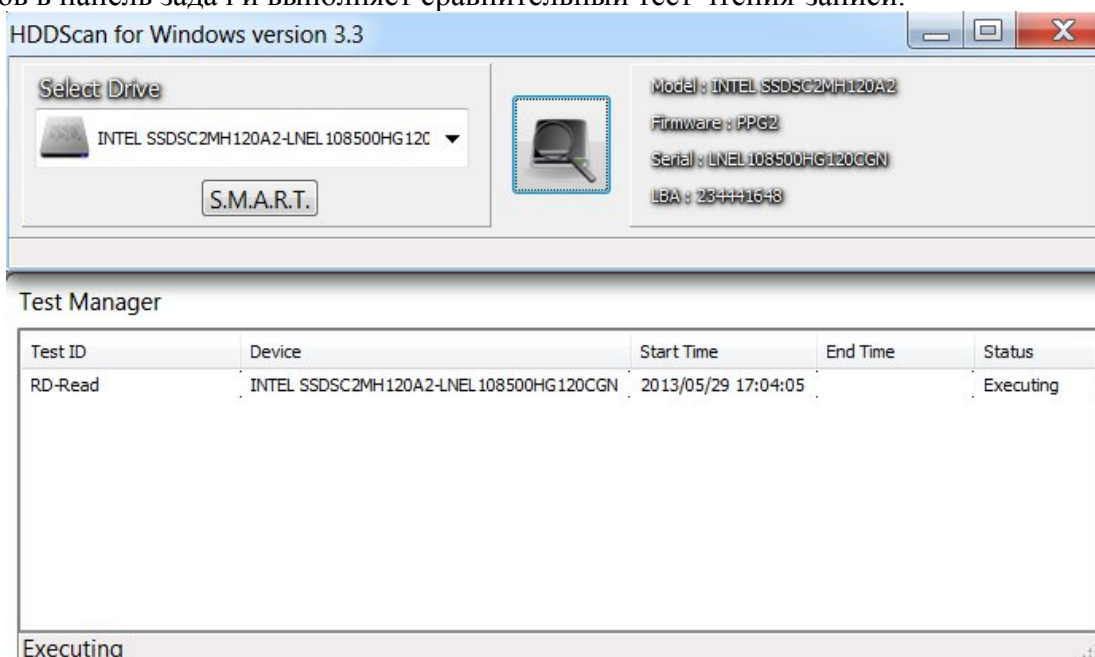
13. Программа **TreeSize** показывает дерево находящихся на жестком диске директорий с предоставлением информации об их размерах, а также проводит анализ использования дискового пространства.



Размеры папок выводятся на экран от самых больших до самых маленьких. Таким образом становится понятно, какие папки занимают большую часть места.

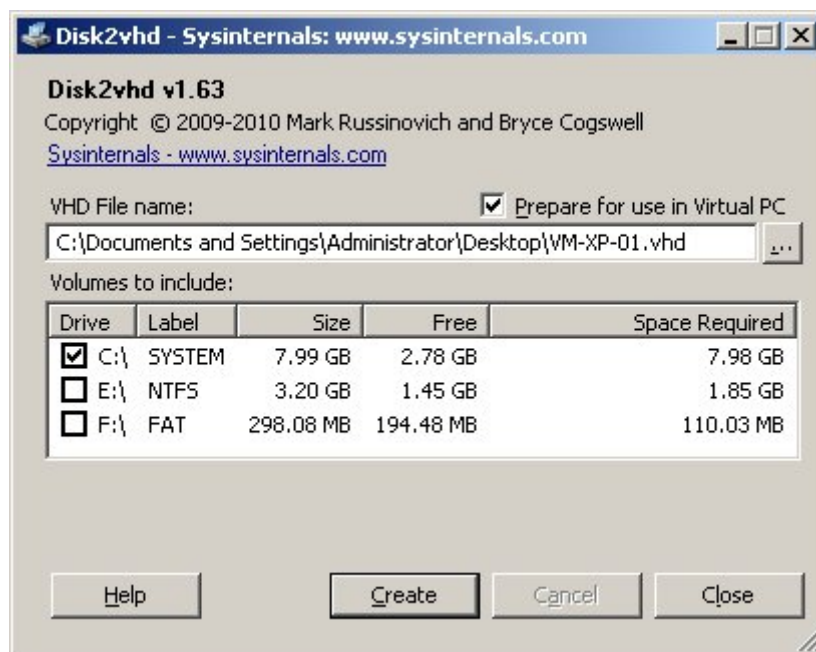
Примечание: при наличии Defraggler, Recuva и TreeSize, можно инициировать функции Defraggler и Recuva для определенной папки непосредственно из TreeSize – все три приложения эффективно интегрируются.

14. HDDScan – утилита диагностики жесткого диска, используется для тестирования накопителей информации (HDD, RAID, Flash) с целью выявления ошибок. Просматривает S.M.A.R.T. атрибуты, выводит показания датчиков температуры жестких дисков в панель задач и выполняет сравнительный тест чтения-записи.



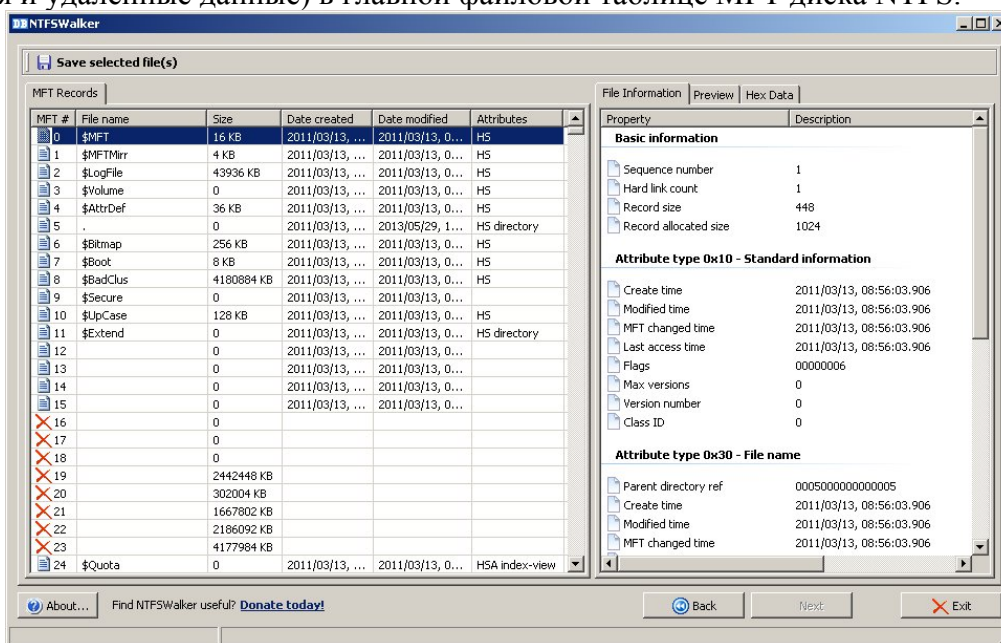
HDDScan предназначена для тестирования накопителей SATA, IDE, SCSI, USB, FireWire (IEEE 1394).

15. Бесплатная утилита **Disk2vhd** преобразует действующую физический диск в виртуальный Virtual Hard Disk (VHD) для платформы Microsoft Hyper-V. Причем, VHD-образ можно создавать прямо с запущенной операционной системы.



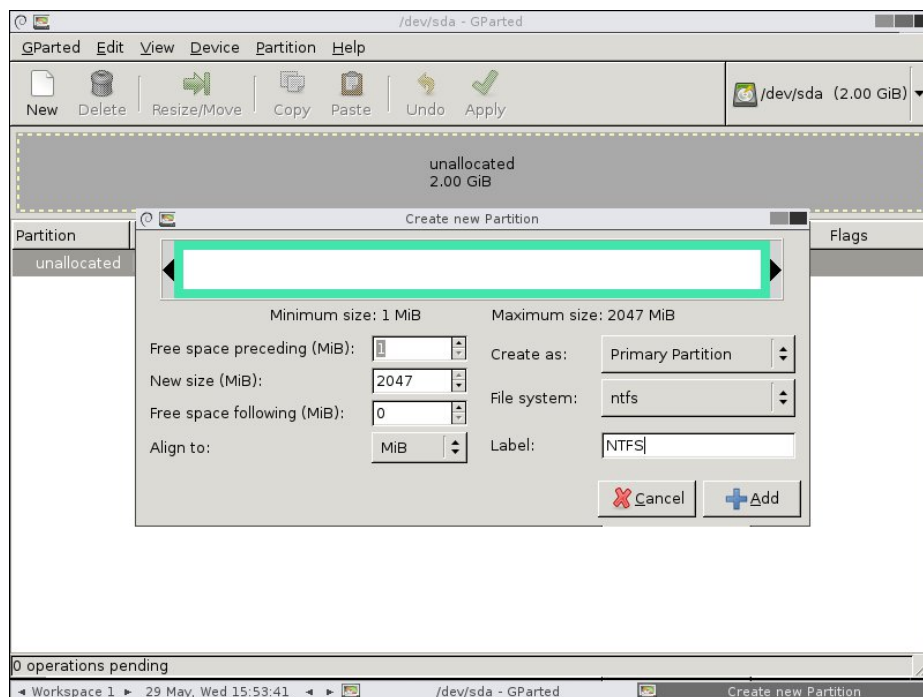
Disk2vhd создает один VHD-файл для каждого диска с избранными томами, сохраняя информацию о разделах диска и копируя только те данные, которые относятся к выбранному тому.

16. Портативная утилита **NTFSWalker** позволяет проводить анализ всех записей (включая и удаленные данные) в главной файловой таблице MFT диска NTFS.



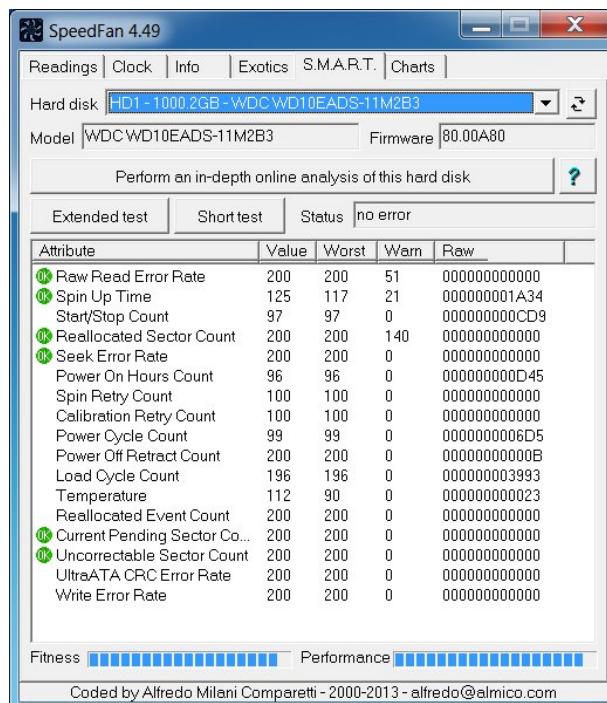
Наличие собственных драйверов NTFS дает возможность просматривать файловую структуру без помощи Windows на любых носителях чтения компьютера. К просмотру доступны удаленные файлы, обычные файлы, а также подробные атрибуты для каждого файла.

17. **GParted** – редактор дисковых разделов с открытым исходным кодом. Осуществляет эффективное и безопасное управление разделами (создание, удаление, изменение размера, перемещение, копирование, проверка) без потери данных.



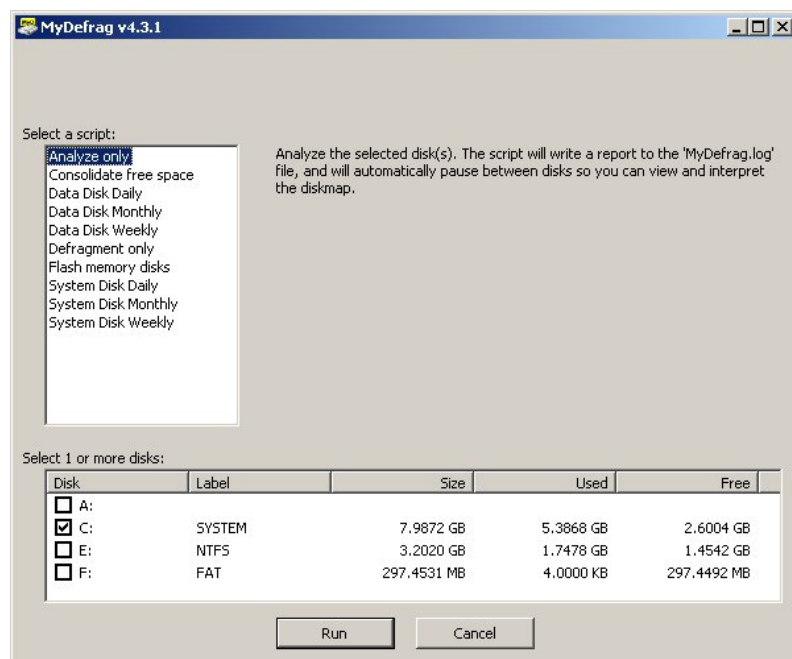
GParted позволяет создавать таблицы разделов (MS-DOS или GPT), включать, отключать и изменять атрибуты, выравнивать разделы, восстанавливать данные с поврежденных разделов и многое другое.

18. Компьютерная программа **SpeedFan** следит за показателями датчиков материнской платы, видеокарты и жёстких дисков, с возможностью регулирования скорости вращения установленных вентиляторов. Есть возможность проводить автоматическую и ручную регулировку.



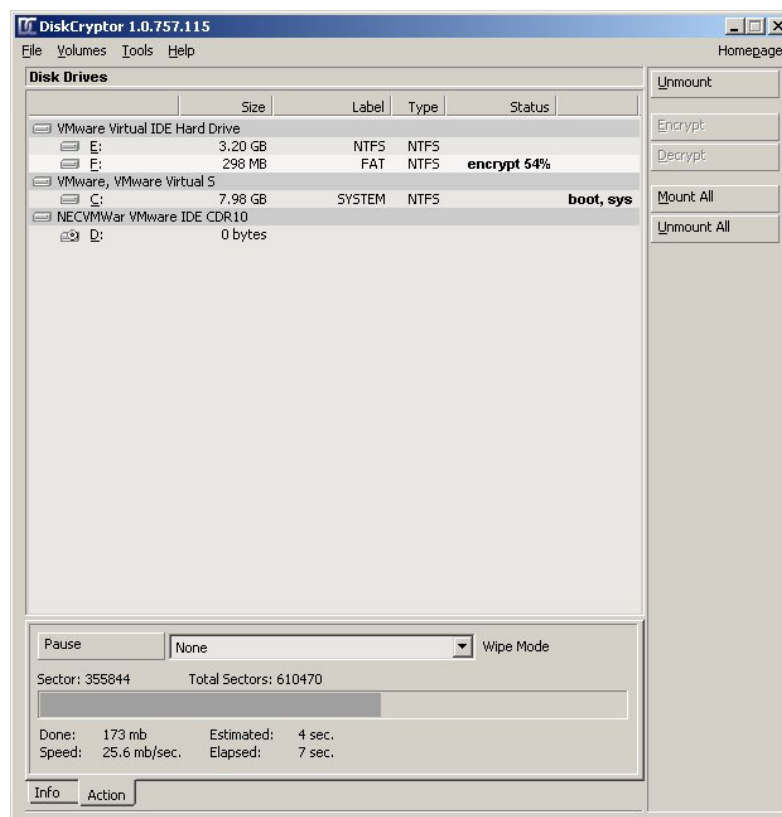
SpeedFan работает с жесткими дисками с интерфейсом SATA, EIDE и SCSI.

19. **MyDefrag** – бесплатный дисковой дефрагментатор, который используется для упорядочивания данных, размещенных на жестких дисках, дискетах, дисках USB и картах памяти.



У программы есть удобная функция работы в режиме скринсейвера, в результате чего дефрагментация будет производиться во время, назначенное для запуска хранителя экрана. MyDefrag также позволяет создавать или настраивать собственные сценарии.

20. С помощью шифровальной программы **DiskCryptor** с открытым исходным кодом, можно полностью зашифровать диск (все дисковые разделы, включая системный).



У DiskCryptor довольно высокая производительность – это один из самых быстрых драйверов шифрования дисковых томов. Программа поддерживает FAT12, FAT16, FAT32, NTFS и exFAT файловые системы, позволяя шифровать внутренние или внешние диски.

Задания к работе

1. Изучите утилиты, представленные в теоретической части работы.
2. Установите 2 утилиты, заданные преподавателем
3. Воспользуйтесь функциями установленных утилит

Вопросы для самоконтроля

1. Что такое дисковые утилиты?
2. Чем полезны дисковые утилиты?
3. Назовите основные классы дисковых утилит, а также их представителей.

Практическое занятие №5

Установка и настройка систем программирования

Цель и содержание: изучение принципов установки и настройки систем разработки программного обеспечения.

Теоретические сведения

Для разработки программного обеспечения используются среды разработки, которые включают в себя текстовый редактор, компилятор, компоновщик и отладчик.

Работа с программой Keil mVision версии 5.14

Для использования кириллицы в Keil следует воспользоваться подменой файла UvEdit.dll, разработанной

Установка среды разработки

Запустите файл mdk514.exe .

В появившемся окне программы Setup MDK ARM V5.14 нажмите кнопку Next.

Далее поставьте галочку напротив надписи I agree to all terms of the preceding License Agreement (соглашение с условиями лицензии). Нажмите кнопку Next.

Далее программа установки предложит выбрать путь для установки программы C:\Keil_v5 и путь к пакетам C:\Keil_v5\ARM\PACK. Предлагается оставить всё как есть, и нажать кнопку Next.

Затем программа попросит ввести First Name (имя), Last Name (фамилия), Company Name (название компании) и E-mail (адрес электронной почты). Введите данные и нажмите Next.

Далее программа установки начнёт процесс инсталляции файлов на компьютер, а затем распаковку пакетов и драйверов.

В конце установки снимите флажок напротив надписи Show Release Notes и нажмите кнопку Finish, установка программы завершена.

Установка пакетов

Для возможности загрузки разрабатываемых программ в микроконтроллер компании «Миландр» необходимо запустить файл [mdr_spl_v1.4.1.pack](#).

Для возможности в среде разработки печатать буквы кириллицы необходимо файл UvEdit.dll, находящийся в папке C:\Keil_v5\UV4, заменить на одноимённый ему, [UvEdit.dll](#).

Настройка среды разработки

Запустите среду разработки uVision. Для удобства работы разверните окно. Пройдите по пути Edit->Configuration и в появившемся окне перейдите на вкладку Colors & Fonts.

В окне Window выберите C/C++ Editor files и затем выберите шрифт Courier вместо Courier New (это необходимо для возможности печатать и отображать буквы кириллицы). Нажмите OK.

Задания к работе

1. Изучите программу Keil mVision
2. Установите, запустите Keil mVision и выполните её первичную настройку, либо установите среду разработки известного вам языка программирования.
3. Напишите исходный код произвольной программы, произведите её компиляцию и запуск.

Вопросы для самоконтроля

1. Назовите известные вам системы программирования, их возможности и сферы использования.
2. Из каких этапов обычно состоит установка и настройка среды программирования?
3. В чем главные отличия между существующими системами программирования?

Практическое занятие №6

Введение в программирование с использованием WinAPI

Цель и содержание: решение

1. Изучить

В операционной системе Windows реализована объектноориентированная идеология. Базовый объект системы – окно, поведение которого определяется методом, называемым функцией окна. Графический образ окна на экране дисплея – прямоугольная рабочая область. Независимо от своего типа любой объект Windows идентифицируется описателем или дескриптором (handle). Дескриптор – это ссылка на объект. Все взаимоотношения программного кода с объектом осуществляются только через его дескриптор.

Интерфейс прикладного программирования (API – Application Programming Interface) представляет собой совокупность 32-битных функций (Win32 API), которые предназначены для создания приложений (программ), работающих под управлением Microsoft Windows. Функции объявлены в заголовочных файлах. Главный из них – файл windows.h, в котором содержатся ссылки на другие заголовочные файлы.

В Win32 единицей работы компьютера является поток – ход выполнения программы в рамках процесса (в контексте процесса). Поток выполняет программный код, принадлежащий процессу. Процесс – это экземпляр выполняемой программы (но не ход ее выполнения). Он не является динамическим объектом и включает виртуальное адресное пространство, код и данные, файлы, синхронизирующие объекты, динамические библиотеки. Каждое приложение создает, по меньшей мере, один первичный поток, но может создать и много потоков.

Любое приложение Windows представлено на экране дисплея как минимум одним окном с набором стандартных элементов управления. Различают следующие типы окон:

- перекрывающие (overlapped window);
- всплывающие (pop-up window);
- дочерние (child window);
- слоистые (layered window) – особые окна, которые позволяют улучшить визуальный эффект, включая прозрачность. Перекрывающие окна создаются функцией CreateWindowEx() со стилем WS_OVERLAPPEDWINDOW. Этот стиль определяет наличие заголовка, системного меню, кнопок минимизации и максимизации, кнопки закрытия окна и «толстой» рамки, позволяющей изменять размеры окна. Перекрывающие окна предназначены для главных окон приложений и могут иметь меню. Всплывающие окна создаются функцией CreateWindowEx() со стилем WS_POPUP и предназначены для окон диалогов, окон сообщений и других окон временного использования, которые могут находиться вне главного окна приложения.

Для того чтобы временное окно имело заголовок, рамку и системное меню, необходимо при его создании использовать комбинацию стилей WS_POPUPWINDOW | WS_CAPTION. Дочерние окна создаются функцией CreateWindowEx() со стилем WS_CHILD и обычно используются для разделения клиентской области родительского окна на отдельные функциональные области. Дочерние окна могут иметь заголовок, системное меню, кнопки минимизации и максимизации, рамку и полосы прокрутки, но не могут иметь меню. Дочерние окна всегда находятся в пределах клиентской области родительского окна, т.е. их координаты всегда отсчитываются от левого верхнего угла родительского окна. Родительское окно может быть перекрывающим, всплывающим или даже другим дочерним окном.

Задания к работе

1. Изучите главные функции Win API

2. Создайте произвольное оконное приложение с использованием WinAPI, содержащее несколько элементов управления, таких как кнопки, флажки, поля ввода.
3. Добавьте в приложение всплывающее окно.

Вопросы для самоконтроля

1. Назовите главную функцию приложения, написанного с использованием WinAPI.
2. Перечислите достоинства использования WinAPI.
3. Назовите виды окон с точки зрения WinAPI.

Практическое занятие №7

Работа с файлами

Цель и содержание: изучение возможностей WinAPI по работе с файлами: файловый ввод-вывод, управление каталогами, дисками

Теоретические сведения

Операционные системы, построенные на основе Win32, предоставляют широкий набор функций по управлению файловыми объектами и дисками. Чтобы выполнить какую-либо операцию над файловым объектом, его вначале необходимо *создать* (если он отсутствует) либо *открыть* (если он уже имеется). Будет создан дескриптор этого файлового объекта, который необходимо передавать во все функции Win32 API для выполнения операций над этим файловым объектом. После выполнения всех необходимых операций файловый объект необходимо *заккрыть*.

Основные функции для файлового ввода/вывода.

HANDLE CreateFile(

LPCTSTR lpzName, /* указатель на нуль-терминированную строку, которая описывает имя файлового объекта */

DWORD fdwAccess, // режим доступа (чтение-запись)

DWORD fdwShareMode, /* режим разделения файла между различными процессами */

LPSECURITY_ATTRIBUTES lpSa, /* указатель на атрибуты защиты (на структуру SECURITY_ATTRIBUTES) */

DWORD fdwCreate, // флаги настройки создания

DWORD fdwFlagsAndAttributes, /* файловые атрибуты (при открытии игнорируются) */

HANDLE hTemplateFile /* описатель файла, чьи атрибуты копируются либо NULL */

) – создает либо открывает файл, канал, почтовый слот, коммуникационный ресурс, дисковое устройство, консоль, каталог (только открывает). Возвращает описатель созданного или открытого файлового объекта при успешном завершении и INVALID_HANDLE_VALUE при возникновении ошибки. Для получения информации об ошибке нужно использовать функцию GetLastError.

Возможные значения параметра fdwAccess – любая комбинация следующих значений:

- 0 – определяет доступ для запроса к дисковому устройству;
- GENERIC_READ – определяет доступ для чтения;
- GENERIC_WRITE – определяет доступ для записи.

Возможные значения параметра fdwShareMode – любая комбинация одного или нескольких следующих значений:

- 0 – файловый объект не может быть разделен между процессами;
- FILE_SHARE_DELETE – последующая операция открытия файлового объекта будет успешной, только если требуется доступ для удаления;
- FILE_SHARE_READ – файловый объект можно открыть еще раз, но только для чтения;
- FILE_SHARE_WRITE – файловый объект можно открыть еще раз, но только для записи.

Параметр lpSa указывает на структуру SECURITY_ATTRIBUTES, которая определяет, может ли возвращаемый описатель наследоваться дочерним процессом. Если указать NULL, то описатель не будет наследоваться. Член lpSecurityDescriptor структуры определяет дескриптор безопасности файлового объекта. Если lpSa есть NULL, то объект получает дескриптор безопасности по умолчанию.

Возможные значения параметра `fdwCreate`:

- `CREATE_NEW` – создать новый файл, только если файла с таким именем еще не существует;
- `CREATE_ALWAYS` – создать новый файл, даже если файл с таким именем уже существует (в последнем случае файл замещается);
- `OPEN_EXISTING` – открыть файл, только если он уже существует;
- `OPEN_ALWAYS` – открыть файл, если он уже существует, и создать новый, если файл не существует;
- `TRUNCATE_EXISTING` – открыть файл, только если он существует, и урезать его до нулевой длины (необходимо также указать флаг `GENERIC_WRITE`).

Параметр `fdwFlagsAndAttributes` определяет файловые атрибуты и флаги для файла. Допустима любая комбинация следующих атрибутов файла:

- `FILE_ATTRIBUTE_ARCHIVE` – файл является архивным (при создании файла устанавливается автоматически);
- `FILE_ATTRIBUTE_COMPRESSED` – файл или каталог является сжатым. Для файла это означает, что данные сжимаются. Для каталога это означает, что сжимаются вновь создаваемые в ней файлы и подкаталоги;
- `FILE_ATTRIBUTE_HIDDEN` – файл является скрытым (не включается в список каталога);
- `FILE_ATTRIBUTE_NORMAL` – у файла нет других атрибутов (допустим, если используется один);
- `FILE_ATTRIBUTE_OFFLINE` – данные файла имеются в наличии не немедленно (данные перемещены в offline хранилище);
- `FILE_ATTRIBUTE_READONLY` – файл только для чтения;
- `FILE_ATTRIBUTE_SYSTEM` – файл является частью операционной системы и используется только ею;
- `FILE_ATTRIBUTE_TEMPORARY` – файл используется как временное хранилище. Файловая система пытается хранить данные в памяти для быстрого доступа к данным (должен быть удален, если более не требуется приложению).

Допустима любая комбинация следующих флагов:

- `FILE_FLAG_WRITE_THROUGH` – отключает кэширование при записи, заставляя систему записывать прямо на диск;
- `FILE_FLAG_OVERLAPPED` – открывает асинхронный доступ к файлу, в этом случае операционная система не поддерживает файловый указатель;
- `FILE_FLAG_NO_BUFFERING` – сообщает драйверу дискового устройства, что поддержка буферизации будет осуществляться непосредственно приложением; операционная система не должна выполнять опережающее чтение или кэширование этого файла; чтение и запись необходимо выполнять по границам секторов, адреса буферов в памяти нужно выровнять по границам секторов диска;
- `FILE_FLAG_RANDOM_ACCESS` – система ожидает, что доступ к файлу будет произвольным;
- `FILE_FLAG_SEQUENTIAL_SCAN` – система ожидает, что доступ к файлу будет последовательным от начала к концу;
- `FILE_FLAG_DELETE_ON_CLOSE` – заставляет систему удалить файл после его закрытия;
- `FILE_FLAG_BACKUP_SEMANTICS` – открывает файл только для резервного копирования или восстановления, если у процесса есть соответствующие права доступа;
- `FILE_FLAG_POSIX_SEMANTICS` – указывает системе использовать при доступе к файлу правила POSIX.

Если функция `CreateFile` открывает на стороне клиента именованный канал, то параметр `fdwFlagsAndAttributes` может также содержать Security степень безопасности информации службы. Когда вызывающее приложение определяет флаг

SECURITY_SQOS_PRESENT, параметр fdwFlagsAndAttributes может содержать одно и более следующих значений:

- SECURITY_ANONYMOUS – определяет исполнять роль клиента на анонимном уровне исполнения роли;
- SECURITY_IDENTIFICATION – определяет исполнять роль клиента на идентификационном уровне исполнения роли;
- SECURITY_IMPERSONATION – определяет исполнять роль клиента на уровне исполнения роли Impersonation;
- SECURITY_DELEGATION – определяет исполнять роль клиента на уровне исполнения роли *делегирование*;
- SECURITY_CONTEXT_TRACKING – определяет динамическую безопасность режима прослеживания. Если этот флаг не определен, то безопасность режима прослеживания является статической;
- SECURITY_EFFECTIVE_ONLY – определяет, что только позволенные аспекты контекста безопасности клиента доступны для сервера. Если Вы не определяете этот флаг, все аспекты контекста безопасности клиента доступны. Этот флаг позволяет клиенту ограничивать группы и привилегии, что сервер может использовать, исполняя роль клиента.

Параметр hTemplateFile определяет описатель файла с доступом GENERIC_READ, либо равен NULL. Если не NULL, то создаваемый файл имеет атрибуты файла с описателем hTemplateFile. Параметр fdwFlagsAndAttributes в этом случае игнорируется. Если открывается существующий файл, то этот параметр игнорируется.

BOOL CloseHandle(

HANDLE hObject // описатель закрываемого объекта

) – закрывает открытый описатель объекта. В случае успешного завершения возвращает TRUE, иначе – FALSE.

Функция CloseHandle закрывает описатели следующих объектов: консольный ввод, консольный вывод, объект событие, проекцию файла, объект мьютекс, именованный канал, процесс, объект семафор, поток, объект знак (token), файловый объект, возвращенный функцией CreateFile.

BOOL ReadFile(

HANDLE hFile, // описатель файла для чтения

LPVOID lpBuffer, /* указатель на буфер, который получает данные из файла */

DWORD nNumberOfBytesToRead, // число прочитанных из файла байт

LPDWORD lpNumberOfBytesRead, /* Указывает на число прочитанных байт;

функция устанавливает его в нуль перед выполнением какой либо работы или проверке ошибки; если этот параметр равен нулю, когда функция возвращает TRUE на именованном канале, другой конец канала в режиме сообщения вызывает функцию WriteFile function with nNumberOfBytesToWrite равным нулю. Если lpOverlapped равен NULL, lpNumberOfBytesRead не может быть NULL. Если lpOverlapped не равен NULL, lpNumberOfBytesRead может быть NULL. Если это асинхронная операция чтения, то получить число прочитанных байт можно вызовом функции GetOverlappedResult. Если hFile связан с портом (completion) ввода / вывода, то число прочитанных байт можно получить вызовом функции GetQueuedCompletionStatus. */

LPOVERLAPPED lpOverlapped /* Указывает на структуру OVERLAPPED. Эта структура требуется, если файл создан с флагом FILE_FLAG_OVERLAPPED. В этом случае операция чтения начинается со смещения, определенного в структуре OVERLAPPED, и функция может вернуть управление до выполнения операции чтения. В этом случае функция возвращает FALSE и функция GetLastError возвращает ERROR_IO_PENDING. Заданное в структуре OVERLAPPED событие устанавливается в сигнальное состояние до завершения операции чтения. */ – читает данные из файла, начиная с позиции, заданной указателем файла. После операции чтения указатель файла

передвигается на число фактически прочитанных байт, если файл не был открыт с флагом `FILE_FLAG_OVERLAPPED`. Если файл создан для асинхронного ввода / вывода, то перемещением файлового указателя после чтения управляет приложение. При успешном выполнении возвращает `TRUE`, иначе – `FALSE`. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию `GetLastError`. При синхронном чтении на конец файла при операции чтения указывает возврат функцией значения `TRUE` и число прочитанных байт равно нулю. При асинхронном чтении на конец файла при операции чтения указывает возврат функцией значения `FALSE` возврат функцией `GetLastError` значения `ERROR_HANDLE_EOF`.

Если `hFile` не был открыт с флагом `FILE_FLAG_OVERLAPPED` и `lpOverlapped` равен `NULL`, то операция чтения начинается с текущей позиции файлового указателя и функция `ReadFile` не возвращает управление, пока операция чтения не будет выполнена.

Если `hFile` не был открыт с флагом `FILE_FLAG_OVERLAPPED` и `lpOverlapped` не равен `NULL`, то операция чтения начинается со смещения, определенного в структуре `OVERLAPPED` и функция `ReadFile` не возвращает управление, пока операция чтения не будет выполнена.

```
BOOL WriteFile(
    HANDLE hFile, // описатель файла для записи
    LPCVOID lpBuffer, /* указатель на буфер, из которого данные записываются в файл
*/
```

```
    DWORD nNumberOfBytesToWrite, // число записанных в файл байт
    LPDWORD lpNumberOfBytesWritten, /* Указывает на число записанных байт;
    функция устанавливает его в нуль перед выполнением какой либо работы или проверке
    ошибки. Если lpOverlapped равен NULL, lpNumberOfBytesWritten не может быть NULL.
    Если lpOverlapped не равен NULL, lpNumberOfBytesWritten может быть NULL. Если это
    асинхронная операция записи, то получить число записанных байт можно вызовом
    функции GetOverlappedResult. Если hFile связан с портом (completion) ввода / вывода, то
    число записанных байт можно получить вызовом функции GetQueuedCompletionStatus. */
```

```
    LPOVERLAPPED lpOverlapped /* Указывает на структуру OVERLAPPED. Эта
    структура требуется, если файл создан с флагом FILE_FLAG_OVERLAPPED. В этом
    случае операция записи начинается со смещения, определенного в структуре
    OVERLAPPED, и функция может вернуть управление до выполнения операции записи. В
    этом случае функция возвращает FALSE и функция GetLastError возвращает
    ERROR_IO_PENDING. Заданное в структуре OVERLAPPED событие устанавливается в
    сигнальное состояние до завершения операции записи. */
```

) – записывает данные в файл, начиная с позиции, заданной указателем файла. После операции записи указатель файла передвигается на число фактически записанных байт, если файл не был открыт с флагом `FILE_FLAG_OVERLAPPED`. Если файл создан для асинхронного ввода / вывода, то перемещением файлового указателя после записи управляет приложение. При успешном выполнении возвращает `TRUE`, иначе – `FALSE`. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию `GetLastError`.

Если `hFile` не был открыт с флагом `FILE_FLAG_OVERLAPPED` и `lpOverlapped` равен `NULL`, то операция записи начинается с текущей позиции файлового указателя и функция `WriteFile` не возвращает управление, пока операция записи не будет выполнена.

Если `hFile` не был открыт с флагом `FILE_FLAG_OVERLAPPED` и `lpOverlapped` не равен `NULL`, то операция записи начинается со смещения, определенного в структуре `OVERLAPPED` и функция `WriteFile` не возвращает управление, пока операция записи не будет выполнена.

```
DWORD SetFilePointer(
    HANDLE hFile, // описатель файла
```

LONG lDistanceToMove, /* Определяет число байт, на которое перемещается файловый указатель. Положительное значение передвигает указатель вперед в файле, а отрицательное – назад. */

PLONG lpDistanceToMoveHigh, /* Указывает на старшее слово 64-х битового расстояния перемещения. Если значение параметра равно 0, то функция может оперировать файлами, чьи максимальные размеры $2^{32} - 2$. Если параметр определен, то максимальный размер файла – $2^{64} - 2$. Этот параметр также получает старшее слово прежнего (по документации нового) файлового указателя. */

DWORD dwMoveMethod /* Определяет стартовую точку для передвижения файлового указателя. Этот параметр может принимать следующие значения: FILE_BEGIN – начало файла (нуль) есть стартовая точка, FILE_CURRENT – текущее значение файлового указателя есть стартовая точка, FILE_END – текущая позиция конца файла есть стартовая точка. Если определен FILE_BEGIN, то параметр lDistanceToMove интерпретируется как беззнаковое расположение нового файлового указателя. */

) – перемещает файловый указатель открытого файла. Если функция завершилась успешно, то она возвращает младшую часть двойного слова прежнего (по документации нового) файлового указателя, и, если параметр lpDistanceToMoveHigh не равен NULL, то функция помещает двойное слово старшей части прежнего (по документации нового) файлового указателя в переменную, на которую указывает параметр lpDistanceToMoveHigh. Если функция завершилась неуспешно и параметр lpDistanceToMoveHigh равен NULL, то возвращаемое значение равно 0xFFFFFFFF. Чтобы получить расширенную информацию об ошибке, необходимо вызвать функцию GetLastError. Если функция завершилась неуспешно и параметр lpDistanceToMoveHigh не равен NULL, то возвращаемое значение равно 0xFFFFFFFF и функция GetLastError возвращает значение отличное от NO_ERROR.

DWORD **GetFileSize**(

HANDLE hFile, // описатель файла

LPDWORD lpFileSizeHigh /* Указывает на переменную, в которую возвращается старшее двойное слово размера файла. Этот параметр может быть равен NULL, если приложению не требуется старшее двойное слово размера файла. */

) – возвращает размер файла в байтах. Если функция завершается успешно, то возвращаемое значение – младшее двойное слово размера файла, и, если параметр lpFileSizeHigh не равен NULL, то функция помещает двойное слово старшей части размера файла в переменную, указанную этим параметром. Если функция завершилась неуспешно и параметр lpFileSizeHigh равен NULL, возвращаемое значение равно 0xFFFFFFFF. Чтобы получить расширенную информацию об ошибке, необходимо вызвать функцию GetLastError. Если функция завершилась неуспешно и параметр lpFileSizeHigh не равен NULL, то возвращаемое значение равно 0xFFFFFFFF и функция GetLastError возвращает значение отличное от NO_ERROR.

BOOL **SetEndOfFile**(

HANDLE hFile // описатель файла

) – устанавливает позицию конца файла в текущую позицию файлового указателя. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

BOOL **LockFile**(

HANDLE hFile, // описатель файла

DWORD dwFileOffsetLow, /* Определяет младшее слово стартового смещения байта в файле, с которого начинается блокировка. */

DWORD dwFileOffsetHigh, /* Определяет старшее слово стартового смещения байта в файле, с которого начинается блокировка. */

DWORD nNumberOfBytesToLockLow, /* Определяет младшее слово длины диапазона блокируемых байтов. */

DWORD nNumberOfBytesToLockHigh /* Определяет старшее слово длины диапазона блокируемых байтов. */

) – блокирует участок в открытом файле, что предотвращает доступ к нему со стороны других процессов. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

BOOL UnlockFile(

HANDLE hFile, // описатель файла

DWORD dwFileOffsetLow, /* Определяет младшее слово стартового смещения байта в файле, с которого начинается блокированный участок. */

DWORD dwFileOffsetHigh, /* Определяет старшее слово стартового смещения байта в файле, с которого начинается блокированный участок. */

DWORD nNumberOfBytesToUnlockLow, /* Определяет младшее слово длины диапазона разблокируемых байтов. */

DWORD nNumberOfBytesToUnlockHigh /* Определяет старшее слово длины диапазона разблокируемых байтов. */

) – разблокирует участок в открытом файле, что делает возможным доступ к нему со стороны других процессов. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

DWORD GetFileAttributes(

LPCTSTR lpFileName /* указатель на нуль оканчивающуюся строку, которая определяет имя файла или каталога */

) – возвращает атрибуты файла или каталога при успешном завершении. При неуспешном завершении возвращает значение 0xFFFFFFFF. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

Возможные значения атрибутов – один или более следующих значений:

- **FILE_ATTRIBUTE_ARCHIVE** – файл или каталог является архивным (при создании файла устанавливается автоматически);

- **FILE_ATTRIBUTE_COMPRESSED** – файл или каталог является сжатым. Для файла это означает, что данные сжимаются. Для каталога это означает, что сжимаются вновь создаваемые в ней файлы и подкаталоги;

- **FILE_ATTRIBUTE_HIDDEN** – файл является скрытым (не включается в список каталога);

- **FILE_ATTRIBUTE_NORMAL** – у файла нет других атрибутов (допустим, если используется один);

- **FILE_ATTRIBUTE_OFFLINE** – данные файла имеются в наличии не немедленно (данные перемещены в offline хранилище);

- **FILE_ATTRIBUTE_READONLY** – файл или каталог только для чтения;

- **FILE_ATTRIBUTE_SYSTEM** – файл или каталог является частью операционной системы и используется только ею;

- **FILE_ATTRIBUTE_TEMPORARY** – файл используется как временное хранилище. Файловая система пытается хранить данные в памяти для быстрого доступа к данным (должен быть удален, если более не требуется приложению).

- **FILE_ATTRIBUTE_DIRECTORY** – файловый объект есть каталог.

BOOL SetFileAttributes(

LPCTSTR lpFileName, /* указатель на нуль оканчивающуюся строку, которая определяет имя файла */

DWORD dwFileAttributes /* определяет устанавливаемые атрибуты файла */

) – устанавливает атрибуты файла. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

Возможные значения атрибутов – один или более следующих значений:

- `FILE_ATTRIBUTE_ARCHIVE` – файл является архивным (при создании файла устанавливается автоматически);
- `FILE_ATTRIBUTE_HIDDEN` – файл является скрытым (не включается в список каталога);
- `FILE_ATTRIBUTE_NORMAL` – у файла нет других атрибутов (допустим, если используется один);
- `FILE_ATTRIBUTE_OFFLINE` – данные файла имеются в наличии не немедленно (данные перемещены в offline хранилище);
- `FILE_ATTRIBUTE_READONLY` – файл только для чтения;
- `FILE_ATTRIBUTE_SYSTEM` – файл является частью операционной системы и используется только ею;
- `FILE_ATTRIBUTE_TEMPORARY` – файл используется как временное хранилище. Файловая система пытается хранить данные в памяти для быстрого доступа к данным (должен быть удален, если более не требуется приложению).
- `FILE_ATTRIBUTE_DIRECTORY` – файловый объект есть каталог.

BOOL *FlushFileBuffers*(

`HANDLE hFile` // описатель файла

) – заставляет записать все буферизуемые файловые данные, сопоставленные с файлом. При успешном выполнении возвращает `TRUE`, иначе – `FALSE`. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию `GetLastError`.

BOOL *DeleteFile*(`LPCTSTR lpzFileName` /* указатель на ноль оканчивающуюся строку, которая определяет имя удаляемого файла */

) – удаляет существующий файл. При успешном выполнении возвращает `TRUE`, иначе – `FALSE`.

BOOL *CopyFile*(

`LPCTSTR lpExistingFileName`, /* указатель на ноль оканчивающуюся строку, которая определяет имя существующего файла */

`LPCTSTR lpNewFileName`, /* указатель на ноль оканчивающуюся строку, которая определяет имя нового файла */

`BOOL bFailIfExists` /* флаг протекания операции, если файл определяемый вторым параметром уже существует: `TRUE` – функция завершается неудачно, `FALSE` – функция переписывает существующий файл */

) – копирует существующий файл в новый файл. При успешном выполнении возвращает `TRUE`, иначе – `FALSE`.

BOOL *MoveFile*(

`LPCTSTR lpExistingFileName`, /* указатель на ноль оканчивающуюся строку, которая определяет имя существующего файла или каталога */

`LPCTSTR lpNewFileName` /* указатель на ноль оканчивающуюся строку, которая определяет имя нового файла или каталога; файловый объект с новым именем не должен существовать; новый файл может быть на другой файловой системе или дисковом устройстве; новый каталог должен быть на том же самом дисковом устройстве */

) – переименовывает или перемещает существующий файл или каталог (включая его дочерние элементы). При успешном выполнении возвращает `TRUE`, иначе – `FALSE`.

DWORD *SearchPath*(

`LPCTSTR lpPath`, /* указатель на ноль оканчивающуюся строку, которая определяет путь, по которому должен осуществляться поиск файла; если этот параметр равен `NULL`, функция ищет соответствующий файл в следующих каталогах в следующей последовательности: каталог приложения, текущий каталог, системный каталог Windows (`system` или `system32`), каталог Windows, каталоги, перечисленные в переменной окружения `Path` */

LPCTSTR lpFileName, /* указатель на нуль оканчивающуюся строку, которая определяет имя файла для поиска */

LPCTSTR lpExtension, /* указатель на нуль оканчивающуюся строку, которая определяет расширение, которое нужно добавить к имени файла для поиска; первый символ расширения имени файла должен быть точкой (.); расширение добавляется только, если имя файла не заканчивается расширением */

DWORD nBufferLength, /* определяет длину в символах буфера, который получает действительный путь и имя файла */

LPTSTR lpBuffer, /* указатель на буфер для действительного пути имени найденного файла */

LPTSTR *lpFilePart /* указывает на адрес (в lpBuffer) последней компоненты действительного пути и имени файла, который есть адрес символа непосредственно следующего за конечной обратной косой чертой (\) в пути (файловая компонента) */

) – выполняет поиск файла. При успешном завершении возвращает длину в символах строки, скопированной в буфер, не включая завершающий нулевой символ. Если возвращенное значение больше, чем nBufferLength, то возвращенное значение – размер буфера, необходимого чтобы вместить путь к файлу. При неудачном завершении возвращает нуль. Чтобы получить расширенную информацию об ошибке, необходимо вызвать функцию GetLastError.

Основные функции по управлению каталогами.

BOOL CreateDirectory(

LPCTSTR lpPathName, /* указатель на нуль оканчивающуюся строку, которая определяет путь к каталогу, который должен быть создан */

LPSECURITY_ATTRIBUTES lpSecurityAttributes /* Указатель на структуру SECURITY_ATTRIBUTES, которая определяет, может ли возвращаемый описатель наследоваться дочерним процессом. Если указать NULL, то описатель не будет наследоваться. Если параметр равен NULL, то каталог получает дескриптор безопасности по умолчанию. */

) – создает новый каталог и устанавливает для него дескриптор безопасности. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

BOOL RemoveDirectory(

LPCTSTR lpPathName /* указатель на нуль оканчивающуюся строку, которая определяет путь к каталогу, который должен быть удален */

) – удаляет существующий пустой каталог. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

HANDLE FindFirstFile(

LPCTSTR lpFileName, /* Указатель на нуль оканчивающуюся строку, которая определяет действительный каталог или путь и имя файла, которые могут содержать символы подстановки (* и ?). */

LPWIN32_FIND_DATA lpFindFileData /* Указатель на структуру WIN32_FIND_DATA, которая получает информацию о файле или каталоге. */

) – выполняет поиск файла, чье имя соответствует заданной в первом аргументе спецификации, начиная с каталога, заданного в этом же параметре. Исследует имена подкаталогов также как и имена файлов. При успешном завершении возвращает описатель поиска, используемый в последующих вызовах функций FindNextFile и FindClose. В случае неудачи возвращает значение INVALID_HANDLE_VALUE. Чтобы получить расширенную информацию об ошибке, необходимо вызвать функцию GetLastError.

Структура WIN32_FIND_DATA описывает файл, найденный функциями FindFirstFile и FindNextFile.

```
typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    DWORD dwReserved0;
    DWORD dwReserved1;
    TCHAR cFileName[ MAX_PATH ];
    TCHAR cAlternateFileName[ 14 ];
} WIN32_FIND_DATA, *PWIN32_FIND_DATA, *LPWIN32_FIND_DATA;
```

Поле dwFileAttributes определяет атрибуты найденного файла. Может быть одним либо комбинацией следующих значений: FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_COMPRESSED, FILE_ATTRIBUTE_DIRECTORY, FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_NORMAL, FILE_ATTRIBUTE_OFFLINE, FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_SYSTEM, FILE_ATTRIBUTE_TEMPORARY.

Поле ftCreationTime содержит время создания файла в формате UTC.

Поле ftLastAccessTime содержит время последнего доступа к файлу в формате UTC.

Поле ftLastWriteTime содержит время последней модификации файла в формате UTC.

Поле nFileSizeHigh содержит старшее двойное слово размера файла в байтах.

Поле nFileSizeLow содержит младшее двойное слово размера файла в байтах.

Размер файла равен $nFileSizeHigh * MAXDWORD + nFileSizeLow$ байт.

Поля dwReserved0, dwReserved1 зарезервированы для будущего использования.

Поле cFileName содержит нулем заканчивающуюся строку, которая содержит имя найденного файла.

Поле cAlternateFileName содержит нулем заканчивающуюся строку, которая содержит альтернативное имя найденного файла (сгенерированное в формате 8.3).

Обнаружив файл, соответствующий заданной спецификации и расположенный в заданном каталоге, функция заполняет элементы структуры и возвращает описатель поиска. Нет – возвращает INVALID_HANDLE_VALUE и содержимое структуры не изменяется. Структура и возвращенный описатель поиска могут использоваться в последующих вызовах функций FindNextFile и FindClose для ссылки на файл или подкаталог.

BOOL FindNextFile(

HANDLE hFindFile, /* Описатель поиска, возвращенный функцией FindFirstFile. */

LPWIN32_FIND_DATA lpFindFileData /* Указывает на структуру WIN32_FIND_DATA, которая получает информацию о найденном файле или подкаталоге. Структура может быть использована в последующих вызовах функции FindNextFile для ссылки на найденный файл или каталог. */

) – продолжает поиск следующего файла, удовлетворяющего спецификации, переданной при вызове функции FindFirstFile. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError. Если не удалось найти соответствующий файл, то функция GetLastError возвращает ERROR_NO_MORE_FILES.

BOOL FindClose(

HANDLE hFindFile /* описатель поиска, предварительно открытый функцией FindFirstFile */

) – закрывает описатель поиска. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

BOOL SetCurrentDirectory(

LPCTSTR lpPathName /* Указатель на ноль завершающуюся строку, которая определяет путь к новому текущему каталогу. Путь может быть относительным и абсолютным. */

) – изменяет текущий каталог для текущего процесса. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

DWORD GetCurrentDirectory(

DWORD nBufferLength, /* Определяет длину буфера в символах для строки с названием текущего каталога. Буфер должен иметь пространство для завершающего нулевого символа. */

LPTSTR lpBuffer /* Указывает на буфер для строки с именем текущего каталога. Эта завершающаяся нулем строка определяет абсолютный путь к текущему каталогу. */

) – возвращает текущий каталог для текущего процесса. При успешном завершении функция возвращает число записанных в буфер символов, исключая завершающий нулевой символ. При неудаче возвращает нулевое значение. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

BOOL DoesDirExist(

LPCTSTR szDir //определяет имя каталога

) – определяет, существует ли данный каталог. Если каталог существует, то возвращает TRUE, иначе – FALSE.

Основные функции по управлению дисками:

DWORD GetLogicalDrives(VOID) – при успешном завершении возвращает 32-битовое значение, каждый бит которого указывает, существует ли данное дисковое устройство. Бит в позиции 0 установлен, если в системе есть диск A, бит в позиции 1 установлен, если в системе есть диск B, бит в позиции 2 установлен, если в системе есть диск C и так далее. При неуспешном завершении возвращает нулевое значение.

DWORD GetLogicalDriveStrings(

DWORD nBufferLength, /* Определяет максимальный размер буфера в символах, на который указывает второй параметр. Этот размер не включает завершающий нулевой символ. */

LPTSTR lpBuffer /* Указывает на буфер, который получает список завершающихся нулевым символом строк, каждая из которых представляет имя корневого каталога действительного логического дискового устройства. Список завершается вторым нулевым символом. */

) – заполняет буфер строками, которые описывают корневые каталоги логических дисковых устройств. При успешном завершении возвращает длину в символах строк, скопированных в буфер, исключая завершающий нулевой символ. Если буфер не достаточного размера, то возвращаемое значение больше значения параметра nBufferLength. Это размер буфера, требуемый для вмещения строк корневых каталогов логических дисковых устройств. При неудаче функция возвращает значение нуль. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

UINT GetDriveType(

LPCTSTR lpRootPathName /* Указывает на завершающуюся нулевым символом строку, которая определяет корневой каталог диска, относительно которого необходимо получить информацию. Если равен NULL, то функция использует корень текущего каталога. */

) – определяет тип дискового устройства (гибкий, жесткий, компакт-диск, эмулированный ОЗУ, сетевой). Возможны следующие значения:

- 0 – тип устройства определить нельзя;
- 1 – корневой каталог не существует;
- DRIVE_REMOVABLE – дисковод со сменным носителем – возвращается для гибких дисков;
- DRIVE_FIXED – дисковод с несменным носителем – возвращается для жестких дисков;
- DRIVE_REMOTE – удаленный дисковод – возвращается для сетевых дисков;
- DRIVE_CDROM – привод компакт-дисков;
- DRIVE_RAMDISK – диск, эмулируемый в оперативной памяти – RAM-диск.

BOOL GetVolumeInformation(

LPTSTR lpRootPathName, /* Указатель на строку, которая содержит имя корневого каталога тома. Если параметр равен NULL, то используется корень текущего каталога. */

LPTSTR lpVolumeNameBuffer, /* Указатель на буфер, в который возвращается имя тома. */

DWORD nVolumeNameSize, /* Определяет длину буфера в символах, в который возвращается имя тома. */

LPDWORD lpVolumeSerialNumber, /* Указатель на переменную, в которую возвращается серийный номер тома. Этот параметр может равняться NULL, если серийный номер не требуется. */

LPDWORD lpMaximumComponentLength, /* Указывает на переменную типа двойного слова, в которую возвращается максимальное число символов, допустимое в именах файлов и каталогов. */

LPDWORD lpFileSystemFlags, /* Указатель на переменную типа двойного слова, в которую возвращаются флаги файловой системы. */

LPTSTR lpFileSystemNameBuffer, /* Указатель на буфер, в который возвращается имя файловой системы. */

DWORD nFileSystemNameSize /* Определяет длину в символах буфера с именем файловой системы. */

) – возвращает информацию о файловой системе и томе, чей корневой каталог определен в первом параметре. Если требуемая информация получена, то функция возвращает TRUE. Если не вся требуемая информация получена, то функция возвращает FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

Параметр lpFileSystemFlags может быть комбинацией следующих флагов, с одним исключением: флаги FS_FILE_COMPRESSION и FS_VOL_IS_COMPRESSED взаимно исключаются. Возможные флаги:

- FS_CASE_IS_PRESERVED – при записи файла на диск регистр букв в его имени сохраняется;
- FS_CASE_SENSITIVE – файловая система поддерживает поиск файлов с учетом регистра букв в именах;
- FS_UNICODE_STORED_ON_DISK – файловая система поддерживает хранение на диске имен файлов в Unicode;
- FS_PERSISTENT_ACLS – файловая система способна оперировать со списками контроля доступа (только в NTFS);
- FS_FILE_COMPRESSION – файловая система поддерживает сжатие основанное на файлах;
- FS_VOL_IS_COMPRESSED – том является сжатым.

BOOL SetVolumeLabel(

LPCTSTR lpRootPathName, /* Указатель на завершающуюся нулевым символом строку, определяющую корневой каталог тома, метка которого должна быть установлена. Если параметр равен NULL, то устанавливается метка тома текущего диска процесса. */

LPCTSTR lpVolumeName /* Указатель на строку, определяющую устанавливаемую метку тома. Если этот параметр равен NULL, то функция удаляет метку тома. */

) – устанавливает метку тома файловой системы. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

BOOL **GetDiskFreeSpace**(

LPCTSTR lpRootPathName, /* Указатель на завершающуюся нулем строку, которая определяет корневой каталог диска, о котором нужно получить информацию. Если параметр равен NULL, то функция использует корень текущего каталога. */

LPDWORD lpSectorsPerCluster, /* Указатель на переменную для возвращения числа секторов на кластер. */

LPDWORD lpBytesPerSector, /* Указатель на переменную для возвращения числа байт в секторе. */

LPDWORD lpNumberOfFreeClusters, /* Указатель на переменную для возвращения числа свободных кластеров на диске. */

LPDWORD lpTotalNumberOfClusters /* Указатель на переменную для возвращения полного числа кластеров на диске. */

) – возвращает информацию о дисковом пространстве на томе. При успешном выполнении возвращает TRUE, иначе – FALSE. Чтобы получить расширенную информацию об ошибке нужно вызвать функцию GetLastError.

Ещё одна функция, которая стоит особняком *SHFileOperation* – осуществляет копирование, перемещение и удаление целого файлового объекта (каталога со всем его содержимым).

Performs a copy, move, rename, or delete operation on a file system object.

WINSHELLAPI int WINAPI **SHFileOperation**(

LPSHFILEOPSTRUCT lpFileOp /* Указатель на структуру SHFILEOPSTRUCT, которая содержит информацию необходимую функции для выполнения операции. */

) – выполняет копирование, перемещение, переименование и удаление целого файлового объекта. При успешном завершении возвращает нулевое значение, при возникновении ошибки – не нулевое значение.

Описание структуры `_SHFILEOPSTRUCT`, содержащую информацию необходимую функции *SHFileOperation* для выполнения операций.

```
typedef struct _SHFILEOPSTRUCT {  
    HWND          hwnd;  
    UINT          wFunc;  
    LPCSTR        pFrom;  
    LPCSTR        pTo;  
    FILEOP_FLAGS fFlags;  
    BOOL          fAnyOperationsAborted;  
    LPVOID        hNameMappings;  
    LPCSTR        pszProgressTitle;  
} SHFILEOPSTRUCT, FAR *LPSHFILEOPSTRUCT;
```

Поле `hwnd` – описатель диалогового окна, используемого для отображения информации о статусе операции.

Поле `wFunc` – операция, которую нужно выполнить. Может принимать одно из следующих значений:

- `FO_COPY` – копировать файлы, определенные параметром `pFrom` в место, определенное параметром `pTo`;

- FO_DELETE – удаление файлов, определенных параметром pFrom;
- FO_MOVE – перемещение файлы, определенные параметром pFrom в место, определенное параметром pTo;

- FO_RENAME - переименование файлов, определенных параметром pFrom.

Поле pFrom – указатель на буфер, определяющих один или несколько исходных файловых имен. Если имен несколько, то они должны быть разделены нулевым символом, а список имен должен завершаться двумя нулевыми символами.

Поле pTo – указатель на буфер, определяющих имя файла или каталога назначения. Буфер может содержать множественные имена, если поле fFlags определяет FOF_MULTIDESTFILES. Если имен несколько, то они должны быть разделены нулевым символом, а список имен должен завершаться двумя нулевыми символами.

Поле fFlags управляет файловой операцией и может быть комбинацией следующих значений:

FOF_ALLOWUNDO – позволяет вернуться назад, если возможно;

FOF_CONFIRMMOUSE – не осуществлен;

FOF_FILESONLY – выполнять операцию только на файлах, если определено имя файла с подстановочными символами (*.*);

FOF_MULTIDESTFILES – указывает, что поле pTo определяет множественные файлы назначения (один для каждого исходного файла), а не один каталог, в который все исходные файлы должны быть размещены;

FOF_NOCONFIRMATION – отвечает 'да ко всем' для любого диалогового окна, которое показано (не требует от пользователя подтверждения);

FOF_NOCONFIRMMKDIR – не подтверждает создание нового каталога, если операция требует, чтобы он был создан;

FOF_RENAMEONCOLLISION – дает файлу, над которым выполняется операция, новое название (типа 'Копии *1 из ...') при выполнении операции перемещения, копирования, переименования, если целевое имя файла уже существует;

FOF_SILENT – не показывает диалоговое окно продвижения процесса;

FOF_SIMPLEPROGRESS – показывает диалоговое окно продвижения, но не показывает имена файла;

FOF_WANTMAPPINGHANDLE – заполняется в поле hNameMappings; описатель должен быть освобожден при использовании функции SHFreeNameMappings.

Поле fAnyOperationsAborted получает значение TRUE, если пользователь прерывал какие-нибудь файловые операции прежде, чем они были закончены и FALSE в противном случае.

Поле hNameMappings – описатель объекта отображения имени файла, который содержит множество структур SHNAMEMAPPING. Каждая структура содержит старые и новые названия путей для каждого файла, который был перемещен, скопирован, или переименован. Этот член используется, только если fFlags включает FOF_WANTMAPPINGHANDLE.

```
typedef struct _SHNAMEMAPPING {
    LPSTR pszOldPath; // указатель на старое имя пути
    LPSTR pszNewPath; // указатель на новое имя пути
    int cchOldPath; // число символов в старом имени пути
    int cchNewPath; // число символов в новом имени пути
} SHNAMEMAPPING, FAR *LPSHNAMEMAPPING;
```

Поле lpzProgressTitle указывает на строку, используемую как название для диалогового окна продвижения. Это поле используется, только если fFlags включает FOF_SIMPLEPROGRESS.

Дополнительные сведения о разработке графического интерфейса приложений, необходимые для выполнения работы.

BOOL ListBox_AddString(

HWND hwndLB, //Описатель списка listbox
 LPTSTR szLine /* указатель на завершающуюся нулевым символом строку*/
) – добавляет в список строку.
 BOOL **ComboBox_AddString**(
 HWND hwndCB, //Описатель списка combobox
 LPTSTR szLine /* указатель на завершающуюся нулевым символом строку*/
) – добавляет в комбинированный список строку.
 int **_sprintf**(TCHAR *buff, const TCHAR *format[, argument, ...]) - подставляет
 элементы списка argument, в шаблон format и записывает результат в buff.
 TCHAR ***_tcscat**(TCHAR *dest, const TCHAR *src) - добавляет строку src к строке
 dest. Возвращает dest.

Задания к работе

Задание 1.

В среде Visual C++ создайте проект DiskInfo – приложение, выдающее в диалоговое окно информацию о диске, выбираемом из списка.

Сравните Ваше решение с имеющимся вариантом выполнения. *Видоизмените приложение следующим образом.*

Приложение должно сразу выводить информацию обо всех дисках (друг за другом). Выдачу информации осуществите в диалоговое окно со списком (используйте заготовку проекта DiskInfoL).

Разработайте консольный вариант (модифицированного) приложения (используйте заготовку проекта DiskInfo_C).

Задание 2.

В среде Visual C++ создайте проект DirList – приложение, выводящее список содержимого выбранного каталога. При запуске приложение должно выдавать содержимое текущего каталога.

Сравните Ваше решение с имеющимся вариантом выполнения. *Видоизмените приложение следующим образом.*

Добавить выдачу информации о файле: размер файла и его атрибуты. Сделайте так, чтобы при запуске приложения оно ничего первоначально не выдавало.

Задание 3.

В среде Visual C++ создайте проект SearchFiles – приложение для поиска файлов в заданном каталоге (и его подкаталогах), имена которых удовлетворяют заданному шаблону.

Сравните Ваше решение с имеющимся вариантом выполнения. *Видоизмените приложение следующим образом.*

Добавить выдачу информации о файле: размер файла и его атрибуты.

Задание 4.

В среде Visual C++ создайте проект CopyDir – приложение, копирующее (перемещающее) файлы из одного каталога (и всех его подкаталогов) в другой.

Сравните Ваше решение с имеющимся вариантом выполнения. *Видоизмените приложение следующим образом.*

Добавить флажок “Копировать с подкаталогами” и обеспечить соответствующую функциональность.

Задание 5.

В среде Visual C++ создайте проект SimplTxtEdt – приложение, реализующее функции простейшего текстового редактора. Текст, вводимый в многострочное поле ввода, должен сохраняться в файле, задаваемом в диалоговом окне «сохранить как». Текстовый файл должен также открываться и отображаться в многострочном поле ввода.

Сравните Ваше решение с имеющимся вариантом выполнения. *Видоизмените приложение следующим образом.*

Добавьте режим автосохранения с выбором соответствующего временного периода (использовать обработчик по таймеру).

Задание 6.

В среде Visual C++ создайте проект FM – приложение, выполняющее следующие функции. По команде создает файл с именем, задаваемым в поле ввода (если файл уже существует, то он перезаписывается), с заданным количеством записей (из одного поля – одного вещественного числа), а затем заполняет его записи случайными вещественными числами от -100 до 100. С помощью кнопок навигации позволяет перемещаться по записям файла, при этом отображается как номер записи, так и ее поле. Используйте заготовку FM.

Вопросы для самоконтроля

- 1) Какова последовательность работы с файловыми объектами в Win32 API? Что такое дескриптор файлового объекта? Для чего он служит?
- 2) Какие функции по управлению файловым вводом/выводом Вы знаете? Опишите их.
- 3) Какие функции по управлению каталогами Вы знаете? Опишите их.
- 4) Какие функции по управлению дисками Вы знаете? Опишите их.
- 5) Для чего используется блокировка файла либо его части?

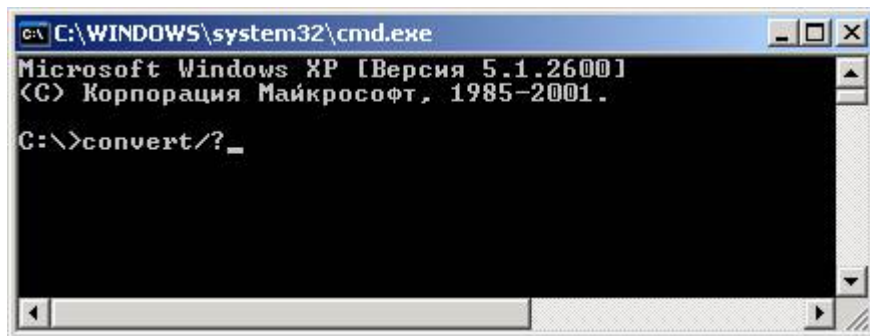
Практическое занятие №8

Файловые системы FAT

Цель и содержание: изучить сведения о файловых системах Windows, научиться использовать утилиту конвертации файловой системы.

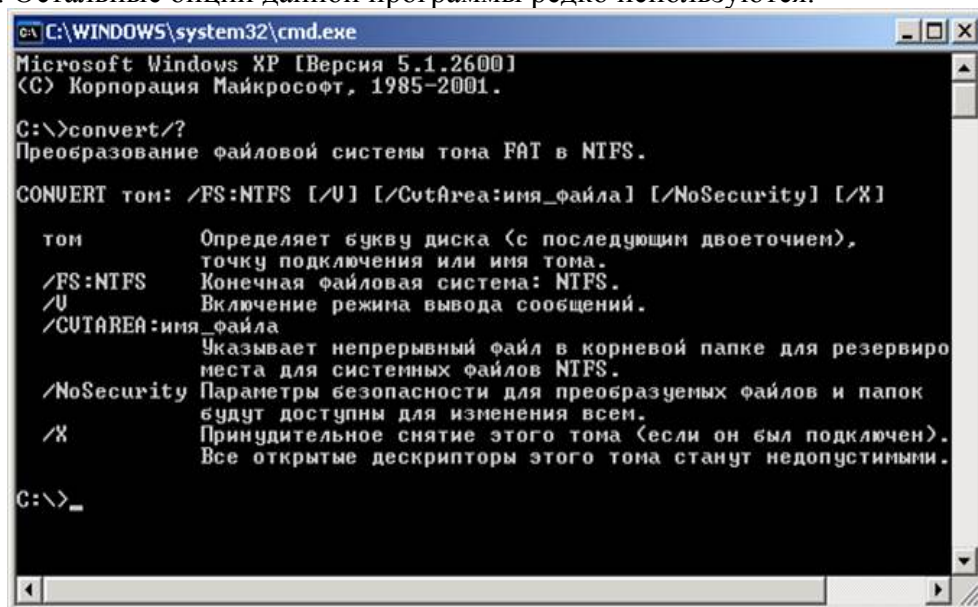
Теоретические сведения

Если на компьютере установлена файловая система *FAT*, то для установки файловой системы *NTFS* следует воспользоваться программой-конвертером файловых систем *convert.exe*, входящей в стандартный комплект поставки *Windows*. Запуск этой программы производится в текстовом командном окне, например, программы *cmd.exe*. Программу можно запустить, указав ключ *"/help"* в качестве аргумента.



Запуск программы конвертации файловых систем

Программа предназначена для конвертации файловых систем *FAT* в *NTFS*. При конвертации в простейшем случае необходимо указать после имени программы имя (букву) диска с двоеточием, а затем ключ *«/FS:NTFS»*. Полезно указать ключ *«/V»* – программа будет выводить на экран больше сообщений, в результате чего можно лучше контролировать процесс конвертации и ошибки, которые могут возникнуть по ходу этого процесса. Остальные опции данной программы редко используются.



Помощь при работе с программой *convert.exe*

Программа *convert.exe* берет за образец для конвертирования файловую систему *FAT* и делает из нее *NTFS*. Однако в *FAT* нет ни системных прав, ни каких-либо других объектов данных, сопровождающих файлы и папки в файловой системе *NTFS*. Поэтому

следует с помощью специальной программы, входящей в состав *Windows XP*, поставить права системы безопасности на созданной системе *NTFS*. Для этого нужно в текстовом командном окне выполнить следующую команду:

Secedit /configure /db C : \ %SystemRoot% \ temp \ temp.mdb /Cfg c : \ %SystemDistrib% \ inf \ defltwk.inf /areas filestore,

где: %systemRoot% – каталог, в который установлена система;

%SystemDistrib% – каталог, в котором находится дистрибутив ОС *Windows XP*.

Проводить такую операцию нужно из пользовательской учетной записи, имеющей права администратора системы. В результате выполнения системой данной команды восстановятся права по умолчанию для папок и файлов в каталоге, где установлена система: *C:\WINDOWS* и *C:\Program Files*. Для папки *Documents and Settings* следует устанавливать права вручную. После завершения данного процесса диск будет находиться под управлением файловой системы *NTFS*.

Задания к работе

1. Изучите теоретические сведения о дисковой утилите *Convert*
2. Создайте клон учебной виртуальной машины
3. Произведите конвертацию файловой системы *FAT* в *NTFS* и установку прав системы безопасности.

Вопросы для самоконтроля

1. Что, по Вашему мнению, представляет собой файловая система?
2. Перечислите типы файловых систем, используемых линейкой *Windows*.
3. Опишите строение файловой системы *FAT*.
4. Перечислите существенные различия между видами файловых систем под общим названием *FAT*.
5. Почему файловой системе *FAT* необходимо постоянное обслуживание?
6. Какие еще функции следует периодически выполнять в файловой системе *FAT* и почему?
7. Какие средства управления файловой системой *FAT* Вы знаете?
8. Какие программы для ускорения доступа к диску Вы знаете? Опишите одну из них.
9. Перечислите недостатки файловой системы *FAT*. Какой самый основной недостаток?
10. Опишите механизм проникновения вирусов в файловую систему *FAT*?

Практическое занятие №9

Сложные обработчики и взаимодействие резидентных программ

Цель и содержание: изучение принципов использования сложных обработчиков прерываний для организации работы резидентных программ.

Теоретические сведения

Основная особенность резидентной программы – событийный характер ее выполнения: вместо предсказуемого регулярного выполнения алгоритма имеют место асинхронные, происходящие в непредсказуемые моменты времени вызовы обработчиков прерываний. При этом их активизация может происходить на фоне любых других программ, находящихся в этот момент в заранее не известном состоянии. Состояние отдельных обработчиков самого резидента также может быть различным. Эти проблемы в целом характерны для параллельно выполняющихся и взаимодействующих программ, а обработчики прерываний и использующие их резидентные программы представляют собой элементы параллелизма в изначально однозадачной среде.

Повторное вхождение в программу (подпрограмму) может возникать либо в результате явной или неявной рекурсии либо, в случае обработчика прерывания, из-за повторного возникновения нового прерывания до завершения обработки предыдущего. Если рекурсивные вызовы по крайней мере прогнозируемы при анализе исходного текста, то повторная активизация обработчиков зависит от стечения «внешних» обстоятельств. Обработчики, допускающие повторный вызов («вхождение»), называют *реентерабельными*.

Обеспечение реентерабельности является сложной задачей. В первую очередь проблему создают глобальные переменные и обращения к глобальным ресурсам вообще. Обращение к такой переменной (или набору переменных) может быть прервано повторным вызовом того же обработчика, по завершении которого их значения могут оказаться изменены, что приведет к некорректной дальнейшей работе прерванного кода. В более общем виде это представляет собой проблему «критического ресурса», характерную для многозадачных систем.

Реентерабельность (и пригодность для рекурсий) обычных подпрограмм обеспечивается в большинстве языков использованием «автоматических» переменных (класс хранения *auto* в терминологии C/C++): они выделяются в стеке заново для каждой «копии» вызова подпрограммы. Однако полностью исключить использование глобальных переменных не всегда возможно (как минимум, регистры процессора являются своего рода «переменными», глобальными для всех программ), а доступное пространство в стеке для резидентных программ всегда ограничено, так как по умолчанию они вынуждены использовать стек той программы, на фоне которой был вызван обработчик. Для резидента можно зарезервировать собственный стек и переключаться на него при каждом вызове обработчика, но тогда сама эта область вместе с обслуживающими ее счетчиками окажется разделяемой между «копиями» вызовов, а динамическое выделение и управление несколькими стеками будет иметь очень сложную реализацию, особенно учитывая общие ограничения на доступную память.

В результате на практике обычно удается сделать реентерабельными лишь относительно простые обработчики. Для более сложных приходится ограничиваться «частичной» реентерабельностью – повторный вызов безопасен, но функции при этом не выполняются или выполняются частично. Типична реализация такого подхода с помощью флага активности (простейшего семафора):

```
is_active DB 0 ; переменная-семафор
Int_Handler PROC FAR
    cmp cs:is_active, 0 ; проверка семафора
    jz work
```

```

iret ;выход – обработчик уже активен
work:
inc cs:is_active ; закрыть семафор
... ;функции обработчика
inc cs:is_active ; открыть семафор
Int_Handler ENDP

```

Важно обеспечить *атомарность* (непрерывность) проверки значения семафора и его изменения, иначе сохраняется вероятность повторного вызова между этими инструкциями. В данном случае непрерывность обеспечивается предварительным запретом прерываний перед передачей управления обработчику, но на это можно рассчитывать не всегда. Система команд **x86** содержит инструкцию `xchg` – элементарный, гарантированно непрерываемый обмен двух значений, но воспользоваться им не всегда удобно. Заметим, что проблему представляют также и «вложенные» запреты, и разрешения прерываний флагом `IF`, когда приходится корректно восстанавливать его значение на каждом «уровне».

Аналогичные проблемы свойственны не только пользовательским программам: реентерабельность стандартных «системных» обработчиков в общем случае также не гарантируется. Практически прерывания BIOS реентерабельны или имеют собственную блокировку критических участков, поэтому их вызов предположительно безопасен в любое время (документация этого не гарантирует). Прерывания DOS заведомо нереентерабельны.

Наиболее часто требуются обращения к прерыванию DOS `int 21h`. Так как полностью отказаться от него не всегда возможно, необходимо выбирать моменты для безопасного вызова. Имеются следующие основные возможности.

Контроль флагов `InDOS` (нахождение в обработчике функций DOS) и `CriticalError` (нахождение в обработчике критической ошибки). Адрес байта `InDOS` возвращается функцией `int 21h AH = 34h` (необходимо вызывать заранее в секции инициализации так как нереентерабельность распространяется и на нее), `CriticalError` расположен в предыдущем байте памяти либо может быть получен недокументированным вызовом `int 21h AX = 6D06h`. Начальные значения флагов – нулевые. Обработчики функций DOS инкрементируют `InDOS` при входе и декрементируют при выходе. Обработчик критической ошибки устанавливает `CriticalError` и сбрасывает `InDOS`.

Обработка прерывания «холостого хода» `int 28h` позволяет определить период, когда DOS находится в состоянии ожидания ввода-вывода. Внутри обработчика `int 28h` можно безопасно обращаться к функциям DOS с номерами выше `0Ch` независимо от состояния флагов.

В обоих случаях удобной будет описанная выше схема «отложенного вызова»: «исполнительный» обработчик устанавливается на прерывание холостого хода или проверяет возможность безопасного вызова DOS по флагам.

Кроме того, можно установить собственный монитор прерываний DOS и самостоятельно контролировать обращения к ее функциям. Так, безопасным считается совмещение консольного и файлового ввода-вывода. Но этот способ трудоемкий и не вполне надежный.

Кроме того, обработчики многих функций DOS предполагают, что вызывающая их в данный момент программа является также и текущей с точки зрения системы, однако в случае вызова из резидента это правило нарушается. Так как идентификатором программы служит адрес её `PSP`, необходимо, дождавшись возможности безопасного обращения к DOS, в первую очередь сохранить текущий `PSP` (функция `AH = 62h`) и зарегистрировать в качестве текущего свой (функция `AH = 50h`), а по окончании работы – восстановить сохраненный.

Для взаимодействия с резидентной программой нужно в общем случае обнаружить ее, а затем получить доступ к ее данным и процедурам. Возможны следующие варианты:

- прямое обращение к переменным и процедурам резидентной программы– необходимо знание ее внутренней структуры, то есть требуется или документирование, или наличие отдельной программы, обладающей этим «знанием» (например, транзитный запуск той же программы, из которой устанавливается резидент, с соответствующими ключами);

- захват и использование в качестве программного интерфейса определенных прерываний и/или их функций– универсально и эффективно, но может быть недостаточно гибко и создавать конфликты с другими программами;

- поддержка резидентом «горячих» клавиш– только интерактивное управление, но не интерфейс между программами.

В свою очередь обнаружение резидента также может быть решено несколькими способами:

- поиск сигнатуры (характерного достаточно длинного значения), содержащейся в определенном месте кода программы, путем сканирования памяти;

- обнаружение характерных для этого резидента изменений программной среды, в первую очередь прерываний и/или их функций;

- выделение специальных «диагностических» прерываний и/или их функций.

Наиболее эффективным, но достаточно сложным и трудоёмким решением для взаимодействия с резидентами является унифицированный интерфейс, основанный на использовании специально выделенного для него *мультиплексного* (или *мультиплексорного*) прерывания int 2Fh.

Основная идея состоит в наличии у резидента обработчика int 2Fh. Все эти обработчики каскадируются. В ходе установки каждая новая программа получает свободный идентификатор, который запоминается для сравнения. В дальнейшем этот идентификатор передается при обращениях к функциям int 2Fh: опознав его, программа выполняет функцию, иначе передает управление дальше по цепочке обработчиков. Кроме того, программа должна содержать по фиксированному адресу унифицированный блок параметров, которые описывают её и могут быть использованы для управления извне.

Отдельной, достаточно часто встречающейся задачей является деинсталляция резидентных программ. Она распадается на две подзадачи: деактивация (отключение) обработчиков и удаление резидентной части из памяти. Готового решения для каждой из них MS-DOS не предоставляет.

При деактивации обработчиков основную трудность представляет восстановление существовавшей ранее цепочки обработчиков. Как правило, это не удастся сделать, если отключаемый обработчик был перекрыт следующим, установленным на то же прерывание, что можно определить, например, сравнивая реальный адрес обработчика со значением в таблице векторов прерываний. Тогда обработчик обычно просто деактивируется без удаления из цепочки– запрещается его выполнение, например с помощью флага:

```
old_handler_off DW ?
old_handler_seg DW ?
is_enabled DB ?
Int_Handler PROC FAR
    cmp cs:is_enabled, 0
    jnz work
    iret
work:
    ... ;функции обработчика
Int_Handler ENDP
```

Для отключения обработчика достаточно обнулить флаг разрешения, для включения– записать туда ненулевое значение.

Более сложный, но и более интересный способ— внутренняя «таблица переходов» (на примере единственного обработчика):

```
act_handler_off DW ?
act_handler_seg DW ?
old_handler_off DW ?
old_handler_seg DW ?
;общая часть обработчика – переход по таблице
Int_Handler PROC FAR
jmp dword ptr cs:act_handler_off
Int_Handler ENDP
;рабочая часть обработчика
Handler_Work PROC FAR
...
call dword ptr cs:old_handler_off
...
Handler_Work ENDP
```

Для включения и выключения такого обработчика в ячейки `act_handler` записывается точка входа в «рабочую» часть `Handler_Work` или сохраненный адрес старого обработчика. Этот подход хорош тем, что «таблицу переходов» и устанавливаемые в таблицу векторов обработчики могут быть компактно сгруппированы в начале программы, тогда при выгрузке резидента а памяти остаются только они, а «рабочие» функции могут быть отброшены.

Выгрузка резидента из памяти включает в себя освобождение занимаемого им блока памяти и всех выделенных ему ресурсов, включая открытые файлы. Хороший способ— завершение резидента стандартной функцией `int 21h AH = 4Ch`. Однако DOS всегда предполагает завершение текущей программы, поэтому предварительно надо временно переключить адрес текущего PSP на PSP резидента. Альтернатива— выполнять освобождение «вручную».

В любом случае до выгрузки резидентной программы должны быть отключены все ее обработчики. Если обработчики не исключаются из цепочек полностью, а замещаются заглушками, полная выгрузка резидента невозможна: как минимум код заглушек должен оставаться в памяти.

Задания к работе

1. Изучите теоретические сведения об обработчиках прерываний
2. Создайте простую резидентную программу, осуществляющую вывод на экран счетчика событий передачи ей управления посредством прерывания.

Вопросы для самоконтроля

1. Каскадные обработчики прерываний.
2. Проблемы идентификации обработчиков прерываний.
3. Мультиплексное прерывание.
4. Проблемы при удалении обработчиков прерываний.
5. Проблема реентерабельности обработчиков прерываний.
6. Повторное вхождение в прерывания DOS.

Практическое занятие №10

Приложения Windows с использованием WinAPI

Цель и содержание: изучить последовательность действий по созданию исполняемой программы в ОС Windows из исходного текста. Получить практические навыки использования функций, предоставляемых операционной системой Windows.

Теоретические сведения

Интерфейс прикладного программирования

Интерфейс прикладного программирования (Application Programming Interface) – набор готовых констант, структур и функций, используемых при разработке программных приложений и обеспечивающих правильное взаимодействие между приложением и операционной системой. В операционной системе windows интерфейс прикладного программирования называют WinAPI.

Окна и сообщения

В основе системы windows лежит механизм обработки сообщений, который транслирует практически каждое событие (нажатие клавиши, перемещение мыши...) в сообщение. Типичное приложение построено на основе цикла сообщений, который принимает сообщения и перенаправляет их к соответствующим функциям – обработчикам. Хотя сообщения передаются приложениям, они адресованы не им, а другим компонентам операционной системы – окнам. Окно – представляет некую абстрактную сущность, через которую взаимодействуют пользователь и компьютер. Все окна находятся в иерархической зависимости: некоторые из них являются окнами верхнего уровня, другие подчинены своим родительским окнам. В Windows существует множество типов окон. Самыми очевидными типами окон являются большие прямоугольные окна, связанные с приложениями, которые можно переместить, изменить их размеры, минимизировать и т. д. Многие элементы, отображаемые внутри главного или диалогового окна, сами по себе являются окнами. Каждая кнопка, поле ввода, полоса прокрутки, окно списка, пиктограмма, даже фон экрана обрабатывается операционной системой как окно.

Классы окон

Основное поведение окна определяется классом окна. Класс окна несет информацию о начальном внешнем виде окна, пиктограмме по умолчанию, курсоре и ресурсе меню, связанном с окном, а также – адрес оконной функции. Когда приложение обрабатывает сообщения, оно обычно делает это посредством вызова функции DispatchMessage() для каждого принятого сообщения - это происходит в цикле обработки сообщений. В свою очередь, функция DispatchMessage() вызывает соответствующую оконную функцию, проверяя, для какого класса окна предназначено сообщение. Именно эта оконная функция обрабатывает сообщение, переданное окну.

Существует множество стандартных классов окон, предусмотренных самой операционной системой Windows. Эти системные глобальные классы реализуют функциональные возможности общих элементов управления. Любое приложение может использовать эти классы в своих окнах, например можно реализовать элемент управления – поле ввода, используя класс окна Edit.

Приложение также может определять собственные классы окон с помощью функции RegisterClass(). Эта функция позволяет программисту реализовать поведение окна, не являющегося частью ни одного из поддерживаемых системой глобальных классов. Например, таким образом, приложение реализует функциональные возможности своего главного окна, регистрирует пиктограмму окна и ресурса меню.

Типы сообщений

Существует много разновидностей сообщений, представляющих события на разных уровнях. Каждое простое событие, каждое простое действие посылается окну в виде сообщения. Приложению приходит множество сообщений, однако приложение не должно заботиться о смысле каждого отдельного сообщения. Вместо обработки всех возможных сообщений приложение имеет свободу выбора: необработанные сообщения передаются в функции обработки сообщений операционной системы по умолчанию.

Windows-сообщение состоит из нескольких частей, для его представления используется структура MSG:

```
typedef struct tagMSG {  
    HWND hwnd; // окно, которому посылается это сообщение UINT message; //  
    значение самого сообщения  
    WPARAM wParam; // для передачи дополнительной информации LPARAM  
    lParam; // для передачи дополнительной информации DWORD time; // время, когда  
    произошло событие  
    POINT pt; // точка, где произошло события (для мыши)  
} MSG;
```

Поле hwnd идентифицирует окно, которому посылается сообщение.

Каждое окно в Windows имеет такой идентификатор. Поле message идентифицирует само сообщение. Это поле может иметь множество различных значений, означающих одно из многих различных сообщений, которые могут получать приложения Windows.

Для идентификаторов сообщений обычно используются символьные представления (WM_PAINT, WM_TIMER), а не числовые значения. Эти символьные значения определены в заголовочных файлах Windows (программисту необходимо включать в исходный программы текст файл windows.h – он содержит директивы #include для остальных заголовочных файлов).

Оконная функция

Windows посылает сообщения окну, вызывая оконную функцию. ОС Windows вызывает оконную функцию каждый раз:

- при создании окна;
- при удалении окна;
- при изменении размеров окна, перемещении, при свертывании окна;
- при выборе пункта меню;
- при манипуляции с полосами прокрутки или с мышью;
- чтобы сообщить программе о необходимости перерисовать рабочую область окна.

Все эти вызовы имеют форму сообщений. В большинстве Windows- программ основная часть программы направлена на обработку этих сообщений. Любое событие, относящееся к окну, передается оконной функции в виде сообщения. Затем оконная функция соответствующим образом реагирует на это сообщение или передает сообщение в функцию DefWindowProc() для обработки по умолчанию.

Сообщения могут быть и синхронные (queued), и асинхронные (nonqueued). Синхронными сообщениями называются сообщения, которые Windows помещает в очередь сообщений программы, и которые извлекаются и диспетчеризируются в цикле обработки сообщений. Асинхронные сообщения передаются непосредственно окну, когда Windows вызывает оконную функцию. Синхронные сообщения помещаются (отправляются) в очередь сообщений (post), а асинхронные посылаются прямо в оконную функцию (send).

Точка входа в программу

При разработке консольных приложений на языке C или C++, от разработчика требуется наличие функции main. Операционная система вызывает ее при запуске

программы. Если программе необходимо «узнать» информацию о том, какие клавиши нажаты, или воспользоваться сервисом операционной системы, она вызывает соответствующие функции.

При запуске в Windows операционная система вызывает в программе функцию WinMain(). В любом приложении обязательно должна присутствовать эта функция, на которую возлагается ряд специфических задач. Самая важная из них – создание основного окна программы, с которым должны быть связаны операторы, способные обрабатывать сообщения, передаваемые операционной системой этому окну. Точкой входа программы для Windows является функция WinMain(), которая всегда определяется следующим образом:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,  
LPSTR lpCmdLine, int nCmdShow)
```

Эта функция использует последовательность вызовов WINAPI и при завершении, возвращает операционной системе Windows целое число.

Описание параметров функции WinMain():

- hInstance - дескриптор экземпляра приложения. Дескриптор экземпляра приложения - это уникальное число, которое идентифицирует программу, в системе Windows. Каждая копия одной и той же запущенной несколько раз программы называется «экземпляром» и у каждой свое значение hInstance;
- hPrevInstance в настоящее время устарел и всегда равен NULL;
- LpCmdLine - указатель на оканчивающуюся нулем строку, в которой содержатся параметры командной строки;
- nCmdShow - определяет, как приложение первоначально отображается на дисплее: пиктограммой (nCmdShow = SW_SHOWMINNOACTIVE) или в виде открытого окна (nCmdShow = SW_SHOWNORMAL).

Функция WinMain() выполняет следующие действия:

- сохранение дескриптора экземпляра приложения в переменной;
- регистрация класса окна приложения и другие инициализации;
- создание главного окна приложения и, возможно, других, подчиненных окон;
- отображение созданного окна и прорисовка содержимого его внутренней части;
- запуск цикла обработки сообщений, помещаемых в очередь приложения;
- завершение работы приложения при извлечении из очереди сообщения WM_QUIT.

Регистрация класса окна

Обычно приложение создает окно в два шага. Сначала с помощью функции RegisterClass() регистрируется класс окна, а затем создается окно зарегистрированного класса с помощью функции CreateWindow(). Класс окна определяет общее поведение нового типа окон, включая адрес новой оконной функции. Такие второстепенные аспекты как размер, расположение и внешний вид окна определяются при его создании.

Новый класс окна регистрируется при вызове функции: `ATOM RegisterClass(const WNDCLASS *lpwc);`

Параметр lpwc указывает на структуру WNDCLASS, которая описывает класс (тип) нового окна. Возвращаемое значение - атом Windows это 16- разрядное значение, которое идентифицирует уникальную символьную строку во внутренней таблице Windows.

Поля структуры WNDCLASS:

- lpszClassName - название нового класса окон;
- lpfnWndProc – (указатель на функцию) указывает адрес оконной функции. Эта функция отвечает за обработку всех сообщений, получаемых окном. Она может обрабатывать эти сообщения сама или вызывать функцию по умолчанию DefWindowProc;

- style – определяет некоторые глобальные характеристики окна. Для этого параметра можно установить комбинацию значений, используя операцию поразрядного ИЛИ. Например, значение CS_DBLCLKS указывает Windows генерировать события о двойном щелчке кнопкой мыши. Значения CS_HREDRAW и CS_VREDRAW указывают, что окно должно полностью перерисовываться каждый раз при изменении горизонтального или вертикального размера;

- hIcon – дескриптор пиктограммы, используемой для представления минимизированных окон этого класса;

- hCursor – дескриптор стандартного указателя мыши для окон этого класса;
- hbrBackground – дескриптор кисти интерфейса GDI, для прорисовки фона окна;
- lpszMenuName - определяет ресурс меню (по имени или с помощью макроса MAKEINTRESOURCE по целому идентификатору), который используется для стандартного меню этого класса;

- cbClsExtra и cbWndExtra - используются, чтобы выделить дополнительную память для класса окна и для отдельных окон, через них задается размер дополнительной памяти. Приложения могут использовать эту память для хранения специальной информации, относящейся к классу окна или к отдельным окнам.

Создание окна

Регистрация нового класса является первым шагом в создании окна. Затем приложение должно создать окно с помощью функции CreateWindow(), которая возвращает дескриптор созданного окна типа HWND:

HWND CreateWindow(LPCSTR lpClassName, LPCSTR lpWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWndParent, HMENU hMenu, HANDLE hInstance, LPVOID *lpParam)

У каждого окна в Windows имеется уникальный дескриптор типа HWND. Дескриптор окна – это один из важнейших описателей, которыми оперирует программа для Windows. Для многих функций Windows требуется дескриптор окна, благодаря которому Windows знает, к какому окну применить функцию.

Описание параметров функции CreateWindow():

- lpClassName - указывает имя класса, поведение которого наследует данное окно. Этот класс должен быть зарегистрирован с помощью функции RegisterClass() или быть одним из предопределенных классов элементов управления. Эти переопределенные классы включают в себя стандартные классы элементов управления с именами “button”, “combobox”, “listbox”, “edit”, “scrollbar” и т. д.

- lpWindowName - определяет строку, которая выводится в заголовке окна;
- dwStyle - определяет стиль окна. Используется для инициализации локальных свойств окна. Как и в случае стиля класса, стиль окна также обычно является комбинацией значений (объединенных операцией поразрядного ИЛИ). Для определения стиля главного окна чаще всего используют стиль перекрывающегося окна, для чего через параметр dwStyle передают символическую константу WS_OVERLAPPEDWINDOW, определенную во включаемых файлах следующим образом:

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED| \
WS_CAPTION|WS_SYSMENU|WS_THICKFRAME| \
WS_MINIMIZEBOX|WS_MAXIMIZEBOX)
```

- x и y – координаты расположения окна на экране;
- nWidth и nHeight - геометрические размеры окна;
- hWndParent – дескриптор родительского окна. Если окно является главным окном приложения, то параметру hWndParent присваивается значение NULL;

- hMenu - дескриптор родительского меню. Значение NULL на месте дескриптора меню hMenu говорит о том, что у окна будет только меню класса, общее для всех окон этого класса;

- hInstance - экземпляр приложения, которое создает окно;
- lpParam - используется для передачи окну дополнительных данных (если их нет, то он должен быть равен NULL).

Отображение окна

Не смотря на то, что функция CreateWindow() создает окно, это не значит, что оно будет автоматически отображаться на экране дисплея. Для отображения окна следует воспользоваться функцией ShowWindow(). Первым параметром в эту функцию передается дескриптор окна, вторым параметром является константа, которая задает начальный вид окна на экране.

BOOL ShowWindow(HWND hwnd, int nCmdShow)

Функция ShowWindow() выводит окно на экран. Если второй параметр этой функции имеет значение SW_SHOWNORMAL, то фон рабочей области окна закрашивается той кистью, которая задана в классе окна. Если второй параметр имеет значение SW_HIDE – окно будет невидимым.

Функция UpdateWindow() передает функции окна сообщение WM_PAINT. Получив это сообщение, функция обновляет содержимое экрана.

void UpdateWindow(HWND hwnd)

Цикл обработки очереди сообщений

После создания и отображения окна функция WinMain должна подготовить приложение к получению событий от клавиатуры, мыши и т. д. Windows предоставляет очередь сообщений для каждой программы, работающей в данный момент в системе. Когда происходит ввод информации, Windows преобразовывает ее в сообщение, которое помещается в очередь сообщений приложения. Программа извлекает сообщения из очереди, выполняя блок операторов, известный как цикл обработки сообщений (message loop). Простейший цикл обработки сообщений имеет следующий вид:

```
...
MSG msg;
while(GetMessage(&msg, NULL, 0, 0)){
// Преобразование некоторых сообщений, полученных с
// помощью клавиатуры TranslateMessage(&msg);

// Отправить сообщение оконной функции DispatchMessage(&msg);
}
```

...
Цикл обработки начинается с извлечения сообщений из очереди при помощи функции:

BOOL GetMessage(MSG FAR* lpmsg,HWND hwnd, UINT uMsgFilterMin,UINT uMsgFilterMax)

Если при вызове этой функции указать вторым аргументом NULL, то программа будет получать сообщения от всех окон, созданных программой. При помощи параметров uMsgFilterMin и uMsgFilterMax можно отфильтровать сообщения, получаемые программой. Если на их месте передать 0, то программа будет получать все сообщения. Функция GetMessage() заполняет структуру MSG информацией о сообщении. Структура MSG содержит информацию об окне, которому предназначено сообщение, численное значение самого сообщения, время и координаты (для сообщений от мыши)

возникновения сообщения, а также два дополнительных параметра `wParam` и `lParam`, значения которых зависят от особенностей сообщения.

Каждое сообщение (за исключением `WM_QUIT`) направлено одному из окон приложения. Поскольку приложение не должно напрямую вызывать оконную функцию, то для передачи сообщения нужному окну используется функция:

LONG DispatchMessage(const MSG FAR* lpmsg)

Эта функция передает сообщение обратно в Windows. Windows отправляет сообщение для его обработки соответствующей оконной функции – таким образом Windows вызывает оконную функцию.

После того, как оконная функция обработает сообщение, Windows возвращает управление в приложение к следующему оператору после `DispatchMessage()`, цикл обработки сообщений в очередной раз возобновляет работу, вызывая `GetMessage()`. Если поле `message` сообщения `msg`, равно любому значению, кроме `WM_QUIT`, то функция `GetMessage()` возвращает ненулевое значение. Сообщение `WM_QUIT`, извлеченное из очереди заставляет прервать цикл обработки сообщений.

Перед вызовом функции `DispatchMessage()` могут быть вызваны специальные функции. Например, для преобразования некоторых сообщений, полученных с помощью клавиатуры, в более понятные сообщения можно при помощи функции **BOOL TranslateMessage(const MSG FAR* lpmsg)**

Завершение работы приложения

Приложение завершает свою работу тогда, когда функция `WinMain()` передает управление Windows. Передать управление можно в любом месте `WinMain()`.

После входа в цикл обработки очереди сообщений, единственным способом завершить приложение является посылка в очередь приложения сообщения `WM_QUIT`. Сообщение `WM_QUIT`, является средством завершения работы приложения. Оно обычно помещается в очередь сообщений только оконной функцией данного приложения, при помощи функции `PostQuitMessage()`.

Когда функция `GetMessage()` извлекает сообщение `WM_QUIT` из очереди, она возвращает нуль (`FALSE`), что приводит к завершению работы приложения. При завершении приложения следует освободить занятые ресурсы.

Обработка сообщений

При обработке сообщений главная роль отводится функции окна (оконной функции). Оконная функция может иметь любое имя. В программе может содержаться более одной оконной функции. Оконная функция всегда связана с определенным классом окна.

Оконная функция всегда определяется следующим образом:

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);

Все четыре параметра оконной функции идентичны первым четырем полям структуры `MSG`. Первым параметром является дескриптор окна, получающего сообщение. Если в программе создается несколько окон на основе одного и того же класса окна тогда параметр `hWnd` идентифицирует конкретное окно, которое получает сообщение.

Функция вызывается непосредственно операционной системой Windows и не может вызываться приложением напрямую.

Каждое получаемое окном сообщение идентифицируется номером, который содержится в параметре `msg` оконной функции. Если оконная функция обрабатывает сообщение, то ее возвращаемым значением должен быть 0.

Все сообщения, не обрабатываемые оконной функцией, должны передаваться в функцию `DefWindowProc()` (такой механизм позволяет Windows обрабатывать окно совместно с приложением). При этом значение, возвращаемое функцией `DefWindowProc()`, должно быть возвращаемым значением оконной функции. Функция

DefWindowProc() играет ключевую роль в формировании информационных потоков сообщений Windows, и ее вызов в функции окна обязателен.

Удаление окна

Сообщение WM_DESTROY является еще одним важным сообщением. Это сообщение показывает, что Windows находится в процессе ликвидации окна в ответ на полученную от пользователя команду (пользователь вызывает поступление этого сообщения, если нажмет мышью на пиктограмме “Close”, выберет пункт “Close” из системного меню или нажмет комбинацию клавиш Alt+F4).

Главное окно стандартно реагирует на это сообщение, вызывая функцию PostQuitMessage(). Эта функция ставит сообщение WM_QUIT в очередь сообщений приложения. Это заставляет функцию WinMain прервать цикл обработки сообщений и выйти в систему, завершив работу приложения.

Создание проекта в среде VisualStudio

Для создания проекта необходимо:

1. выбрать пункт меню File → New Project;
2. в левой части окна (Project types) выбрать тип проекта Visual C++ Win32;
3. в правой части окна (Templates) выбрать Win32 Project;
4. в поле «Name» ввести имя проекта, в поле «Location» выбрать каталог проекта и нажать «Ok» затем «Next»;
5. в окне «Application Settings» установить поле «Empty project»
6. выбрать пункт главного меню Project Properties, и в левой части окна выбрать пункт Configuration Properties → General, затем в правой части окна установить параметр «Character Set» в значение «Not Set» и нажать «Ok»;
7. для добавления нового файла в проект необходимо нажать правой кнопкой мыши на имени проекта и в выпадающем меню выбрать пункт Add New Item, в появившемся окне выбрать «C++ File (.cpp)», в поле Name указать имя файла, например main и нажать кнопку «Add»;
8. в новом файле можно разместить текст программы (например, из раздела 1.4 этой лабораторной работы)
9. для компиляции проекта нужно нажать правой кнопкой мыши на имени проекта и в выпадающем меню выбрать пункт «Build» или «Rebuild»

Порядок выполнения работы

На занятии выполните следующие учебные эксперименты:

Написать программу, которая создает стандартное окно windows, при нажатии левой кнопки мыши, на экране появляется сообщение «Mouse click». Текст программы приведен ниже.

```
#include <windows.h>
// Заголовок оконной функции
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam,
LPARAM lParam);
// Функция WinMain
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow){
    char class_name[] = "main_window"; // Название класса окна char app_title[] =
"Simple window"; // Заголовок главного окна
    WNDCLASS wc; // Структура для информации о классе окна HWND hWnd; //
Дескриптор главного окна приложения MSG msg; // Структура для хранения
сообщения
    // Заполнение структуры WNDCLASS для регистрации класса окна.
    memset(&wc, 0, sizeof(wc)); // Заполнение структуры нулями
```

Вопросы для самоконтроля

1. Назовите основные компоненты оконного приложения Windows
2. Опишите принципы исполнения оконного приложения операционной системой.
3. Назовите известные вам сообщения, передаваемые между окнами.

Практическое занятие №11

Создание и использование элементов управления

Цель и содержание: изучить основной набор элементов управления, научиться создавать элементы управления, а также обрабатывать события элементов управления.

Теоретические сведения

Окно (window) – один из наиболее важных объектов в ОС Windows. Окно представляет собой прямоугольную область, в которой приложение отображает выводимую информацию и из которой получает вводимую. Кроме того, окно участвует и в программных интерфейсах – оно является получателем *сообщений*. Существует много разновидностей окон: диалоговые, окна многодокументных интерфейсов (MDI), окна сообщений, окна элементов управления. Присутствующие в системе окна образуют иерархию, в которой они могут находиться в отношениях владелец – подчиненный и предок – потомок. Окна имеют заголовки (*Title, Caption*), но в большинстве случаев обращение к ним происходит по описателям – *Handle* (системный тип *HWND*)

Класс окна (Window Class) определяет общие свойства и особенности поведения группы окон. Классы делятся на системные глобальные (к ним относятся, например, стандартные элементы управления), прикладные глобальные (реализуются в DLL, требуют регистрации, после чего могут использоваться любыми приложениями) и прикладные локальные (регистрируются как доступные единственному приложению). Именно класс связывает окно с его оконной процедурой. Классы идентифицируются, как правило, по их именам.

Для создания и регистрации классов служат функции *RegisterClass()* и *RegisterClassEx()*, для создания окон любых видов – функции *CreateWindow()* и *CreateWindowEx()*.

Оконная процедура (Window Procedure или WndProc) является получателем всех сообщений, адресованных окну. Обычно представляет собой конструкцию вида *switch – case*, распознающую поступившие сообщения и выполняющую их обработку:

```
LRESULT CALLBACK
MainWndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case M_CLOSE: //команда на закрытие окна DestroyWindow( hWnd);
            break;
        case WM_DESTROY: //подтверждение о готовности деинициализации
            PostQuitMessage( 0);
            break;
        case ...
            break;
        default: //для всех остальных сообщений
            return DefWindowProc( hWnd, uMsg, wParam, lParam);
    }
    return 0;
}
```

(В данном примере правильнее было бы проверять адресата сообщений и начинать завершение приложения, только если это его главное окно.)

Сообщения (Window Message) лежат в основе механизмов событийного управления приложениями. Они представляют собой структуры, передаваемые между приложениями и несущими информацию о событиях, команды, данные и так далее. Для представления сообщений служит структура *MSG*, поля которой содержат: тип сообщения, пару

параметров wParam, lParam, адресата сообщения, время и экранные координаты его возникновения (имеет смысл для некоторых сообщений).

Сообщения передаются в приложение либо напрямую, либо через очередь. Прямая передача сообщения оконной процедуре является фактически косвенным вызовом этой процедуры, в том числе и из другого приложения. Для этого служит системная функция SendMessage(). Важно, что выполнение отправителя сообщения блокируется до завершения оконной процедуры.

Многие сообщения передаются не напрямую, а через очередь, что избавляет программы от взаимной зависимости и блокировок. Для этого служит функция PostMessage(). Кроме того, есть некоторые другие функции, сочетающие свойства синхронной и асинхронной отсылки, например SendNotifyMessage(), которая передает сообщение минуя очередь, но и не ожидает окончания его обработки, поэтому блокирующей не является.

Цикл обработки сообщений выбирает сообщения из очереди и перенаправляет их в соответствующие оконные процедуры. Упрощенно цикл образуют три основные функции:

```
MSG msg;
while (GetMessage(&msg,NULL,0,0)) //выборка сообщений из очереди
{
    TranslateMessage(&msg); //дон. обработка (трансляция) сообщения
    DispatchMessage(&msg); //передача сообщения в оконную процедуру
}
```

Здесь выбираются все сообщения для всех окон приложения, однако параметры GetMessage() позволяют накладывать на них фильтры.

Функция GetMessage() является блокирующей – при отсутствии сообщений в очереди она переводит приложение в состояние ожидания. При наличии подходящего по фильтрам сообщения оно удаляется из очереди, и его содержимое переносится в структуру MSG. Функция возвращает нулевое значение при получении единственного сообщения WM_QUIT, для всех остальных – ненулевое. Как следствие цикл прекращается после приема этого сообщения.

Типичное приложение Windows строится по следующей общей схеме:

- 1) WinMain() – головной модуль программы;
- 2) инициализация внутренних переменных и прочие подготовительные операции;
- 3) RegisterClass() – регистрация локального класса;
- 4) CreateWindow() – создание главного окна приложения (если требуется, могут быть и другие классы и окна;
- 5) ShowWindow(), UpdateWindow() – отображение главного окна;
- 6) цикл обработки сообщений: GetMessage() – TranslateMessage() – DispatchMessage().
- 7) освобождение выделенных ресурсов, деинициализация внутренних переменных;
- 8) оконная процедура, прямых вызовов нет, так как предполагается «обратный вызов» (callback) со стороны системы.

Контрольные вопросы

1. Стандартная структура оконного приложения, его основные функции.
2. Окно. Функция создания нового окна и ее параметры.
3. Что такое оконный класс и каким образом он может быть создан.
4. Каким образом для окна можно указать собственные курсор и иконку.
5. За что отвечают функции UpdateWindow() и ShowWindow().
6. Сообщение Windows (Win Message), его основные параметры. Очередь сообщений.

7. Способы отправки сообщений другому окну.
8. Цикл обработки сообщений, функция GetMessage(), завершение цикла.
9. Функции TranslateMessage(), DispatchMessage().
10. Назначение, логика работы и параметры оконной процедуры.
11. Сообщение WM_PAINT. Функция вывода текста в окне, ее параметры.
12. Сообщение WM_COMMAND. Данные, передаваемые в wParam и lParam в данном сообщении.
11. Создание собственного пункта меню и обработка нажатия на него.
12. Функции GetWindowRect() и InvalidateRect().
13. Таймер. Создание и удаление таймера. Обработка сообщений таймера.

Задание

Написать программу, которая создает окно с собственными курсором и иконкой. В главное меню добавить подменю с двумя пунктами Start и Stop. При нажатии на пункт меню Start в окне должна появиться движущаяся по середине окна надпись. После этого выбор пункта меню Stop должно приостанавливать движение надписи, а пункта Start – возобновлять. Для реализации движения использовать обработчик сообщения WM_PAINT и таймер.

Практическое занятие №12

Графический ввод-вывод в оконном приложении

Цель и содержание: изучить графическую подсистему GDI, научиться использовать графику GDI.

Теоретические сведения

GDI (Graphics Device Interface) представляет собой единый унифицированный интерфейс устройств (средств) отображения графической информации в Windows. Работа GDI базируется на понятии *контекста* устройства (*device context – DC*), который абстрагирует свойства реальных устройств: экран (окно на экране), принтер, битовый образ в памяти и так далее. Контекст идентифицируется его описателем (тип handle DC – HDC).

Получив контекст, программа может обращаться к нему с единым набором функций, причем поведение контекста (изображения контекста) должно быть одинаковым независимо от того, с каким устройством он связан.

Для получения контекста служат функции: *GetDC()*, *GetWindowDC()*, *GetDCEx()*. Они применимы для оконных (экранных) контекстов. Функции *CreateDC()* и *CreateCompatibleDC()* создают контексты, связанные с иными (не окна) объектами либо «контексты в памяти», не связанные с реальным устройством. Освобождение выполняется функциями *ReleaseDC()* для оконных контекстов и *DeleteDC()* – для остальных.

Для формирования изображения в контексте служат функции графических примитивов (например, *Ellipse()*, *DrawText()* и так далее) и графические инструменты (объекты). Основными инструментами являются: «перо» (*Pen*),

«кисть» (*Brush*) и «шрифт» (*Font*). Логика GDI такова, что система при отображении примитива сама выбирает соответствующий инструмент: например, при изображении закрашенного полигона текущее «перо» будет использовано для прорисовки его контура, а текущая «кисть» – для заполнения внутренней области.

Объекты типа *Bitmap* или *Metafile* сами способны хранить изображение, поэтому могут служить «рабочей поверхностью» контекста. Объекты «палитра», «область отсечения» и некоторые другие дополнительно влияют на формирование изображения.

Для создания инструментов служат соответствующие функции GDI API, например *CreatePen()*, *CreateBrush()* и так далее. Для сложных объектов

может быть определено несколько функций, различающихся параметрами и получаемым эффектом.

Контекст может иметь только по одному активному объекту каждого вида. Для их переключения служит функция *SelectObject()*, которая принимает экземпляр объекта (инструмента) и делает его текущим в заданном контексте; тип объекта определяется автоматически, и предыдущий объект этого типа выталкивается из контекста.

Общая схема отрисовки следующая:

- получение контекста;
- установка набора инструментов;
- формирование изображения из примитивов, в том числе со сменой используемых инструментов;
- освобождение контекста.

Для оконных контекстов, которые после окончания их использования не удаляются, а лишь освобождаются, перед освобождением следует восстановить те же инструменты, которые были активными при получении контекста.

В типичном случае перерисовка содержимого окна инициируется сообщением *WM_PAINT* – оно сигнализирует, что текущее видимое содержимое в окне было утрачено, и система уже выполнила автоматическую перерисовку самого окна и его фона.

Задание

Написать программу, которая создает окно с двумя элементами управления типа самоотрисовывающейся кнопки (OWNERDRAW), для каждой из которых должен быть задан какой-нибудь рисунок. При нажатии на первую из них в окне выводится рисунок, состоящие не менее чем из двадцати графических примитивов (линия, дуга, полигон, прямоугольник, эллипс – каждый из перечисленных примитивов должен быть использован хотя бы два раза). Вторая из кнопок должна стирать данный рисунок.

Контрольные вопросы

1. Контексты графических устройств.
2. Последовательность формирования изображения в окне.
3. Активные инструменты Pen и Brush, их создание и выбор. Параметры функций создания Pen и Brush.
4. Как могут быть нарисованы линия, эллипс, дуга, прямоугольник. Параметры функций, рисующих данные графические примитивы.
5. Каким образом может быть нарисован Polyline.
6. Самоотрисовывающаяся кнопка (OWNERDRAW). Как она может быть создана, как на ней рисовать собственный рисунок и как обрабатывать нажатия на такие кнопки.

Практическое занятие №13

Динамическое отображение данных на окне

Цель и содержание: изучение способов динамического отображения данных на окне.

Теоретические сведения

Помимо описанных выше графических примитивов GDI работает также и с растровыми изображениями, то есть целыми массивами (матрицами) пикселей. В простейшем случае изображение может формироваться попиксельно с помощью функции SetPixel(), но обычно этот путь слишком медленный и неудобный. (Имеется также функция GetPixel() – доступ к отдельным пикселям изображения.)

GDI предоставляет ряд функций для манипулирования целыми фрагментами изображений:

BitBlt() – копирование прямоугольной области из одного контекста в другой без каких-либо преобразований;

StretchBlt() – копирование области с масштабированием по обеим осям (обычно с уменьшением, так как при растяжении изображение заметно портится);

MaskBlt() – перенос с маскированием части изображения;

PlgBlt() – перенос прямоугольного фрагмента в непрямоугольную область с соответствующим геометрическим искажением.

В ряде случаев формирование всего изображения заново может быть слишком длительным – например, различного рода анимация или изображения из очень большого числа элементов. В этом случае эффективно использование «теневых» контекстов или «контекстов в памяти». Такие контексты не имеют связи с реальным устройством, вместо этого рабочую поверхность для них образует битовая карта (bitmap) или метафайл. (Под метафайлом здесь понимается объект Metafile, фактически хранящий последовательность применявшихся к нему графических примитивов и способный их воспроизвести.)

«Теневой» контекст создается совместимым с текущим экранным режимом, и с ним ассоциируется «битовая карта»:

hShadowDC = CreateCompatibleDC(NULL);

hBitmap = CreateCompatibleBitmap(hPrimDC, width, height);

SelectObject(hShadowDC, hBitmap);

Здесь hPrimDC – «реальный» контекст, в котором необходимо формировать изображение.

После создания «теневого» контекста в нем формируется необходимое изображение. Далее в процессе работы программы это изображение или его фрагменты переносятся на «основной» (видимый) контекст.

Задание к работе

В графическом редакторе сделать несколько (например, четыре) изображений движущегося объекта небольшого размера (например, 32×32 пикселя). Реализовать поочередное отображение их в заданной позиции в окне. При щелчке мышью начинается движение объекта из текущей позиции в позицию указателя мыши с поочередной сменой изображений для имитации фаз движения. При отрисовке используются «теневые» контексты.

Контрольные вопросы

1. Использование таймера.
2. Теневые графические контексты.
3. Формат BMP файла.
4. Отображение BMP картинки.
5. Анимация движения.

Практическое занятие №14

Обмен сообщениями между окнами, обработка сообщений ввода-вывода

Цель и содержание: изучение обмена сообщениями между окнами, обработка сообщений ввода-вывода.

Теоретические сведения

GDI (Graphics Device Interface) представляет собой единый унифицированный интерфейс устройств (средств) отображения графической информации в Windows. Работа GDI базируется на понятии *контекста* устройства (*device context – DC*), который абстрагирует свойства реальных устройств: экран (окно на экране), принтер, битовый образ в памяти и так далее. Контекст идентифицируется его описателем (тип handle DC – HDC).

Получив контекст, программа может обращаться к нему с единым набором функций, причем поведение контекста (изображения контекста) должно быть одинаковым независимо от того, с каким устройством он связан.

Для получения контекста служат функции: *GetDC()*, *GetWindowDC()*, *GetDCEx()*. Они применимы для оконных (экранных) контекстов. Функции *CreateDC()* и *CreateCompatibleDC()* создают контексты, связанные с иными (не окна) объектами либо «контексты в памяти», не связанные с реальным устройством. Освобождение выполняется функциями *ReleaseDC()* для оконных контекстов и *DeleteDC()* – для остальных.

Для формирования изображения в контексте служат функции графических примитивов (например, *Ellipse()*, *DrawText()* и так далее) и графические инструменты (объекты). Основными инструментами являются: «перо» (*Pen*), «кисть» (*Brush*) и «шрифт» (*Font*). Логика GDI такова, что система при отображении примитива сама выбирает соответствующий инструмент: например, при изображении закрашенного полигона текущее «перо» будет использовано для прорисовки его контура, а текущая «кисть» – для заполнения внутренней области.

Объекты типа *Bitmap* или *Metafile* сами способны хранить изображение, поэтому могут служить «рабочей поверхностью» контекста. Объекты «па- литра», «область отсечения» и некоторые другие дополнительно влияют на формирование изображения.

Для создания инструментов служат соответствующие функции GDI API, например *CreatePen()*, *CreateBrush()* и так далее. Для сложных объектов может быть определено несколько функций, различающихся параметрами и получаемым эффектом.

Контекст может иметь только по одному активному объекту каждого вида. Для их переключения служит функция *SelectObject()*, которая принимает экземпляр объекта (инструмента) и делает его текущим в заданном контексте; тип объекта определяется автоматически, и предыдущий объект этого типа выталкивается из контекста.

Общая схема отрисовки следующая:

- получение контекста;
- установка набора инструментов;
- формирование изображения из примитивов, в том числе со сменой используемых инструментов;
- освобождение контекста.

Для оконных контекстов, которые после окончания их использования не удаляются, а лишь освобождаются, перед освобождением следует восстановить те же инструменты, которые были активными при получении контекста.

В типичном случае перерисовка содержимого окна инициируется сообщением *WM_PAINT* – оно сигнализирует, что текущее видимое содержимое в окне было утрачено, и система уже выполнила автоматическую перерисовку самого окна и его фона.

Задание к работе

Написать программу, которая создает окно с двумя элементами управления типа самоотрисовывающейся кнопки (OWNERDRAW), для каждой из которых должен быть задан какой-нибудь рисунок. При нажатии на первую из них в окне выводится рисунок, состоящие не менее чем из двадцати графических примитивов (линия, дуга, полигон, прямоугольник, эллипс – каждый из перечисленных примитивов должен быть использован хотя бы два раза). Вторая из кнопок должна стирать данный рисунок.

Контрольные вопросы

1. Контексты графических устройств.
2. Последовательность формирования изображения в окне.
3. Активные инструменты Pen и Brush, их создание и выбор. Параметры функций создания Pen и Brush.
4. Как могут быть нарисованы линия, эллипс, дуга, прямоугольник. Параметры функций, рисующих данные графические примитивы.
5. Каким образом может быть нарисован Polyline.

Практическое занятие №15

Использование потоков

Цель и содержание: изучить способы использования потоков и управления ими с использованием средств WinAPI.

Теоретические сведения

Поток (Thread) – системный объект, соответствующий последовательности («потоку») команд, выполняемой независимо и асинхронно по отношению к другим подобным последовательностям. Поток – базовый объект, для которого планировщик задач ОС распределяет время центрального процессора. Каждый процесс имеет как минимум один главный (*первичный – primary*) поток. Главный поток может порождать подчиненные (*вторичные*) потоки, которые будут выполняться одновременно с ним, равно как и с потоками прочих процессов.

Многопоточность, свойственная Win32 и ряду других современных ОС, удачно дополняет их многозадачность и существенно повышает гибкость системы. Процесс в Win32 играет роль владельца ресурсов и «контейнера потоков» планирования ресурсов. Ресурсы процесса доступны всем его потокам, и все потоки одного процесса совместно используют его виртуальное адресное пространство (но каждый поток имеет собственный стек). В то же время процессорное время распределяется именно между потоками, а не между процессами.

Поток может находиться в нескольких состояниях, начиная от его инициализации до завершения, но наибольший интерес представляют следующие:

- *выполнение (running)* или *активность (active)* – поток обладает всеми необходимыми ресурсами, включая процессорное время, и выполняется;
- *готовность (ready)* – поток обладает ресурсами для выполнения, но время ему не выделено, и он ожидает активизации в очереди планировщика;
- *ожидание (wait)*, или *заблокированное (blocked)*, или «спящее» (*sleep*) – поток не обладает требуемыми ресурсами и не может выполняться, он ожидает выполнения необходимых условий или наступления каких-либо событий.

Например, активный поток переходит в состояние ожидания после обращения к *блокирующему* системному вызову до его завершения, затем перемещается в очередь планировщика, откуда выбирается для следующей активизации. В простейшем случае переход в состояние ожидания произойдет при выполнении вызова Sleep().

Планировщик задач ОС в рамках принятой в Win32 модели многозадачности выделяет процессорное время отдельным потокам *квантами*. Переключение потоков и изменение их состояний происходит по мере израсходования выделенных квантов и в соответствии с приоритетами потоков.

Потоки идентифицируются их *описателями (handle)*, уникальными в рамках одного процесса, и идентификаторами (уникальны в системе). Однако большинство функций работают именно с описателями. Поток всегда сам является обладателем специального *локального* описателя, поэтому закрытие

«внешнего» его описателя не приводит к удалению объекта. Одновременно поток как объект системы не может быть удален даже после прекращения его выполнения, если «внешний» описатель не был закрыт. Неверная работа с описателями может приводить при интенсивном создании и разрушении потоков к заметному накоплению «мусора» и дальнейшим ошибкам.

Новые потоки порождаются из *функций потока* с помощью вызова CreateThread(). Функция потока – одна из функций головной программы, соответствующая требованиям по типу, параметрам и формату вызова: DWORD WINAPI ThreadProc(LPVOID pParam)

```

{
... //действия, выполняемые в потоке return result;
}

```

Интерпретации как параметра, так и возвращаемого значения произвольны и зависят от конкретной программы. Например, параметр может быть адресом структуры или просто целым числом (используя приведение типа).

Все операции, выполняемые этой функцией, включая вызовы других подпрограмм, будут выполняться внутри созданного потока. Поток завершается после завершения его функции. Возвращаемое ею значение будет доступно после завершения потока до его окончательного разрушения.

Последним параметром `CreateThread()` передается указатель на ячейку типа `DWORD`, куда записывается возвращаемый идентификатор созданного потока. В Windows 9x этот указатель не должен быть нулевым. В Windows NT такого ограничения нет.

Уже существующий поток может быть открыт вызовом `OpenThread()`, для этого необходимо знать его идентификатор.

Для корректного завершения потока служит функция `ExitThread()`, вызываемая «изнутри» потока, её аргументом является код возврата потока. Фактически результат ничем не отличается от стандартного завершения функции потока. Принудительное завершение потока, в том числе и «извне», возможно с помощью функции `TerminateThread()`, но при этом глобальные данные и другие ресурсы, с которыми он в этот момент работал, могут остаться в некорректном состоянии. Поэтому предпочтительнее организовать взаимодействие так, чтобы завершение потоков согласовывалось с их текущим состоянием, оптимально – обеспечивалось самим же потоком.

Кроме завершения, поток может быть приостановлен и вновь запущен функциями `SuspendThread()` и `ResumeThread()` соответственно. В действительности для потока ведется счетчик, при нулевом значении которого он может выполняться, при ненулевом (положительном) – останавливается. `SuspendThread()` увеличивает «счетчик остановок», `ResumeThread()` уменьшает его, но не ниже нулевого значения. Это позволяет делать «вложенные» приостановки потока без риска ошибочно разблокировать его, но требует следить за сбалансированностью приостановок.

Остановленный поток не участвует в планировании выполнения, и его разблокирование зависит только от описанного счетчика.

Варианты заданий

В каждом из заданий на главном окне должны быть созданы две кнопки `Start` и `Stop`. Нажатие кнопки `Start` запускает указанные в задании потоки. Нажатие кнопки `Stop` приостанавливает их. Выполнение потоков можно возобновить повторным нажатием кнопки `Start`. При выполнении задания по возможности использовать одну поточную функцию, передавая ей необходимые данные как параметр.

1. Должны быть реализованы три потока, каждый из которых осуществляет передвижение собственной надписи по главному окну. Все надписи должны быть различны и двигаться с разной скоростью.

2. Главное окно должно быть разделено на четыре части. Также должны быть созданы четыре потока, каждый из которых раз в секунду изменяет цвет фона в своей части окна.

3. На главном окне должны быть созданы три элемента управления `Edit`. Необходимо создать три потока, каждый из которых раз в две секунды установит случайное число в соответствующий `Edit`.

4. При помощи трех потоков необходимо реализовать движение трех квадратиков по главному окну в случайных направлениях. Каждый поток должен двигать собственный квадратик.
5. При помощи пяти потоков реализовать падение пяти букв с различной скоростью по главному окну.
6. При помощи двух потоков необходимо реализовать движение двух шариков друг за другом по синусоиде.
7. При помощи двух потоков реализовать вращение двух палочек по экрану с различной скоростью.
8. Главное окно должно быть разделено диагональю на две части. Необходимо создать два потока, каждый из которых раз в полсекунды закрашивает соответствующую часть главного окна в случайный цвет.
9. При помощи трех потоков реализовать механические часы: первый поток должен двигать часовую стрелку, второй минутную и третий секундную.
10. При помощи трех потоков реализовать электронные часы: первый поток должен выводить значение часа, второй – минуты, а третий – секунды.
11. Реализовать шесть потоков, каждый из которых двигает свой шарик по окружности.

Контрольные вопросы

1. Понятие потока.
2. Создание потока, параметры вызова.
3. Что такое функция потока? Каковы ее параметры?
4. Функции приостановки и возобновления потока.
5. Функции Sleep(), SleepEx().

Практическое занятие №16

Синхронизация доступа к ресурсам

Цель и содержание: изучить предусмотренные в Win32 средства синхронизации и соответствующие системные объекты, научиться синхронизировать ресурсы при помощи различных объектов синхронизации

1. Изучить

Теоретические сведения

В многозадачной (многопоточной) среде возникает проблема одновременного доступа к одним и тем же ресурсам (данным) со стороны нескольких потоков. В случае если конкретный ресурс не допускает такого использования, возникает конфликт, требующий разрешения. Это называют задачей *синхронизации (взаимного исключения)*.

Критический ресурс – ресурс (объект, данные), который в силу своей физической природы либо логики использования не может быть доступен одновременно нескольким пользователям. В зависимости от контекста речь может идти об определенном сочетании обращений или любых обращениях вообще (например, одновременное чтение допустимо, но одновременная запись или чтение и запись – нет).

Примерами критических ресурсов могут служить устройство вывода (одновременно выводимые данные нескольких источников будут перемешаны), записываемый файл (чтение до окончания записи дает неверные результаты), счетчик цикла (постороннее изменение нарушает работу цикла) и так далее.

Критическая секция – участок кода, выполняющий обращение или логически связную последовательность обращений к критическому ресурсу. В зависимости от контекста может быть удобнее представлять, что критическая секция одна и определяется связью ее с критическим ресурсом, либо однотипные критические секции принадлежат различным пользователям, но связаны посредством единого критического ресурса.

В этих терминах задача исключения сводится к обеспечению единственности пользователя, находящегося внутри критической секции, связанной с данным критическим ресурсом.

Проблема усугубляется еще и тем, что проверка условий возможности доступа должна быть неотделима от самого доступа или хотя бы блокировки его для других пользователей, то есть *атомарность*. Так как прикладная программа обычно не может обеспечить это самостоятельно, многозадачные системы обязательно предоставляют механизмы синхронизации.

Простейшее средство исключения в Windows – объект CriticalSection. Он представляет собой обычную структуру и не имеет глобальной идентификации, поэтому может использоваться только потоками одного процесса. Использование CriticalSection выглядит в общем случае следующим образом:

```
CRITICAL_SECTION cs;  
InitializeCriticalSection( &cs);  
...  
EnterCriticalSection( &cs);  
... //защищенная критическая секция  
LeaveCriticalSection( &cs);  
...  
DeleteCriticalSection( &cs);
```

Вызов EnterCriticalSection – попытка входа в критическую секцию. Если секция уже занята другим потоком, то поток блокируется до момента ее освобождения. Повторное вхождение в критическую секцию допускается только для одного и того же потока, что помогает избегать самоблокировок. Блокировка управляется счетчиком,

каждое вхождение в секцию увеличивает счетчик, освобождение – уменьшает, поэтому их количество должно быть сбалансировано.

Для синхронизации потоков разных процессов предусмотрены специальные объекты: Event, Mutex, Semaphore, WaitableTimer. Все они являются именованными и допускают глобальную идентификацию по именам.

Сам перевод потока в состояние ожидания осуществляется функциями WaitForSingleObject(), WaitForMultipleObjects() и их разновидностями. В зависимости от параметров эти функции блокируют выполнение потока до обнаружения одного или нескольких переданных им объектов в состоянии

«готовности» (*signaled*).

Объект «событие» (Event) – наиболее простая разновидность. Объекты создаются функцией CreateEvent() и бывают двух типов: «ручные» и «автоматические». Открытие существующего объекта «события» по его имени – OpenEvent(). Установка события (перевод в состояние signaled) происходит всегда явно функцией SetEvent(), сброс – для «ручных» событий явно, функцией ResetEvent(), для «автоматических» – автоматически при успешном выполнении Wait-функции. Также имеется функция PulseEvent() – временная установка события, активизация всех ожидавших его потоков и автоматический сброс. Кроме того, объекты Event ассоциируются с файлами при организации асинхронного ввода-вывода.

Объект «мьютекс» (Mutex) – простейший двузначный семафор для организации критических секций. Принято считать, что он «захватывается» потоком, и если мьютекс в этот момент уже занят, очередной захватывающий его поток блокируется. Подобно CriticalSection, Mutex допускает повторный захват, но только одним и тем же потоком. Создание нового мьютекса – вызов CreateMutex(), открытие существующего по имени – OpenMutex(), освобождение – ReleaseMutex(), захват с возможной блокировкой – Wait-функции.

Объект «семафор» (Semaphore) – отличается от мьютекса тем, что является счетчиком и может принимать множество значений от нуля и выше.

«Занятым» считается семафор с нулевым значением, с ненулевым – свободным. При попытке опустить значение ниже нуля происходит блокировка. Работа семафора не зависит от того, разные потоки обращаются к нему или один и тот же. Создание нового семафора – вызов CreateSemaphore(), открытие существующего по имени – OpenSemaphore(), «подъем» счетчика (разблокирование) – ReleaseSemaphore(), проверка и «опускание», в том числе блокирование – Wait-функции.

Кроме специализированных объектов синхронизации Wait-функции могут работать также и с другими объектами, например:

- процессы и потоки – ожидание завершения;
- файлы – ожидание окончания.

В каждом из заданий необходимо создать несколько потоков и защищенный ресурс. Каждый из потоков должен делать следующее:

проверить, свободен ли защищенный ресурс; если занят, то дожидаться освобождения; если свободен, то занять его, выполнить какие-то действия (указанные в задании), сделать паузу на одну секунду и освободить ресурс. Если в задании указано два объекта синхронизации, то необходимо выполнить отдельную программу для каждого из них.

1. Каждый из трех потоков должен пытаться закрасить главное окно в свой цвет: первый – в синий, второй – в красный и третий – в зеленый. В результате каждую секунду цвет фона главного окна будет изменяться. Реализовать синхронизацию доступа к ресурсам через Event, а затем через CriticalSection.

2. Каждый из трех потоков должен пытаться закрасить главное окно в свой цвет: первый – в желтый, второй – в голубой и третий – в черный. В результате каждую секунду

цвет фона главного окна будет изменяться. Реализовать синхронизацию доступа к ресурсам через Mutex, а затем через Semaphore.

3. На главном окне необходимо создать Edit. Каждый из трех потоков должен пытаться установить в данный Edit соответствующий текст: First, Second или Third. Реализовать синхронизацию доступа к ресурсам через Event, а затем через CriticalSection.

4. На главном окне необходимо создать Edit. Каждый из трех потоков должен пытаться установить в данный Edit соответствующий текст: String1, String2, String3. Реализовать синхронизацию доступа к ресурсам через Mutex, а затем через Semaphore.

5. На главном окне необходимо нарисовать движущуюся справа налево фигуру (например квадрат). Также необходимо создать два потока: первый из них будет опускать фигуру вниз, а второй – поднимать вверх. Синхронизацию доступа к ресурсам реализовать через Event, а затем через CriticalSection.

Контрольные вопросы

1. Что такое синхронизация доступа к ресурсам и зачем она нужна?
2. Объекты синхронизации в Win32.
3. Объект синхронизации CriticalSection, его использование.
4. Функция WaitForSingleObject, ее параметры и возвращаемые значения. Использование данной функции.
5. Объект синхронизации Event, его создание, уничтожение и использование. Параметры данных функций.
6. Объекты Event с автоматическим сбросом.
7. Объект синхронизации Mutex, его создание, уничтожение и использование.

Практическое занятие №17

Приоритеты процессов и потоков

Цель и содержание: изучить систему приоритетов процессов и потоков ОС Windows, а также средства управления приоритетами.

Теоретические сведения

Процессорное время выделяется потокам в соответствии с их уровнем приоритета. Потоку с более низким приоритетом не выделяется время, если на него претендует поток с более высоким уровнем приоритета. Более того, процесс с более низким приоритетом прерывается до истечения кванта времени, если на процессор претендует более приоритетный поток.

Уровни приоритетов варьируются в диапазоне от 0 (низший) до 31 (высший).

Действующий приоритет каждого потока образуют три составляющие: *класс приоритета* процесса, *уровень приоритета* потока внутри класса приоритета процесса, *динамический* уровень приоритета.

Класс приоритета процесса и уровень приоритета потока внутри класса определяют базовый уровень приоритета потока. Привилегированными считаются приоритеты от 16 до 31, они резервируются за системными программами реального времени. Прикладная программа получить приоритет выше 15 не может.

Определены следующие классы приоритетов, которым соответствуют константы:

- *Idle* (простаивающий) – IDLE_PRIORITY_CLASS (4), процесс активизируется только при простое других процессов;
- *Normal* (нормальный) – NORMAL_PRIORITY_CLASS (7), большинство процессов в системе, в частности, все процессы пользователя; приоритет владеющего активным окном процесса повышается на 2 и составляет 9;
- *High* (высокий) – HIGH_PRIORITY_CLASS (13), системные процессы, реагирующие на соответствующие события;
- *Real time* (реального времени) – REALTIME_PRIORITY_CLASS (24), некоторые системные процессы в "особых случаях".

Внутри классов приоритетов процессов определены уровни приоритетов потоков:

- *низший* (THREAD_PRIORITY_LOWEST) – –2 от уровня класса;
- *пониженный* (THREAD_PRIORITY_BELOW_NORMAL) – –1 от уровня класса;
- *нормальный* (THREAD_PRIORITY_NORMAL) – равен уровню класса;
- *повышенный* (THREAD_PRIORITY_ABOVE_NORMAL) – +1 к уровню класса;
- *высший* (THREAD_PRIORITY_HIGHEST) – +2 к уровню класса;
- *простаивающий* (THREAD_PRIORITY_IDLE) – равен 16 для REALTIME_PRIORITY_CLASS и 1 для остальных классов;
- «*критический*» (THREAD_PRIORITY_TIME_CRITICAL) – равен 31 для REALTIME_PRIORITY_CLASS и 15 для остальных классов.

Для класса REALTIME_PRIORITY_CLASS может использоваться также расширенный диапазон значений – от –7 до +6. Начиная с Windows 2003 были добавлены два специальных значения: THREAD_MODE_BACKGROUND_BEGIN и THREAD_MODE_BACKGROUND_END, они связаны с дополнительными возможностями планировщика.

Динамический уровень приоритета образуется повышением базового уровня потока на две единицы при поступлении сообщений в его очередь; по истечении некоторого времени восстанавливается исходное значение. Временное повышение приоритета делается также и для долго не получавшего управление потока. Эти правила действуют только для потоков с уровнем приоритета не выше 15.

Для управления приоритетами выполнения процессов и потоков служат следующие функции:

GetPriorityClass() – получение текущего класса приоритета для процесса;
SetPriorityClass() – установка класса приоритета для процесса;
GetThreadPriority() – получение текущего приоритета выполнения потока;
SetThreadPriority() – установка приоритета выполнения потока.

В сбалансированной системе высокоприоритетные потоки выполняют, как правило, короткие операции, связанные с реагированием на события. Кроме того, само событийное управление процессами предполагает частое переключение в состояние ожидания. Это дает шанс на исполнения потокам с низким уровнем приоритета.

Задание к работе

Написать программу, на главном окне которой будет показан ListBox, в который должны быть занесены все доступные из выполняющихся в данный момент процессов и их приоритеты. При выборе какого-нибудь из них во втором ListBox-е должны быть показаны его модули. Также должно быть реализовано всплывающее при нажатии правой кнопки мышки меню, в которое должны быть занесены основные классы приоритетов. При выборе какого-либо пункта данного меню приоритет выбранного в первом ListBox процесса должен измениться на заданный.

Контрольные вопросы

1. Понятия приоритета процесса и потока. Роль приоритетов в планировании выполнения задач.
2. Классы приоритетов. Краткая характеристика основных классов приоритетов.
3. Функция для получения всех выполняющихся в данный момент процессов. Ее параметры и использование.
4. Функция для получения всех модулей заданного процесса. Ее параметры и принцип использования.
5. Основные функции для управления приоритетами.
6. Создание всплывающих меню.

Практическое занятие №18

Реестр Windows

Цель и содержание: изучение системного реестра Windows, а также API для работы с реестром

Теоретическая часть

Реестр (Registry) – специальная системная база данных, в которой приложения и операционная система могут сохранять информацию о конфигурации.

Системный реестр служит для осуществления ряда функций:

- хранение конфигураций оборудования и сведений об устройствах «Plug-and-Play»;
- хранение списка драйверов и их параметров;
- описания программных интерфейсов (например, интерфейсов COM- серверов);
- таблица ассоциаций файлов данных;
- хранение конфигурации и значений параметров программ;
- обслуживание различных административных программ, например, панели управления (Control Panel).

Реестр имеет иерархическую древовидную структуру. Узлы дерева называются *ключами (key)*. Каждый ключ может содержать любое количество *подключей (sub-key)* и значений (*values*), причем и те, и другие организованы в виде неупорядоченных списков, элементам которых присвоены индексы, начиная с нуля. Различие между ключами и подключами в действительности условно, фактически все ключи являются подключами различного уровня нескольких предопределенных ключей. Значения ключей могут быть строковыми, двоичными и числовыми; также ключ может быть ссылкой на другой ключ. Каждый ключ идентифицируется его именем, уникальным относительно вышестоящего ключа; открытым ключам присваиваются описатели (handle) HKEY. Значения идентифицируются именами и индексами в списке.

Данные в системном реестре хранятся в двоичном виде. Для работы с ними приложения должны использовать специальные системные функции. Различают *hive* (букв. «улей») ключей, то есть их двоичный образ в структурах в памяти, называемых собственно реестром, и файлы данных реестра.

Для работы с ключом приложение должно открыть его. При открытии ключа необходимо указать открытый ранее ключ в качестве вышестоящего. Система всегда предоставляет четыре предопределенных (*predefined*) ключа верхнего уровня, которые считаются открытыми всегда и могут использоваться как точки входа в реестр: HKEY_LOCAL_MACHINE – описание известных на этот момент конфигураций компьютера; HKEY_CLASSES_ROOT – описание текущей конфигурации машины, ссылка на одну из конфигураций HKEY_LOCAL_MACHINE); HKEY_USERS –

описание всех имеющихся пользователей; HKEY_CURRENT_USER – описание текущего пользователя, ссылка на одного из пользователей HKEY_USERS; HKEY_CURRENT_CONFIG – текущий подключ Config (ссылка) ключа HKEY_LOCAL_MACHINE;

Система предоставляет ряд функций для доступа к реестру. Перечислим некоторые из них:

- RegCreateKey(), RegCreateKeyEx() – создание нового или открытие существующего ключа в реестре; создаваемый или открываемый ключ обязательно должен быть подключом уже открытого ключа;
- RegOpenKey(), RegOpenKeyEx() – только открытие существующего ключа, в остальном аналогичны предыдущим;
- RegCloseKey() – закрытие открытого ключа;

- RegDeleteKey() – удаление ключа;
- RegFlushKey() – выгрузка содержимого ключа в соответствующий файл реестра;
- RegSaveKey(), RegLoadKey() – выгрузка содержимого ключа в отдельный файл и загрузка из него;
- RegEnumKey(), RegEnumKeyEx() – получение списка подключей;
- RegEnumValue() – получение списка значений;
- RegQueryValue(), RegQueryValueEx() – чтение значения по его имени;
- RegQueryMultipleValues() – чтение нескольких значений.

Ввиду относительной сложности и громоздкости обращений к реестру можно реализовать собственные «оберточные» функции для доступа по именам и «путям» реестре, аналогично работе с файловой системой.

На пользовательском уровне для работы с реестром служит утилита regedit, поддерживающая как интерактивный, так и командный (выполнение пакетных файлов) режимы.

Варианты заданий

1. Поиск в реестре значений, в том числе по шаблону (для текстовых), и вывод в удобной форме результатов поиска.
2. Поиск в реестре неиспользуемых и/или недействительных ключей и значений, например ссылающихся на несуществующие файлы и пути.
3. Получение и модификация отдельных элементов реестра (например, чтение текущей конфигурации, изменение отдельных настроек, чистка «историй» программ и т.п.).

Контрольные вопросы

1. Назначение реестра
2. Структура реестра
3. Поддерживаемые типы данных в реестре
4. API для работы с реестром
5. Порядок и правила работы с реестром

Практическое занятие №19

Прикладное программное обеспечение, реализующее типовые процедуры обработки экономической информации

Цель и содержание: изучение прикладного ПО, реализующего типовые процедуры обработки экономической информации.

Теоретическая часть

Прикладное ПО по характеру применения делится на прикладные программные средства коммерческого использования (программные продукты) и индивидуально разрабатываемые программные средства (не распространяемые программы).

Прикладные программные *средства коммерческого использования* представляют собой программное обеспечение, изготовленное на продажу. Программные продукты имеют программную документацию, обеспечивающую установку и эксплуатацию программ сторонними пользователями.

Прикладные программные средства коммерческого использования позволяют решить большинство проблем, стоящих перед пользователем при реализации типовой обработки экономической информации.

Индивидуально разрабатываемые программные средства обеспечивают комплексность и функциональную завершенность прикладного программного обеспечения. Эффективность разработки и использования прикладного программного обеспечения достигается оптимальным сочетанием данных составных частей.

Можно выделить следующие виды прикладных систем:

1. Во-первых, системы управления базами данных (СУБД), которые позволяют управлять большими информационными массивами. Одной из наиболее известных систем является Visual FoxPro. Данная СУБД позволяет с помощью средств программирования обеспечить для определенного пользователя комплексную обработку экономической информации. Для создания многопользовательских информационных систем, как правило, используется СУБД типа клиент-сервер. Сервер принимает от программ, выполняемых на других компьютерах (клиентов), запросы на получение информации из базы данных или осуществление манипуляций с данными. Запросы на обработку информации, как правило, делаются с помощью языка SQL (Structured Query Language). Наиболее известные многопользовательские системы - Oracle, Microsoft SQL.

2. Во-вторых, табличные процессоры, которые обеспечивают работу с большими таблицами. При работе с табличным процессором на экран выводится прямоугольная таблица, в клетках которой могут находиться числа, пояснительные тексты и формулы для расчета значений. Табличные процессоры позволяют использовать графики, включать в таблицу рисунки, применять макросы. Наибольшей популярностью пользуются табличные процессоры Microsoft Excel, Quatro Pro.

3. В-третьих, текстовые редакторы, которые обеспечивают подготовку служебной документации. Редакторы документов позволяют эффективно набирать и редактировать текст, использовать различные шрифты, создавать таблицы и диаграммы, проверять правописание. Одним из наиболее известных текстовых редакторов является Microsoft Word.

4. В-четвертых, бухгалтерские программы, которые предназначены для ведения бухгалтерского учета. Сложность и функции бухгалтерских программ прежде всего зависят от размеров и функциональной направленности предприятия, для которого они предназначены. Для ведения бухгалтерского учета на небольших и средних предприятиях могут быть использованы следующие программы: «1С:Бухгалтерия», «Парус».

5. В-пятых, статистические программы, которые обеспечивают анализ данных с использованием инструмента математической статистики. Статистические программы один из видов прикладных программных продуктов, используемых в аналитических и

плановых отделах банков, предприятий, страховых фирм и др.. В качестве примера статистических программ можно назвать следующие программы: Statistica (производится фирмой StatSoft inc), Forecast PRO.

6. В-шестых, графические редакторы, которые позволяют создавать и редактировать рисунки. Графические редакторы используются для повышения наглядности в сфере обучения, составления деловой документации, создания рекламных объявлений и др. К популярным графическим редакторам можно отнести Corel Draw.

7. В-седьмых, системы, основанные на знаниях (экспертные системы). Данные системы позволяют получать экспертные оценки и рекомендации в различных ситуациях хозяйственной деятельности. Примером, экспертной системы служит система «XCON», которая обеспечивает составление конфигурации компьютерных систем.

Задание для выполнения

Подготовить развёрнутое сообщение о произвольном средстве обработки экономической информации, рассказать о его разработчике, схеме распространения, назначении, принципах установки и настройки.

Контрольные вопросы

1. Классификация ПО по характеру применения
2. Виды прикладных систем
3. Основные разработчики прикладных систем различных видов
4. Схемы распространения и лицензирования ПО для обработки экономической информации.

Практическое занятие № 20

Управление задачами в ОС Windows

1.1 Цель работы

Целью работы является изучение процесса управления заданиями в ОС Windows.

1.2 Информация об организации вычислительных задач

Современные операционные системы содержат встроенные средства, предоставляющие информацию о компонентах вычислительного процесса. Диспетчер задач (Task Manager) операционных систем Windows (например, Windows XP) позволяет получить обобщенную информацию об организации вычислительного процесса с детализацией до выполняющихся прикладных программ (приложений) и процессов. Однако диспетчер задач не позволяет отслеживать потоки.

Для запуска диспетчера задач и просмотра компонентов вычислительного процесса нужно выполнить следующие действия:

1. Щелкнуть правой кнопкой мыши по панели задач и выбрать строку «Диспетчер задач», или нажать клавиши Ctrl+Alt+Del, или нажать последовательно Пуск -> Выполнить -> taskmgr (рис. 1.1).

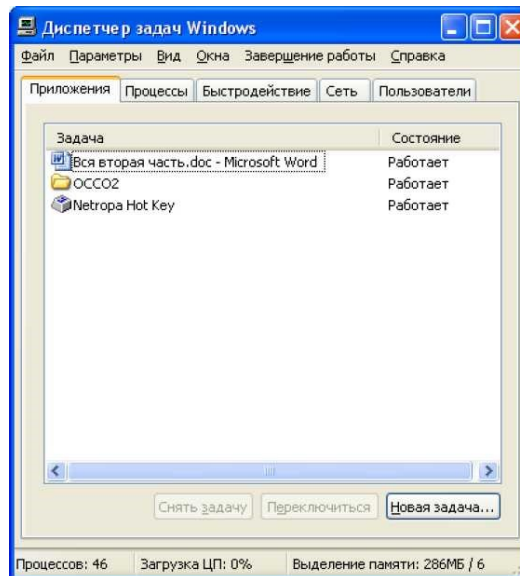


Рис. 1.1 - Окно диспетчера задач в ОС Windows XP

2. Для просмотра приложений перейти на вкладку «Приложения». Здесь можно завершить приложение (кнопка «Снять задачу»), переключиться на другое приложение (кнопка «Переключиться») и создать новую задачу (кнопка «Новая задача»). В последнем случае после нажатия кнопки «Новая задача» в появившемся окне (рис. 1.2) нужно ввести имя задачи.

- 3.

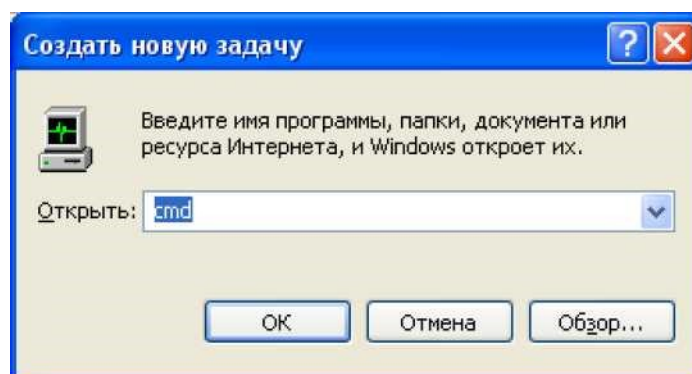


Рис. 1.2 - Окно создания новой задачи в ОС Windows XP

4. Просмотр (мониторинг) процессов осуществляется переходом на вкладку «Процессы». Таблица процессов включает в себя все процессы, запущенные в собственном адресном пространстве, в том числе все приложения и системные сервисы. Обратите внимание на процесс «Бездействие системы» – фиктивный процесс, занимающий процессор при простое системы.
5. Если требуется просмотреть 16-разрядные процессы, то в меню «Параметры» необходимо выбрать команду «Отображать 16-разрядные задачи».
6. Для выбора просматриваемых показателей (характеристик) с помощью команды «Выбор столбцов» (меню «Вид») необходимо установить флажки рядом с показателями, которые требуется отображать (рис. 1.3).

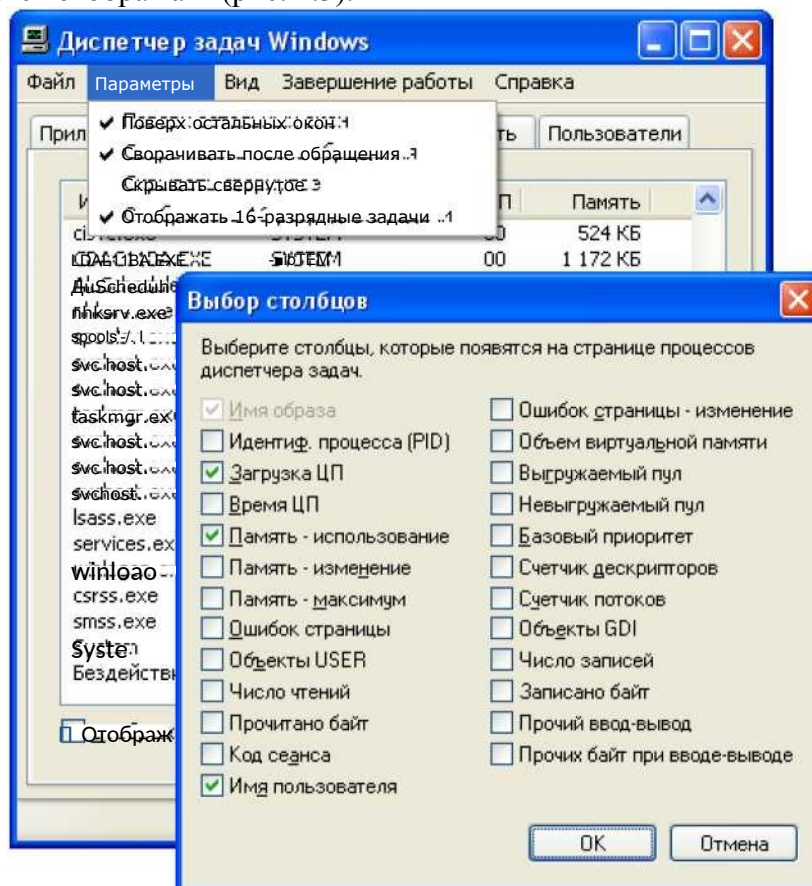


Рис. 1.3 - Окно диспетчера задач в ОС Windows XP на вкладке процессы с окном настройки отображения столбцов

В качестве примера можно рассмотреть процессы приложения MS Word. Для этого нужно выполнить следующие действия:

1. Запустить MS Word. Щелкнуть правой клавишей мыши по названию приложения и в появившемся контекстном меню выбрать строку «Перейти к процессам». Произойдет переход на вкладку «Процессы». Можно просмотреть число потоков и другие характеристики процесса.
2. Изменить приоритет процесса. На вкладке «Процессы» необходимо щелкнуть правой клавишей мыши по названию процесса и выбрать в контекстном меню строку «Приоритет». Изменив приоритет, можно увидеть в колонке «Базовый приоритет» его новое значение.
3. Изменить скорости обновления данных. Войти в меню «Вид» и выбрать команду «Скорость обновления». Установить требуемую скорость (высокая – каждые полсекунды, обычная – каждую секунду, низкая – каждые 4 секунды, приостановить

– обновления нет). Следует иметь в виду, что с повышением скорости мониторинга возрастают затраты ресурсов компьютера на работу операционной системы, что в свою очередь вносит погрешность в результаты мониторинга.

Диспетчер задач позволяет получить обобщенную информацию об использовании основных ресурсов компьютера. Для этого необходимо сделать следующее:

1. Перейти на вкладку «Быстродействие» (рис. 1.4). Верхние два окна показывают интегральную загрузку процессора и хронологию загрузки. Нижние два окна – те же показатели, но по использованию памяти.
2. Для просмотра использования процессора в режиме ядра (красный цвет) войти в меню «Вид» и щелкнуть на строке Вывод времени ядра.

В нижней части окна вкладки «Быстродействие» отображается информация о количестве процессов и потоков, участвующих в мультипрограммном вычислительном процессе, об общем количестве дескрипторов (описателей) объектов, созданных операционной системой, а также информация о доступной и выделенной памяти для реализации приложений. Кроме того, приводятся сведения о выделении памяти под ядро операционной системы с указанием выгружаемой и невыгружаемой памяти ядра и объеме системного кэша.

Также в диспетчере задач имеются вкладки для отображения состояния сети (вкладка «Сеть») и информации о вошедших в систему пользователях (вкладка «Пользователи»). Ряд программ, как производителей операционных систем, так и сторонних производителей могут предоставить более детальную информацию о компонентах вычислительного процесса и механизмы управления им: Process Explorer, Process Viewer, Microsoft Spy++, CPU Stress, Scheduling Lab, Job Lab и др.

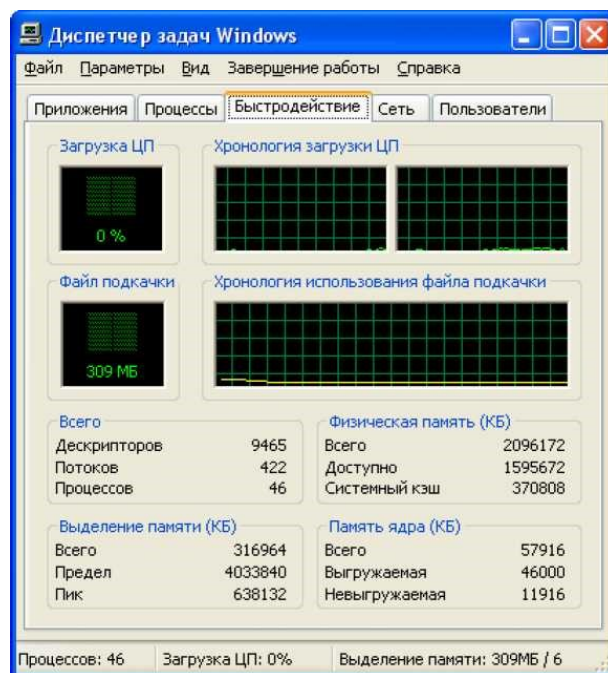


Рис. 1.4 - Окно диспетчера задач в ОС Windows XP на вкладке быстродействие

На рис. 1.5 показан окно с получением информации о потоках в программе Process Explorer. В данной программе можно получить исчерпывающую информацию о количестве и состоянии задач в операционной системе Windows.

Любой поток состоит из двух компонентов:

- объекта ядра, через который операционная система управляет потоком. Там же хранится статистическая информация о потоке;

– стека потока, который содержит параметры всех функций и локальные переменные, необходимые потоку для выполнения кода.

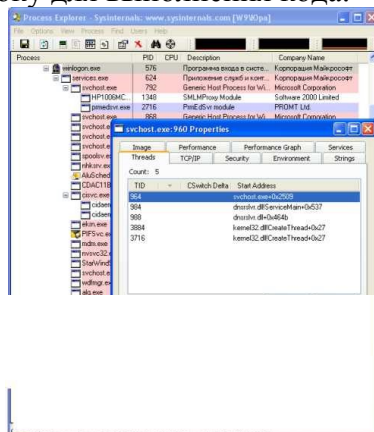


Рис. 1.5 - Окно с информацией о потоках в программе Process Explorer

Создав объект ядра «поток», система присваивает счетчику числа его пользователей начальное значение, равное двум. Затем система выделяет стеку потока память из адресного

пространства процесса (по умолчанию резервирует 1 Мбайт адресного пространства процесса и передает ему всего две страницы памяти, далее память может добавляться). После этого система записывает в верхнюю часть стека два значения (стеки строятся от старших адресов памяти к младшим). Первое из них является значением параметра **pvParam**, который позволяет передать функции потока какое-либо инициализирующее значение. Второе значение определяет адрес функции потока **pfnStartAddr**, с которой должен будет начать работу создаваемый поток.

У каждого потока собственный набор регистров процессора, называемый контекстом потока. Контекст отображает состояние регистров процессора на момент последнего исполнения потока и записывается в структуру **CONTEXT**, которая содержится в объекте ядра «ПОТОК».

7

Указатель команд (IP) и указатель стека (SP) – два самых важных регистра в контексте потока. Когда система инициализирует объект ядра «ПОТОК», указателю стека в структуре **CONTEXT** присваивается тот адрес, по которому в стек потока было записано значение **pfnStartAddr**, а указателю команд – адрес недокументированной функции **BaseTbreadStart** (находится в модуле **Kernel32.dll**).

Новый поток начинает выполнение этой функции, в результате чего система обращается к функции потока, передавая ей параметр **pvParam**. Когда функция потока возвращает управление, **BaseTbreadStart** вызывает **ExitTbread**, передавая ей значение, возвращенное функцией потока. Счетчик числа пользователей объекта ядра «ПОТОК» уменьшается на 1, и выполнение потока прекращается.

При инициализации первичного потока его указатель команд устанавливается на другую недокументированную функцию – **BaseProcessStart**. Она почти идентична **BaseTbreadStart**. Единственное различие между этими функциями в отсутствии ссылки на параметр **pvParam**. Функция **BaseProcessStart** обращается к стартовому коду библиотеки C/C++/C#, который выполняет необходимую инициализацию, а затем вызывает входную функцию **main**, **wmain**, **WinMain**, **Main**. Когда входная функция возвращает управление, стартовый код библиотеки C/C++/C# вызывает **ExitProcess**.

1.3 Исследование производительности

В операционных системах Windows имеются средства, позволяющие детально анализировать вычислительные процессы. К таким средствам относится «Системный

монитор» и «Оповещения и журналы производительности». Для доступа к этим средствам нужно выполнить последовательность действий: Пуск -> Панель управления -> Администрирование -> Производительность.

Откроется окно Производительность, содержащее две оснастки: «Системный монитор» и «Оповещения и журналы производительности» (рис. 1.6).

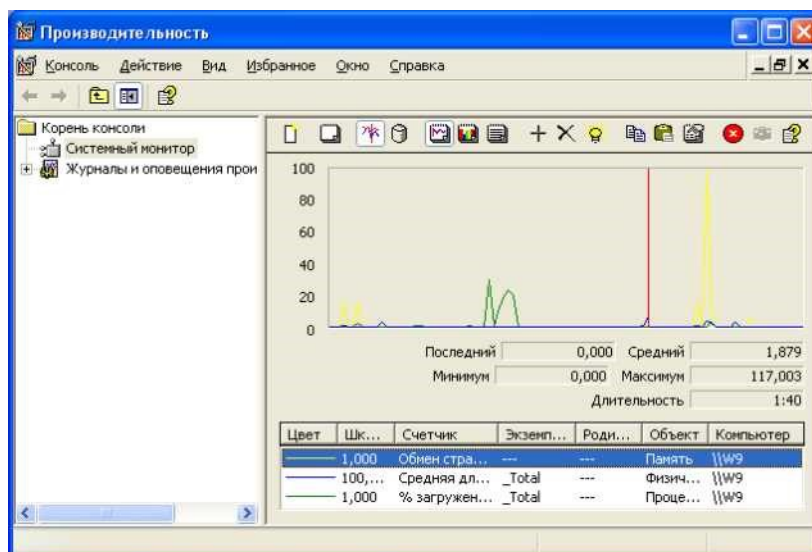


Рис. 1.6 - Окно Производительность в ОС Windows XP на вкладке быстродействие

Системный монитор позволяет анализировать вычислительный процесс, используя различные счетчики. Объектами исследования являются практически все компоненты компьютера: процессор, кэш, задание, процесс, поток, физический диск, файл подкачки, очереди сервера, протоколы и др.

Для просмотра и выбора объектов мониторинга и настройки счетчиков нужно выполнить следующие действия:

1. Открыть оснастку «Производительность». По панели результатов (правая панель) щелкнуть правой клавишей мыши и выбрать в контекстном меню строку «Добавить счетчики» или щелкнуть по кнопке «Добавить» (значок +) на панели инструментов.
2. В появившемся окне «Добавить счетчики» (рис. 1.7) выбрать объект мониторинга, например процессор, а затем выбрать нужные счетчики из списка «Выбрать счетчики из списка», например «% времени прерываний», нажимая кнопку Добавить, для потока можно определить:
 - число контекстных переключений в сек.;
 - состояние потока (для построения графа состояний и переходов); 8
 - текущий приоритет (для анализа его изменения);
 - базовый приоритет;
 - % работы в привилегированном режиме и др.

Нажав кнопку «Объяснение», можно получить информацию о счетчике. При выборе нескольких однотипных объектов, например потоков, нужно их указать в правом поле «Выбрать вхождения из списка».

Для удобства работы предусмотрена настройка вида отображаемой информации.

Просмотр информации производительности возможен в виде графика, гистограммы и отчета. Для настройки внешнего вида окна нужно щелкнуть по графику правой кнопкой мыши и выбрать команду «Свойства».

На вкладке «Общие» можно задать вид информации (график, гистограмма, отчет), отображаемые элементы (легенда, строка значений, панель инструментов), данные отчета и гистограммы (максимальные, минимальные и т.д), период обновления данных и др.

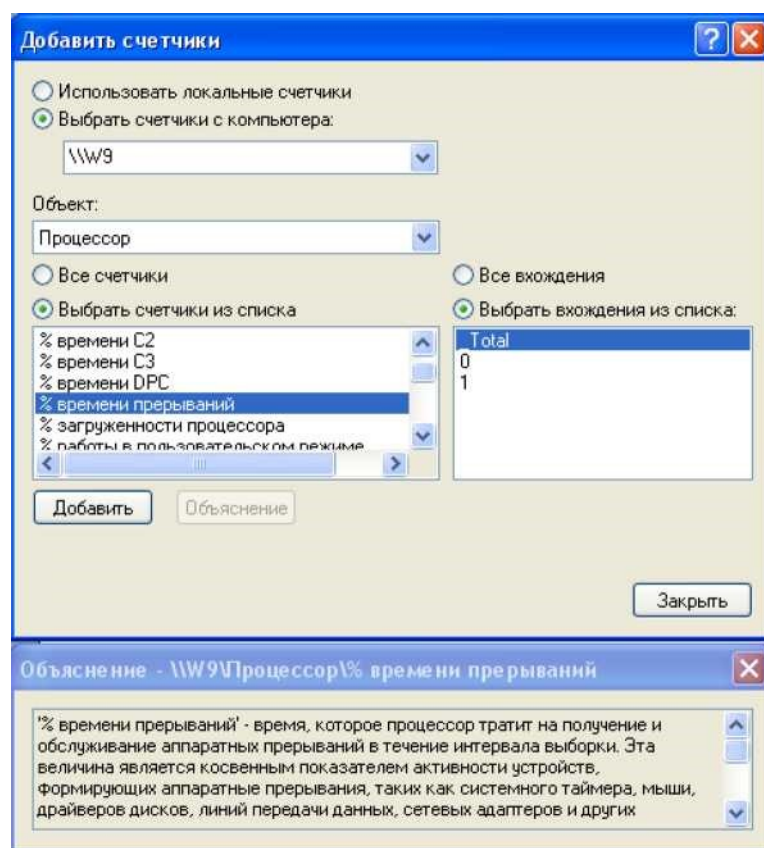


Рис. 1.7 - Окно Добавить счетчики в программе оценки производительности в ОС Windows XP

На вкладке «Источник» задается источник данных. На вкладке «Данные» можно для каждого счетчика задать цвет, ширину линии, масштаб и др.

На вкладке «График» можно задать заголовок, вертикальную и горизонтальную сетку, диапазон значений вертикальной шкалы. На вкладках «Цвета и шрифты» можно изменить набор цветов и шрифт.

Режимы «График» и «Гистограмма» не всегда удобны для отображения результатов анализа, например, при большом количестве счетчиков, меняющих свое значение в разных диапазонах величин. Режим «Отчет» позволяет наблюдать реальные значения счетчиков, так как не использует масштабирующих множителей. В этом режиме доступна только одна опция – изменение интервала опроса.

Полученная с помощью «Монитора производительности» информация позволяет наглядно произвести экспресс-анализ функционирования нужного компонента вычислительного процесса или устройства компьютера.

Оснастка «Оповещения и журналы производительности» содержит три компонента:

Журналы счетчиков, Журналы трассировки и Оповещения, которые можно использовать для записи и просмотра результатов исследования вычислительного процесса. Данные, созданные при помощи оснастки, можно просматривать как в процессе сбора, так и после его окончания.

Файл журнала счетчиков состоит из данных для каждого указанного счетчика на указанном временном интервале. Для создания журнала необходимо выполнить следующие действия:

1. запустить оснастку «Производительность»;
2. дважды щелкнуть по значку «Оповещения и журналы производительности»;
3. выбрать значок «Журналы счетчиков», щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры журнала»;

4. в открывшемся окне ввести произвольное имя журнала и нажать кнопку «ОК»;
5. в новом окне на вкладке «Общие» добавить нужные счетчики и установить интервал съема данных;
6. на вкладке «Файлы» журналов можно выбрать размещение журнала, имя файла, добавить комментарий, указать тип журнала и ограничить его объем. Возможны следующие варианты: - текстовый файл - CVS (данные сохраняются с использованием запятой в качестве разделителя);
 - текстовый файл - TSV (данные сохраняются с использованием табуляции в качестве разделителя);
 - двоичный файл для регистрации прерывающейся информации;
 - двоичный циклический файл для регистрации данных с перезаписью;
7. на вкладке «Расписание» выбрать режим запуска и остановки журнала (вручную или по времени). Для запуска команды после закрытия журнала установить флажок «Выполнить команду» и указать путь к исполняемому файлу;
8. после установки всех значений нажать кнопки «Применить» и «ОК».

В отличие от журналов счетчиков, журналы трассировки находятся в ожидании определенных событий. Для интерпретации содержимого журнала трассировки необходимо использовать специальный анализатор.

Для создания журнала трассировки необходимо выполнить следующие действия:

1. запустить оснастку «Производительность»;
2. щелкнуть по значку «Журналы трассировки»;
3. щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры журнала»;
4. в открывшемся окне ввести произвольное имя журнала и нажать кнопку «ОК»;
5. по умолчанию файл журнала создается в папке PerfLogs в корневом каталоге и к имени журнала присоединяется серийный номер;
6. на вкладке «Общие» указать путь и имя созданного журнала (по умолчанию оно уже есть);
7. на этой же вкладке выбрать «События», протоколируемые системным поставщиком или указать другого поставщика;
8. на вкладке «Файлы журналов» выбрать тип журнала:
 - файл циклической трассировки (журнал с перезаписью событий, расширение etl);
 - файл последовательной трассировки (данные записываются, пока журнал не достигнет предельного размера, расширение etl);
9. на этой же вкладке выбрать и размер файла;
10. на вкладке «Дополнительно» можно указать размер буфера журнала;
11. на вкладке «Расписание» выбрать режим запуска и остановки журнала (вручную или по времени).

В ряде случаев для обнаружения неполадок в организации вычислительного процесса удобно использовать оповещения. С помощью этого компонента можно установить оповещения для выбранных счетчиков. При превышении или снижении относительно заданного значения выбранными счетчиками оснастка посредством сервиса «Messenger» оповещает пользователя.

Для создания оповещений необходимо выполнить следующие действия:

1. щелкнуть по значку «Оповещения»;
2. щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры оповещений»;
3. в открывшемся окне ввести произвольное имя оповещения и нажать кнопку «ОК»;

4. в появившемся окне на вкладке «Общие» можно задать комментарий к оповещению и выбрать нужные счетчики; 1
5. в поле «Оповещать» выбрать предельные значения для счетчиков;
6. в поле «Снимать показания» выбрать период опроса счетчиков;
7. на вкладке «Действие» можно выбрать действие, которое будет происходить при запуске оповещения, например, послать сетевое сообщение и указать имя компьютера;
8. на вкладке «Расписание» выбрать режим запуска и остановки наблюдения.

Если в компьютере произойдет событие, предусмотренное в оповещениях, в журнал событий «Приложение» будет сделана соответствующая запись. Для ее просмотра нужно зайти в оснастку «Просмотр событий», где и можно увидеть сведения о событии.

1.4 Задание на выполнение

Часть 1.

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Исследовать мультипрограммный вычислительный процесс на примере выполнения самостоятельно разработанных трех задач (например, заданий по курсу программирования).
2. Для одной из задач определить PID, загрузку ЦП, время ЦП, базовый приоритет процесса, использование памяти. Изменить приоритет процесса и установить, влияет ли это на время выполнения приложения.
3. Монопольно выполнить каждую из трех задач, определить время их выполнения.
4. Запустить одновременно (друг за другом) три задачи, определить время выполнения пакета.

Ответьте на вопросы:

1. В каком случае суммарное время выполнения задач больше? При последовательном выполнении или одновременном выполнении?
2. Как изменилось время выполнения каждой отдельной задачи?
3. Как изменится время выполнения отдельной задачи при изменении ее приоритета?
4. Окажет ли влияние изменение приоритета одной задачи на время выполнения другой задачи? Объяснить результаты.

Часть 2.

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Запустить некоторое количество программ. Используя возможности оснастки Производительность, получить диаграммы, характеризующие использование процессора при его нагрузке различным количеством потоков, меняя их активность и уровни приоритета.
2. Исследовать свои задачи (например, задания по курсу программирования). Определить характеристики процессов: % загрузки процессора (в пользовательском и привилегированном режиме), % времени прерываний, количество прерываний, базовый приоритет, обращения к диску, время выполнения процесса.
3. Исследовать свои приложения с записью результатов в Журнал счетчиков, выбрав следующие счетчики: % загруженности, работы процессора в привилегированном и пользовательском режимах, % времени прерываний, % использования выделенной памяти, частота обращений к диску, скорость обмена с диском.
4. Выполнить следующие действия:
 - Запустить журнал (частота съема данных 10 сек., файл типа CVS).
 - Запустить исследуемую программу. - Через 2 - 3 мин. остановить журнал.

- Просмотреть Результаты, открыв файл журнала в Excel. Объяснить полученные результаты. - Исследовать программу еще раз, указав тип журнала – двоичный (чтобы потом можно было просмотреть диаграммы).

5. Создать журнал трассировки для исследования своего приложения. Создать Оповещения по выбранным счетчикам для своего приложения. Просмотреть журнал событий. Объяснить полученные результаты.

Ответьте на вопросы:

1. Что можно просматривать, используя счетчики в системном мониторе?
2. В каких видах можно просматривать информацию о производительности? 1

Часть 3. Дополнительное задание

1. Найти в Интернет бесплатную программу Process Explorer для Microsoft Windows.
2. Установить ее на компьютер.
3. Произвести исследование ее работы.

Ответьте на вопрос:

1. Сколько потоков содержит запущенная на вашем компьютере программа Microsoft Word?

Практическое занятие № 21

Исследование блоков управления памятью

2.1 Цель работы

Изучение структуры системных таблиц реального режима Windows и организации цепочек блоков памяти.

2.2 Организация хранения байтов в памяти

При просмотре памяти имейте в виду, что двухбайтовые слова хранятся в виде {младший байт} {старший байт} - т.е. порядке обратном естественному представлению многобайтного числа.

То же самое относится к порядку расположения слов в двойном слове - сначала младшее слово, потом старшее. Всегда действует общий принцип - младшее лежит в ячейке памяти с младшим адресом. Таким образом, полный 4-х байтный указатель (например, на таблицу таблиц) 1234:5678H будет в дампе памяти выглядеть как:

78 56 34 12

\\ \\

| | старшее слово с переставленными байтами

|

| | младшее слово с переставленными байтами

2.3 Информация о структурах памяти

Это список указателей, каждый из которых представляет собой двойное слово (4 байта). Старшее слово - это сегментный адрес, младшее - смещение в сегменте. Например, для указателя, у которого сегментный адрес=1234H, а смещение 5678H, абсолютный физический адрес ячейки памяти образуется, как сумма сегментного адреса * 16 + смещение (т.е. сегментный адрес сдвинут влево на 1 шестнадцатеричный разряд).

Таким образом 0110:0026 -это тоже, что и 0112:0006.

2.4 Структура таблицы таблиц

Смещение	Длина	Содержимое
-2	2	сегментный адрес 1 MCB
0	4	указатель на 1 DPB (Disk Parameters Block)
+ 4	4	указатель на список таблиц открытых файлов
+ 8	4	указатель на первый драйвер DOS (CLOCK\$)

Данная структура является НЕДОКУМЕНТИРОВАННОЙ и используется для изучения низкоуровневой информации о структурах памяти.

2.5 Структура блока управления памятью (MCB)

Смещение	Длина	Содержимое
+0	1	'M' (4dH) - за этим блоком есть еще блоки 'Z' (5aH) - данный блок является последним

MCB – это НЕДОКУМЕНТИРОВАННЫЙ управляющий блок, который используется при распределении, модификации и освобождении блоков системной памяти.

+1	2	Владелец, параграф владельца (для FreeMem); 0 = владеет собой
+3	2	Размер, число параграфов в этом блоке распределения. Параграф равен 16 байтам
+5	0Bh	Зарезервировано
+10h	?	Блок памяти начинается здесь и имеет длину (Размер*10H) байт

Замечания:

- блоки памяти всегда выровнены на границу параграфа («сегмент блока»); - блоки М-типа: следующий блок находится по (сегмент блока + Размер):0000; - блоки Z-типа: (сегмент блока + Размер):0000 = конец памяти (a000H=640K).

В любом MCB указан его владелец - сегментный адрес PSP (префикс программного сегмента) программы владельца данного блока памяти. А в PSP есть ссылка на окружение данной программы, в котором можно найти имя программы - DOS и в языке C, представляет собой последовательность символов ASCII, заканчивающуюся байтом 00H), пока не дойдете до пустой строки (нулевой длины), что указывает конец окружения. Обычно 'имя' в каждой строке окружения задано прописными буквами, но это необязательно.



Следует помнить, что сама программа (и PSP в том числе) окружении ее сами располагаются в блоках памяти. Поэтому, в MCB блока памяти программы самой в хозяина собственный качестве самого себя.

Использование окружения. Окружение не превышает 32 Кбайт и начинается на границе параграфа. Смещение 2Ch в PSP текущей программы содержит номер параграфа окружения.

Вы можете найти нужное 'имя' серией сравнений строк ASCIIZ (Строка ASCIIZ, используемая во многих функциях

Более подробную информацию о структурах памяти можно получить из справочника TECH Help!

2.6 Задание на выполнение

Часть 1.

1. Подготовиться к работе, используя материалы лекций, данное пособие, справочник TEACNHELP.
2. Познакомиться с работой одной из программ, позволяющих просмотреть содержимое ОЗУ в виде шестнадцатеричного дампа - например, PEEK.COM (во время работы доступен HELP - F1, карта памяти - F8 и информация о блоке памяти - F6).
3. Найти в памяти таблицу таблиц (для получения ее адреса - запусти lol.com), познакомиться с ее содержимым и посмотреть указатель на 1 MCB (упр. блок памяти).
4. Проследить в памяти цепочку блоков, определяя их принадлежность и сравнивая с информацией из карты памяти (F8).

5. Написать отчет о найденной цепочке блоков памяти с их адресами и размерами.

Часть 2. Дополнительное задание

1. Найти в Интернет бесплатную программу Process Explorer для Microsoft Windows.
2. Установить ее на компьютер.
3. Произвести исследование ее работы.

Ответьте на вопрос:

1. Сколько потоков содержит запущенная на вашем компьютере программа Microsoft Word?

Практическое занятие № 22

Диагностика IP-протокола

3.1 Цель работы

Целью работы является проверка работоспособности сетевого подключения в ОС Windows, через диагностику IP- протокола.

3.2 Просмотр свойств сетевого окружения

Получить информацию о свойствах сетевого окружения возможно с использованием следующих действий: Нажмите кнопку «Пуск» и в появившемся окне щелкните правой кнопкой мыши по пункту «Сетевое окружение». В появившемся контекстном меню выберите пункт «Свойства». Перед вами появится окно, показанное на рис. 3.1.

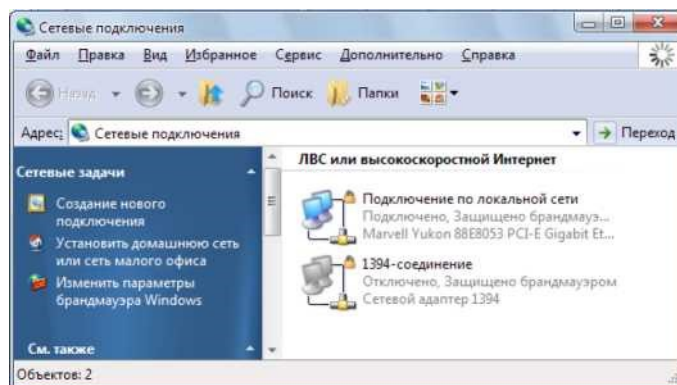


Рис. 3.1. Свойства сетевого окружения

Чтобы получить информацию о свойствах подключения по локальной сети, щелкните по надписи «Подключение по локальной сети» правой кнопкой мыши и также в появившемся меню выберите свойства. В появившемся окне (рис. 3.2) вы можете настраивать протоколы сетевых взаимодействий.

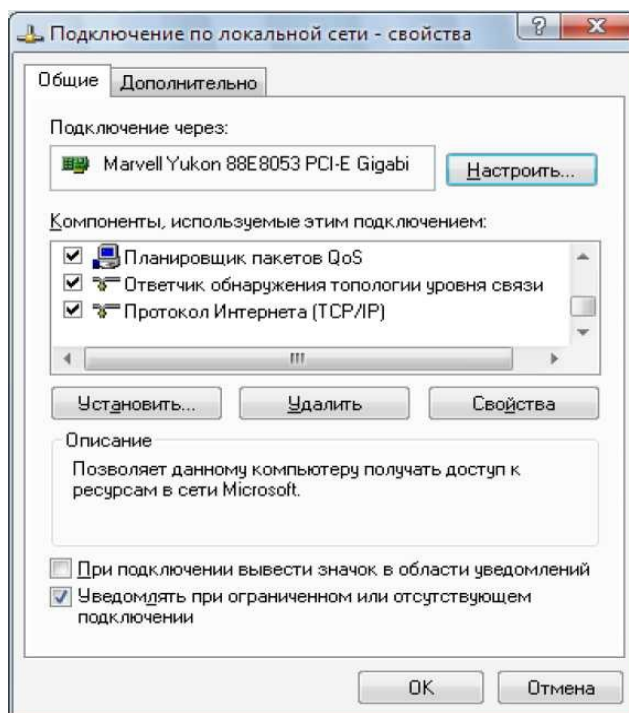


Рис. 3.2 - Свойства подключения по локальной сети

Важным элементом в свойствах подключения по локальной сети, является протокол Интернета TCP/IP. Выбрав это компонент и нажав кнопку «Свойства» откроется

окно (рис. 3.3) где можно устанавливать настройки сетевого подключения по протоколу TCP/IP.

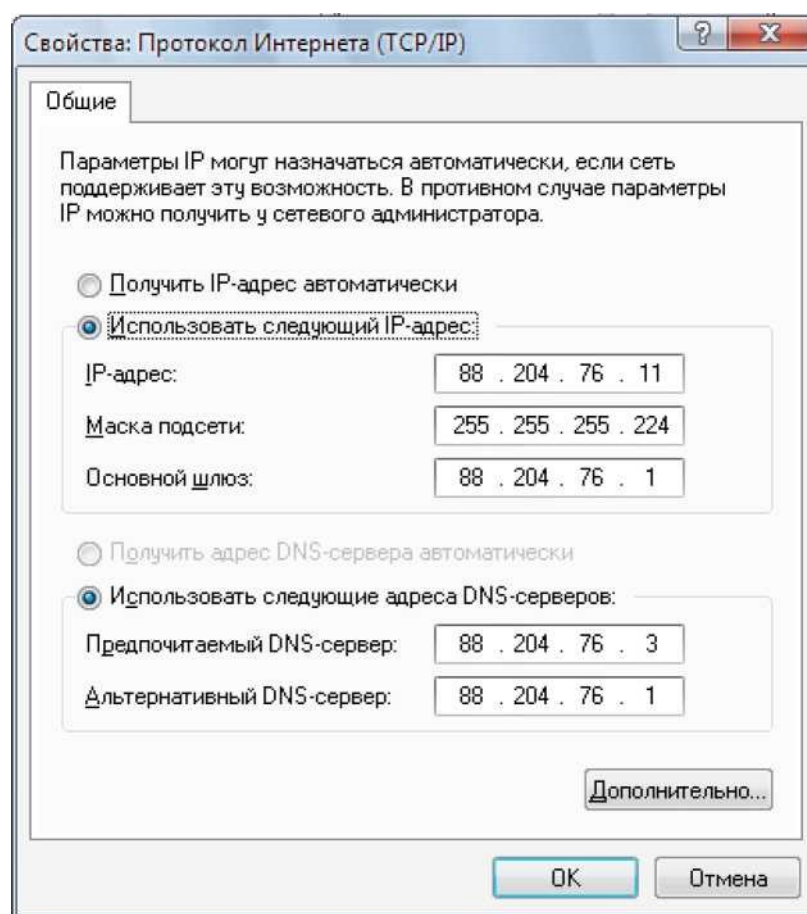


Рис. 3.3 Свойства протокола Интернета (TCP/IP)

3.3 Утилита диагностики сети

Существуют различные утилиты, позволяющие быстро протестировать IP-подключение. Однако большинство операций легко может быть выполнено с использованием команд самой операционной системы.

Пользователи Windows XP для диагностики сетевого подключения могут воспользоваться специальным мастером. Эта программа вызывается из меню задачи «Сведения о системе». Произведите следующие действия (Пуск > Все программы > Стандартные > Служебные > Сведения о системе > меню Сервис > Диагностика сети). На рисунке 3.4 показан процесс работы утилиты «Диагностики сети». На рис. 3.5. результат работы утилиты по диагностике сетевого подключения.

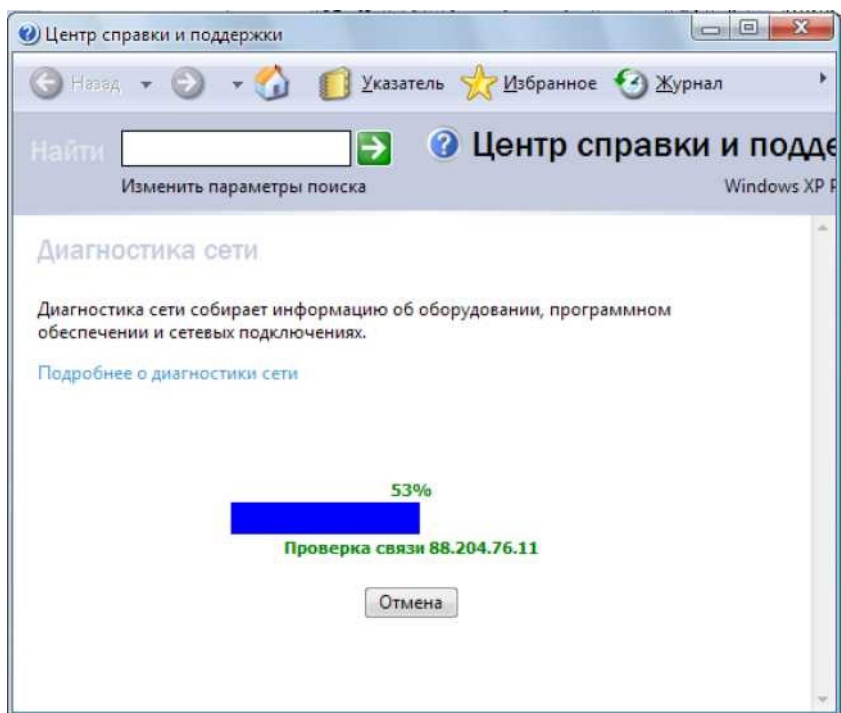


Рис. 3.4 Ход работы утилиты «Диагностика сети»

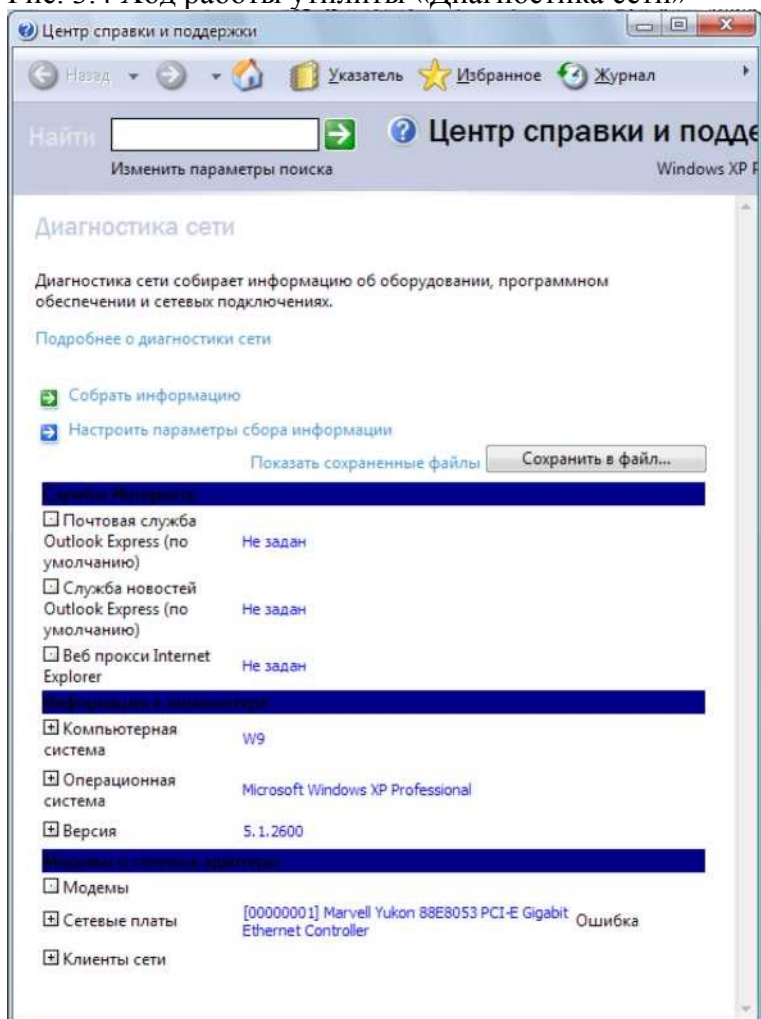


Рис. 3.5 Ход работы утилиты «Диагностика сети»

3.4 Утилита «Ipconfig»

Для отображения параметров IP-протокола в ОС на платформе Windows NT используются утилиты `ipconfig`. Эта утилита выводит на экран основные параметры настройки протокола TCP/IP: значения адреса, маски, шлюза.

1. Нажмите кнопку «Пуск», выберите строку меню «Выполнить», наберите символы `cmd` (запуск консоли командной строки) и нажмите клавишу `Enter` на клавиатуре.
2. Введите команду: `ipconfig /all`. При нормальной работе компьютера на экран должен выводиться примерно такой листинг:

```
Windows IP Configuration
Host Name ..... :
Primary Dns Suffix ..... : w9 aoi.tusur.ru
Node Type ..... : Hybrid
IP Routing Enabled ..... : No
WINS Proxy Enabled ..... : No aoi.tusur.ru
DNS Suffix Search List ..... : tomsk.ru
Ethernet adapter Local Area Connection:   aoi.tusur.ru
Connection-specific DNS Suffix . : Intel(R) PRO/100 S
Description ..... :
00-03-BA-8D-42-5B
Desktop Adapter
Yes
Physical Address ..... :
Yes 83.192.12.54
Dhcp Enabled ..... :
255.255.255.0
Autoconfiguration Enabled . . . . :
83.192.12.254
IP Address ..... :
83.192.12.2
Subnet Mask ..... :
192.168.0.1
Default Gateway ..... :
83.192.12.2
DHCP Server ..... :
83.192.12.2
DNS Servers ..... :
213.183.109.8
Primary WINS Server ..... : 27 августа 2012 г.
Secondary WINS Server ..... :
13 октября 2012 г.
Lease Obtained ..... : 19:20:22
Lease Expires ..... :
19:20:22
```

Отключите сетевое подключение, повторите команду. При отсутствующем соединении на экран выводится примерно такой листинг:

```
Windows IP Configuration
Host Name ..... : Primary Dns Suffix ..... :
Node Type ..... :
IP Routing Enabled ..... :
WINS Proxy Enabled ..... :
DNS Suffix Search List ..... :
Ethernet adapter Local Area Connection:
Media State ..... :
```

Description :
Desktop Adapter
Physical Address : w9 aoi.tusur.ru
Hybrid
No
No aoi.tusur.ru tusur.ru
Media disconnected Intel(R) PRO/100 S
00-03-BA-8D-42-5

Обратите внимание, что программа вывела на экран только данные о «физических» параметрах сетевой карты и указала, что отсутствует подключение сетевого кабеля (Media disconnected).

3.5. Утилита «Ping»

Утилита «Ping» используется для проверки протокола TCP/IP и достижимости удаленного компьютера. Она выводит на экран время, за которое пакеты данных достигают заданного в ее параметрах компьютера.

1. Проверка правильности установки протокола TCP/IP. Откройте командную строку и выполните команду: ping 127.0.0.1

Адрес 127.0.0.1 – это личный адрес любого компьютера. Таким образом, эта команда проверяет прохождение сигнала «на самого себя». Она может быть выполнена без наличия какого-либо сетевого подключения. Вы должны увидеть приблизительно следующие строки:

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Ping statistics for 127.0.0.1:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss) Approximate round trip times in milli-seconds:

Minimum = 0ms, Maximum = 0ms, Average = 0ms

По умолчанию команда посылает пакет 32 байта. Размер пакета может быть увеличен до 65 Кбайт. Так можно обнаружить ошибки при пересылке пакетов больших размеров. За размером тестового пакета отображается время отклика удаленной системы (в нашем случае – меньше 1 миллисекунды). Потом показывается еще один параметр протокола – значение TTL. TTL – «время жизни» пакета. На практике это число маршрутизаторов, через которые может пройти пакет. Каждый маршрутизатор уменьшает значение TTL на единицу. При достижении нулевого значения пакет уничтожается. Такой механизм введен для исключения случаев заикливания пакетов. Если будет показано сообщение о недостижимости адресата, то это означает ошибку установки протокола IP. В этом случае целесообразно удалить протокол из системы, перезагрузить компьютер и вновь установить поддержку протокола TCP/IP.

Проверка видимости локального компьютера и ближайшего компьютера сети. Выполните команду:

ping 192.168.0.19

На экран должны быть выведены примерно такие строки:

Pinging 212.73.124.100 with 32 bytes of data:
Reply from 192.168.0.19: bytes=32 time=5ms TTL=60
Reply from 192.168.0.19: bytes=32 time=5ms TTL=60
Reply from 192.168.0.19: bytes=32 time=4ms TTL=60
Reply from 192.168.0.19: bytes=32 time=4ms TTL=60
Ping statistics for 212.73.124.100:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

Minimum = 4ms, Maximum = 5ms, Average = 4ms

Наличие отклика свидетельствует о том, что канал связи установлен и работает.

3.6 Утилита «Tracert»

При работе в сети одни информационные серверы откликаются быстрее, другие медленнее, бывают случаи недостижимости желаемого хоста. Для выяснения причин подобных ситуаций можно использовать специальные утилиты.

Например, команда tracert, которая обычно используется для показа пути прохождения сигнала до желаемого хоста. Зачастую это позволяет выяснить причины плохой работоспособности канала. Точка, после которой время отклика резко увеличено, свидетельствует о наличии в этом месте узла, не справляющегося с нагрузкой.

В командной строке введите команду: tracert 192.168.0.19

Вы должны увидеть примерно такой листинг:

Tracing route to 192.168.0.19 over a maximum of 30 hops:

1 <1 ms <1 ms <1 ms 192.168.0.19

Trace complete.

3.7 Утилита «Route»

Команда Route позволяет просматривать маршруты прохождения сетевых пакетов при передаче информации.

Выведите на экран таблицу маршрутов TCP/IP, для этого в командной строке введите команду route print.

3.8 Утилита «Net view»

Выводит список доменов, компьютеров или общих ресурсов на данном компьютере. Вызванная без параметров, команда net view выводит список компьютеров в текущем домене.

1. net view и вы увидите список компьютеров своей рабочей группы.
2. net view \192.165.0.12 для просмотра общих ресурсов, расположенных на компьютере 192.165.0.12

3.9 Утилита «Net send»

Служит для отправки сообщений другому пользователю, компьютеру или псевдониму, доступному в сети.

1. Введите net send 192.168.0.1 Привет. Проверка связи.

Ваше сообщение получит пользователь 192.168.0.1 2. Введите net send * Привет. Проверка связи. Ваше сообщение получают все пользователи рабочей группы.

3.10 Задание на выполнение

1. Просмотрите через оконный интерфейс ОС Windows XP свойства протокола TCP/IP. Выпишите IP-адрес.

2. Осуществите диагностику сети.

3. Последовательно исследуйте все возможности сетевых утилит. **Ответьте на**

вопросы:

1. Какие сетевые протоколы установлены на вашем компьютере?
2. Чему равно «время жизни» пакета, посылаемого с вашего компьютера?
3. Сколько компьютеров в вашей рабочей группе?
4. Чему равна длина маршрута пакета, отправляемого вами на соседний компьютер?

Практическое занятие № 23

Управление устройствами ввода-вывода и файловыми системами в ОС Windows

4.1 Цель работы

Целью работы является изучение процесса управления устройствами ввода-вывода, файловыми системами.

4.2 Диспетчер устройств и драйвера устройств

Задача системы ввода-вывода ОС Windows заключается в предоставлении основных средств (каркаса) для эффективного управления широким спектром устройств ввода-вывода. Основу этих средств образует набор независимых от устройств процедур для определенных аспектов ввода-вывода и набор загруженных драйверов для общения с устройствами. Формирует этот каркас **менеджер ввода-вывода**, который предоставляет остальной операционной системе независимый от устройств ввод-вывод, вызывая для выполнения физического ввода-вывода соответствующий драйвер.

Файловые системы формально являются драйверами устройств, работающих под управлением менеджера ввода-вывода. В операционной системе Windows существует два драйвера для файловых систем FAT и NTFS, которые независимы друг от друга и управляют различными разделами диска или различными дисками.

Чтобы гарантировать, что драйверы устройств хорошо работают с остальной частью ОС, корпорация Microsoft определила для драйверов модель **Windows Driver Model**, которой должны соответствовать драйверы устройств. Разработчикам драйверов предоставляется набор инструментов, который должен помочь в создании драйверов, удовлетворяющих требованиям этой модели.

Существует набор утилит позволяющий контролировать работу программ управляющих аппаратными устройствами. Так утилита **Drivers из набора средств Microsoft Windows Resource Kit** позволяет получить детальную информацию о загруженных драйверах в текстовом формате.

Корпорацией Microsoft разработана утилита **Bootvis**, позволяющая выявлять проблемы, возникающие в процессе загрузки операционной системы. Эта утилита выполняет трассировку всех этапов загрузки системы, в том числе этапов загрузки системного ядра, драйверов устройств и запуска процессов. Утилита не входит в стандартную поставку Windows, но ее можно загрузить из Интернета (<http://download.microsoft.com:80/download/whistler/BTV/1.0/WXP/EN-US/BootVis-Tool.exe>).

В самой операционной системе Windows имеется программа «**Диспетчер устройств**», которую используют для обновления драйверов (или программного обеспечения) оборудования, изменения настроек оборудования, а также для устранения неполадок. Драйверы устройств для аппаратных продуктов с эмблемой «Для Microsoft Windows XP» или какой-либо другой более поздней версии снабжаются цифровой подписью корпорации Microsoft, которая подтверждает, что данный продукт проверен на совместимость с Windows и не изменился после проведения проверки. В окне диспетчера устройств представлено графическое отображение оборудования, установленного на компьютер. Для открытия окна диспетчера устройств нужно щелкнуть правой клавишей мыши по значку «Мой компьютер» и выбрать в контекстном меню строку «Свойства». В открывшемся окне «Свойства системы» следует перейти на вкладку «Оборудование» и нажать кнопку «Диспетчер устройств».

В окне диспетчера устройств (рис. 4.1) можно, раскрывая соответствующие узлы, видеть устройства, которые либо подключены и работают, либо отключены. Диспетчер устройств обычно используется для проверки состояния оборудования, подключения-отключения оборудования и обновления драйверов устройств, установленных на компьютере. Кроме того, возможности диагностики диспетчера устройств могут использоваться опытными пользователями, обладающими глубокими знаниями о

компьютерном оборудовании, для разрешения конфликтов устройств и изменения параметров ресурсов, однако при этом следует соблюдать большую осторожность.

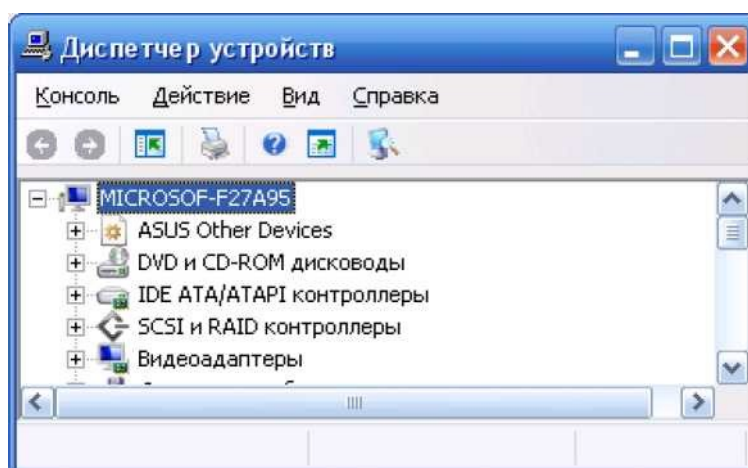


Рис. 4.1 - Окно программы «Диспетчер устройств»

При установке устройства **Plug and Play** Windows автоматически настраивает его, обеспечивая его правильную работу с другими установленными на компьютере устройствами. В ходе процесса настройки Windows назначает устанавливаемому устройству уникальный набор системных ресурсов. Эти ресурсы могут включать в себя один или несколько из следующих параметров: - номера строк запросов на прерывание (IRQ);

- каналы прямого доступа к памяти (DMA);
- адреса портов ввода/вывода (I/O);
- диапазоны адресов памяти.

При установке устройств не Plug and Play автоматическая настройка ресурсов не производится. Некоторые типы устройств требуется настраивать вручную. Необходимые инструкции содержатся в руководстве, поставляемом вместе с устройством. Изменять параметры ресурсов вручную обычно не рекомендуется, поскольку при этом значения фиксируются, что снижает возможности Windows по выделению ресурсов ДЛЯ других устройств. Если зафиксировано слишком много значений параметров для отдельных ресурсов, Windows не сможет автоматически устанавливать новые устройства Plug and Play.

Используя, Диспетчер устройств, можно отключать подсоединенные к компьютеру устройства и удалять их из конфигурации компьютера. Хотя для удаления устройства Plug and Play обычно достаточно его отключить или удалить из конфигурации, для удаления некоторых устройств необходимо сначала выключить компьютер.

Удаление устройств, не являющихся устройствами Plug and Play, обычно состоит из двух шагов:

отмена установки устройства с помощью диспетчера устройств и удаление устройства из конфигурации компьютера.

Не обязательно удалять устройство, которое требуется отключить, не отсоединяя от компьютера. Не отменяя установку самонастраиваемого устройства, его можно просто отключить. При отключении такого устройства оно физически остается подключенным к компьютеру, но Windows обновляет системный реестр таким образом, что драйверы отключенного устройства не загружаются при запуске компьютера. При включении устройства драйверы снова становятся доступными. Эта возможность полезна при необходимости переключения между двумя устройствами, например сетевым адаптером и модемом, или при устранении неполадок в оборудовании.

4.3 Диски и файловая система

Для получения доступа к просмотру состояния и управлению дисками нужно щелкнуть правой клавишей мыши по значку «Мой компьютер», выбрать строку «Управление» и щелкнуть по ней. В открывшемся окне щелкнуть по строке «Управление дисками» (рис. 4.2). В правой части окна будут отображены все дисковые устройства компьютера и основные параметры их состояния.

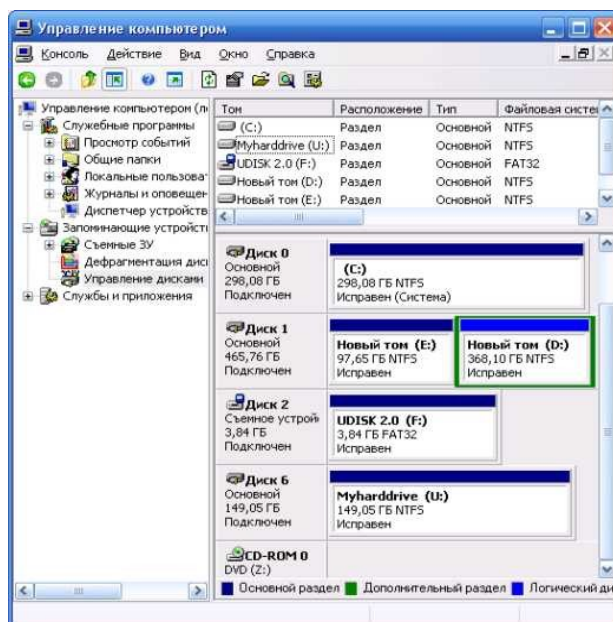


Рис. 4.2 - Вид окна «Управление компьютером» на вкладке «Управление дисками»

В окне можно управлять разделами дисковых устройств. Можно создать или удалить раздел или логический диск, можно сделать первичный раздел активным, чтобы при перезагрузке операционной системы обращение к загрузочной записи осуществлялось с указанного раздела. Активный раздел может быть только один. Здесь же можно отформатировать диск и изменить букву или путь диска. Все эти действия вызываются щелчком правой кнопки мыши по выбранному разделу в окне, представленном на рисунке 4.2.

При работе с жестким диском всегда имеет место фрагментация. С течением времени после установки программ диск заполняется, а после их удаления файлы фрагментируются и операционной системе приходится искать свободные фрагменты на диске для размещения файлов. Это может привести к заметному снижению быстродействия компьютера. Негативный эффект фрагментации устраняется с помощью встроенной в Windows программы дефрагментации, запустить которую можно, указав предварительно имя диска, в левой панели оснастки «Управление компьютером» (рис. 4.3).

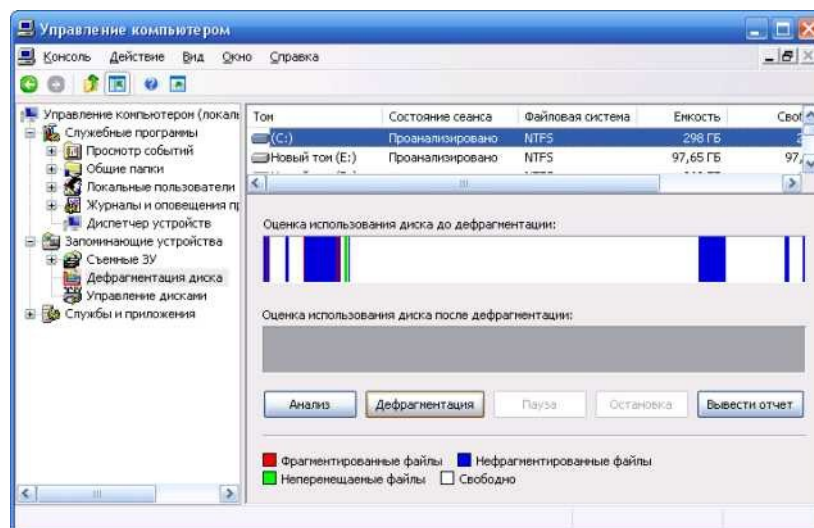


Рис. 4.3 - Вид окна «Управление компьютером» на вкладке «Дефрагментация диска»

Результаты дефрагментации можно просмотреть, нажав на кнопку «Вывести отчет», которая становится доступной после завершения дефрагментации.

4.4 Дисковые квоты

При совместном использовании дисковой памяти несколькими пользователями, работающими на одном компьютере, необходим контроль расходования дискового пространства. В Windows на платформе NT эта проблема решается квотированием дискового пространства по каждому тому (независимо от количества физических дисков) и для каждого пользователя.

После установки квот дискового пространства пользователь сможет хранить на томе ограниченный объем данных, в то время как на этом томе может оставаться свободное пространство. Если пользователь превышает выданную ему квоту, в журнал событий вносится соответствующая запись. Затем, в зависимости от конфигурации системы, пользователь либо сможет записать информацию на том (более мягкий режим), либо ему будет отказано в записи.

Устанавливать и просматривать квоты на диске можно только в разделе NTFS 5.0 и при наличии необходимых полномочий (задаваемых с помощью локальных или доменных групповых политик) у пользователя, устанавливающего квоты.

Чтобы установить квоты, нужно выполнить следующие действия:

1. Щелкнуть правой кнопкой мыши по конфигурируемому тому и выбрать в контекстном меню команду «Свойства». В появившемся окне перейти на вкладку Квота (рис. 4.4).
2. Установить флажок «Включить управление квотами». В этом случае будет установлен мягкий режим контроля используемого дискового пространства. Для задания жесткого режима контроля нужно установить флажок «Не выделять место на диске при превышении квоты». На этой же вкладке устанавливается размер выделяемой квоты и порог, превышение которого вызовет запись предупреждений в журнале событий.

Чтобы узнать, какие пользователи превысили выделенную им квоту (в мягком режиме), нужно нажать кнопку «Записи квот», где будет отражен список пользователей с параметрами квот и объемом используемого ими пространства диска.

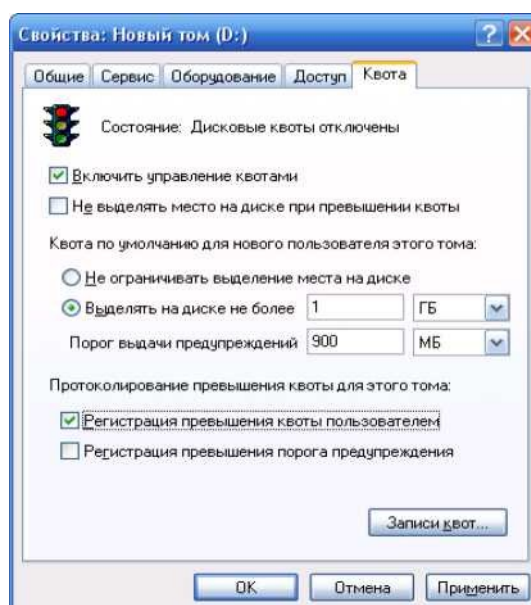


Рис. 4.4 - Вид окна по просмотру свойств диска на вкладке «Квота»

4.5 Обеспечение надежности хранения данных на дисковых накопителях с файловой системой NTFS 5.0

Устанавливая пользователям определенные **разрешения для файлов и каталогов (папок)**, администраторы системы могут защищать конфиденциальную информацию от несанкционированного доступа¹. Каждый пользователь имеет определенный набор разрешений на доступ к конкретному объекту файловой системы (рис. 4.5). Администратор может назначить себя владельцем любого объекта файловой системы.

Действующие разрешения в отношении конкретного файла или каталога образуются из всех прямых и косвенных разрешений, назначенных пользователю для данного объекта с помощью логической функции ИЛИ.

Пользователь может назначить себя владельцем какого-либо объекта файловой системы, если у него есть необходимые права, а также передать права владельца другому пользователю.

Точки соединения (аналог монтирования в UNIX) позволяют отображать целевую папку (диск) в пустую папку, находящуюся в пространстве имен файловой системы NTFS 5.0 локального компьютера. Целевой папкой может служить любой допустимый путь Windows 2000² или выше. Точки соединений прозрачны для приложений, это означает, что приложение или пользователь, осуществляющий доступ к локальной папке NTFS, автоматически перенаправляется к другой папке.

¹ Для практического исследования данных возможностей необходимо использовать Windows Server на платформе 2000 или выше.

² Поддерживаются только в NTFS 5.0

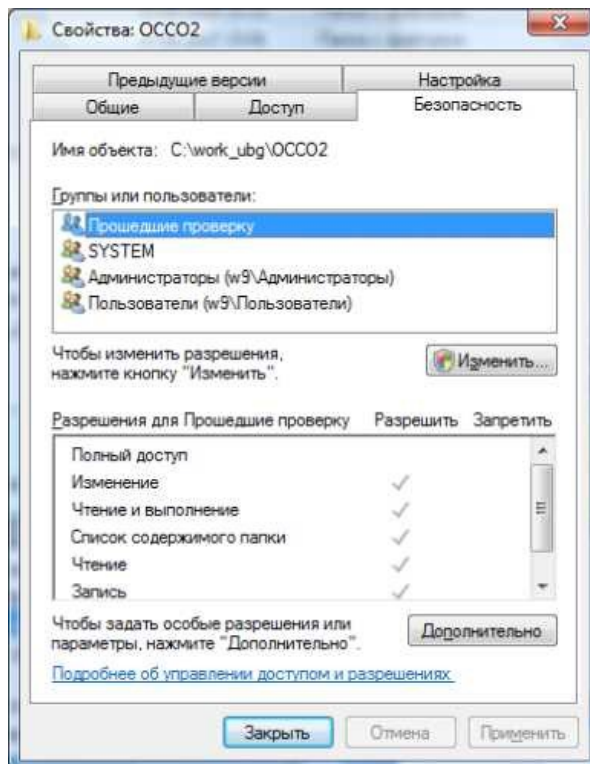


Рис. 4.5 - Вид окна установки разрешений на доступ к конкретному объекту файловой системы

Для работы с точками соединения на уровне томов можно использовать стандартные средства системы – *утилиту Mountvol* (рис. 4.6) и оснастку «Управление дисками». Для монтирования папок нужна утилита Linkd (из Windows 2000 Resource Kit).

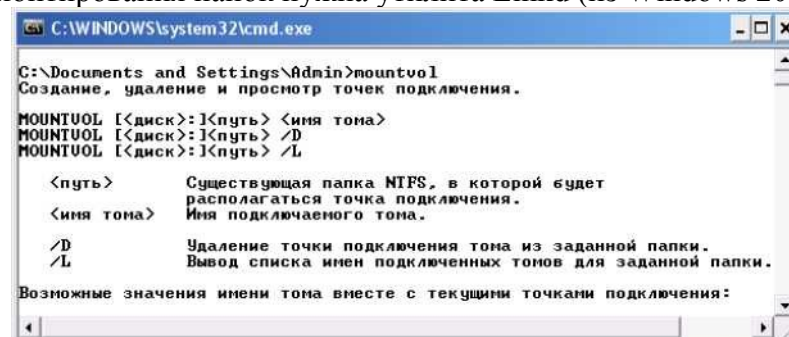


Рис. 4.6 - Вызов утилиты mountvol

С помощью утилиты Mountvol можно выполнить следующие действия:

- отобразить корневую папку локального тома в некоторую целевую папку NTFS, т.е. подключить или монтировать том;
- вывести на экран информацию о целевой папке точки соединения NTFS, использованной при подключении тома;
- просмотреть список доступных для использования томов файловой системы; - уничтожить точки подключения томов.

Оснастка «Управление дисками» позволяет также создать соединения для дисков компьютера.

Шифрующая файловая система EFS (Encrypting File System). Поскольку шифрование и дешифрование выполняются автоматически, пользователь может работать с файлом так же, как и до установки его криптозащиты. Все остальные пользователи, которые попытаются получить доступ к зашифрованному файлу, получают сообщение об

ошибке доступа, поскольку они не владеют необходимым личным ключом, позволяющим им расшифровать файл.

Шифрование информации задается в окне свойств файла или папки. В окне свойств файла на вкладке Общие нужно нажать кнопку другие. Появится окно диалога «Дополнительные атрибуты». В группе «Атрибуты сжатия и шифрования» необходимо установить флажок «Шифровать содержимое для защиты данных» и нажать кнопку ОК. Далее следует нажать кнопку ОК в окне свойств зашифровываемого файла или папки. Появится окно, в котором надо указать режим шифрования.

При шифровании папки можно указать следующие режимы применения нового атрибута: «Только к этой папке» или «К этой папке и ко всем вложенным папкам и файлам». Для дешифрования файла или папки на вкладке «Общие» окна свойств соответствующего объекта нажать кнопку «Другие» и в открывшемся окне сбросить флажок «Шифровать содержимое для защиты данных».

В процессе шифрования файлов и папок система EFS формирует специальные атрибуты (Data Decryption Field – Поле дешифрования данных), содержащие список зашифрованных ключей (FEK – File Encryption Key), что позволяет организовать доступ к файлу со стороны нескольких пользователей. Для шифрования набора FEK используется открытая часть пары ключей каждого пользователя. Информация, требуемая для дешифрования, привязывается к самому файлу. Секретная часть ключа пользователя используется при дешифровании FEK. Она хранится в безопасном месте, например на смарт-карте или устройстве высокой степени защищенности.

FEK применяется для создания ключей восстановления, которые хранятся в другом специальном атрибуте – DRF (Data Recovery Field – Поле восстановления данных). Сама процедура восстановления выполняется довольно редко (при уходе пользователя из организации или забывании секретной части ключа).

Система EFS имеет встроенные средства восстановления зашифрованных данных в условиях, когда неизвестен личный ключ пользователя. Пользователи, которые могут восстанавливать зашифрованные данные в условиях утраты личного ключа, называются агентами восстановления данных. Они обладают сертификатом (X.509 v.3) на восстановление данных и личным ключом, с помощью которого выполняется операция восстановления зашифрованных данных.

4.6 Задание на выполнение

1. Исследуйте работу диспетчера устройств.
2. Опишите структуру дисков и файловых систем на вашем компьютере.
3. Исследуйте механизм раздачи дисковых квот.
4. Исследуйте механизм надежности хранения информации.

Ответьте письменно на вопросы по самостоятельной подготовке:

1. Опишите структуру файловой системы NTFS.
2. Опишите преимущества и недостатки файловой системы NTFS.

Практическое занятие № 24

Файлы пакетной обработки

1.1 Цель работы

Целью данной работы является:

- изучение назначения и основных возможностей командных файлов (Файлов пакетной обработки) операционных систем, построенных на платформе Windows NT;
- знакомство со специальными командами, используемыми в командных файлах;
- исследование стандартных потоков ввода-вывода и их перенаправление

1.2 Язык командных файлов

Командный файл - это текстовый файл (в коде ASCII), состоящий из группы команд. Правила идентификации командных файлов совпадают с общими правилами идентификации файлов. Единственное исключение – командный файл всегда записывается на диск с расширением «.BAT» и/или «.CMD» (для операционных систем Windows на платформе NT).

Обратиться к командному файлу крайне просто. Набирается команда старта - имя файла, и нажимается клавиша Enter. После введения команды файл выбирается из рабочего каталога указанного или рабочего диска. Если в рабочем каталоге его нет, то поиск файла будет производиться в каталогах, описанных системной переменной %PATH%. При нахождении файла первая из его команд загружается в память, отображается на экране и выполняется. Этот процесс повторяется последовательно для всех команд файла (от первой до последней команды).

Выполнение командного файла можно прервать в любой момент, нажав на клавиши Ctrl-Break (Ctrl-C).

Организация командного файла. Существует несколько способов организации командных файлов. Файл можно создать с помощью любого текстового редактора или введением команд непосредственно с клавиатуры. В этом случае ввод оформляется файлом и записывается на диск.

В сеансе DOS клавиатура называется «CON» (CONsole) Для организации файла используется команда «COPY CON:». Наберите команду и имя создаваемого файла. Например, для создания файла «SAMPLE.BAT» введите:

```
C:\>COPY CON: SAMPLE.BAT
```

После этого введите составляющие файл команды. Набрав последнюю команду, одновременно нажмите клавиши Ctrl-Z (или функциональную клавишу F6) и клавишу Enter.

Стандартные потоки ввода-вывода и перенаправление потоков. Термин CONsole используется для обозначения стандартных потоков ввода-вывода. Когда говорят о вводе с консоли, подразумевается ввод с клавиатуры. Когда говорят о выводе на консоль, подразумевают вывод на экран монитора. Существуют специальные символы для перенаправления стандартных потоков ввода-вывода.

> приемник – перенаправить стандартный вывод в приемник (если файл-приемник существует, то он будет создан заново).

>> приемник – перенаправить стандартный вывод в приемник (если файл-приемник существует, то он будет сохранен, а информация будет записана в конец файла).

< источник – перенаправить стандартный ввод из источника. передатчик | приемник – передает вывод одной команды на вход другой.

Замещаемые параметры. Внутри командного файла допускается использование замещаемых параметров. Параметр – это символьная переменная, расположенная в командной строке после имени команды. Он содержит дополнительную информацию, необходимую операционной системе при обработке команды. Параметром, например, может быть имя файла, к которому относится действие команды. Замещаемый параметр – это специальная переменная, которая в процессе выполнения команды подменяется

обычным параметром (например, именем файла). В командном файле замещаемый параметр обозначается знаком процента % и цифрой от 0 до 9. Таким образом, командный файл может включать до десяти замещаемых параметров. Символьные переменные, предназначенные для подмены замещающего параметра, вводятся в командной строке при обращении к командному файлу – набирается команда старта (имя файла) и список параметров в порядке, соответствующем последовательности замещаемых параметров внутри файла.

Параметры заменяются в порядке следования символьных переменных в командной строке. Первая переменная подменяет параметр %1, вторая - параметр %2 и т.д. Вместо замещаемого параметра %0 автоматически подставляется спецификация (имя) командного файла.

При введении замещаемых параметров командный файл становится более гибким. Поясним это на примере. Предположим, что на диске имеется несколько файлов, которые нужно копировать после каждой корректировки. В рассмотренном выше примере командный файл использовался для копирования конкретного файла. Этим же командным файлом можно воспользоваться и для копирования любого файла. В этом случае вместо имени копируемого файла подставляется замещаемый параметр.

Имя копируемого файла будет вводиться в командной строке при обращении к командному файлу. Назовем наш командный файл «COPYALL.BAT». Введем в нем: COPY %1 A:

При обращении к файлу набирается его имя и через пробел - имя копируемого файла (в нашем примере «SHOPLIST.DOC»). Введите команду: C:\>COPYALL.BAT SHOPLIST.DOC

На экран выводится следующая команда:

C:\>COPY SHOPLIST.DOC A:

1 File(s) copied

DOS автоматически подставила имя файла на место замещаемого параметра %1. Усложним пример. Организуем командный файл «DIFNUM.BAT», автоматически копирующий любой указанный файл и присваивающий копии любое указанное имя:

COPY %1 A:%2

Для обращения к этому файлу наберите его имя, имя копируемого файла, в нашем примере «NEW.DOC», и имя копии «OLD.DOC»:

C:\>DIFNUM NEW.DOC OLD.DOC

На экране появляется следующая команда файла «DIFNUM.BAT»:

C:\>COPY NEW.DOC A:OLD.DOC

1 File(s) copied

Первое имя в командной строке «NEW.DOC» поставлено вместо замещаемого параметра %1. Второе имя «OLD.DOC» - вместо замещаемого параметра %2.

Замещаемые параметры и замещаемые символы. Параметр в командной строке команды старта командного файла может включать замещаемые символы «?» и «*». Если замещаемый символ вводится для обозначения группы параметров, то команда выполняется по количеству параметров в группе (т.е. один раз для каждого параметра). Рассмотрим командный файл: COPY %1 CON:

Этот файл копирует на экран (CON) файл, описанный замещаемым параметром %1 (DISPLAY.BAT). Имя копируемого файла указывается в командной строке при обращении к командному файлу. Если указанный файл найден, его содержимое выводится на экран.

Этот файл копирует на экран (con) файл, описанный замещаемым параметром %1. Имя копируемого файла указывается в командной строке при обращении к командному файлу. Если указанный файл найден, его содержимое выводится на экран. Итак, командный файл «DISPLAY.BAT» записан на диск. Введем команду:

C:\>DISPLAY *.TXT

Все файлы рабочего диска с соответствующей спецификацией будут выведены на экран. Если имя копируемого файла включает обозначение процента, то при введении его в командную строку знак процента набирается два раза подряд. Например, имя «НІНО%.ТХТ» в командной строке должно быть представлено как «НІНО%%.ТХТ».

1.3 Некоторые команды DOS (Windows)

Для получения полного списка команд DOS, поддерживаемых вашей операционной системой Windows, построенной на платформе NT, необходимо ввести команду³: **HELP**

Вот ее возможный результат:

Для получения сведений об определенной команде наберите **HELP <имя команды>**

ASSOC – Вывод либо изменение сопоставлений по расширениям имен файлов.

AT – Выполнение команд и запуск программ по расписанию.

ATTRIB – Отображение и изменение атрибутов файлов.

BREAK – Включение/выключение режима обработки комбинации клавиш CTRL+C.

CACLS – Отображение/редактирование списков управления доступом (ACL) к файлам.

CALL – Вызов одного пакетного файла из другого.

CD – Вывод имени либо смена текущей папки.

CHCP – Вывод либо установка активной кодовой страницы.

CHDIR – Вывод имени либо смена текущей папки.

CHKDSK – Проверка диска и вывод статистики

CHKNTFS– Отображение или изменение выполнения проверки диска во время загрузки.

Синтаксис представлен для ОС Windows XP построенной на базе технологии NT, в ОС MS-DOS и Windows 9x количество аргументов и команд несколько меньше.

CLS – Очистка экрана.

CMD – Запуск еще одного интерпретатора командных строк Windows.

COLOR – Установка цвета текста и фона, используемых по умолчанию. **COMP** – Сравнение содержимого двух файлов или двух наборов файлов.

COMPACT– Отображение/изменение сжатия файлов в разделах NTFS.

CONVERT– Преобразование дисковых томов FAT в NTFS. Нельзя выполнить преобразование текущего активного диска.

COPY – Копирование одного или нескольких файлов в другое место.

DATE – Вывод либо установка текущей даты.

DEL – Удаление одного или нескольких файлов.

DIR – Вывод списка файлов и подпапок из указанной папки.

DISKCOMP– Сравнение содержимого двух гибких дисков.

DISKCOPY–Копирование содержимого одного гибкого диска на другой.

DOSKEY– Редактирование и повторный вызов командных строк; создание макросов.

ECHO – Вывод сообщений и переключение режима отображения команд на экране.

ENDLOCAL– Конец локальных изменений среды для пакетного файла.

ERASE – Удаление одного или нескольких файлов.

EXIT – Завершение работы программы CMD.EXE (интерпретатора командных строк).

FC – Сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FIND – Поиск текстовой строки в одном или нескольких файлах.

FINDSTR – Поиск строк в файлах.

FOR – Запуск указанной команды для каждого из файлов в наборе.

FORMAT – Форматирование диска для работы с Windows.

FTYPE – Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.

GOTO – Передача управления в отмеченную строку пакетного файла.

GRAFTABL – Позволяет Windows отображать расширенный набор символов в графическом режиме.

HELP – Выводит справочную информацию о командах Windows.

IF – Оператор условного команд выполнения в пакетном файле.

LABEL – Создание, изменение меток тома для дисков.

MD – Создание папки. и удаление

MKDIR – Создание папки.

MODE – Конфигурирование системных устройств.

MORE – Последовательный вывод данных по частям размером в один экран.

MOVE – Перемещение одного или нескольких файлов из одной папки в другую.

PATH – Вывод либо установка пути поиска исполняемых файлов.

PAUSE – Приостановка выполнения пакетного файла и вывод сообщения.

POPD – Восстановление значения текущей активной папки, с помощью команды PUSHHD.

PRINT – Вывод на печать содержимого текстовых файлов.

PROMPT – Изменение приглашения в командной строке Windows. предыдущего сохраненного

PUSHHD – Сохранение значения текущей активной папки и переход к другой папке.

RD – Удаление папки.

RECOVER – Восстановление читаемой информации с плохого или поврежденного диска.

REM – Помещение комментариев в пакетные файлы и файл CONFIG.SYS. REN – Переименование файлов и папок. RENAME – Переименование файлов и папок. REPLACE – Замещение файлов. RMDIR – Удаление папки.

SET – Вывод, установка и удаление переменных среды Windows.

SETLOCAL – Начало локальных изменений среды для пакетного файла.

SHIFT – Изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

SORT – Сортировка ввода.

START – Запуск программы или команды в отдельном окне.

SUBST – Сопоставляет заданному пути имя диска.

TIME – Вывод и установка системного времени.

TITLE – Назначение заголовка окна для текущего сеанса интерпретатора командных строк CMD.EXE.

TREE – Графическое отображение структуры папок заданного диска или заданной папки.

TYPE – Вывод на экран содержимого текстовых файлов.

VER – Вывод сведений о версии Windows.

VERIFY – Установка режима проверки правильности записи файлов на диск.

VOL – Вывод метки и серийного номера тома для диска.

XCOPY – Копирование файлов и дерева папок.

Чтобы получить информацию о какой-либо команде операционной системы можно также в командной строке набрать имя команды и через пробел указать знак /?. Например, C:\>PAUSE /?

Далее приводится основной синтаксис некоторых команд, необходимых для выполнения лабораторной работы.

ECHO

ECHO [ON | OFF] – переключение режима отображения команд на экране. ECHO [сообщение] – вывод сообщений.

Введите ECHO без параметра для определения текущего значения этой команды. Введите ECHO. (с точкой) для получения пустой строки.

@ – знак экранирования. Отключает вывод на экран текущей строки.

GOTO – передача управления содержащей метку строке пакетного файла. GOTO метка метка – строка пакетного файла, оформленная как метка.

Метка должна находиться в отдельной строке и начинаться с двоеточия.

IF – оператор условного выполнения команд в пакетном файле.

IF [NOT] ERRORLEVEL число команда

IF [NOT] строка1==строка2 команда

IF [NOT] EXIST имя_файла команда

NOT – обращает истинность условия: истинное условие становится ложным, а ложное – истинным.

ERRORLEVEL число – условие является истинным, если код возврата последней выполненной программы не меньше указанного числа.

строка1==строка2 – это условие является истинным, если указанные строки совпадают. IF (%1)==() – проверка на пустой параметр.

EXIST имя_файла – это условие является истинным, если файл с указанным именем существует.

команда – задает команду, выполняемую при истинности условия. За этой командой может следовать ключевое слово ELSE, служащее для указания команды, которая должна выполняться в том случае, если условие ложно.

Предложение ELSE должно располагаться в той же строке, что и команда, следующая за ключевым словом IF. Например:

```
IF EXIST имя_файла. ( del имя_файла. ) ELSE ( echo имя_файла. missing.
)
```

Следующий пример содержит ОШИБКУ, поскольку команда del должна заканчиваться переходом на новую строку:

```
IF EXIST имя_файла. del имя_файла. ELSE echo имя_файла. missing
```

Следующий пример также содержит ОШИБКУ, поскольку команда ELSE должна располагаться в той же строке, что и команда, следующая за IF:

```
IF EXIST имя_файла. del имя_файла. ELSE echo имя_файла. missing
```

Вот правильный пример, где все команды расположены в одной строке:

```
IF EXIST имя_файла. (del имя_файла.) ELSE echo имя_файла. missing
```

PAUSE – приостановка выполнения пакетного файла и вывод сообщения: Для продолжения нажмите любую клавишу . . .

DIR – вывод списка файлов и подкаталогов из указанного каталога.

```
DIR [диск:][путь][имя_файла] [/A[:]атрибуты] [/B] [/C] [/D] [/L] [/N]
[/O[:]порядок] [/P] [/Q] [/S] [/T[:]время] [/W] [/X] [/4]
```

[диск:][путь][имя_файла]

Диск, каталог и/или файлы, которые следует включить в список.

/A Вывод файлов с указанными атрибутами. атрибуты:

- D Каталоги
- R Доступные только для чтения
- H Скрытые файлы
- A Файлы для архивирования
- S Системные файлы
- Префикс «-» имеет значение НЕ /B Вывод только имен файлов.

/C Применение разделителя групп разрядов для вывода размеров файлов (по умолчанию). Для отключения этого режима служит ключ /-C.

/D Вывод списка в несколько столбцов с сортировкой по столбцам.

/L Использование нижнего регистра для имен файлов.

/N Отображение имен файлов в крайнем правом столбце.

/O Сортировка списка отображаемых файлов. порядок:

- N По имени (алфавитная)
- S По размеру (сперва меньшие)
- E По расширению (алфавитная)
- D По дате (сперва более старые)
- G Начать список с каталогов
- Префикс «-» обращает порядок

/P Пауза после заполнения каждого экрана.

/Q Вывод сведений о владельце файла.

/S Вывод списка файлов из указанного каталога и его подкаталогов.

/T Выбор поля времени для отображения и сортировки время:

- C Создание
- A Последнее использование
- W Последнее изменение

/W Вывод списка в несколько столбцов.

/X Отображение коротких имен для файлов, чьи имена не соответствуют стандарту 8.3. Формат аналогичен выводу с ключом /N, но короткие имена файлов выводятся слева от длинных. Если короткого имени у файла нет, вместо него выводятся пробелы.

/4 Вывод номера года в четырехзначном формате

Стандартный набор ключей можно записать в переменную среды DIRCMD. Для отмены их действия введите в команде те же ключи с префиксом «-», например: /-W.

MD – создание каталога.

MKDIR [диск:]путь

MD [диск:]путь

CD – вывод имени либо смена текущего каталога.

CHDIR [/D] [диск:][путь]

CHDIR [..]

CD [/D] [диск:][путь]

CD [..]

.. обозначает переход в родительский каталог.

Команда **CD** диск: отображает имя текущего каталога указанного диска.

Команда **CD** без параметров отображает имена текущих диска и каталога.

Параметр /D используется для одновременной смены текущих диска и каталога.

RD – удаление каталога.

RMDIR [/S] [/Q] [диск:]путь **RD** [/S] [/Q] [диск:]путь

/S Удаление дерева каталогов, т. е. не только указанного каталога, но и всех содержащихся в нем файлов и подкаталогов.

/Q Отключение запроса подтверждения при удалении дерева каталогов с помощью ключа /S.

COPY – копирование одного или нескольких файлов в другое место.

COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/A | /B] источник [/A | /B] [+ источник [/A | /B] [+ ...]] [результат [/A | /B]]

источник Имена одного или нескольких копируемых файлов.

/A Файл является текстовым файлом ASCII.

/B Файл является двоичным файлом.

/D Указывает на возможность создания зашифрованного файла результат Каталог и/или имя для конечных файлов.

/V Проверка правильности копирования файлов.

/N Использование, если возможно, коротких имен при копировании файлов, чьи имена не удовлетворяют стандарту 8.3.

/Y Подавление запроса подтверждения на перезапись существующего конечного файла.

/-Y Обязательный запрос подтверждения на перезапись существующего конечного файла. /Z Копирование сетевых файлов с возобновлением.

Ключ /Y можно установить через переменную среды COPYCMD.

Ключ /-Y командной строки переопределяет такую установку.

По умолчанию требуется подтверждение, если только команда COPY не выполняется в пакетном файле.

Чтобы объединить файлы, укажите один конечный и несколько исходных файлов, используя подстановочные знаки или формат «файл1+файл2+файл3+...».

REN – переименование одного или нескольких файлов.

RENAME [диск:][путь]имя_файла1 имя_файла2.

REN [диск:][путь]имя_файла1 имя_файла2.

Для конечного файла нельзя указать другой диск или каталог.

DEL – удаление одного или нескольких файлов.

DEL [/P] [/F] [/S] [/Q] [/A[:]атрибуты] имена

ERASE [/P] [/F] [/S] [/Q] [/A[:]атрибуты] имена

имена – Имена одного или нескольких файлов. Для удаления сразу нескольких файлов

используются подстановочные знаки. Если указан каталог, из него будут удалены все файлы.

/P Запрос на подтверждение перед удалением каждого файла.

/F Принудительное удаление файлов, доступных только для чтения.

/S Удаление указанных файлов из всех подкаталогов.

/Q Отключение запроса на подтверждение при удалении файлов.

/A Отбор файлов для удаления по атрибутам. атрибуты:

- S Системные файлы
- R Доступные только для чтения
- H Скрытые файлы
- A Файлы для архивирования
- Префикс «-» имеет значение НЕ

TYPE – вывод содержимого одного или нескольких текстовых файлов.

TYPE [диск:][путь]имя_файла

FOR – выполнение указанной команды для каждого файла набора.

FOR %переменная IN (набор) DO команда [параметры]

%переменная - подставляемый параметр;

(набор) - набор, состоящий из одного или нескольких файлов. Допускается использование подстановочных знаков; команда - команда, которую следует выполнить для каждого файла; параметры - параметры и ключи для указанной команды.

В пакетных файлах для команды FOR используется запись %%переменная вместо %переменная. Имена переменных учитывают регистр букв (%i отличается от %I).

Добавление поддерживаемых вариантов команды FOR при включении расширенной обработки команд:

FOR /D %переменная IN (набор) DO команда [параметры]

Если набор содержит подстановочные знаки, команда выполняется для всех подходящих имен каталогов, а не имен файлов.

FOR /R [[диск:]путь] %переменная IN (набор) DO команда [параметры]

Выполнение команды для каталога [диск:]путь, а также для всех подкаталогов этого пути. Если после ключа /R не указано имя каталога, выполнение команды начинается с текущего каталога.

Если вместо набора указана только точка (.), команда выводит список всех подкаталогов.

FOR /L %переменная IN (начало, шаг, конец) DO команда [параметры]

Набор раскрывается в последовательность чисел с заданными началом, концом и шагом приращения. Так, набор (1,1,5) раскрывается в (1 2 3 4 5), а набор (5,-1,1) заменяется на (5 4 3 2 1)

FOR /F [«ключи»] %переменная IN (набор) DO команда [параметры]

FOR /F [«options»] %variable IN («literal string») DO command [command-parameters]

FOR /F [«options»] %variable IN ('command') DO command [command-parameters] или, если использован параметр usebackq:

FOR /F [«options»] %variable IN (filename) DO command [command-parameters]

FOR /F [«options»] %variable IN ('literal string') DO command [command-parameters]

FOR /F [«options»] %variable IN ('command') DO command [command-parameters]

Набор содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбиении его на отдельные строки текста и выделения из каждой строки заданного числа подстрок (в том числе нуля). Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла. По умолчанию ключ /F выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательные параметры «ключи» служат для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую указанные параметры. Ключевые слова:

eol=c – определение символа комментариев в конце строки (допускается задание только одного

символа); skip=n – число пропускаемых при обработке строк в начале файла;

delims=xxx – определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции; tokens=x,y,m-n – определение номеров подстрок, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла. При использовании этого ключа создаются дополнительные переменные. Формат m-n представляет собой диапазон подстрок с номерами от m по n. Если последний символ в строке tokens= является звездочкой, создается дополнительная переменная, значением которой будет весь оставшийся текст в строке после обработки последней подстроки; usebackq – применение новой семантики, при которой строки, заключенные в обратные кавычки, выполняются как команды, строки, заключенные в прямые одиночные кавычки, являются строкой литералов команды, а строки, заключенные в двойные кавычки, используются для выделения имен файлов в списках имен файлов.

Поясняющий пример:

FOR /F “eol=; tokens=2,3* delims=,” %i in (myfile.txt) do @echo %i %j %k

– эта команда обрабатывает файл myfile.txt, пропускает все строки, которые начинаются с символа точки с запятой, и передает вторую и третью подстроки из каждой строки в тело цикла, причем подстроки разделяются запятыми и/или пробелами. В теле цикла переменная %i используется для второй подстроки, %j – для третьей, а %k получает все оставшиеся подстроки после третьей.

Имена файлов, содержащие пробелы, необходимо заключать в двойные кавычки.

Для того чтобы использовать двойные кавычки, необходимо использовать параметр usebackq, иначе двойные кавычки будут восприняты как границы строки для обработки.

Переменная %i явно описана в инструкции for, а переменные %j и %k описываются неявно с помощью ключа tokens=. Ключ tokens= позволяет извлечь из одной строки файла до 26 подстрок, при этом, не допускается использование переменных больших чем буквы 'z' или 'Z'. Следует помнить, что имена переменных FOR являются глобальными, поэтому одновременно не может быть активно более 52 переменных.

Синтаксис команды FOR /F также позволяет обработать отдельную строку, с указанием параметра filename, заключенным в одиночные кавычки.

Строка будет обработана как единая строка из входного файла.

Наконец, команда FOR /F позволяет обработать строку вывода другой команды.

Для этого следует ввести строку вызова команды в апострофах вместо набора имен файлов в скобках. Строка передается для выполнения обработчику команд CMD.EXE, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующая команда:

FOR /F "usebackq delims==" %%i IN ('set') DO @echo %%i

– выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

SHIFT – изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

SHIFT [/n] – команда SHIFT при включении расширенной обработки команд поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть от 0 до 9.

Например, в следующей команде:

SHIFT /2

%3 заменяется на %2, %4 на %3 и т.д., а %0 и %1 остаются без изменений.

CALL – вызов одного пакетного файла из другого. CALL [диск:][путь]имя_файла [параметры] параметры - набор параметров командной строки, необходимых пакетному файлу.

CHOICE³ – ожидает ответа пользователя. CHOICE [/C:]варианты] [/N] [/S] [/T[:]с,nn] [текст]

/C:]варианты – варианты ответа пользователя.

По умолчанию строка включает два варианта: YN

/N Ни сами варианты, ни знак вопроса в строке приглашения не отображаются.

/S Учитывать регистр символов.

/T[:]с,nn Ответ «с» выбирается автоматически после nn секунд ожидания текст

Строка приглашения

После выполнения команды переменная ERRORLEVEL приобретает значение, равное номеру выбранного варианта ответа.

FC – сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FC [/A] [/C] [/L] [/LBn] [/N] [/OFF[LINE]] [/T] [/U] [/W] [/nnnn] [диск 1:] [путь1]имя_файла1

[диск2:] [путь2]имя_файла2

FC /B [диск1:] [путь1]имя_файла1

[диск2:] [путь2]имя_файла2

/A Вывод только первой и последней строк для каждой группы различий.

/B Сравнение двоичных файлов.

/C Сравнение без учета регистра символов.

/L Сравнение файлов в формате ASCII.

/LBn Максимальное число несоответствий для заданного числа строк.

/N Вывод номеров строк при сравнении текстовых файлов ASCII.

/OFF[LINE] Не пропускать файлы с установленным атрибутом «Автономный».

/T Символы табуляции не заменяются эквивалентным числом пробелов.

/U Сравнение файлов в формате UNICODE.

/W Пропуск пробелов и символов табуляции при сравнении.

/nnnn Число последовательных совпадающих строк, которое должно встретиться после группы несовпадающих.

[диск1:] [путь1]имя_файла1

³ CHOICE — это внешняя команда.

Указывает первый файл или набор файлов для сравнения.
[диск2:][путь2]имя_файла2 Указывает второй файл или набор файлов для сравнения.

FIND – поиск текстовой строки в одном или нескольких файлах.

FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] «строка»

[[диск:][путь]имя_файла[...]]

/V Вывод всех строк, НЕ содержащих заданную строку.

/C Вывод только общего числа строк, содержащих заданную строку.

/N Вывод номеров отображаемых строк.

/OFF[LINE] Не пропускать файлы с установленным атрибутом «Автономный».

/I Поиск без учета регистра символов.

«строка» Искомая строка.

[диск:][путь]имя_файла

Один или несколько файлов, в которых выполняется поиск.

Если путь не задан, поиск выполняется в тексте, введенном с клавиатуры либо переданном по конвейеру другой командой.

SORT – осуществляет сортировку файла.

SORT [/R] [/+n] [/M килобайтов] [/L язык] [/REC символов]

[[диск1:][путь1]имя_файла1] [/T [диск2:][путь2]]

[/O [диск3:][путь3]имя_файла3]

/+n Задаёт число символов, n, до начала каждого сравнения. /+3 показывает, что каждое сравнение будет начинаться с третьего символа каждой строки. Строки меньше чем n символов собираются перед всеми остальными строками.

По умолчанию, сравнение начинается с первого символа каждой строки.

/L[OCAL] язык Перекрывает установленные в системе по умолчанию язык и раскладку заданными. Пока существует возможность только одного выбора: ««С»» - наиболее быстрый способ упорядочивания последовательности.

Сортировка всегда идет без учета регистра.

/M[EMORY] килобайтов Задаёт количество основной памяти, используемой для сортировки, в килобайтах. Размер памяти должен быть не менее 160КБ.

/REC[ORD_MAXIMUM] символов Определяет максимальное число символов в записи (по умолчанию 4096, максимальное 65535).

/R[EVERSE] Обратный порядок сортировки; т.е. сортировка идет от Я до А, и затем от 9 до 0.

[диск1:][путь1]имя_файла1 Определяет имя сортируемого файла. Если оно опущено, то будет использоваться стандартный поток ввода. Явное задание сортируемого файла работает быстрее, чем перенаправление того же файла в качестве стандартного потока ввода.

/T[EMPORARY] [диск2:][путь2] Определяет путь к папке, содержащей рабочие файлы сортировки, в том случае, когда данные не помещаются в основной памяти. По умолчанию используется системная временная папка.

/O[UTPUT] [диск3:][путь3]имя_файла3 Определяет имя файла, в котором сохраняются отсортированные результаты. Если оно опущено, данные записываются в стандартный поток вывода. Явное задание файла вывода работает быстрее, чем перенаправление стандартного потока вывода в этот же файл.

1.4 Варианты заданий на выполнение

При разработке учтите возможность неправильного запуска ваших программ (например, с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки. При выполнении работы используйте данные методические указания и справочник THelp.

Вариант 1. Разработать командный файл создающий, копирующий или удаляющий файл, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 2. Разработать командный файл создающий, копирующий или удаляющий каталог, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 3. Разработать командный файл, добавляющий вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей (замещаемого параметра) /b /e).

Вариант 4. Разработать командный файл, регистрирующий время своего запуска в файле протокола run.log и автоматически запускающий некоторую программу (например, антивирусную и т. п.) по пятницам или 13 числам.

Вариант 5. Разработать командный файл, копирующий произвольное число файлов, заданных аргументами из текущего каталога в указываемый каталог.

Вариант 6. Разработать командный файл, который помещает список файлов текущего каталога в текстовый файл и в зависимости от ключа сортирует по какому-либо полю. Реализовать два варианта: с использованием только команды DIR, с использованием команд DIR и SORT.

Вариант 7. Разработать командный файл, который в интерактивном режиме мог бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.

Вариант 8. Разработать командный файл, который дописывал бы имя файла, полученного входным параметром в сам файл N количество раз. N - также задается параметром.

Вариант 9. Разработать командный файл, который бы запускал бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает. В работе используйте для сравнения дат команду FC.

Вариант 10. Разработать командный файл, который бы запускал бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает. Сравнение дат реализуйте через переменные, а не через файлы.

Вариант 11. Разработать командный файл, который получал в качестве параметра какое-либо имя, и проверял, определена ли такая переменная среды или нет, и выводил соответствующее сообщение.

Вариант 12. Разработать командный файл, который получал в качестве параметра какой-либо символ и в зависимости от второго параметра вырезал или сохранял в заданном файле все строки, начинающиеся на этот символ.

Вариант 13. В некотором файле храниться список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос - копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да» то программа копирует файл.

Вариант 14. Разработать командный файл, который бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования.

Вариант 15. Разработать командный файл, который бы получал в качестве аргумента имя текстового файла и выводил на экран информацию о том, сколько символов, слов и строк в текстовом файле.

Вариант 16. Разработать командный файл (аналог команды tail в Unix). Командный файл печатает конец файла. По умолчанию - 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.

Вариант 17. Разработать командный файл, который бы склеивал текстовые файлы, заданные в качестве аргументов, и сортировал бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Вариант 18. Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные).

Вариант 19. Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы измененные).

Вариант 20. Выполняющий в зависимости от ключа один из 3-х вариантов работы:

- с ключом /n дописывает в начало указанных текстовых файлов строку с именем текущего файла;
- с ключом /b создает резервные копии указанных файлов; с ключом /d удаляет указанные файлы после предупреждения.

Практическое занятие №25

Программирование на языке SHELL в ОС Unix

2.1 Цель работы

Изучение языка Shell, использование переменных среды, переменных Shell и предопределенных переменных.

2.2 Программирование в языке Shell

Версии Shell

Shell – интерпретатор команд, подаваемых с терминала или из командного файла. Это обычная программа (т.е. не входит в ядро операционной системы UNIX). Ее можно заменить на другую или иметь несколько.

Две наиболее известные версии:

- Shell (версии 7 UNIX) или Bourne Shell (от фамилии автора S.R.Bourne из фирмы Bell Labs);
- C-Shell (версии Berkley UNIX).

Они похожи, но есть и отличия: C-Shell мощнее в диалоговом режиме, а обычный Shell имеет более элегантные управляющие структуры.

Shell – язык программирования, так как имеет:

- переменные;
- управляющие структуры (типа if);
- подпрограммы (в том числе командные файлы);
- передачу параметров;
- обработку прерываний.

Файл начала сеанса (login-файл)

Независимо от версии Shell при входе в систему UNIX ищет файл начала сеанса с предопределенным именем, чтобы выполнить его как командный файл;

- для UNIX версии 7 это: .profile;
- для C-Shell это: .login и/или .cshrc.

В этот файл обычно помещают команды:

- установки характеристик терминала;
- оповещения типа who, date;
- установки каталогов поиска команд (обычно: /bin, /usr/bin); - смена подсказки с \$ на другой символ и т.д.

Процедура языка Shell

Это командный файл. Два способа его вызова на выполнение:

1. \$ sh dothat (где dothat – некоторый командный файл); 2. \$ chmod 755 dothat (сделать его выполнимым, т.е. -rwxr-xr-x) \$ dothat.

Следует знать порядок поиска каталогов команд (по умолчанию):

- текущий;
- системный /bin;
- системный /usr/bin.

Следовательно, если имя вашего командного файла дублирует имя команды в системных каталогах, последняя станет недоступной (если только не набирать ее полного имени).

Переменные Shell

В языке Shell версии 7 определение переменной содержит имя и значение: var = value.

Доступ к переменной – по имени со знаком \$ спереди: fruit = apple (определение); echo \$fruit (доступ); apple (результат echo).

Таким образом, переменная – это строка. Возможна конкатенация строк:

```
$ fruit = apple
$ fruit = pine$fruit
$ echo $fruit pineapple
```

```
$ fruite = apple
$ wine = ${fruite}jack
$ echo $wine applejack $
```

Другие способы установки значения переменной – ввод из файла или вывод из команды, а также присваивание значений переменной – параметру цикла for из списка значений, заданного явно или по умолчанию.

Переменная может быть:

- Частью полного имени файла: \$d/filename, где \$d – переменная (например, d = /usr/bin).
- Частью команды:
\$ S = "sort + 2n + 1 - 2" (наличие пробелов требует кавычек "") \$ \$S tennis/lpr
\$ \$S basketball/lpr
\$ \$S pingpong/lpr \$

Однако внутри значения для команды не могут быть символы |, >, <, & (обозначающие канал, перенаправления и фоновый режим).

Предопределенные переменные Shell

Некоторые из них можно только читать. Наиболее употребительные:

HOME – "домашний" каталог пользователя; служит аргументом по умолчанию для cd;

PATH – множество каталогов, в которых UNIX ищет команды; PS1 – первичная подсказка (строка) системы (для v.7 - \$).

Изменение PS1 (подсказки) обычно делается в login- файле, например: PS1 = ? или PS1 = "? " (с пробелом, что удобнее). Изменение PATH:

```
$ echo $PATH          - посмотреть;
:/bin:/usr/bin       - значение PATH;
$ cd - "домой";
$ mkdir bin          - новый каталог;
$ echo $HOME          - посмотреть;
/users/maryann        - текущий каталог;
$ PATH = :$HOME/bin:$PATH - изменение PATH;
$ echo $PATH          - посмотреть;
/users/maryann/bin:/bin:/usr/bin - новое значение PATH.
```

Установка переменной Shell выводом из команды Пример 1:

```
$ now = 'date' (где " - обратные кавычки)
$ echo $now
Sun Feb 14 12:00:01 PST 1985
$
```

Пример 2: (получение значения переменной из файла):

```
$ menu = 'cat food'
$ echo $menu apples cheddar chardonnay (символы возврата каретки заменяются на пробелы).
```

Переменные Shell – аргументы процедур Это особый тип переменных, именуемых цифрами.

Пример:

```
$ dothis grapes apples pears (процедура).
```

Тогда позиционные параметры (аргументы) этой команды доступны по именам:

```
$1 = 'grapes'
$2 = 'apples'
```

\$3 = 'pears' и т.д. до \$9. Однако есть команда shift, которая сдвигает имена на остальные аргументы, если их больше 9 (окно шириной 9).

Другой способ получить все аргументы (даже если их больше 9): \$*, что эквивалентно \$1\$2 ...

Количество аргументов присваивается другой переменной: \$#(диез).

Наконец, имя процедуры - это \$0; переменная \$0 не учитывается при подсчете \$#.

Структурные операторы Shell

Оператор цикла **for**

Пусть имеется командный файл makelist (процедура)

```
$ cat makelist sort +1 -2 people | tr -d -9 | pr -h Distribution | lpr.
```

Если вместо одного файла people имеется несколько, например: adminpeople, hardpeople, softpeople,..., то необходимо повторить выполнение процедуры с различными файлами. Это возможно с помощью for – оператора. Синтаксис:

```
for <переменная> in <список значений> do <список команд> done
```

Ключевые слова for, do, done пишутся с начала строки.

Пример (изменим процедуру makelist): for file in adminpeople, hardpeople, softpeople
do

```
Sort +1 -2 $file | tr ... | lpr done.
```

Можно использовать метасимволы Shell в списке значений.

Пример: for file in *people (для всех имен, кончающихся на people) do done.

Если in опущено, то по умолчанию в качестве списка значений берется список аргументов процедуры, в которой содержится цикл, а если цикл не в процедуре, то – список параметров командной строки (то есть в качестве процедуры выступает команда).

Пример: for file do done

Для вызова makelist adminpeople hardpeople softpeople будет сделано то же самое.

Условный оператор **if**

Используем имена переменных, представляющие значения параметров процедуры:

```
sort +1 -2 $1 | tr ... | lpr
```

Пример неверного вызова:

makelist (без параметров), где \$1 неопределен. Исправить ошибку можно, проверяя количество

аргументов - значение переменной \$# посредством if - оператора.

Пример: (измененной процедуры makelist): if test \$# -eq 0 then echo "You must give a filename" exit 1

```
else sort +1 -2 $1 | tr ... | lpr
```

Здесь test и exit - команды проверки и выхода. Таким образом, синтаксис оператора if: if <если эта команда выполняется успешно, то>; then <выполнить все следующие команды до else или, если его нет, до fi>;

```
[else <иначе выполнить следующие команды до fi>]
```

Ключевые слова if, then, else и fi пишутся с начала строки.

Успешное выполнение процедуры означает, что она возвращает значение true = 0 (zero) (успех - возвращаемое значение не равно 0).

Оператор exit 1 задает возвращаемое значение 1 для неудачного выполнения makelist и завершает процедуру.

Возможны вложенные if. Для else if есть сокращение elif, которое одновременно сокращает fi.

Команда **test**

Не является частью Shell, но применяется внутри Shell- процедур. Имеется три типа проверок:

- оценка числовых значений; - оценка типа файла; - оценка строк.

Для каждого типа свои примитивы (операции ор).

1. Для чисел синтаксис такой: N ор M, где N, M - числа или числовые переменные; ор принимает значения: -eq, -ne, gt, -lt, -ge, -le.

2. Для файла синтаксис такой: ор filename, где ор принимает значения: - s (файл существует и не пуст);

- -f (файл, а не каталог);

- -d (файл-директория (каталог)); - -w (файл для записи); - -r (файл для чтения).
Для строк синтаксис такой: S op R, где S, R - строки или строковые переменные или op1 S, где op принимает значения: - = (эквивалентность);
- != (не эквивалентность); op1 принимает значения:
- -z (строка нулевой длины);
- -n (не нулевая длина строки).

Наконец, несколько проверок разных типов могут быть объединены логическими операциями -a (AND) и -o (OR).

Примеры:

```
$ if test -w $2 -a -r $1
> then cat $1 >> $2
> else echo "cannot append"
> fi
$
```

В некоторых вариантах ОС UNIX вместо команды test используются квадратные скобки, т.е. if

[...] вместо if test

Оператор цикла **while** Синтаксис:

```
while <команда>
do
<команды> done
```

Если "команда" выполняется успешно, то выполнить "команды", завершаемые ключевым словом done.

Пример:

```
if test $# -eq 0
then echo "Usage: $0 file ..." > &2 exit
fi
while test $# -gt 0
do
if test -s $1 then echo "no file $1" > &2
sort + 1 - 2 $1 | tr -d ... (процедуры) fi shift (* перенумеровать аргументы *) done
```

Процедуры выполняются над всеми аргументами.

Оператор цикла **until**

Инвертирует условие повторения по сравнению с while Синтаксис: until <команда>
do

```
<команды> done
```

Пока "команда" не выполнится успешно, выполнять команды, завершаемые словом done. Пример:

```
if test $# -eq 0
then echo "Usage $0 file..." > &2 exit fi until test $# -eq 0 do if test -s $1 then
echo "no file $1" > &2 else
sort +1 -2 $1 | tr -d ... (процедура) fi shift (сдвиг аргументов) done
```

Исполняется аналогично предыдущему.

Оператор выбора **case**

Синтаксис: case <string> in string1) <если string = string1, то выполнить все следующие команды до ;; > ;; string2) <если string = string2, то выполнить все следующие команды до ;; > ;; string3) ... и т.д. ... esac

Пример:

```
Пусть процедура имеет опцию -t, которая может быть подана как первый параметр:
together = no
case $1 in
-t)    together = yes
```

```

shift ;;
-?)    echo "$0: no option $1"
exit ;;
esac if test $together = yes then sort ...
fi

```

где ? - метасимвол (если -?, т.е. "другая" опция, отличная от -t, то ошибка). Можно употреблять все метасимволы языка Shell, включая ?, *, [-]. Легко добавить (в примере) другие опции, просто расширяя case.

Использование временных файлов в каталоге /tmp

Это специальный каталог, в котором все файлы доступны на запись всем пользователям.

Если некоторая процедура, создающая временный файл, используется несколькими пользователями, то необходимо обеспечить уникальность имен создаваемых файлов. Стандартный прием - имя временного файла \$0\$\$, где \$0 - имя процедуры, а \$\$ - стандартная переменная, равная уникальному идентификационному номеру процесса, выполняющего текущую команду.

Хотя администратор периодически удаляет временные файлы в /tmp, хорошей практикой является их явное удаление после использования.

Комментарии в процедурах

Они начинаются с двоеточия: которое считается нуль- командой, а текст комментария - ее аргументом. Чтобы Shell не интерпретировал метасимволы (\$, * и т.д.), рекомендуется заключать текст комментария в одиночные кавычки.

В некоторых вариантах ОС UNIX примечание начинается со знака #.

Пример процедуры

```

:'Эта процедура работает с файлами, содержащими имена'
:'и номера телефонов,'
:'сортирует их вместе или порознь и печатает результат на'
:'экране или на принтере'
:'Ключи процедуры:'
:'-t (together) - слить и сортировать все файлы вместе'
:'-p (printer) - печатать файлы на принтере'
if test $# - eq 0 then echo "Usage: $ 0 file ... " > & 2 exit
fi
together = no print = no while test $# -gt 0 do case $1 in
-t)    together = yes
shift ;;
-p)    print = yes
shift ;;
-?)    echo "$0: no option $1" exit ;;
*) if test $together = yes then sort -u +1 -2 $1 | tr ... > /tmp/$0$$ if $print = no then cat
/tmp/$0$$ else
lpr -c /tmp/$0$$
fi
rm /tmp/$0$$ exit
else if test -s $1 then echo "no file $1" > &2 else    sort +1 -2 $1 | tr...> /tmp/$0$$ if
$print = no then cat /tmp/$0$$
else lpr -c /tmp/$0$$
fi
rm /tmp/$0$$
fi shift fi;; esac done.

```

Процедура проверяет число параметров \$# и, если оно равно нулю, завершается. В противном случае она обрабатывает параметры (оператор case). В качестве параметра

может выступать либо ключ (символ, предваряемый минусом), либо имя файла (строка, представленная метасимволом *). Если ключ отличен от допустимого (метасимвол ? отличен от t и r), процедура завершается. Иначе в зависимости от наличия ключей t и r выполняются действия, заявленные в комментариях в начале процедуры.

Обработка прерываний в процедурах

Если при выполнении процедуры получен сигнал прерывания (от клавиши BREAK или DEL, например), то все созданные временные файлы останутся неудаленными (пока это не сделает администратор) ввиду немедленного прекращения процесса.

Лучшим решением является обработка прерываний внутри процедуры оператором trap:

Синтаксис: trap 'command arguments' signals...

Кавычки формируют первый аргумент из нескольких команд, разделенных точкой с запятой. Они будут выполнены, если возникнет прерывание, указанное аргументами signals (целые):

2 - когда вы прерываете процесс;

1 - если вы "зависли" (отключены от системы) и др.

Пример (развитие предыдущего): case \$1 in

) trap 'rm /tmp/; exit' 2 1 (удаление временных файлов) if test -s \$1

rm /tmp/*

Лучше было бы:

trap 'rm /tmp/* > /dev/null; exit' 2 1

так как прерывание может случиться до того, как файл /tmp/\$0\$\$ создан и аварийное сообщение об

этом случае перенаправляется на null-устройство.

Выполнение арифметических операций: expr

Команда expr вычисляет значение выражения, поданного в качестве аргумента, и посылает результат на стандартный вывод. Наиболее интересным применением является выполнение операций над переменными языка Shell.

Пример суммирования 3 чисел:

\$ cat sum3 expr \$1 + \$2 + \$3 \$ chmod 755 sum3

\$ sum3 13 49 2

64

\$

Пример непосредственного использования команды:

\$ expr 13 + 49 + 2 + 64 + 1 129 \$

В expr можно применять следующие арифметические операторы: +, -, *, /, % (остаток). Все операнды и операции должны быть разделены пробелами.

Заметим, что знак умножения следует заключать в кавычки (одинарные или двойные), например:

'*', так как символ * имеет в Shell специальный смысл.

Более сложный пример expr в процедуре (фрагмент):

num = 'wc -l < \$1' tot = 100 count = \$num avint = 'expr \$tot / \$num' avdec = 'expr \$tot % \$num' while test \$count -gt 0 do ...

Здесь wc -l осуществляет подсчет числа строк в файле, а далее это число используется в выражениях.

Отладка процедур Shell

Имеются три средства, позволяющие вести отладку процедур.

- Размещение в теле процедуры команд echo для выдачи сообщений, являющихся трассой выполнения процедуры.

- Опция -v (verbose = многословный) в команде Shell приводит к печати команды на экране перед ее выполнением.

- Опция -x (execute) в команде Shell приводит к печати команды на экране по мере ее выполнения с заменой всех переменных их значениями; это наиболее мощное средство.

Утилита AWK

Awk - утилита, подобная gher. Однако, кроме поиска по образцу, она позволяет проверять отношения между полями строк (записей) и выполнять некоторые действия над строками (генерировать отчеты). Название не является акронимом, оно образовано первыми буквами фамилий авторов (A.V.Aho, P.Y.Weinberger, B.W.Kernighan).

Задание поиска-действия следует синтаксису:

/<образец>/ {<действие>}

И образец, и действие могут отсутствовать. Найденные по образцу строки при отсутствии заданного действия выводятся в стандартный вывод (на экран).

Образец задается регулярным выражением, как и в gher. Если образец отсутствует, обрабатываются все строки.

Рассмотрим примеры действий, которые можно выполнить командой awk.

Перестановка полей строки выполняется с помощью ссылки на поле \$n, где n - номер поля.

Например:

```
$ cat people
```

```
Mary Clark 101
```

```
Henry Morgan 112
```

```
Bill Williams 100
```

```
$ awk '{print $2      $1 "4" $3}' people
```

```
Clark, Mary 101
```

Morgan, Henry 112 Williams, Bill 100 где ^ (control - I) - знак табуляции для подвода каретки к очередной позиции табуляции (для выравнивания третьего поля).

Действия для awk могут быть заданы в файле.

Например:

```
$ cat swap
```

```
{print $2      $1 "ЛГ" $3} $ awk -f swap people
```

Awk имеет встроенные образцы и переменные. Образцы BEGIN и END означают начало и конец файла соответственно. Переменная NR (Number of Records) означает число записей (строк) в файле, NF - число полей в записи. Можно использовать переменные, объявленные пользователем. Пример, подсчитывающий среднее значение третьего поля файла tennis (программа действий для awk - в файле average):

```
$ cat > average
```

```
{total = total + $3}
```

```
END {print "Average value is", total/NR}
```

```
ЛД
```

```
$ awk -f average tennis Average value is 8.9 $
```

Образец поиска в awk может содержать условные выражения. Пример, в котором в файле tennis пишутся все записи, значение третьего поля в которых не меньше 10:

```
$ awk '$3 >= 10 {print $0}' tennis
```

```
Steve Daniel 11
```

```
Hank Parker 18
```

```
Jack Austen 14
```

```
$
```

Знак \$0 (доллар-ноль) есть ссылка на всю запись (строку). В общем случае выражение для условия подчиняется синтаксису, близкому к синтаксису выражений в языке C. Кроме того, в команде awk допустимо указывать отрезок образцов. Пример выборки всех записей, сделанных с 1976 до 1978 г.: \$ sort -n -o chard.s chard

```
$ awk '/1976/, /1978/ {if($2 < 8.00 print $0}' chard.s 1976 7.50 Chateau 1977 7.75
Chateau 1978 5.99
Charles
```

Как видно из примера, в программах действий для awk можно использовать управляющие структуры с синтаксисом, близким к языку C.

Пример цикла для печати полей всех записей файла в обратном порядке: `$ awk {for (i = NF; i > 0; --i) print $i} f1`, где NF - число полей в записи.

Встроенные функции AWK `length(arg)` - Функция длины arg. Если arg не указан, то выдает длину текущей строки.

`exp(), log(), sqrt()` - Математические функции экспонента, логарифм и квадратный корень.

`int()` - Функция целой части числа. `substr(s, m, n)` - Возвращает подстроку строки s, начиная с позиции m, всего n символов.

`index(s, t)` - Возвращает начальную позицию подстроки t в строке s. (Или 0, если t в s не содержится.) `sprintf(fmt, exp1, exp2, ...)` - Осуществляет форматированную печать (вывод) в строку, идентично PRINTF.

`split(s, array, sep)` - Помещает поля строки s в массив array и возвращает число заполненных

элементов массива. Если указан sep, то при анализе строки он понимается как разделитель.

Операции отношения awk `X == Y` - X равно Y?

`X != Y` - X не равно Y?

`X > Y` - X больше чем Y?

`X >= Y` - X больше чем или равно Y?

`X < Y` - X меньше чем Y?

`X <= Y` - X меньше чем или равно Y?

`X ~ Re` - X совпадает с регулярным выражением Re?

`X !~ Re` - X не совпадает с регулярным выражением Re?

Старшинство операций в awk

Группа	Операции
1	<code>= += -= *= /= %=</code>
2	<code> </code>
3	<code>&&</code>
4	<code>> >= < <= == != ~ !~</code>
5	Строка конкатенации «x» «y» становится «xy»
6	<code>+ -</code>
7	<code>* / %</code>
8	<code>++ --</code>

Стандартные переменные

ARGC - число аргументов в командной строке; ARGV - массив с аргументами командной строки; FILENAME - строка текущего файла ввода; FNR - номер текущей записи в текущем файле; FS - разделитель полей ввода;

NF - число полей в текущей записи;

NR - номер текущей записи;

OFMT - формат вывода чисел (по умолчанию % 6g); OFS - разделитель полей ввода (по умолчанию пробел); ORS - Разделитель выводимых записей (по умолчанию новая строка); RS - Разделитель полей ввода (по умолчанию новая строка).

2.3 Список команд Shell

date - вывод даты; who - вывод пользователей; who am i - вывод собственного имени; exit - выход из системы (для передачи кода завершения); mail

– почта;
 write – передача сообщения другому пользователю; man – информация о команде;
 news – новости;
 ed – текстовый редактор (a/... ./w имя/ctrl-d) ls – перечень имен файлов в каталоге; ls -t – перечень файлов во временном порядке; ls -l – перечень файлов в полном виде; ls -li – перечень файлов в расширенном виде; cat – распечатка файла (saОимя – создание файла);
 pr – распечатка по 66 строк; mv – перенос файла; cp – копирование файла; rm – удаление файла; ln – назначение связи; rmdir – удалить каталог; mkdir – создать каталог; pwd – определение своего рабочего каталога; cd – смена каталога; wc – подсчет числа строк, слов и символов; tail +n – вывод файла начиная со строки с номером n; str – поиск различий между файлами (до первого различия); diff – поиск всех различий; echo – вывод строки (' – результата, ' – команды); echo \$? – выдача кода завершения команды (0, 1, 2); wait – ждать завершения всех процессов;
 kill – убить процесс (kill -9 #_процесса);
 ps – список процессов; nohup – выполнение команды после отключения (nohup кмд&); nice – запуск с пониженным приоритетом (nice кмд&); at – запуск в определенное время (at команды ctrl-d); export – сообщение интерпретатору о использовании переменных; sh – переход в порожденный shell; du – определение занятого пространства; df – свободное пространство диска; chmod – смена права доступа; mesg – (n – запрет, y – разрешение) сообщения; sleep – пауза; set – показать все ранее определенные переменные; set " – установить значение переменной; time – информация о времени выполнения команды; uname – информация о системе (uname -a – полная); read – присваивает переменной значение последующей строки;
 touch – заменяет время модификации файла на настоящее;
 for – цикл (for i in список/ do команды/ done); case – выбор (case слово in/шаблон) команды ;;/esac); if – условие (if команда / then команды, если условие верно / else команды, если условие ложно/fi; while – цикл (while команда/do тело цикла, выполняется пока команда возвращает истина/done; until – цикл (аналог while, но ждет ложь);
 trap – последовательность действий выполняемая при прерывании (trap 'rm -f \$old; exit 1' 1 2 15), где 0 – выход из интерпретатора
 1 – отбой
 2 – прерывание (DEL)
 3 – останов (ctrl-\; вызывает распечатку содержимого памяти программы)
 9 – уничтожение
 15 – окончание выполнения.

Встроенные переменные интерпретатора

\$# – число аргументов;
 \$* – все аргументы, передаваемые интерпретатору (\$@);
 \$- – флаги, передаваемые интерпретатору;
 \$? – возвращение значения последней выполненной команды;
 \$\$ – номер процесса интерпретатора;
 \$! – номер процесса последней команды, запущенной с &;

Правила сопоставления шаблонов в интерпретаторе

* – задание любой строки, в том числе и пустой.
 ? – любой одиночный символ;
 "... " – задает в точности ...; ""("") защищает от спецсимволов;
 \c – задает с буквально;
 a|b – только для выражения выбора, a или b.

Значения переменных

\$var – значение var;
 \${var-thing} – значение var, если оно определено, в противном случае thing;

`${var=thing}` – значение `var`, если `var` не определено, то присваивается значение `thing`;
`${var?строка!}` – если `var` определено – `$var`, в противном случае выводится строка и инт. прекращает работу;

`${var+thing}` – `thing`, если `$var` определено, в противном случае ничего.

Метасимволы

`|` – конвейер (связь выходного потока одной программы с выходным потоком другой);
`&` – асинхронный запуск;
`;` – последовательное выполнение;
`>` – помещение выходного потока;
`>>` – добавление выходного потока;
`*` – любая строка;
`?` – любой символ;
`[ccc]` – задает любой символ из `[ccc]` в имени файла;
`'...'` – иницирует выполнение команды;
`()` – иницирует выполнение команды в порожденном shell;
`{ }` – иницирует выполнение команды в текущем shell;
`$1` – заменяется аргументом командного файла;
`$var` – значение переменной `var` в программе на языке shell; `${var}` – значение `var`;
`\` – перевод строки;
`'...'` – непосредственное использование;
`"..."` – непосредственное использование после того, как `'...'` и `\` будут интерпретированы;
`#` – оставшаяся строка – комментарий; `p1&& p2` – выполнить `p1`, в случае успеха `p2`; `p1||p2` – выполнить `p1`, в случае неудачи `p2`; `2>file` – переключить поток диагностики на файл; `2>&1` – поместить стандартный поток диагностики в выходной поток;
`1>&2` – добавление выходного потока к стандартному потоку диагностики.

2.4 Варианты заданий на выполнение

Вариант 1. Разработать программу, отправляющую почту (содержимое файла) группе пользователей, выбираемых из общего списка (хранящегося в другом файле) в интерактивном режиме. Например, вы отвечаете "Y" для тех, кому надо посылать, "N" – не надо, "Q" – конец выбора.

Вариант 2. Разработать программу, выводящую через определенный интервал времени информацию о пользователях в системе: кто вошел, кто вышел.

Вариант 3. Разработать программу, выполняющую в зависимости от ключа один из 3-х вариантов работы:

- с ключом `/n` дописывает в начало указанных текстовых файлов строку с именем текущего файла; - с ключом `/b` создает резервные копии указанных файлов; - с ключом `/d` удаляет указанные файлы после предупреждения.

Вариант 4. Разработать программу создающую, копирующую или удаляющую файл, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) `/n` , `/c` , `/d`.

Вариант 5. Разработать программу, добавляющую вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей (замещаемого параметра) `/b` /`e`).

Вариант 6. Разработать программу, регистрирующую время своего запуска в файле протокола `gun.log` и автоматически запускающую некоторую программу (например, антивирусную и т. п.) по пятницам или 13 числам.

Вариант 7. Разработать программу, копирующую произвольное число файлов, заданных аргументами из текущего каталога в указываемый каталог.

Вариант 8. Разработать программу, которая в интерактивном режиме могла бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.

Вариант 9. Разработать программу, которая бы запускала бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает.

Вариант 10. Разработать программу, которая получала бы в качестве параметра какой-либо символ и в зависимости от второго параметра вырезала или сохраняла в заданном файле все строки, начинающиеся на этот символ.

Вариант 11. В некотором файле храниться список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос - копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да» то программа копирует файл.

Вариант 12. Разработать программу, которая бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования.

Вариант 13. Разработать программу (аналог команды `wc`), которая бы получала бы в качестве аргумента имя текстового файла и выводила на экран информацию о том, сколько символов, слов и строк в текстовом файле.

Вариант 14. Разработать программу (аналог команды `tail`), которая печатает конец файла. По умолчанию - 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.

Вариант 15. Разработать программу, которая склеивала бы текстовые файлы, заданные в качестве аргументов, и сортировала бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Вариант 16. Разработать программу, которая формировала бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные).

Вариант 17. Разработать программу, разбирающую содержимое письма (файл или входной поток), выделяющую заголовок письма с адресом отправителя (поля `From:` или `From`) и отправляющую содержимое письма без заголовка обратно отправителю.

Вариант 18. Разработать программу, которая изменяет текстовый файл так, что четные и нечетные строки меняются местами.

Вариант 19. Разработать программу, которая бы в зависимости от параметров, строила бы выборку по какому бы условию (числовые значения) из табличного файла.

Вариант 20. Разработать программу, которая инвертирует текстовый файл или его строки.

Практическое занятие №26

Процессы в ОС QNX

3.1 Цель работы

Познакомиться с визуальным интерфейсом ОС QNX, возможностями различных функций управления процессами и изучить принципы работы с компилятором C в среде ОС QNX.

3.2 Создание процессов

На самом высоком уровне абстракции система состоит из множества процессов. Каждый процесс ответственен за обеспечение служебных функций определенного характера. Разделение объектов на множество процессов дает ряд преимуществ:

1. возможность декомпозиции задачи и модульной организации решения;
2. удобство сопровождения;
3. надежность.

Любой поток может осуществить запуск процесса. Однако необходимо учитывать ограничения, вытекающие из основных принципов защиты.

Из курса «Операционные системы» вы уже должны быть знакомы с возможностями запуска процессов из командного интерпретатора (*shell*). Например:

\$ program1 – запуск приложения в режиме переднего плана; *\$ program2 &* – запуск приложения в режиме заднего плана.

\$ nice program3 – запуск приложения с заниженным приоритетом.

Обычно разработчиков программного обеспечения не заботит тот факт, что командный интерпретатор создает процессы – это просто подразумевается. Однако в большой мультипроцессорной системе вы можете пожелать, чтобы одна главная программа выполняла запуск всех других процессов вашего приложения.

Рассмотрим некоторые функции, которые ОС QNX использует для запуска других процессов:

- *system()*;
- *fork()*;
- *vfork()*;
- *exec()*; • *spawn()*.

Какую из этих функций применять, зависит от двух требований: переносимости и функциональности.

system() – самая простая функция; она получает на вход одну командную строку, такую же,

которую вы набрали бы в ответ на подсказку командного интерпретатора, и выполняет ее.

Фактически, для обработки команды функция *system()* запускает копию командного интерпретатора.

fork() – порождает процесс, являющийся его точной копией. Новый процесс выполняется в том

же адресном пространстве и наследует все данные порождающего процесса.

Между тем, родительский и дочерний процесс имеют различные идентификаторы процессов, так как в системе не может быть двух процессов с одинаковыми идентификаторами. Есть и еще одно отличие, это значение, возвращаемое функцией *fork()*. В дочернем процессе функция возвращает ноль, а в родительском процессе идентификатор дочернего процесса. Пример, использования функции *fork()*:

```
printf("PID родителя равен %d\n", getpid()); if (child_pid = fork()) {  
    printf("^ТО родитель, PID сына %d\n", child_pid); } else {  
    printfC^!^ сын, PID %d\n", getpid());  
}
```

vfork() – так же порождает процесс. В отличие от функции *fork()* она позволяет существенно сэкономить на ресурсах, поскольку она делает разделяемое адресное пространство родителя. Функция *vfork()* создает дочерний процесс, а затем приостанавливает родительский до тех пор, пока дочерний процесс не вызовет функцию *exec()* или не завершится.

exec() – заменяет образ порождающего процесса образом нового процесса. Возврата управления из нормально отработавшего *exec()* не существует, т.к. образ нового процесса накладывается на образ порождающего процесса. В системах стандарта POSIX новые процессы обычно создаются без возврата управления порождающему процессу – сначала вызывается *fork()*, а затем из порожденного процесса – *exec()*.

spawn() – создает новый процесс по принципу «отец»- «сын». Это позволяет избежать использования примитивов *fork()* и *exec()*, что ускоряет обработку и является более эффективным средством создания новых процессов. В отличие от *fork()* и *exec()*, которые по определению создают процесс на том же узле, что и порождающий процесс, примитив *spawn()* может создавать процессы на любом узле сети.

3.3 Задание на выполнение

1. Познакомиться с интерфейсом ОС QNX.
2. Изучить процедуру компиляции (компилятор командной строки *gcc*). Повторить стандартный ввод - вывод, разбор аргументов и переменных среды. Исследовать работу функций по работе с файлами языка C.
3. Написать программу, которая бы запускала в памяти еще один процесс и оставляла бы его работать в бесконечном цикле. При повторном запуске программа должна убирать запущенный ранее процесс из памяти (можно использовать *kill*).
4. Подготовиться к ответам на теоретическую часть лабораторной работы.
5. Посмотреть задание на вторую лабораторную работу, с целью вспомнить численные методы, чтобы быть готовым к выполнению следующей лабораторной работы.

Практическое занятие №27

Потоки в ОС QNX

4.1 Цель работы

Познакомиться возможностями функций управления потоками в ОС QNX.

4.2 Создание потоков

Процесс может содержать один или несколько потоков. Число потоков варьируется. Один разработчик программного обеспечения, используя только единственный поток, может реализовать те же самые функциональные возможности, что и другой, используя пять потоков. Некоторые задачи сами по себе приводят к многопоточности и дают относительно простые решения, другие в силу своей природы, являются однопоточными, и свести их к многопоточной реализации достаточно трудно.

Любой поток может создать другой поток в том же самом процессе. На это не налагается никаких ограничений (за исключением объема памяти). Как правило, для этого применяется функция POSIX *pthread_create()*:

```
#include <pthread.h> int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine) (void
*),
void *arg);
```

thread – указатель на *pthread_t*, где храниться идентификатор потока; *attr* – атрибутивная запись; *start_routine* – подпрограмма, с которой начинается поток; *arg* – параметр, который передается подпрограмме *start_routine*.

Первые два параметра необязательные вместо них можно передавать NULL.

Параметр *thread* можно использовать для хранения идентификатора вновь создаваемого потока.

Пример однопоточной программы. Предположим, что мы имеем программу, выполняющую алгоритм трассировки луча. Каждая строка раstra не зависит от остальных. Это обстоятельство (независимость строк раstra) автоматически приводит к программированию данной задачи как многопоточной.

```
int main ( int argc, char **argv)
{ int x1;
... // Выполнить инициализации
109
for (x1 = 0; x1 < num_x_lines; x1++)
{
do_one_line (x1);
}
. // Вывести результат
}
```

Здесь видно, что программа независимо по всем значениям *x1* рассчитывает необходимые растровые строки.

Пример первой многопоточной программы. Для параллельного выполнения функции *do_one_line (x1)* необходимо изменить программу следующим образом:

```
int main ( int argc, char **argv)
{ int x1;
... // Выполнить инициализации
for (x1 = 0; x1 < num_x_lines; x1++) { pthread_create
(NULL, NULL, do_one_line,
(void *) x1);
}
. // Вывести результат
}
```

Пример второй многопоточной программы. В приведенном примере непонятно, когда нужно выполнять вывод результатов, так как приложение запустило массу потоков, но не знает когда они завершаться. Можно поставить задержку выполнения программы (sleep 1), но это не будет правильно. Для этого лучше использовать функцию pthread_join().

Есть еще один минус у приведенной выше программы, если у нас много строк в изображении не факт, что все созданные потоки будут функционировать параллельно, как правило, процессоров системе, гораздо меньше. Для этого лучше модифицировать программу так, чтобы запускалось столько потоков, сколько у нас процессоров в системе.

```
int num_lines_per_cpu; int num_cpus;
int main (int argc, char **argv)
{
    int cpu; pthread_t *thread_ids;
    // Выполнить инициализации
    // Получить число процессоров num_cpus = _syspage_ptr->num_cpu;
    thread_ids = malloc (sizeof (pthread_t) * num_cpus); num_lines_per_cpu = num_x_lines
/ num_cpus;
    for (cpu = 0; cpu < num_cpus; cpu++)
    { pthread_create (&thread_ids [cpu], NULL, do_one_batch,
(void *) cpu);
    }
    // Синхронизировать с завершением всех потоков for (cpu = 0; cpu < num_cpus;
cpu++)
    {
        pthread_join (thread_ids [cpu], NULL); }
    110
    ... // Вывести результат
    }
    void *do_one_batch (void *c)
    {
        int cpu = (int) c; int x1;
        for (x1 = 0; x1 < num_lines_per_cpu; x1++) { do_one_line(x1 + cpu
num_lines_per_cpu); }
    }
}
```

4.3 Задание на выполнение

1. Выполнить задание согласно варианту.
2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Вариант 1. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки обменом «пузырьком».

Вариант 2. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки простых вставок.

Вариант 3. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки выбором.

Вариант 4. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную

обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом прямоугольников.

Вариант 5. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом трапеций.

Вариант 6. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом Симпсона.

Вариант 7. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод оптимизации функций (min, max) - золотого сечения.

Вариант 8. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод оптимизации функций (min, max) - общего поиска.

Вариант 9. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод оптимизации функций (min, max) - дихотомии.

Вариант 10. Написать программу, которая принимает в качестве параметров набор имен файлов 111 данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод решения системы нелинейных уравнений - метод касательных.

Практическое занятие №28

Обмен сообщениями

5.1 Цель работы

Познакомиться с механизмами обмена сообщениями в ОС QNX.

5.2 Связь между процессами посредством сообщений

Механизм передачи межпроцессных сообщений занимается пересылкой сообщений между процессами и является одной из важнейших частей операционной системы, так как все общение между процессами, в том числе и системными, происходит через сообщения. Сообщение в QNX - это последовательность байтов произвольной длины (0-65535 байтов) и произвольного формата.

Протокол обмена сообщениями выглядит таким образом: задача блокируется для ожидания сообщения. Другая же задача посылает первой сообщение и при этом блокируется сама, ожидая ответа. Первая задача становится разблокированной, обрабатывает сообщение и отвечает, разблокируя при этом вторую задачу.

Сообщения и ответы, пересылаемые между процессами при их взаимодействии, находятся в теле отправляющего их процесса до того момента, когда они могут быть приняты. Это значит, что, с одной стороны, уменьшается вероятность повреждения сообщения в процессе передачи, а с другой - уменьшается объем оперативной памяти, необходимый для работы ядра. Кроме того, уменьшается число пересылок из памяти в память, что разгружает процессор. Особенностью процесса передачи сообщений является то, что в сети, состоящей из нескольких компьютеров под управлением QNX, сообщения могут прозрачно передаваться процессам, выполняющимся на любом из узлов.

Передача сообщений служит не только для обмена данными между процессами, но, кроме того, является средством синхронизации выполнения нескольких взаимодействующих процессов.

Рассмотрим более подробно функции Send(), Receive() и Reply().

Использование функции Send(). Предположим, что процесс А выдает запрос на передачу сообщения процессу В. Запрос оформляется вызовом функции Send().

Send (pid, smsg, rmsg, smsg_len, rmsg_len)

Функция Send() имеет следующие аргументы:

- pid – идентификатор процесса-получателя сообщения (т.е. процесса В); pid – это идентификатор, посредством которого процесс опознается операционной системой и другими процессами;
- smsg – буфер сообщения (т.е. посылаемого сообщения);
- rmsg – буфер ответа (т.е. сообщения посылаемого в ответ); 109
- smsg_len – длина посылаемого сообщения;
- rmsg_len – максимальная длина ответа, который должен получить процесс А.

Обратите внимание на то, что в сообщении будет передано не более чем smsg_len байт и принято в ответе не более чем rmsg_len байт – это служит гарантией того, что буферы никогда не будут переполнены.

Использование функции Receive(). Процесс В может принять запрос Send(), выданный процессом А, с помощью функции Receive().

pid = Receive (0, msg, msg_len)

Функция Receive() имеет следующие аргументы:

- pid – идентификатор процесса, пославшего сообщение (т.е. процесса А);
- 0 – (ноль) указывает на то, что процесс В готов принять сообщение от любого процесса;
- msg – буфер, в который будет принято сообщение;
- msg_len – максимальное количество байт данных, которое может поместиться в приемном буфере.

В том случае, если значения `smsg_len` в функции `Send()` и `msg_len` в функции `Receive()` различаются, то количество передаваемых данных будет определяться наименьшим из них.

Использование функции `Reply()`. После успешного приема сообщения от процесса А процесс В должен ответить ему, используя функцию `Reply()`.

`Reply(pid, reply, reply_len)`

Функция `Reply()` имеет следующие аргументы:

- `pid` – идентификатор процесса, которому направляется ответ (т.е. процесса А);
- `reply` – буфер ответа;
- `reply_len` – длина сообщения, передаваемого в ответе.

Если значения `reply_len` в функции `Reply()` и `rmsg_len` в функции `Send()` различаются, то количество передаваемых данных определяется наименьшим из них.

Дополнительные возможности передачи сообщений. В системе QNX имеются функции, предоставляющие дополнительные возможности передачи сообщений, а именно:

- условный прием сообщений;
- чтение и запись части сообщения;
- передача составных сообщений.

Обычно для приема сообщения используется функция `Receive()`. Этот способ приема сообщений в большинстве случаев является наиболее предпочтительным.

Однако иногда процессу требуется предварительно «знать», было ли ему послано сообщение, чтобы не ожидать поступления сообщения в RECEIVE-блокированном состоянии. Например, процессу требуется обслуживать несколько высокоскоростных устройств, не способных генерировать прерывания и, кроме того, процесс должен отвечать на сообщения, поступающие от других процессов. В этом случае используется функция `Creceive()`, которая считывает сообщение, если оно становится доступным, или немедленно возвращает управление процессу, если нет ни одного отправленного сообщения.

ВНИМАНИЕ. По возможности следует избегать использования функции `Creceive()`, так как она позволяет процессу непрерывно загружать процессор на соответствующем приоритетном уровне.

5.3 Примеры обмена сообщениями

5.3.1 Клиент

Передача сообщения со стороны клиента осуществляется применением какой-либо функции из семейства `MsgSend()`.

Мы рассмотрим это на примере простейшей из них – `MsgSend()`:

```
110
```

```
include <sys/neutrino.h>
```

```
int MsgSend(int coid, const void *smsg, int sbytes, void *rmsg, int rbytes);
```

Для создания установки соединения между процессом и каналом используется функция `ConnectAttach()`, в параметрах которой задаются идентификатор процесса и номер канала.

```
#include <sys/neutrino.h> int ConnectAttach( uint32_t nd,
```

```
pid_t pid, int chid, unsigned index, int flags );
```

Пример:

Передадим сообщение процессу с идентификатором 77 по каналу 1:

```
#include <sys/neutrino.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h> #include <errno.h>
```

```
char *smsg = "Это буфер вывода"; char rmsg[200]; int coid; int main(void);
```

```
{
```

```
// Установить соединение coid = ConnectAttach(0, 77, 1, 0, 0); if (coid == -1)
```

```

    { fprintf(stderr, "Ошибка ConnectAttach к 0/77/1!\n"); perror(NULL);
    exit(EXIT_FAILURE);
    }
    // Послать сообщение if(MsgSend(coid, smsg, strlen(smsg)+1, rmsg, sizeof(rmsg)) == -
1)
    { fprintf(stderr, "Ошибка MsgSendNn"); perror(NULL); exit(EXIT_FAILURE);
    }
    if (strlen(rmsg) > 0)
    { printf("Процесс с ID 77 возвратил \"%s\"\n", rmsg);
    } exit(0); }

```

Предположим, что процесс с идентификатором 77 был действительно активным сервером, ожидающим сообщение именно такого формата по каналу с идентификатором 1.

После приема сообщения сервер обрабатывает его и в некоторый момент времени выдает ответ с результатами обработки. В этот момент функция `MsgSend()` должна вернуть ноль (0), указывая этим, что все прошло успешно.

Если бы сервер послал нам в ответ какие-то данные, мы смогли бы вывести их на экран с помощью последней строки в программе (с тем предположением, что обратно мы получаем корректную ASCIIZ-строку).

5.3.2 Сервер

Создание канала. Сервер должен создать канал – то, к чему присоединялся клиент, когда вызывал функцию `ConnectAttach()`.

Обычно сервер, однажды создав канал, бережет его «впрок». Канал создается с помощью ¹¹¹ функции `ChannelCreate()` и уничтожается с помощью функции `ChannelDestroy()`:

```

#include <sys/neutrino.h> int ChannelCreate(unsigned flags); int
ChannelDestroy(int chid);

```

Пока на данном этапе будем использовать для параметра `flags` значение 0 (ноль). Таким образом, для создания канала сервер должен сделать так:

```

int chid; chid = ChannelCreate (0);

```

Теперь у нас есть канал. В этом пункте клиенты могут подсоединиться (с помощью функции `ConnectAttach()`) к этому каналу и начать передачу сообщений. Сервер отрабатывает схему сообщений обмена в два этапа – этап «приема» (receive) и этап «ответа» (reply).

```

#include <sys/neutrino.h> int MsgReceive(int chid, void *rmsg, int rbytes, struct msg
info *info);

```

```

int MsgReply(int rvid, int status, const void *msg, int nbytes);

```

Для каждой буферной передачи указываются два размера (в случае запроса от клиента это `sbytes` на стороне клиента и `rbytes` на стороне сервера; в случае ответа сервера это `sbytes` на стороне сервера и `rbytes` на стороне клиента). Это сделано для того, чтобы разработчики каждого компонента смогли определить размеры своих буферов – из соображений дополнительной безопасности.

В нашем примере размер буфера функции `MsgSend()` совпадал с длиной строки сообщения.

Пример (Структура сервера):

```

#include <sys/neutrino.h> #include <sys/iomsg.h>
#include <errno.h> #include <stdio.h>
#include <process.h>
void main(void)
{ int rvid; int chid; char message [512];
// Создать канал chid = ChannelCreate(0);
// Выполняться вечно – для сервера это обычное дело while (1) {

```

```

// Получить и вывести сообщение rcvid=MsgReceive(chid, message, sizeof(message),
NULL);
printf ("Получил сообщение, rcvid %X\n", rcvid); printf ("Сообщение такое:
\"%s\\",\n", message);
// Подготовить ответ – используем тот же буфер strcpy (message, "Это ответ");
MsgReply (rcvid, EOK, message,
sizeof (message));
}
}

```

Как видно из программы, функция MsgReceive() сообщает ядру том, что она может обрабатывать сообщения размером вплоть до sizeof(message) (или 512 байт).

Наш клиент (представленный выше) передал только 28 байт (длина строки).

5.3.3 Определение идентификаторов узла, процесса и канала (ND/PID/CHID) нужного сервера

Для соединения с сервером функции ConnectAttach() необходимо указать дескриптор узла (Node

Descriptor – ND), идентификатор процесса (process ID – PID), а также идентификатор канала (Channel ID – CHID).

Если один процесс создает другой процесс, тогда это просто – вызов создания процесса возвращает идентификатор вновь созданного процесса. Создающий процесс может либо передать собственные PID и CHID вновь созданному процессу в командной строке, либо вновь созданный процесс может вызвать функцию *getppid()* для получения идентификатора родительского процесса, и использовать некоторый «известный» идентификатор канала.

```
#include <process.h> pid_t getpid( void );
```

Вопрос: «Как сервер объявляет о своем местонахождении?»

Существует множество способов сделать это; мы рассмотрим только три из них, в порядке возрастания «элегантности»:

1. Открыть файла с известным именем и сохранить в нем ND/PID/CHID. Такой метод является традиционным для серверов UNIX, когда сервер открывает файл (например, /etc/httpd.pid), записывает туда свой идентификатор процесса в виде строки ASCII и предполагают, что клиенты откроют этот файл, прочитают из него идентификатор.

2. Использовать для объявления идентификаторов ND/PID/ CHID глобальные переменные. Такой способ обычно применяется в многопоточных серверах, которые могут посылать сообщение сами себе. Этот вариант по самой своей природе является очень редким.

3. Занять часть пространства имен путей и стать администратором ресурсов.

5.4 Задание на выполнение

1. Необходимо создать два приложения - клиент и сервер, которые бы обменивались между собой сообщениями, используя стандартные механизмы операционной системы QNX/Neutrino2.

При реализации клиент-серверных приложений требуется предусмотреть возможность работы с сервером нескольких клиентов.

В клиенте имеется возможность консольного ввода команд, которые должны обрабатываться сервером. Сервер должен обрабатывать не менее трех команд, посылаемых клиентом:

- help – помощь по командам;
- list /dir – получение содержимого указанного каталога;
- get filename – передача клиенту указанного файла.

Также сервер должен отвечать клиенту, если тот послал ему не верную команду.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Практическое занятие №29

Таймер и периодические уведомления

6.1 Цель работы

Познакомиться с механизмом управления временем ОС QNX – системным таймером.

6.2 Управление таймером

В QNX управление временем основано на использовании системного таймера. Этот таймер содержит текущее координатное универсальное время (UTC) относительно 0 часов 0 минут 0 секунд 1 января 1970 г. Для установки местного времени функции управления временем используют переменную среды TZ.

Процесс может создать один или несколько таймеров. Таймеры могут быть любого поддерживаемого системой типа, а их количество ограничивается максимально допустимым количеством таймеров в системе.

Функция создания таймера позволяет задавать следующие типы механизма ответа на события:

- перейти в режим ожидания до завершения. Процесс будет находиться в режиме ожидания начиная с момента установки таймера до истечения заданного интервала времени;
- оповестить с помощью rgoxu. Rgoxu используется для оповещения процесса об истечении времени ожидания;
- оповестить с помощью сигнала. Сформированный пользователем сигнал выдается процессу по истечении времени ожидания.

Вы можете задать таймеру следующие временные интервалы:

- *абсолютный*. Время относительно 0 часов, 0 минут, 0 секунд, 1 января 1970 г.;
- *относительный*. Время относительно значения текущего времени.

Можно также задать повторение таймера на заданном интервале. Например, вы установили таймер на 9 утра завтрашнего дня. Его можно установить так, чтобы он срабатывал, каждые пять минут после истечения этого времени. Можно также установить новый временной интервал существующему таймеру.

Результат этой операции зависит от типа заданного интервала:

- для абсолютного таймера новый интервал замещает текущий интервал времени;
- для относительного таймера новый интервал добавляется к оставшемуся временному интервалу.

При использовании множества параллельно работающих таймеров 114 – например, когда необходимо активизировать несколько потоков в различные моменты времени – ядро ставит запросы в очередь, отсортировав таймеры по возрасту их времени истечения (в голове очереди при этом окажется таймер с минимальным временем истечения). Обработчик прерывания будет анализировать только переменную, расположенную в голове очереди.

При использовании периодических и однократных таймеров у вас появляется выбор:

- послать импульс;

- послать сигнал;
- создать поток.

В данной лабораторной работе будем использовать вторую возможность.

Пример 1. Данная программа запускает в цикле ожидания на 10 секунд и прерывает это ожидание с помощью сигнала.

```
#include <unistd.h>
#include <stdio.h>
#include <sys/siginfo.h>
#include <sys/neutrino.h>
#include <signal.h>
```

```

#include <time.h>
#include <errno.h>
#include <unistd.h> #include <stdlib.h>
int main(void)
{
    struct itimerspec timer; // структура с описанием
    //таймера
    timer_t timerid; // ID таймера
    extern void handler(); //Обработчик таймера struct sigaction act;
    //Структура описывающая
    //действие
    //по сигналу sigset_t set; //Набор сигналов нам
    //необходимый //для таймера
    sigemptyset( &set ); //Обнуление набора sigaddset( &set, SIGALRM);
    //Включение в набор //сигнала
    //от таймера act.sa_flags = 0;
    act.sa_mask = set;
    act.sa_handler = &handler; // Вешаем обработчик
    // на действие sigaction( SIGALRM, &act, NULL);
    //Зарядить сигнал,
    // присваивание структуры
    // для конкретного сигнала
    // (имя сигнала,
    // структура-действий)
    //Создать таймер
    if (timer_create (CLOCK_REALTIME, NULL, &timerid) == -1) { fprintf (stderr, "%s:
ne udalos timer %d\n", "TM", errno);
    }
    // Данный макрос для сигнала SIGALRM не нужен // Его необходимо вызывать
    // для пользовательских сигналов
    // SIGUSR1 или SIGUSR2. Функция timer_create() // в качестве второго параметра
должна 115
    // использовать &event.
    // SIGEV_SIGNAL_INIT(&event, SIGALRM);
    timer.it_value.tv_sec= 3; //Ввести таймер
    //на 3 секунды timer.it_value.tv_nsec= 0;
    timer.it_interval.tv_sec= 3;
    //Перезагружать //таймер timer.it_interval.tv_nsec= 0; // через 3 секунды
timer_settime (timerid, 0, &timer, NULL); //Включить таймер
    for (;;)
    {
        sleep(10); // Спать десять секунд.
        // Использование таймера задержки
        printf("More time!\n");
    } exit(0); }
    void handler( signo )
    {
        //Вывести сообщение в обработчике. printf( "Alarm clock ringing!!!\n");
        // Таймер заставляет процесс проснуться. }
    Здесь it_value и it_interval принимает одинаковые значения. Такой таймер
    сработает один раз (с задержкой it_value), а затем будет циклически перезагружаться с
    задержкой it_interval.

```

Оба параметра *it_value* и *itinterval* фактически являются структурами типа **struct timespec** – еще одного POSIX-объекта. Эта структура позволяет вам обеспечить разрешающую способность на уровне долей секунд. Первый ее элемент, *tv_sec*, – это число секунд, второй элемент, *tv_nsec*, – число наносекунд в текущей секунде. (Это означает, что никогда не следует устанавливать параметр *tv_nsec* в значение, превышающее 1 миллиард – это будет подразумевать смещение на более чем 1 секунду).

Пример 2:

```
t_value.tv_sec = 5; it_value.tv_nsec = 500000000; it_interval.tv_sec = 0;
it_interval.tv_nsec = 0;
```

Это сформирует однократный таймер, который сработает через 5,5 секунды. (5,5 секунд складывается из 5 секунд и 500,000,000 наносекунд.)

Мы предполагаем здесь, что этот таймер используется как относительный, потому что если бы это было не так, то его время срабатывания уже давно бы его истекло (5.5 секунд с момента 00:00 по Гринвичу, 1 января 1970).

Пример 3:

```
it_value.tv_sec = 987654321; it_value.tv_nsec = 0; it_interval.tv_sec = 0; it_interval.tv_nsec = 0;
```

Данная комбинация параметров сформирует однократный таймер, который сработает в четверг, 19 апреля 2001 года

В программе могут быть использованы следующие структуры и функции:

TimerCreate(), TimerCreate_r() – Функция создание таймера

```
#include <sys/neutrino.h> int TimerCreate( clockid_t id,
116
```

```
const struct sigevent *event );
int TimerCreate_r( clockid_t id,
const struct sigevent *event );
```

Обе функции идентичны, исключение составляют лишь способы обнаружения ими ошибок:

- TimerCreate() – если происходит ошибка, то возвращается значение -1.
- TimerCreate_r() – при возникновении ошибки ее конкретное значение возвращается из секции Errors.

Struct sigevent – Структура, описывающая событие таймера

```
#include <sys/siginfo.h> union sigval { int sival_int;
void *sival_ptr;
};
```

Файл <sys/siginfo.h> определяет также некоторые макросы для более облегченной инициализации структуры *sigevent*. Все макросы ссылаются на первый аргумент структуры *sigevent* и устанавливают подходящее значение *sigev_notify* (уведомление о событии).

SIGEV_INTR – увеличить прерывание. В этой структуре не используются никакие поля.

Инициализация макроса: *SIGEV_INTR_INIT(event)*

SIGEV_NONE – Не посылать никаких сообщений. Также используется без полей.

Инициализация: *SIGEV_NONE_INIT(event)*

SIGEV_PULSE – посылать периодические сигналы. Имеет следующие поля:

int sigev_coid – ID подключения. По нему происходит связь с каналом, откуда будет получен

сигнал; *short sigev_priority* – установка приоритета сигналу;

short sigev_code – интерпретация кода в качестве манипулятора сигнала. *sigev_code* может быть любым 8-битным значением, чего нужно избегать в программе. Значение *sigev_code* меньше нуля приводит к конфликту в ядре.

Инициализация макроса: *SIGEV_PULSE_INIT(event, coid, priority, code, value)*

SIGEV_SIGNAL – послать сигнал процессу. В качестве поля используется: `int sigev_signo` – повышение сигнала. Может принимать значение от 1 до -1. Инициализация макроса:

```
SIGEV_SIGNAL_INIT( event, signal )
```

SignalAction(), SignalAction_r() // функция определяет действия для сигналов.

```
#include <sys/neutrino.h>
```

```
int SignalAction( pid_t pid, void (*sigstub)(), int signo,
```

```
const struct sigaction* act, struct sigaction* oact );
```

```
int SignalAction_r( pid_t pid, void* (sigstub)(), int signo, const struct sigaction* act,  
struct sigaction* oact );
```

Все значения ряда сигналов идут от `_SIGMIN` (1) до `_SIGMAX` (64).

Если `act` не `NULL`, тогда модифицируется указанный сигнал. Если `oact` не `NULL`, то предыдущее действие сохраняется в структуре, на которую он указывает. Использование комбинации `act` и `oact` позволяет запрашивать или устанавливать (либо и то и другое) действия сигналу. Структура `sigaction` содержит следующие параметры:

- `void (*sa_handler)()` – возвращает адрес манипулятора сигнала или действия для неполученного сигнала, действие- обработчик.
- `void (*sa_sigaction) (int signo, siginfo_t *info, void *other)` – возвращает адрес манипулятора сигнала или действия для полученного сигнала.
- `sigset_t sa_mask` – дополнительная установка сигналов для изолирования (блокирования) функций, улавливающих сигнал в течение исполнения.
- `int sa_flags` – специальные флаги, для того, чтобы влиять на действие сигнала. Это два флага: `SA_NOCLDSTOP` и `SA_SIGINFO`.

- `SA_NOCLDSTOP` используется только когда сигнал является дочерним 117 (`SIGCHLD`). Система не создает дочерний сигнал внутри родительского, он останавливается через `SIGSTOP`.

- `SA_SIGINFO` сообщает Neutrino поставить в очередь текущий сигнал. Если установлен флаг `SA_SIGINFO`, сигналы ставятся в очередь и все передаются в порядке очередности.

Добавление сигнала на установку:

```
#include <signal.h> int sigaddset( sigset_t *set, int signo );
```

Функция ***sigaddset()*** – добавляет `signo` в `set` по указателю. Присвоение сигнала набору. ***sigaddset()*** возвращает – 0, при удачном исполнении; -1, в случае ошибки;

Функция ***sigemptyset()*** – обнуление набора сигналов

```
#include <signal.h> int sigemptyset( sigset_t *set );
```

Возвращает – 0, при удачном исполнении; -1, в случае ошибки.

6.3 Задание на выполнение

1. Модифицировать программу созданную в третьей лабораторной работе, так чтобы в клиенте работали два таймера с разной периодичностью, например. 3 и 4,44 секунды. При срабатывании каждого таймера серверу посылался бы сигнал с номером сигнала таймера.
2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Практическое занятие №30

Среда визуальной разработки программ PHOTON APPLICATION BUILDER - PHAB

7.1 Цель работы

Ознакомление со средой визуальной разработки программ и построение пробного приложения в Phab (Photon Application Builder).

7.2 Основы работы с Phab

Для запуска среды Phab необходимо выбрать Launch=>Development=>Phab.

После запуска идем в пункты меню File=>New и в окне диалога выбираем тип проекта. Далее сохраняем проект под некоторым именем в своей домашней директории

Компиляция, компоновка и запуск. Осуществляются следующим образом:

1. Application=>Generate;
2. Выбрать платформу gcc;
3. Выбрать Make - создание исполняемого модуля;
4. Выбрать Run - запуск приложения.

Запустите приложение из терминального режима независимо от Phab.

Для того, что бы узнать какие объекты, функции и сообщения необходимы для построения приложения, используйте Launch=>HelpViewer. Ключевым словом может служить «widget».

Компоновка формы:

Чтобы создать рабочую форму с полями ввода и кнопкой, нужно сделать следующее:

1. Разместить элементы на форме.
2. Задать им уникальные имена.
3. Создать действие на нажатие кнопки:
 - a. Дать имя компилируемому модулю, в котором будет находиться обработчик кнопки.
 - b. Применить настройки.
 - c. Создать новый модуль.
4. Написание обработчика кнопки делается непосредственно в созданном модуле.

Функции, используемые для работы с сообщениями:

PtGetResource//взять данные по ресурсу из компоненты формы, например, из поля для ввода текста

изъять сам текст. 119

#define PtGetResource(widget, type, value, len) ...

widget - название ресурса (в данном случае - название поля, компоненты, в которую вводится

сообщение, посылаемое клиентом серверу); type - тип ресурса, например Pt_ARG_COLOR, Pt_ARG_TXT.

value - адрес, то есть куда отправлять сообщение, либо в какую переменную записать. len - определяется в зависимости от типа ресурса, здесь это длина посылаемого сообщения.

Для того, чтобы взять текст, посланный сервером клиенту в ответ на его сообщение, и поместить в окно редактирования ввода, необходимо использовать функцию

SetResource//установить ресурс для данного элемента формы (например, для поля ввода текста) #define PtSetResource(widget, type, value, len) ...

Пример:

PtWidget_t *widget;

PtSetResource(widget, Pt_ARG_FILL_COLOR, Pg_BLUE, 0);

Обе функции возвращают значение 0 при удачной работе и -1 при возникновении ошибки.

7.3 Задание на выполнение

1. Построить пробные приложения в Phab (Photon Application Builder).

Создать клиентское приложение, использующее графический интерфейс: 2 поля для ввода текста и кнопка отправки сообщения. Это приложение должно получать от пользователя текст через поле ввода и отправлять его по нажатию кнопки серверу в виде сообщения.

Сервер должен получать сообщение и отвечать ровно через 4,75 секунды. Ответное сообщение сервера должно приходить во второе поле формы.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Практическое занятие №31

Улучшение навыков программирования

8.1 Цель работы

Осмысление знаний о механизмах и ресурсах ОС QNX и с получение дополнительного опыта программирования в среде ОС QNX.

8.2 Задания на выполнение

Выполняется в течении двух лабораторных работ.

Вариант 1. Система безопасности летательного аппарата

Описание: Система должна следить за температурой носовой части, передней кромки левого и правого крыла. Всего три датчика температуры. Датчик носовой части должен опрашиваться с частотой 4Гц, датчики крыльев - 2Гц. Датчик возвращает значение температуры в диапазоне 0..65535K.

Задание: Написать программы сервера, моделирующие датчики и клиента - системы безопасности. Пусть значения температуры изменяются по закону косинуса (в случае отсутствия библиотеки тригонометрических функций, следует реализовать функцию косинуса с помощью разложения ряда) в заданном диапазоне.

Программа-клиент должна осуществлять опрос серверов и выводить на экран значение температуры в шесть столбцов (временная отметка, температура). Предусмотреть возможность отказа датчика, клиент не должен при этом блокироваться. Вместо отказавшего датчика в столбце должна выводиться -1.

При запуске должно быть три процесса сервера и один процесс клиента.

Смоделировать отказ датчика можно путем уничтожения одного или нескольких процессов серверов (kill). Датчик считается потерянным, если он не ответил на два опроса подряд. Но датчик может восстановить свою работу. Моделируется запуском процесса-сервера.

Опции: Значения температуры выводятся разными цветами в зависимости от диапазона температуры.

0-256 - фиолетовый 257-512 - синий

.. - 65535 - красный

121

Вариант 2. Базовая станция сотовой связи

Описание: Процессы моделируют базовые станции сотовой связи (БСС). Станция «ведет» не более 64 терминальных устройств мобильных телефонов (МТ).

Каждый МТ регистрируется на БСС. Моделирующий процесс при запуске оповещает станцию о номере своего процесса и номере телефона. Это пока не вызов, это - регистрация.

При вызове вызывающий МТ должен сообщить БСС номер вызываемого абонента. Станция ищет данный номер среди зарегистрированных, если таковой находится, то станция разрешает передавать данные.

Задание: Написать программы сервера-БСС и клиента- МТ. Клиент должен узнавать номер процесса БСС через пространство имен (см. руководство по QNX/Neutrino). Далее клиент отправляет импульс (или пару импульсов последовательно) в которых оповещает сервер о номере своего процесса и номере телефона. Вызов осуществляется импульсом специального формата (определить самостоятельно). По этому вызову сервер отыскивает абонента и организует канал связи между абонентами.

По окончании связи станция должна определить, сколько продолжался сеанс. В случае если пытается зарегистрироваться

65-ый по счету МТ. То сервер должен ответить соответствующим импульсом, по которому МТ «поймет», что в доступе отказано.

Вариант 3. Сетевой морской бой

Описание: Для игры в морской бой, запускаются две программы, которые представляют собой графические окна с двумя матрицами-полями 5x5. Одно поле противника, другое свое. В окне есть кнопки, нажатие на которые реализуют функции: подключиться к серверу, расставить корабли перед боем (по пять кораблей на поле), начать игру, сдаться. После начала игры пользователи поочередно делают выстрелы по полям врага, как только, у кого-либо потоплены все корабли, выдается сообщение, что бой закончен: Вы проиграли/выиграли.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер ждет, когда к ней подключаться два клиента. Далее сервер ожидает, когда клиенты проинициализируют свои игровые поля, информация о расположении кораблей храниться на сервере. После инициализации своих игровых полей, любой из клиентов может послать серверу сигнал о начале игры. Сервер случайным образом, выбирает игрока, который начинает первый ход, далее идет игра по правилам морского боя. Сервер получает информацию о выстреле, проверяет его результативность и отвечает клиентам, которые делают отметку в окне на игровом поле. Если у какого-либо клиента потоплены все корабли, то сервер прекращает игру и выдает сообщение о результате игры клиентам.

После окончания игры клиенты могут вновь без перезапуска программы начать игру, проинициализировав игровые поля.

Описание: Для игры в крестики-нолики, запускаются две программы, которые представляют собой графические окна с матрицей 3x3. В окне есть кнопки, нажатие на которые реализуют функции: подключиться к серверу, начать игру, сдаться. После начала игры пользователи поочередно делают ходы, как только, кто-либо проиграл, выдается сообщение, что игра закончена: Вы проиграли/выиграли.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер ждет, когда к ней подключаться два клиента. Далее сервер случайным образом, выбирает игрока, который начинает первый ход и его символ (X или O), далее идет игра по обычным правилам. Сервер получает информацию о ходе, проверяет его результативность и отвечает клиентам, которые делают отметку в окне на игровом поле. Если какой-либо клиент выиграл, то сервер прекращает игру и выдает сообщение о результате игры клиентам.

После окончания игры клиенты могут вновь без перезапуска программы начать игру.

Вариант 5. Банкомат

Описание: Пользователь банкомата может, через банкомат идентифицироваться, посмотреть свой счет, получить информацию об операциях с ним (пополнение или изъятие денег), снять деньги или

Вариант 4. Сетевые крестики-нолики

перевести на другой счет.

Задание: Написать консольное приложение-сервер, исполняющее роль банка, и оконное приложение-клиент, исполняющее роль банкомата. На сервере храниться перечень счетов клиентов, их пароли, количество денег и последние десять операций. Приложение клиент имеет оконный интерфейс, через который серверу посылаются запросы.

Описание: Предварительный подсчет голосов за кандидатов. Число голосов на каждом из 5-х избирательных пунктах постепенно увеличивается. Центризбирком, опрашивает избирательные пункты и выводит результат по каждому из кандидатов. На экране изображаются кандидаты и кол-во голосов по каждому из них. Если у первого больше всего голосов, то он рисуется выше других (не по росту, а по расположению на экране); если у третьего кол-во голосов меньше всех, то он рисуется ниже всех; соответственно второй выше третьего, но ниже первого. Все кандидаты разных цветов.

Задание: Написать консольное приложение-сервер, исполняющее роль избирательного участка, и оконное приложение-клиент, исполняющее роль

Центризбиркома. Число голосов на серверах, растет по таймеру. Клиент, также по таймеру, опрашивает сервера.

Вариант 7. Обмен сообщениями со спутником

Описание: В окне приложения нарисована планета, вокруг нее вращается спутник, в поле окна задается сектор контакта со спутником. Когда спутник заходит в сектор общения он начинает посылать сигнал о готовности к общению. Если в окне нажать кнопку «Опрос спутника» спутник вернет свои координаты, которые отобразятся в окне. Если спутник, находится вне сектора контакта, то данная функция не доступна.

Задание: Написать консольное приложение-сервер, исполняющее роль спутника, и оконное приложение-клиент, исполняющее роль окна на станции наблюдения. Координаты спутника изменяются непосредственно на сервере, а клиент их постоянно опрашивает. Проверяет на вхождение в сектор и отображает спутник на экране.

Описание: При запуске чата, происходит регистрация пользователя, после соединения с сервером, в окне приложения, показывается список пользователей чата. В программе также имеют два способа по обмену сообщениями: публичный (послать сообщение всем пользователям) и приватный (послать сообщение только конкретному пользователю).

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Клиент - это непосредственно программа ЧАТ, а сервер - программа, которая хранит список, присоединившихся пользователей их ID. Клиент формирует сообщение, состоящее из типа (публичное или приватное), имени пользователя (если сообщение приватное) и текста сообщения. Далее клиент отправляет сообщение серверу. Сервер переправляет это сообщение или конкретному клиенту или всем пользователям. Сообщения в зависимости от типа, раскрашиваются в разный цвет (посланные или принятые, приватные или публичные).

Вариант 9. Мониторинг состояния доменной печи **Вариант 8. Реализовать ЧАТ для пользователей.**

Описание: При строительстве доменной печи в ее стенки закладываются термодатчики. Компьютер с заданной периодичностью опрашивает эти датчики и следит за состоянием стенок печи. В случае 123 прогорания стенки печи выдается сигнал тревоги.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер исполняет роль датчика. В нем в специальной переменной хранится информация о длине термодатчика. С определенным интервалом времени длина термодатчика уменьшается. Клиент - это оконное приложение, в котором нарисован план печи с установленными термодатчиками. Клиент опрашивает датчики/сервера об их длине. И отображает полученную информацию на экране. Если длина датчика в пределах 71-100%, то он отображается зеленым цветом. Если длина датчика в пределах 31-70%, то он отображается желтым цветом. Если длина датчика в пределах 130%, то он отображается красным цветом.

Если длина датчика достигла 15%, то на экран выдается красное окно с сообщением об опасности.

В клиенте также отображаются и сами значения длин датчиков.

Клиент может работать с независимым количеством датчиков. **Вариант 10. Управление полетом**

Описание: Диспетчерская станция управления полетами на земле ведет мониторинг за полетами самолетов с земли. Один раз в секунду опрашивая самолеты об их координатах и высоте. Если самолеты находятся в опасной близости, то диспетчер может подать самолету команду об изменении направления движения. Если диспетчер не подал команду об изменении полета, то может произойти авиакатастрофа.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер - это самолеты, при первом запуске у сервера генерируется случайная координата

и высота зоны обслуживания диспетчерской станции. Далее генерируется направление полета (точка с координатами на краю зоны обслуживания). Самолет меняет свое местоположение вдоль направления полета. Клиент - это диспетчерская станция, в которой идет отображение плоскости полета вдоль земли и плоскости с разрезом высот. Если самолеты находятся в опасной близости, то они окрашиваются в желтый цвет. Если произошло столкновение, то они окрашиваются в красный цвет, и сервера самолетов попавших в аварию завершают работу.

Практическое занятие № 32

Изучение структуры программы на ассемблере

1.1 Цель работы

Целью данной работы является изучение структуры программы на ассемблере, использование различных директив сегментации и создание примитивных программ, типа «Hello Word».

1.2 Структура программы на ассемблере

Программа на ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти. Программа может состоять из одного или нескольких таких блоков-сегментов. Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы.

Предложения ассемблера бывают четырех типов:

- *команды* или *инструкции*, представляющие собой символические аналоги машинных команд. В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд микропроцессора;
- *макрокоманды* - оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями;
- *директивы*, являющиеся указанием транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении;
- *строки комментариев*, содержащие любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором.

1.2.1 Синтаксис ассемблера

Предложения, составляющие программу, могут представлять собой синтаксическую конструкцию, соответствующую команде, макрокоманде, директиве или комментарию. Для того чтобы транслятор ассемблера мог распознать их, они должны формироваться по определенным синтаксическим правилам. Формат предложений ассемблера⁴:

Оператор директивы [текст комментария] Оператор команды [текст комментария] Оператор макрокоманды [; текст комментария]

Формат директив:

[Имя] директива [операHgl.операHgN][; комментарий]

Формат команд и макрокоманд

**[Имя метки:] КОП
[операHgl.операHgN][;комментарий]**

Здесь: **Вариант 8.** Реализовать ЧАТ для пользователей.

- *имя метки* - идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает;

125

- *имя* - идентификатор, отличающий данную директиву от других одноименных директив;
- *код операции (КОП) и директива* - это мнемонические обозначения соответствующей машинной команды, макрокоманды или директивы транслятора;
- *операнды* - части команды, макрокоманды или директивы ассемблера, обозначающие объекты, над которыми производятся действия. Операнды ассемблера описываются выражениями с числовыми и текстовыми константами, метками и идентификаторами переменных с использованием знаков операций и некоторых зарезервированных слов.

Допустимыми символами при написании текста программ являются:

- все латинские буквы: **A-Z, a-z**. При этом заглавные и строчные буквы считаются эквивалентными;
- цифры от **0** до **9**;

⁴ Квадратные скобки означают не обязательные параметры

- знаки `?, @, $, _, &`;
- разделители `, . [ ] ( ) < > { } + / * % ! ' " ? \ = #`.

Предложения ассемблера формируются из *лексем*, представляющих собой синтаксически неразделимые последовательности допустимых символов языка, имеющие смысл для транслятора. *Лексемами* являются:

- *идентификаторы* - последовательности допустимых символов, использующиеся для обозначения таких объектов программы, как коды операций, имена переменных и названия меток. Правило записи идентификаторов заключается в следующем: идентификатор может состоять из одного или нескольких символов. В качестве символов можно использовать буквы латинского алфавита, цифры и некоторые специальные знаки - `_`, `?`, `$`, `@`. Идентификатор не может начинаться символом цифры. Длина идентификатора может быть до 255 символов, хотя транслятор воспринимает лишь первые 32, а остальные - игнорирует. Регулировать

длину возможных идентификаторов можно с использованием опции командной строки `mv`. Кроме этого, существует возможность указать транслятору на то, чтобы он различал прописные и строчные буквы либо игнорировал их различие (что и делается по умолчанию). Для этого применяются опции командной строки `/mu`, `/ml`, `/mx`;

- *цепочки символов* - последовательности символов, заключенные в одинарные или двойные кавычки; - *целые числа* в одной из следующих систем счисления: *двоичной, десятичной, шестнадцатеричной*. Отождествление чисел при записи их в программах

на ассемблере производится по определенным правилам:

- **Десятичные числа** не требуют для своего отождествления указания каких-либо дополнительных символов, например 25 или 139.
- Для отождествления в исходном тексте программы **двоичных чисел** необходимо после записи нулей и единиц, входящих в их состав, поставить латинское **"b"**, например 10010101**b**.
- **Шестнадцатеричные числа** имеют больше условностей при своей записи:
- *Во-первых*, они состоят из цифр **0...9**, строчных и прописных букв латинского алфавита **a, b, c, d, e, f** или **A, B, C, D, E, F**.
- *Во-вторых*, у транслятора могут возникнуть трудности с распознаванием шестнадцатеричных чисел из-за того, что они могут состоять как из одних цифр 0...9 (например, 190845), так и начинаться с буквы латинского алфавита (например, **ef15**). Для того чтобы "объяснить" транслятору, что данная лексема не является десятичным числом или идентификатором, программист должен специальным образом выделять шестнадцатеричное число. Для этого, на конце последовательности шестнадцатеричных цифр, составляющих шестнадцатеричное число, записывают латинскую букву **"h"**. Это обязательное условие. Если шестнадцатеричное число начинается с буквы, то перед ним записывается ведущий ноль: **0ef15h**.

1.2.2 Директивы сегментации

В ходе предыдущего обсуждения мы выяснили все основные правила записи команд и операндов в программе на ассемблере. Открытым остался вопрос о том, как правильно оформить последовательность команд, чтобы транслятор мог их обработать, а микропроцессор - выполнить.

При рассмотрении архитектуры микропроцессора мы узнали, что он имеет шесть сегментных регистров, посредством которых может одновременно работать:

- с одним сегментом кода;
- с одним сегментом стека;
- с одним сегментом данных;
- с тремя дополнительными сегментами данных.

Еще раз вспомним, что физически сегмент представляет собой область памяти, занятую командами и (или) данными, адреса которых вычисляются относительно значения в соответствующем сегментном регистре.

Синтаксическое описание сегмента на ассемблере представляет собой следующую конструкцию:

Имя сегмента SEGMENT [тип выравнивания] [тип комбинирования] [класс сегмента] [тип размера сегмента]

содержимое сегмента

Имя сегмента ENDS

Важно отметить, что функциональное назначение сегмента несколько шире, чем простое разбиение программы на блоки кода, данных и стека. Сегментация является частью общего механизма, связанного с концепцией модульного программирования. Она предполагает унификацию оформления объектных модулей, создаваемых компилятором, в том числе с разных языков программирования. Это позволяет объединять программы, написанные на разных языках. Именно для реализации различных вариантов такого объединения и предназначены операнды в директиве SEGMENT. Рассмотрим их подробнее:

- *Атрибут выравнивания сегмента* (тип выравнивания) сообщает компоновщику о том, что нужно обеспечить размещение начала сегмента на заданной границе. Это важно, поскольку при правильном выравнивании доступ к данным в процессорах i80x86 выполняется быстрее. Допустимые значения этого атрибута следующие:

- BYTE - выравнивание не выполняется. Сегмент может начинаться с любого адреса памяти;

- WORD - сегмент начинается по адресу, кратному двум, то есть последний (младший) значащий бит физического адреса равен 0 (выравнивание на границу слова);

- DWORD - сегмент начинается по адресу, кратному четырем, то есть два последних (младших) значащих бита, равны 0 (выравнивание на границу двойного слова);

- PARA - сегмент начинается по адресу, кратному 16, то есть последняя шестнадцатеричная цифра адреса должна быть 0h (выравнивание на границу параграфа);

- PAGE - сегмент начинается по адресу, кратному 256, то есть две последние шестнадцатеричные цифры должны быть 00h (выравнивание на границу 256-байтной страницы);

- MEMPAGE - сегмент начинается по адресу, кратному 4 Кбайт, то есть три последние шестнадцатеричные цифры должны быть 000h (адрес следующей 4-Кбайтной страницы памяти). По умолчанию тип выравнивания имеет значение PARA.

Атрибут комбинирования сегментов (комбинаторный тип) сообщает компоновщику, как нужно комбинировать сегменты различных модулей, имеющие одно и то же имя. Значениями атрибута комбинирования сегмента могут быть:

- PRIVATE - сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля;

- PUBLIC - заставляет компоновщик соединить все сегменты с одинаковыми именами. Новый объединенный сегмент будет целым и непрерывным. Все адреса (смещения) объектов, а это могут быть, в зависимости от типа сегмента, команды и данные, будут вычисляться относительно начала этого нового сегмента;

- COMMON - располагает все сегменты с одним и тем же именем по одному адресу. Все сегменты с данным именем будут перекрываться и совместно

использовать память. Размер полученного в результате сегмента будет равен размеру самого большого сегмента;

- **АТ xxxx** - располагает сегмент по абсолютному адресу параграфа (параграф - объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0);

- **STACK** - определение сегмента стека. Заставляет компоновщик соединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра **ss**.

По умолчанию атрибут комбинирования принимает значение **PRIVATE**.

- **Атрибут класса сегмента** (тип класса) - это заключенная в кавычки строка, помогающая компоновщику определить соответствующий порядок следования сегментов при собирании программы из сегментов нескольких модулей. Компоновщик объединяет в памяти все сегменты с одним и тем же именем класса (имя класса, в общем случае, может быть любым, но лучше, если оно будет отражать функциональное назначение сегмента);

- **Атрибут размера сегмента**. Для процессоров i80386 и выше сегменты могут быть 16 или 32-разрядными. Это влияет, прежде всего, на размер сегмента и порядок формирования физического адреса внутри него. Атрибут может принимать следующие значения:

- **USE16** - это означает, что сегмент допускает 16-разрядную адресацию. При формировании физического адреса может использоваться только 16-разрядное смещение. Соответственно, такой сегмент может содержать до 64 Кбайт кода или данных;

- **USE32** - сегмент будет 32-разрядным. При формировании физического адреса может использоваться 32-разрядное смещение. Поэтому такой сегмент может содержать до 4 Гбайт кода или данных.

Все сегменты сами по себе равноправны, так как директивы **SEGMENT** и **ENDS** не содержат информации о функциональном назначении сегментов. Для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом, для чего используют специальную директиву **ASSUME**. Эта директива сообщает транслятору о том, какой сегмент, к какому сегментному регистру привязан. В свою очередь, это позволит транслятору корректно связывать символические имена, определенные в сегментах.

Далее приведем пример программы с использованием стандартных директив сегментации:

```
data segment para public 'data' ;сегмент данных message db 'Hello World,$' ;  
описание строки data ends  
stk segment stack db 256 dup ('') ; размер сегмента стека stk ends  
code segment para public 'code' ; начало сегмента  
; кода main proc ;начало процедуры main assume cs:code,ds:data,ss:stk mov  
ax,data ; адрес сегмента данных  
; в регистр ax mov ds,ax ; ax в ds  
mov ah,9  
mov dx,offset message  
int 21h ; ah=9 функция 21h прерывания  
; выводит строку на экран, адрес  
; которой храниться в регистре dx,  
; строка должна обязательно  
; заканчиваться символом $  
mov ax,4c00h ; пересылка 4c00h в регистр ax int 21h ; вызов прерывания с  
номером  
21h
```

main endp ; конец процедуры main code ends ; конец сегмента кода
end main ; конец программы с точкой входа main

Для простых программ, содержащих по одному сегменту для кода, данных и стека, хотелось бы упростить ее описание. Для этого в трансляторы MASM и TASM ввели возможность использования *упрощенных директив сегментации*. Но здесь возникла проблема, связанная с тем, что необходимо было как-то компенсировать невозможность напрямую управлять размещением и комбинированием сегментов. Для этого, совместно с упрощенными директивами сегментации, стали использовать директиву указания модели памяти **MODEL**, которая частично стала управлять размещением сегментов и выполнять функции директивы **ASSUME** (поэтому при использовании упрощенных директив сегментации директиву ASSUME можно не использовать). Эта директива связывает сегменты, которые в случае использования упрощенных директив сегментации имеют предопределенные имена с сегментными регистрами (хотя явно инициализировать *ds* все равно придется).

Теперь, перепишем вышеприведенную программу с использованием упрощенных директив сегментации.

мает ; режим работы TASM: ideal или masm model small ; модель памяти
.data ; сегмент данных message db 'Hello World,\$' ; описание строки
.stack ;сегмент стека db 256 dup ('?') ; сегмент стека
.code ;сегмент кода main proc ; начало процедуры main
mov ax,@data ; заносим адрес сегмента данных в
; регистр ax mov ds,ax ;ax в ds
mov ah,9
mov dx,offset message
int 21h ; ah=9 функция 21h прерывания ; выводит строку на экран, адрес
; которой храниться в регистре dx
mov ax,4c00h ; пересылка 4c00h в регистр ax
int 21h ; вызов прерывания с номером 21h main endp ; конец процедуры main
end main ; конец программы с точкой входа main

Обязательным параметром директивы **MODEL** является *модель памяти*. Этот параметр определяет модель сегментации памяти для программного модуля. Предполагается, что программный модуль может иметь только определенные типы сегментов, которые определяются упомянутыми нами ранее *упрощенными директивами описания сегментов*. Эти директивы приведены в таблице 1.

Таблица 1. Упрощенные директивы определения сегмента

Формат директивы (режим MASM)	Формат директивы (режим IDEAL)	Назначение
.CODE [имя]	CODESEG^]	Начало или продолжение сегмента кода
.DATA	DATASEG	Начало или продолжение сегмента инициализированных данных. Также используется для определения данных типа near
.CONST	CONST	Начало или продолжение сегмента постоянных данных (констант) модуля

.FARDATA [имя]	FARDATA [имя]	Начало или продолжение сегмента инициализированных данных типа far
-------------------	---------------	---

Наличие в некоторых директивах параметра **[имя]** говорит о том, что возможно определение нескольких сегментов этого типа. С другой стороны, наличие нескольких видов сегментов данных обусловлено требованием обеспечения совместимости с некоторыми компиляторами языков высокого уровня, которые создают разные сегменты данных для инициализированных и неинициализированных данных, а также констант.

При использовании директивы **MODEL** транслятор делает доступными несколько идентификаторов, к которым можно обращаться во время работы программы с тем, чтобы получить информацию о тех или иных характеристиках данной модели памяти (таблица 3). Перечислим эти идентификаторы и их значения (табл. 2).

Таблица 2. Модели памяти

Модель	Тип кода	Тип данных	Назначение модели
TINY	near	near	Код и данные объединены в одну группу с именем DGROUP. Используется для создания программ формата .com.
SMALL	near	near	Код занимает один сегмент, данные объединены в одну группу с именем DGROUP. Эту модель обычно используют для большинства программ на ассемблере
MEDIUM	far	near	Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль. Все ссылки на передачу управления - типа far. Данные объединены в одной группе; все ссылки на них - типа near
COMPACT	near	far	Код в одном сегменте; ссылка на данные - типа far
LARGE	far	far	Код в нескольких сегментах, по одному на каждый объединяемый программный модуль

Параметр модификатор директивы **MODEL** позволяет уточнить некоторые особенности использования выбранной модели памяти (табл. 3).

Таблица 3. Модификаторы модели памяти

Значение модификатора	Назначение
use16	Сегменты выбранной модели используются как 16-битные (если соответствующей
	директивой указан процессор i80386 или i80486)

use32	Сегменты выбранной модели используются как 32-битные (если соответствующей директивой указан процессор i80386 или i80486)
dos	Программа будет работать в MS-DOS

Необязательные параметры - язык и модификатор языка, определяют некоторые особенности вызова процедур. Необходимость в использовании этих параметров появляется при написании и связывании программ на различных языках программирования.

Описанные нами стандартные и упрощенные директивы сегментации не исключают друг друга. Стандартные директивы используются, когда программист желает получить полный контроль над размещением сегментов в памяти и их комбинированием с сегментами других модулей.

Упрощенные директивы целесообразно использовать для простых программ и программ, предназначенных для связывания с программными модулями, написанными на языках высокого уровня. Это позволяет компоновщику эффективно связывать модули разных языков за счет стандартизации связей и управления.

1.2.3 Создание COM-программ

Все вышеприведенные директивы сегментации и примеры программ предназначены для создания программ в EXE-формате⁶. Компоновщик LINK автоматически генерирует особый формат для EXE-файлов, в котором присутствует специальный начальный блок (заголовок) размером не менее 512 байт.

За исключением модели памяти TINY при использовании упрощенных директив сегментации. Для выполнения можно также создавать COM-файлы. Примером часто используемого COM-файла является COMMAND.COM.

Размер программы. EXE-программа может иметь любой размер, в то время как COM-файл ограничен размером одного сегмента и не превышает 64К. COM-файл всегда меньше, чем соответствующий EXE-файл; одна из причин этого - отсутствие в COM-файле 512-байтового начального блока EXE-файла.

Сегмент стека. В EXE-программе определяется сегмент стека, в то время как COM-программа генерирует стек автоматически. Таким образом, при создании ассемблерной программы, которая будет преобразована в COM-файл, стек должен быть опущен.

Сегмент данных. В EXE программе обычно определяется сегмент данных, а регистр DS инициализируется адресом этого сегмента. В COM-программе все данные должны быть определены в сегменте кода. Ниже будет показан простой способ решения этого вопроса.

Инициализация. EXE-программа записывает нулевое слово в стек и инициализирует регистр DS. Так как COM-программа не имеет ни стека, ни сегмента данных, то эти шаги отсутствуют.

Когда COM-программа начинает работать, все сегментные регистры содержат адрес префикса программного сегмента (PSP), - 256-байтового (шест. 100) блока, который резервируется операционной системой DOS непосредственно перед COM или EXE программой в памяти. Так как адресация начинается с шест. смещения 100 от начала PSP, то в программе после оператора SEGMENT кодируется директива ORG 100H.

Обработка. Для программ в EXE и COM форматах выполняется ассемблирование для получения OBJ-файла, и компоновка для получения EXE-файла. Если программа создается для выполнения как EXE-файл, то ее уже можно выполнить. Если же программа создается для выполнения как COM-файл, то компоновщиком будет выдано сообщение:

Warning: No STACK Segment (Предупреждение: Сегмент стека не определен) Ниже приведем пример COM-программы:

CSEG Segment 'Code' assume CS:CSEG,DS:CSEG,ES:CSEG,SS:CSEG org 100h
start:
mov ah,9 mov dx,offset message
int 21h ; ah=9 функция 21h прерывания
; выводит строку на экран, адрес
; которой храниться в регистре dx
int 20h ; выход из COM-программы
message db 'Hello World,\$' ; описание строки ends end start

1.3 Компиляция программ на ассемблере

Для написания программ на языке ассемблере вы можете воспользоваться любым текстовым редактором, поддерживающим кодировку ASCII-символов, например «Блокнот»/«Notepad» из ОС Windos или встроенным текстовым редактором FAR/DN/NC и др.

Для создания исполняемых файлов из программ написанных на языке ассемблере вам необходимо использовать компилятор TASM.EXE и линковщик TLINK.EXE.

TASM.EXE компилирует программные модули ассемблера в объектные модули OBJ. А TLINK.EXE из нескольких модулей делает один исполняемый файл EXE или COM. Более подробный синтаксис использования TASM.EXE и TLINK.EXE можно получить запустив эти программы без параметров.

1.4 Задание на выполнение

1. Ознакомиться со структурой программы на ассемблере.
2. Написать и скомпилировать программу «Hello World» с использованием стандартных директив компиляции в exe- формате.
3. Написать и скомпилировать программу «Hello World» с использованием упрощенных директив компиляции в exe- формате.
4. Написать и скомпилировать программу «Hello World» с использованием стандартных директив компиляции в сом- формате.
5. Написать и скомпилировать программу «Hello World» с использованием упрощенных директив компиляции в сом- формате.
6. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

Практическое занятие № 33

Изучение функций ввода/вывода

2.1. Цель работы

Целью данной работы является изучение функций ввода/вывода.

2.2 Функции прерываний ввода/вывода

В операционной системе существует большая группа функций 21h прерывания (прерывания DOS). Небольшую часть этих функций составляют функции ввода вывода информации. Для вызова какого-либо прерывания необходимо:

- в регистр АН занести номер функции прерывания;
- в зависимости от типа прерывания в какие-либо регистры занести дополнительные параметры; - использовать команду int с указанием номера прерывания.

С 9h функцией прерывания 21h вы познакомились на предыдущей лабораторной работе.

В данной лабораторной работе вам понадобится помимо функции вывода строки использовать функции ввода и вывода символа.

Для вызова функции ввода символа используют 1h- функцию 21h прерывания. После нажатия символа на клавиатуре в регистре AL сохраняется код ASCII нажатого символа.

Для вывода символа на экран можно воспользоваться функцией 2h прерывания 21h. В регистр AL помещается ASCII- код символа и вызывается прерывание 21h.

Дополнительную информацию по различным функциям прерываний операционной системы можно взять в электронном справочнике thelp (рис. 2.1-2.4).

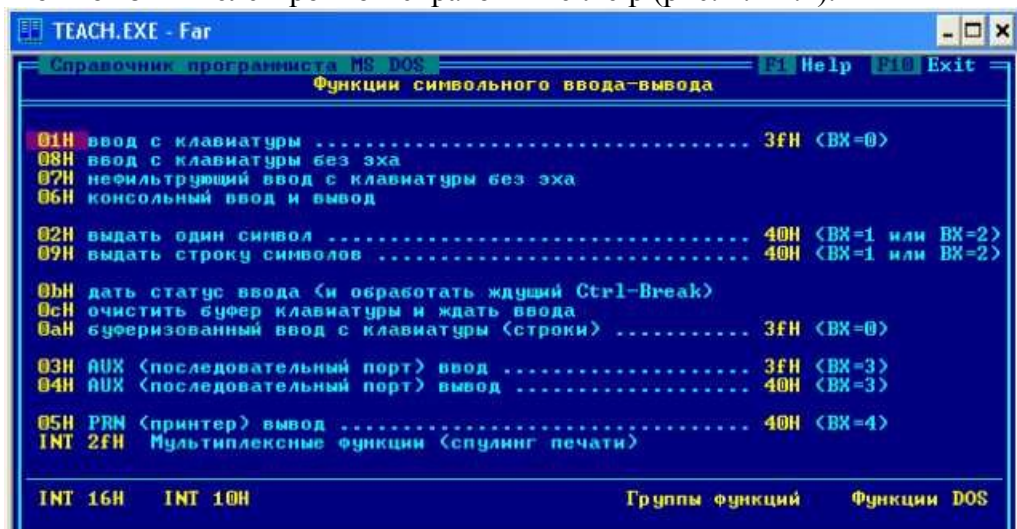


Рис. 2.1 - Функции символического ввода/вывода

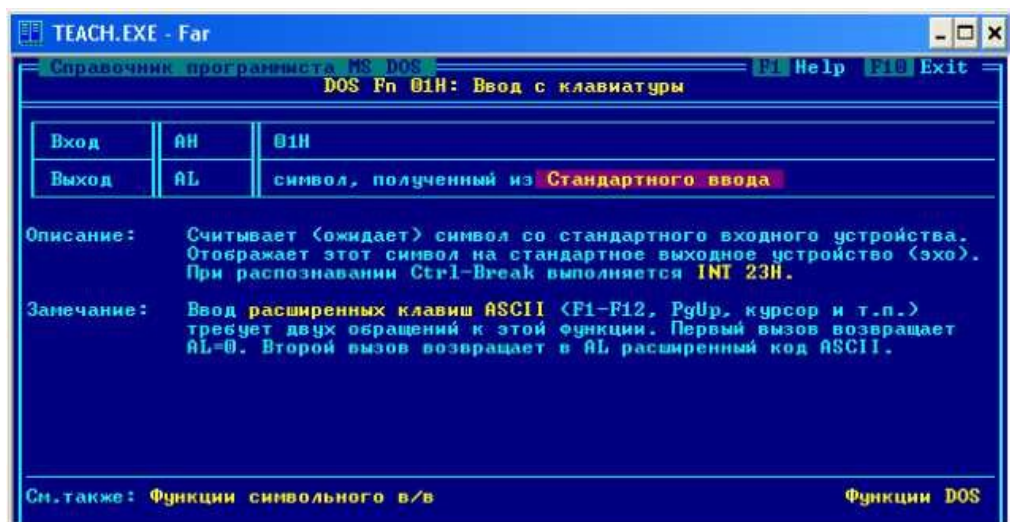


Рис. 2.2 - Функция ввод с клавиатуры

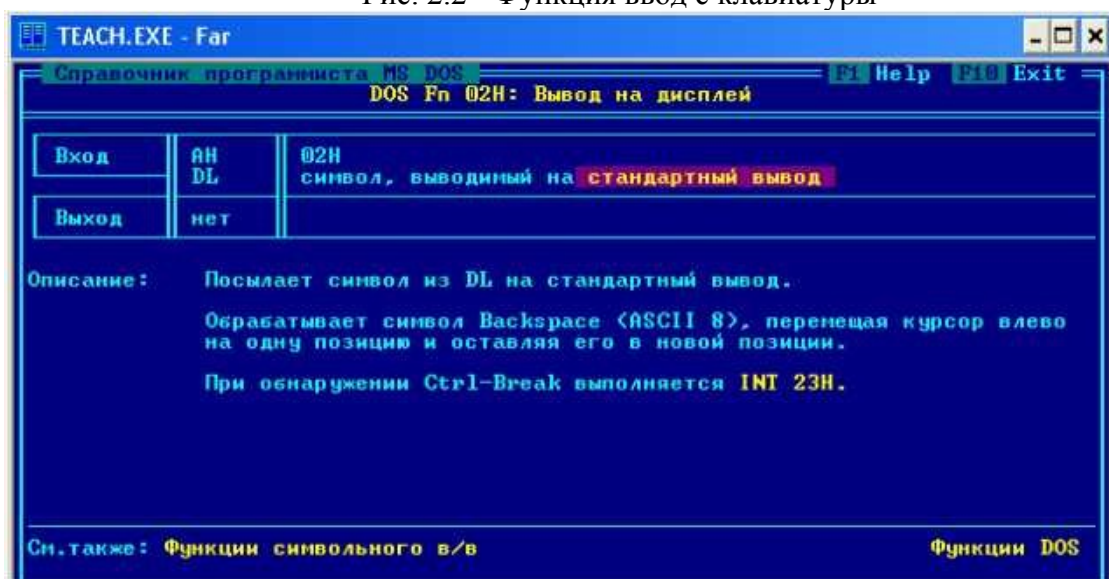


Рис. 2.3 Функция вывода символа на дисплей

2.4 Задание на выполнение

Написать программу ввода двух шестнадцатеричных чисел и вывода на экран этих чисел в двоичном виде.

Практическое занятие №34

Изучение арифметических и логических команд

3.1. Цель работы

Целью данной работы является изучение арифметических и логических команд микропроцессора.

3.2 Арифметические команды

Сложение двоичных чисел без знака. Микропроцессор выполняет сложение операндов по правилам сложения двоичных чисел. Проблем не возникает до тех пор, пока значение результата не превышает размерности поля операнда. Например, при сложении операндов размером в байт результат не должен превышать число 255. Если это происходит, то результат оказывается неверным. К примеру, выполним сложение: $254 + 5 = 259$ в двоичном виде. $11111110 + 0000101 = 1\ 00000011$. Результат вышел за пределы восьми бит и правильное его значение укладывается в 9 бит, а в 8-битовом поле операнда осталось значение 3, что, конечно, неверно. В микропроцессоре этот исход сложения прогнозируется и предусмотрены специальные средства для фиксирования подобных ситуаций и их обработки. Так, для фиксирования ситуации выхода за разрядную сетку результата, как в данном случае, предназначен флаг переноса cf. Он располагается в бите 0 регистра флагов eflags/flags. Именно установкой этого флага фиксируется факт переноса единицы из старшего разряда операнда. Естественно, что программист должен предусматривать возможность такого исхода операции сложения и средства для корректировки. Это предполагает включение участков кода после операции сложения, в которых анализируется флаг cf. Анализ этого флага можно провести различными способами. Самый простой и доступный - использовать команду условного перехода jcc. Эта команда в качестве операнда имеет имя метки в текущем сегменте кода. Переход на эту метку осуществляется в случае, если в результате работы предыдущей команды флаг cf установился в 1. В системе команд микропроцессора имеются три команды двоичного сложения: - inc операнд - операция инкремента, то есть увеличения значения операнда на 1;

- add операнд_1,операнд_2 - команда сложения с принципом действия: операнд_1 = операнд_1 + операнд_2
- adc операнд_1,операнд_2 - команда сложения с учетом флага переноса cf: операнд_1 = операнд_1 + операнд_2 + значение_cf

Обратите внимание на последнюю команду - это команда сложения, учитывающая перенос единицы из старшего разряда. Механизм появления такой единицы мы уже рассмотрели. Таким образом, команда adc является средством микропроцессора для сложения длинных двоичных чисел, размерность которых превосходит поддерживаемые микропроцессором длины стандартных полей.

Рассмотрим пример вычисления суммы чисел:

```
<1> ;prg1
<2> masm
<3> model    small
<4> stack    256
<5> .data
<6> a db     254
<7> .code    ;сегмент кода
<8> main:
<9> mov ax,@data
<10> mov ds,ax
<11> ...
<12> xor ax,ax
<13> add al,17
<14> add al,a
```

```

<15> jnc t1 ;если нет переноса, то перейти ;на m1
<16> adc ah,0;е ax сумма с учетом переноса
<17> m1: ...
<18> exit:
<19> mov ax,4c00h ;стандартный выход
<20> int 21h
<21> end main ;конец программы

```

В строках 13-14 создана ситуация, когда результат сложения выходит за границы операнда. Эта возможность учитывается строкой 15, где команда `jnc` (хотя можно было обойтись и без нее) проверяет состояние флага `cf`. Если он установлен в 1, то это признак того, что результат операции получился больше по размеру, чем размер операнда, и для его корректировки необходимо выполнить некоторые действия. В данном случае мы просто полагаем, что границы операнда расширяются до размера `AX`, для чего учитываем перенос в старший разряд командой `ADC` (строка 15).

Сложение двоичных чисел со знаком. Микропроцессор не подозревает о различии между числами со знаком и без знака. Вместо этого у него есть средства фиксирования возникновения характерных ситуаций, складывающихся в процессе вычислений. Некоторые из них мы рассмотрели при обсуждении сложения чисел без знака:

- флаг переноса `cf`, установка которого в 1 говорит о том, что произошел выход за пределы разрядности операндов;
- команду `adc`, которая учитывает возможность такого выхода (перенос из младшего разряда).

Другое средство - это регистрация состояния старшего (знакового) разряда операнда, которое осуществляется с помощью флага переполнения `of` в регистре `eflags` (бит 11).

Дополнительно к флагу `of` при переносе из старшего разряда устанавливается в 1 и флаг переноса `cf`. Так как микропроцессор не знает о существовании чисел со знаком и без знака, то вся ответственность за правильность действий с получившимися числами ложится на программиста. Проанализировать флаги `cf` и `of` можно командами условного перехода `jc\jnc` и `jo\jno` соответственно.

Что же касается команд сложения чисел со знаком, то они те же, что и для чисел без знака.

Вычитание двоичных чисел без знака. Как и при анализе операции сложения, порассуждаем над сутью процессов, происходящих при выполнении операции вычитания. Если уменьшаемое больше вычитаемого, то проблем нет, - разность положительна, результат верен. Если уменьшаемое меньше вычитаемого, возникает проблема: результат меньше 0, а это уже число со знаком. В этом случае результат необходимо завернуть. Что это означает? При обычном вычитании (в столбик) делают заем 1 из старшего разряда. Микропроцессор поступает аналогично, то есть занимает 1 из разряда, следующего за старшим, в разрядной сетке операнда.

Таким образом, после команды вычитания чисел без знака нужно анализировать состояние флага `cf`. Если он установлен в 1, то это говорит о том, что произошел заем из старшего разряда и результат получился в дополнительном коде.

Аналогично командам сложения, группа команд вычитания состоит из минимально возможного набора. Эти команды выполняют вычитание по алгоритмам, которые мы сейчас рассматриваем, а учет особых ситуаций должен производиться самим программистом. К командам вычитания относятся: - `dec` операнд - операция декремента, то есть уменьшения значения операнда на 1; - `sub` операнд_1,операнд_2 - команда вычитания; ее принцип действия:

операнд_1 = операнд_1 - операнд_2

- `sbb` операнд_1,операнд_2 - команда вычитания с учетом заема (флага `cf`):
операнд_1 = операнд_1 - операнд_2 - значение_сf

Как видите, среди команд вычитания есть команда `sbb`, учитывающая флаг переноса `cf`. Эта команда подобна `adc`, но теперь уже флаг `cf` выполняет роль индикатора заема 1 из старшего разряда при вычитании чисел.

Рассмотрим пример программной обработки ситуации:

```
<1> ;prg2
<2> masm
<3> model    small
<4> stack    256
<5> .data
<6> .code    ;сегмент кода
<7> main:    ;точка входа в программу
<8>
<9> xor ax,ax
<10> mov al,5
<11> sub al,10
<12> jnc m1 ;нет переноса?
<13> neg al ;в al модуль результата <14> m1: ...
<15> exit:
<16> mov ax,4c00h ;стандартный выход <17> int 21h
<18> end main ;конец программы
```

В этом примере в строке 11 выполняется вычитание. С указанными для этой команды вычитания исходными данными результат получается в дополнительном коде (отрицательный). Для того чтобы преобразовать результат к нормальному виду (получить его модуль), применяется команда `neg`, с помощью которой получается дополнение операнда. В нашем случае мы получили дополнение дополнения или модуль отрицательного результата. А тот факт, что это на самом деле число отрицательное, отражен в состоянии флага `cf`. Дальше все зависит от алгоритма обработки.

Вычитание двоичных чисел со знаком. Здесь все несколько сложнее. Последний пример показал то, что микропроцессору незачем иметь два устройства - сложения и вычитания. Достаточно наличия только одного - устройства сложения. Но для вычитания способом сложения чисел со знаком в дополнительном коде необходимо представлять оба операнда - и уменьшаемое, и вычитаемое. Результат тоже нужно рассматривать как значение в дополнительном коде. Но здесь возникают сложности. Прежде всего, они связаны с тем, что старший бит операнда рассматривается как знаковый.

Отследить ситуацию переполнения мантиссы можно по содержимому флага переполнения `of`. Его установка в 1 говорит о том, что результат вышел за диапазон представления знаковых чисел (то есть изменился старший бит) для операнда данного размера, и программист должен предусмотреть действия по корректировке результата.

Умножение чисел без знака. Для умножения чисел без знака предназначена команда `mul` сомножитель_1

Как видите, в команде указан всего лишь один операнд- сомножитель. Второй операнд - сомножитель_2 задан неявно. Его местоположение фиксировано и зависит от размера сомножителей. Так как в общем случае результат умножения больше, чем любой из его сомножителей, то его размер и местоположение должны быть тоже определены однозначно. Варианты размеров сомножителей и размещения второго операнда и результата приведены в таблице 4.

Таблица 4. Расположение операндов и результата при умножении

сомножитель 1	сомножитель 2	Результат
Байт	al	16 бит в ax: al - младшая часть результата; ah – старшая часть результата
Слово	ax	32 бит в паре dx:ax: ax - младшая часть результата; dx - старшая часть результата
Двойное слово	eax	64 бит в паре edx:eax: eax - младшая часть результата; edx - старшая часть результата

Из таблицы видно, что произведение состоит из двух частей и в зависимости от размера операндов размещается в двух местах - на месте сомножитель_2 (младшая часть) и в дополнительном регистре ah, dx, edx (старшая часть). Как же динамически (то есть во время выполнения программы) узнать, что результат достаточно мал и уместился в одном регистре или что он превысил размерность регистра, и старшая часть оказалась в другом регистре? Для этого привлекаются уже известные нам по предыдущему обсуждению флаги переноса cf и переполнения of:

- если старшая часть результата нулевая, то после операции произведения флаги cf = 0 и of = 0;

- если же эти флаги ненулевые, то это означает, что результат вышел за пределы младшей части произведения и состоит из двух частей, что и нужно учитывать при дальнейшей работе. *Умножение чисел со знаком.* Для умножения чисел со знаком предназначена команда `imul` операнд_1[,операнд_2,операнд_3]

Эта команда выполняется так же, как и команда `mul`. Отличительной особенностью команды `imul` является только формирование знака.

Если результат мал и умещается в одном регистре (то есть если cf = of = 0), то содержимое другого регистра (старшей части) является расширением знака - все его биты равны старшему биту (знаковому разряду) младшей части результата.

В противном случае, (если cf = of = 1) знаком результата является знаковый бит старшей части результата, а знаковый бит младшей части является значащим битом двоичного кода результата.

Если вы посмотрите описание команды `imul`, то увидите, что она допускает более широкие возможности по заданию местоположения операндов. Это сделано для удобства использования.

Деление чисел без знака. Для деления чисел без знака предназначена команда:

`div` делитель

Делитель может находиться в памяти или в регистре и иметь размер 8, 16 или 32 бит. Местонахождение делимого фиксировано и так же, как в команде умножения, зависит от размера операндов. Результатом команды деления являются значения частного и остатка.

Варианты местоположения и размеров операндов операции деления показаны в таблице 5.

Таблица 5. Расположение операндов и результата при делении

Делимое	Делитель	Частное	Остаток
16 бит в регистре ax	Байт регистр или ячейка памяти	Байт в регистре al	Байт в регистре ah
32 бит dx - старшая часть ax - младшая часть	Слово 16 бит регистр или ячейка памяти	Слово 16 бит в регистре ax	Слово 16 бит в регистре dx
64 бит edx - старшая часть eax - младшая часть	Двойное слово 32 бит регистр или ячейка памяти	Двойное слово 32 бит в регистре eax	Двойное слово 32 бит в регистре edx

После выполнения команды деления содержимое флагов неопределено, но возможно возникновение прерывания с номером 0, называемого “деление на ноль”. Этот вид прерывания относится к так называемым исключениям. Эта разновидность прерываний возникает внутри микропроцессора из-за некоторых аномалий во время вычислительного процесса. Прерывание 0, “деление на ноль”, при выполнении команды `div` может возникнуть по одной из следующих причин:

- делитель равен нулю;
- частное не входит в отведенную под него разрядную сетку, что может случиться в следующих случаях:
- при делении делимого величиной в слово на делитель величиной в байт, причем значение делимого в более чем 256 раз больше значения делителя;
- при делении делимого величиной в двойное слово на делитель величиной в слово, причем значение делимого в более чем 65 536 раз больше значения делителя;
- при делении делимого величиной в учетверенное слово на делитель величиной в двойное слово, причем значение делимого в более чем 4 294 967 296 раз больше значения делителя. *Деление чисел со знаком.* Для деления чисел со знаком предназначена команда `idiv` делитель

Для этой команды справедливы все рассмотренные положения, касающиеся команд и чисел со знаком. Отметим лишь особенности возникновения исключения 0, “деление на ноль”, в случае чисел со знаком. Оно возникает при выполнении команды `idiv` по одной из следующих причин:

- делитель равен нулю;
- частное не входит в отведенную для него разрядную сетку. Последнее в свою очередь может произойти:
- при делении делимого величиной в слово со знаком на делитель величиной в байт со знаком, причем значение делимого в более чем 128 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от -128 до +127);
- при делении делимого величиной в двойное слово со знаком на делитель величиной в слово со знаком, причем значение делимого в более чем 32 768 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от -32 768 до +32 768);
- при делении делимого величиной в учетверенное слово со знаком на делитель величиной в двойное слово со знаком, причем значение делимого в более чем 2 147 483 648 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от -2 147 483 648 до +2 147 483 647).

3.3 Логические команды

В системе команд микропроцессора есть следующий набор команд, поддерживающих работу с логическими данными:

and операнд_1, операнд_2 - операция логического умножения.

Команда выполняет поразрядно логическую операцию И (конъюнкцию) над битами операндов операнд_1 и операнд_2. Результат записывается на место операнд_1.

or операнд_1, операнд_2 - операция логического сложения.

Команда выполняет поразрядно логическую операцию ИЛИ (дизъюнкцию) над битами операндов операнд_1 и операнд_2. Результат записывается на место операнд_1.

xor операнд_1, операнд_2 - операция логического исключающего сложения.

Команда выполняет поразрядно логическую операцию исключающего ИЛИ над битами операндов операнд_1 и операнд_2. Результат записывается на место операнд_1.

test операнд_1, операнд_2 - операция “проверить” (способом логического умножения). Команда выполняет поразрядно логическую операцию И над битами операндов операнд_1 и операнд_2. Состояние операндов остается прежним, изменяются только флаги zf, sf, и pf, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния.

not операнд - операция логического отрицания. Команда выполняет поразрядное инвертирование (замену значения на обратное) каждого бита операнда. Результат записывается на место операнда.

3.4 Команды сдвига

Команды этой группы также обеспечивают манипуляции над отдельными битами операндов, но иным способом, чем логические команды, рассмотренные выше. Все команды сдвига перемещают биты в поле операнда влево или вправо в зависимости от кода операции.

Количество сдвигаемых разрядов - счетчик_сдвигов - располагается, как видите, на месте второго операнда и может задаваться двумя способами:

- статически, что предполагает задание фиксированного значения с помощью непосредственного операнда;
- динамически, что означает занесение значения счетчика сдвигов в регистр cl перед выполнением команды сдвига.

Команды линейного сдвига делятся на два подтипа: - команды логического линейного сдвига;

- команды арифметического линейного сдвига.

К командам логического линейного сдвига относятся:

- shl операнд, счетчик_сдвигов (Shift Logical Left) - логический сдвиг влево. Содержимое операнда сдвигается влево на количество битов, определяемое значением счетчик_сдвигов. Справа (в позицию младшего бита) вписываются нули;
- shr операнд, счетчик_сдвигов (Shift Logical Right) - логический сдвиг вправо. Содержимое операнда сдвигается вправо на количество битов, определяемое значением счетчик_сдвигов. Слева (в позицию старшего, знакового бита) вписываются нули.

Команды арифметического линейного сдвига отличаются от команд логического сдвига тем, что они особым образом работают со знаковым разрядом операнда:

- sal операнд, счетчик_сдвигов (Shift Arithmetic Left) - арифметический сдвиг влево. Содержимое операнда сдвигается влево на количество битов, определяемое значением счетчик_сдвигов. Справа (в позицию младшего бита) вписываются нули. Команда sal не сохраняет знака, но устанавливает флаг cf в случае смены знака очередным выдвигаемым битом. В остальном команда sal полностью аналогична команде shl;
- sar операнд, счетчик_сдвигов (Shift Arithmetic Right) - арифметический сдвиг вправо. Содержимое операнда сдвигается вправо на количество битов, определяемое

значением счетчик_сдвигов. Слева в операнд вписываются нули. Команда `sar` сохраняет знак, восстанавливая его после сдвига каждого очередного бита.

К командам циклического сдвига относятся команды, сохраняющие значения сдвигаемых бит. Есть два типа команд циклического сдвига:

- команды простого циклического сдвига;
- команды циклического сдвига через флаг переноса `cf`.

К командам простого циклического сдвига относятся:

- `rol` операнд, счетчик_сдвигов (Rotate Left) - циклический сдвиг влево. Содержимое операнда сдвигается влево на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые влево биты записываются в тот же операнд справа.
- `ror` операнд, счетчик_сдвигов (Rotate Right) - циклический сдвиг вправо. Содержимое операнда сдвигается вправо на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые вправо биты записываются в тот же операнд

К командам циклического сдвига через флаг переноса `cf` относятся следующие:

- `rcl` операнд, счетчик_сдвигов (Rotate through Carry Left) - циклический сдвиг влево через перенос. Содержимое операнда сдвигается влево на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые биты поочередно становятся значением флага переноса `cf`.

`rcr` операнд, счетчик_сдвигов (Rotate through Carry Right) - циклический сдвиг вправо через перенос. Содержимое операнда сдвигается вправо на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые биты поочередно становятся значением флага переноса `cf`.

3.5 Задание на выполнение

Выполните задания в соответствии с вашим вариантом. И подготовиться к ответам на теоретическую часть лабораторной работы.

Вариант 1

Пользователь вводит два числа *A* и *B* в шестнадцатеричном виде. Программа должна:

1. Посчитать $C = A + B/2$.
2. Установить все четные биты *C*.
3. Вывести на экран число *C* в двоичном виде.

Вариант 2

Пользователь вводит два числа *A* и *B* в шестнадцатеричном виде. Программа должна:

1. Посчитать $C = A + B/3$.
2. Обнулить все нечетные биты *C*.
3. Вывести на экран число *C* в двоичном виде.

Вариант 3 Пользователь вводит два числа *A* и *B* в шестнадцатеричном виде. Программа должна:

1. Посчитать $C = A + B$.
2. Если третий бит числа *C* установлен, то вывести на экран *C* в двоичном виде, в противном случае, вывести на экран $C/2$ в двоичном виде.

Вариант 4

Пользователь вводит два числа *A* и *B* в шестнадцатеричном виде. Программа должна:

1. Посчитать $C = A/2 + B$.
2. Установить все нечетные биты *C*.
3. Вывести на экран число *C* в двоичном виде.

Вариант 5

Пользователь вводит два числа *A* и *B* в шестнадцатеричном виде. Программа должна:

1. Посчитать $C = (A + B)/4$.

2. Сбросить первый бит числа C , если он установлен.
3. Вывести на экран число C в двоичном виде.

Вариант 6

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A-B)*4$.
2. Выполнить циклический сдвиг полученного числа C на 3 бита вправо.
3. Вывести на экран число C в двоичном виде.

Вариант 7

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A/2+B$.
2. Выполнить арифметический сдвиг C на 3 бита влево.
3. Вывести на экран число C в двоичном виде.

Вариант 8

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A+B*2$.
2. Обнулить все четные биты C .
3. Вывести на экран число C в двоичном виде.

Вариант 9

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A+(B-5h)*2$.
2. Если установлен четвертый бит числа C то вывести на экран A в двоичном виде, в противном случае вывести на экран число B в двоичном виде.

Вариант 10

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A+12h)/2+B$.
2. Обнулить все четные биты C .
3. Вывести на экран число C в двоичном виде.

Вариант 11

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A-14h)*4-B$.
2. Установить все четные биты C .
3. Вывести на экран число C в двоичном виде.

Вариант 12

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A*B-4$.
2. Если третий и пятый бит числа C установлены, вывести на экран A в двоичном виде, если третий и пятый бит числа C сброшены, вывести на экран B в двоичном виде, в других случаях вывести на экран число C в двоичном виде.

Вариант 13

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A+B)/4-16$.
2. Если третий и пятый бит числа C установлены, вывести на экран A в двоичном виде, если второй и четвертый бит числа C сброшены, вывести на экран B в двоичном виде, в других случаях вывести на экран число C в двоичном виде.

Вариант 14

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A-B)*2+1Ah$.
2. Вывести на экран 1 если шестой бит числа С установлен и 0 в противном случае.

Вариант 15

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A-7h)*4-B$.
2. Сбросить третий бит числа С, если он установлен.
3. Вывести на экран число С в двоичном виде.

Вариант 16

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A+4*B$.
2. Выполнить циклический сдвиг числа С на 3 бита вправо.
3. Вывести на экран число С в двоичном виде.

Вариант 17

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A*3+B*2$.
2. Выполнить арифметический сдвиг числа С на 2 бита вправо.
3. Вывести на экран число С в двоичном виде.

Вариант 18

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A*3+B*2$.
2. Выполнить арифметический сдвиг числа С на 2 бита вправо.
3. Вывести на экран число С в двоичном виде.

Вариант 19

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=A*B-4$.
2. Если третий и пятый бит числа С установлены, вывести на экран А в двоичном виде, если третий и пятый бит числа С сброшены, вывести на экран В в двоичном виде, в других случаях вывести на экран число С в двоичном виде.

Вариант 20

Пользователь вводит два числа А и В в шестнадцатеричном виде. Программа должна:

1. Посчитать $C=(A+17h)/2+B$.
2. Обнулить все четные биты С.
3. Вывести на экран число С в двоичном виде.

Практическое занятие №35

Модульное программирование

4.1 Цель работы

Целью данной работы является изучение написания процедур и макросов с использованием модульной структуры программы.

4.2 Процедуры на языке ассемблера

Для оформления процедур существуют специальные директивы `proc/endr` и машинная команда `ret` - возврат управления из процедуры вызывающей программе.

Формат команды `ret`: `ret` число.

Работа команды зависит от типа процедуры:

- для процедур ближнего типа - восстановить из стека содержимое `ebp/ir`;
- для процедур дальнего типа - последовательно восстановить из стека содержимое `ebp/ir` и сегментного регистра `cs`.
- если команда `ret` имеет операнд, то увеличить содержимое `esp/sp` на величину операнда число; при этом учитывается атрибут режима адресации - `use16` или `use32`:
- если `use16`, то $sp^{sp} + \text{число}$, то есть указатель стека сдвигается на число байт, равное значению число;
- если `use32`, то $sp = (sp + 2 * \text{число})$, то есть указатель стека сдвигается на число слов, равное значению число.

Процедуры могут размещаться в программе:

- в начале программы (до первой исполняемой команды);
- в конце программы (после команды выхода в операционную систему);
- промежуточный вариант - тело процедуры располагается внутри другой процедуры или основной программы (с использованием команды безусловного перехода `jmp`); - в другом модуле.

Вызов близкой или дальней процедуры с запоминанием в стеке адреса точки возврата осуществляется *командой `call`* (формат: `call` метка). Выполнение команды не влияет на флаги.

При ближней адресации - в стек заносится содержимое указателя команд `ebp/ir` и в этот же регистр загружается новое значение адреса, соответствующее метке;

При дальней адресации - в стек заносится содержимое указателя команд `ebp/ir` и `cs`. Затем в эти же регистры загружаются новые значения адресов, соответствующие дальней метке;

4.3 Передача аргументов через регистры

Существуют следующие варианты передачи аргументов в процедуру (модуль):

- *Через регистры.* Данные становятся доступными немедленно после передачи управления процедуре. Очень популярный способ при небольшом объеме передаваемых данных.
- *Через общую область памяти.* Необходимо использовать атрибут комбинирования сегментов - `common` (см. лаб. работу 1). Сегменты будут перекрываться в памяти и, следовательно, совместно использовать выделенную память.
- *Через стек.* Наиболее часто используемый способ. Суть его в том, что вызывающая процедура самостоятельно заносит в стек передаваемые данные, после чего производит вызов вызываемой процедуры.
- *С помощью директив `extrn` и `public`.* Директива `extrn` предназначена для объявления некоторого имени внешним по отношению к данному модулю. Это имя в другом модуле должно быть объявлено в директиве `public`. Директива `public` предназначена для объявления некоторого имени, определенного в этом модуле и видимого в других модулях.

Синтаксис директив:

`Extrn` имя:тип, имя:тип

`Public` имя, имя

Здесь имя - идентификатор, определенный в другом модуле. В качестве идентификатора могут выступать:

- имена переменных (определенных операторами db, dw и т.д.); - имена процедур;
- имена констант (определенных операторами = и equ).

Возможные значения типа определяются допустимыми типами объектов для этих директив:

- если имя - это переменная, то тип может принимать значения byte, word, dword, pword, fword, qword и tbyte;
- если имя - это процедура, то тип может принимать значения near или far; - е с л и имя - это константа, то тип должен быть abs.

Остановимся более подробно при передаче параметров через стек. Приведем пример структуры вызываемой процедуры при ближней адресации.

```

asmpr proc near
;пролог push bp mov bp,sp
;доступ к элементам стека mov ax, [bp+4] ;'доступ к N-элементу mov ax, [bp+6]
'доступ к N-1-элементу mov ax, [bp+8] 'доступ к N-2-элементу
' эпилог mov sp,bp 'восстановление sp pop bp 'восстановление bp
ret 'возврат в вызывающую программу asmpr endp 'конец процедуры

```

На рис. 1. показана структура стека при ближней (а) и дальней (б) адресации в нутрии вызываемой процедуры.

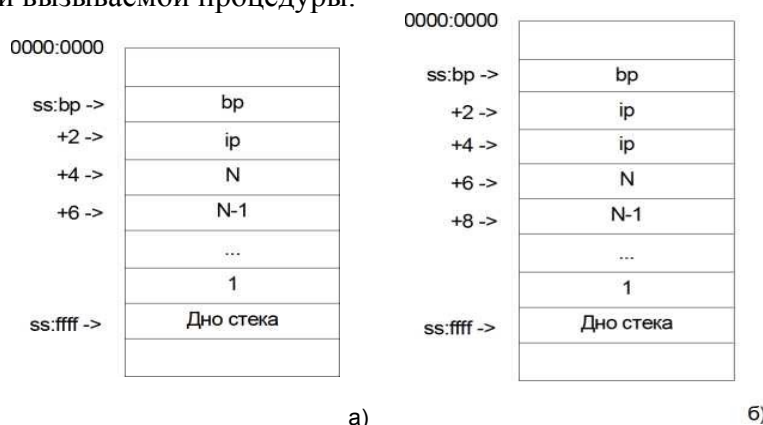


Рис. 1. Структура стека в вызываемой процедуре при использовании различных видов адресации: а - ближняя адресация; б - дальняя адресация.

При использовании дальней адресации для доступа к элементам стека требуется поправка на +2.

```

;доступ к элементам стека при дальней
адресации mov ax, [bp+6] ;'доступ к N-элементу
mov ax, [bp+8] 'доступ к N-1-элементу mov ax, [bp+10] 'доступ к N-2элементу

```

4.4 Возврат результата из процедуры

Существует три варианта возврата результата из процедуры:

- С использованием регистров.
- С использованием общей памяти.
- С использованием стека. Здесь возможны два варианта:
- Использование для возвращаемых аргументов тех же ячеек в стеке, которые применялись для передачи аргументов в процедуру.
- Предварительное помещение в стек наряду с передаваемыми аргументами фиктивных аргументов с целью резервирования места для возвращаемого значения.

4.5 Макросредства языка ассемблера

4.5.1 Псевдооператоры equ и =

К простейшим макросредствам языка ассемблера можно отнести псевдооператоры equ и "=" (равно). Эти псевдооператоры предназначены для присвоения некоторому

выражению символического имени или идентификатора. Впоследствии, когда в ходе трансляции этот идентификатор встретится в теле программы, макроассемблер подставит вместо него соответствующее выражение. В качестве выражения могут быть использованы константы, имена меток, символические имена и строки в апострофах. После присвоения этим конструкциям символического имени его можно использовать везде, где требуется размещение данной конструкции.

Синтаксис псевдооператора **equ**: **имя_идентификатора equ строка или числовое_выражение**

Синтаксис псевдооператора **"="**: **имя_идентификатора = числовое_выражение**

Несмотря на внешнее и функциональное сходство псевдооператоры **equ** и **"="** отличаются следующим:

- из синтаксического описания видно, что с помощью **equ** идентификатору можно ставить в соответствие, как числовые выражения, так и текстовые строки, а псевдооператор **"="** может использоваться только с числовыми выражениями;
- идентификаторы, определенные с помощью **"="**, можно переопределять в исходном тексте программы, а определенные с использованием **equ** - нельзя.

Ассемблер всегда пытается вычислить значение строки, воспринимая ее как выражение. Для того, чтобы строка

воспринималась именно как текстовая, необходимо заключить ее в угловые скобки:

<строка>.

Кстати сказать, угловые скобки являются оператором ассемблера, с помощью которого транслятору сообщается, что заключенная в них строка должна трактоваться как текст, даже если в нее входят служебные слова ассемблера или операторы. Хотя в режиме **Ideal** это не обязательно, так как строка для **equ** в нем всегда трактуется как текстовая.

Псевдооператор **equ** удобно использовать для настройки программы на конкретные условия выполнения, замены сложных в обозначении объектов, многократно используемых в программе более простыми именами и т. п.

Псевдооператор **"="** удобно использовать для определения простых абсолютных (то есть независящих от места загрузки программы в память) математических выражений. Главное условие то, чтобы транслятор мог вычислить эти выражения во время трансляции.

4.5.2 Макрокоманды

Идейно макрокоманда представляет собой дальнейшее развитие механизма замены текста. С помощью макрокоманд в текст программы можно вставлять последовательности строк (которые логически могут быть данными или командами) и даже более того - привязывать их к контексту места вставки.

Макрокоманда представляет собой строку, содержащую некоторое символическое имя - имя макрокоманды, предназначенное для того, чтобы быть замещенной одной или несколькими другими строками. Имя макрокоманды может сопровождаться параметрами.

Обычно программист сам чувствует момент, когда ему нужно использовать макрокоманды в

своей программе. Если такая необходимость возникает, и нет готового ранее разработанного варианта нужной макрокоманды, то вначале необходимо задать ее шаблон-описание, который называют макроопределением.

Синтаксис макроопределения следующий:

имя_макрокоманды macro список_аргументов тело макроопределения endm

Есть три варианта размещения макроопределения:

- В начале исходного текста программы до сегмента кода и данных с тем, чтобы не ухудшать читабельность программы. Этот вариант следует применять в случаях, если определяемые вами макрокоманды актуальны только в пределах одной этой программы.

- В отдельном файле. Этот вариант подходит при работе над несколькими программами одной проблемной области. Чтобы сделать доступными эти макроопределения в конкретной программе, необходимо в начале исходного текста этой программы записать директиву `include` имя_файла.
- В макробиблитеке. Если у вас есть универсальные макрокоманды, которые используются практически во всех ваших программах, то их целесообразно записать в так называемую макробиблитеку. Сделать актуальными макрокоманды из этой библиотеки можно с помощью все той же директивы `include`.

Если в программе некоторая макрокоманда вызывается несколько раз, то в процессе макрогенерации возникнет ситуация, когда в программе один идентификатор будет определен несколько раз, что, естественно, будет распознано транслятором как ошибка. Для выхода из подобной ситуации применяют директиву **local**, которая имеет следующий синтаксис: `local` список_идентификаторов.

4.6. Задание на выполнение

Модифицировать предыдущую программу в виде:

Вариант 1

1. Написать процедуру для вывода результата
2. Написать процедуру для ввода чисел
3. Написать макрос для расчета

Передача параметров через регистры

Вариант 2

1. Написать процедуру для вывода результата
2. Написать макрос для ввода чисел
3. Написать макрос для расчета Передача параметров через стек

Вариант 3

1. Написать макрос для вывода результата
2. Написать процедуру для ввода чисел
3. Написать макрос для расчета Передача параметров через стек

Вариант 4

1. Написать макрос для вывода результата
2. Написать макрос для ввода чисел
3. Написать процедуру для расчета Передача параметров через стек

Вариант 5

1. Написать процедуру для вывода результата
2. Написать процедуру для ввода чисел
3. Написать макрос для расчета

Передача параметров через стек

Вариант 6

1. Написать процедуру для вывода результата
2. Написать макрос для ввода чисел
3. Написать процедуру для расчета Передача параметров через стек

Вариант 7

1. Написать макрос для вывода результата
2. Написать процедуру для ввода чисел
3. Написать процедуру для расчета

Передача параметров через стек

Вариант 8

1. Написать процедуру для вывода результата
2. Написать процедуру для ввода чисел
3. Написать процедуру для расчета Передача параметров через стек

Вариант 9

1. Написать макрос для вывода результата

Практическое занятие №36

Совершенствование навыков работы на языке ассемблера

5.1 Цель работы

Целью работы является совершенствование навыков работы на языке ассемблера.

5.2 Задание на выполнение

Введите матрицу из N,N (или массив из N) элементов на языке и выполните одно из действий согласно варианту.

Вариант 1.

Выбрать все элементы главной диагонали матрицы.

Вариант 2.

Найти максимальный и минимальный элемент матрицы.

Вариант 3.

Построить обратную матрицу.

Вариант 4.

Из всех элементов матрицы вычесть минимальный элемент.

Вариант 5.

Обнулить в матрице все четные числа

Вариант 6.

Обнулить в матрице все нечетные числа.

Вариант 7.

Поменять в матрице четные столбцы с нечетными.

Вариант 8.

Поменять в матрице четные строки с нечетными.

Вариант 9.

Умножить элементы матрицы на минимальный элемент.

Вариант 10.

Найти среднееарифметическое число всех элементов матрицы (целая часть, остаток)

Вариант 11.

Найти определитель матрицы 3×3 .

Вариант 12.

Сложить все элементы матрицы построчно с элементом на главной диагонали текущей строки.

Вариант 13.

Сложить все элементы матрицы по столбцам с элементом на главной диагонали текущего столбца.

Вариант 14.

Посчитать сумму элементов всех строк.

Вариант 15.

Посчитать сумму элементов всех столбцов.

Вариант 16.

Посчитать сумму элементов главной диагонали

Вариант 17.

Посчитать сумму элементов обратной диагонали.

Вариант 18.

Ввести два массива и посчитать сумму и разность соответствующих элементов массивов.

Ввести два массива и найти максимальные элементы. Сравнить их. Вывести тот массив, у которого максимальный элемент меньше, чем максимальный элемент другого массива.

Практическое занятие №37

Интерфейс с языками высокого уровня и обработка массивов

6.1 Цель работы

Целью работы является изучение связи подпрограмм на ассемблере с программами, написанными на языках высокого уровня и обработка массивов на языке ассемблере.

6.2 Формы комбинирования программ на языках высокого уровня с ассемблером

Существуют следующие формы комбинирования программ:

- Использование ассемблерных вставок. Эта форма сильно зависит от синтаксиса языка высокого уровня и конкретного компилятора.
- Использование внешних процедур и функций (на уровне объектных модулей). Это более универсальная форма комбинирования. Она имеет ряд преимуществ:
- Написание и отладку программ можно производить независимо.
- Написанные программы можно использовать в других проектах.
- Облегчается модификация и сопровождение программ в течение жизненного цикла проекта.

Разработка таких подпрограмм на языке ассемблера требует ясного представления о том, каким образом взаимодействуют подпрограммы в разных языках (таблица 6). Передача аргументов, как правило, осуществляется через стек.

Таблица 6. Сравнение механизмов взаимодействия подпрограмм

	Си	Паскаль
Тип подпрограммы	Функция	Процедура или функция
Направление передачи аргументов	Справа налево	Слева направо
Аргументы передаются	По значению (адрес это указатель)	По значению и по адресу
Процедура, чистящая стек	Вызывающая	Вызываемая

6.3 Соглашения о связях для языка Си

Если в программе на языке Си используется обращение к внешнему модулю, написанному на другом языке, необходимо включить в программу прототип внешней функции (описание точки входа), например:

```
extern "C" void asmpoc(char ch, unsigned x, unsigned y)
```

Функция `asmpoc` должна описываться на ассемблере следующим процедурным блоком:

```
public _asmpoc  
_asmpoc proc ...  
_asmpoc endp
```

Блок `extrn "C"` добавляет к имени точки входа функции префикс (символ подчеркивания). Некоторые компиляторы Си не требуют наличия "C", символ подчеркивания добавляется по умолчанию. Тип ассемблерной подпрограммы зависит от используемой компилятором модели памяти. Модели памяти и типы указателей на подпрограммы сведены в таблице 7.

Таблица 7. Соотношение моделей памяти и типов указателей

Модель	Ключ компилятора	BCC-указатель на функцию	указатель на данные
Small	-ms	near, CS	near, DS
Compact	-mc	far	near, DS
Medium	-mm	near, CS	far
Large	-ml	far	far

Для малой модели памяти (Small) сегмент кода ассемблерной подпрограммы объединяется с кодовым сегментом главной программы на СИ, который для малой модели всегда имеет имя `_TEXT`. Это же имя должен иметь сегмент кода ассемблерной внешней подпрограммы или используйте упрощенные директивы сегментации, где нет необходимости указывать имя сегмента

Для большой модели памяти (Large) сегмент кода в ассемблерной подпрограмме не объединяется и может иметь любое имя.

5.4 Соглашение о связях для языка Паскаль

В программу на языке Паскаль необходимо включить прототип (описание) точки входа во внешнюю подпрограмму или функцию по правилам описания заголовка процедуры или функции:

```
Procedure asmproc(ch:char;x,y,kol:integer); external;
```

Директива `EXTERNAL` указывает, что описание подпрограммы не содержится в главной программе. Для указания файла, содержащего объектный модуль внешней подпрограммы, в программе на Паскале используется директива `{ $\$$ L путь_до_объектного_модуля}`.

Компоновщик объединяет этот сегмент с сегментом главной программы.

Занесение аргументов в стек при вызове подпрограммы производится в порядке следования аргументов в списке прототипа. При передаче по значению сами аргументы заносятся в стек, даже массивы.

По умолчанию компилятор Паскаля вставляет в программу команду `CALL` дальнего вызова. Если необходимо организовать ближний вызов, то перед прототипом внешней подпрограммы следует поставить директиву `{ $\$$ F-}` - отмена дальнего вызова. Эта директива действует только на одну подпрограмму и формирует для нее `CALL` ближнего обращения. Директива не действует на адреса аргументов - в стек заносится полный адрес.

6.4 Задание на выполнение

1. Введите матрицу из N,N (или массив из N , согласно предыдущей лабораторной работы) элементов на языке Си или Паскаль.
2. Передайте его в качестве аргументов в процедуру языка Ассемблера.
3. Выполните одно из действий (согласно предыдущей лабораторной работы).
4. Результат передайте в вызывающую программу и выведите на печать.

Практическое занятие №38

Использование цепочечных команд

7.1 Цель работы

Целью работы является изучение работы цепочечных команд при обработке массивов на языке ассемблера.

7.2 Цепочечные команды

Пересылка цепочек в память movs приемник,источник movsb movsw movsd

Алгоритм работы:

Выполнить копирование байта, слова или двойного слова из операнда источника в операнд приемник, при этом адреса элементов предварительно должны быть загружены:

- адрес источника – в пару регистров ds:esi/si (ds по умолчанию, допускается замена сегмента);
 - адрес приемника – в пару регистров es:edi/di (замена сегмента не допускается);
- В зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
- если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
 - если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;

Если есть префикс повторения, то выполнить определяемые им действия.

Пример программы пересылки цепочки в память:

```
assume ds:@data,es: @data
```

```
main: mov ax,@data ;загрузка сегментных регистров
```

```
mov ds,ax ;настройка регистров DS и ES
```

```
mov es,ax ;на адрес сегмента данных
```

```
cld ;сброс флага DF - обработка строки от начала к концу
```

```
lea si,source ;загрузка в SI смещения строки-
```

```
.data source db 'Тестируемая строка','$' ;строка-источник dest db 19 DUP ('') ;строка-приёмник
.code
```

```
lea dx,dest mov ah,09h ;вывод на экран строки-приёмника
int 21h exit:
```

Сравнение цепочек в памяти.

источника lea di,dest ;загрузка в DI смещения строки-приёмника

```
mov cx,20 ;для префикса rep - счетчик повторений movs
```

```
rep dest,source ;пересылка строки
```

```
cmps приемник,источник cmpsb cmpsw cmpsd
```

Алгоритм работы:

- выполнить вычитание элементов (источник - приемник), адреса элементов предварительно должны быть загружены:
- адрес источника – в пару регистров ds:esi/si;
- адрес назначения – в пару регистров es:edi/di;
- в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
- если df=0, то увеличить содержимое этих регистров на длину элемента последовательности;
- если df=1, то уменьшить содержимое этих регистров на длину элемента последовательности;
- в зависимости от результата вычитания установить флаги:
- если очередные элементы цепочек не равны, то cf=1, zf=0;

- если очередные элементы цепочек или цепочки в целом равны, то cf=0, zf=1; - при наличии префикса выполнить определяемые им действия.

Загрузка элемента цепочки в аккумулятор

lods источник lodsb lodsw lodsd

Алгоритм работы:

- загрузить элемент из ячейки памяти, адресуемой парой ds:esi/si, в регистр al/ax/eax. Размер элемента определяется неявно (для команды lods) или явно в соответствии с применяемой командой (для команд lodsb, lodsw, lodsd);

- изменить значение регистра si на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:

- df=0 – значение положительное, то есть просмотр от начала цепочки к ее концу; - df=1 – значение отрицательное, то есть просмотр от конца цепочки к ее началу.

Загрузка элемента из аккумулятора в цепочку stos приемник stosb stosw stosd

Алгоритм работы:

- записать элемент из регистра al/ax/eax в ячейку памяти, адресуемую парой es:di/edi. Размер элемента определяется неявно (для команды stos) или конкретной применяемой командой (для команд stosb, stosw, stosd);

- изменить значение регистра di на величину, равную длине элемента цепочки. Знак этого изменения зависит от состояния флага df:

- df=0 – увеличить, что означает просмотр от начала цепочки к ее концу;

- df=1 – уменьшить, что означает просмотр от конца цепочки к ее началу.

Пример совместной работы stosb и lodsb:

;копировать одну строку в другую до первого пробела str1 db 'Какая-то строка'

len_str1=\$-str str2 db len_str1 dup (' ')

mov ax,@data mov ds,ax mov es,ax

cld mov cx,len_str1 lea si,str1 lea di,str2

m1: lodsb cmp al,' ' jc exit ;выход, если пробел

stosb

loop m1 exit:

7.3 Задание на выполнение

Реализовать задание из предыдущей лабораторной работы полностью на языке ассемблера с применением цепочечных команд.

Практическое занятие №39

Основы программирования VBA

Цель и содержание: ознакомиться с возможностями разработки встроенных в документы приложений-макросов на языке Visual Basic for Applications.

Теоретические сведения

Макрос – это программа, состоящая из списка команд, которые должны быть выполнены приложением. Макрос служит для объединения нескольких различных действий в одну процедуру, которую вы можете легко вызвать. Этот список команд состоит в основном из *макрокоманд*, которые тесно связаны с командами приложения, в котором вы создаете макрос – т. е. с командами Word, Excel или других приложений Microsoft Office. Запись макросов позволяет не просто запомнить последовательность вызовов команд меню, нажатий на кнопки мыши и ввод данных с клавиатуры, но и перевести эти действия на объектно-ориентированный язык программирования Visual Basic для приложений (Visual Basic for Applications, в дальнейшем просто VBA) и сохранить их в виде готовой к выполнению программы.

VBA является полноценным языком программирования, позволяющим записать не только последовательно выполняемые пользователем действия, но и содержащим все необходимые конструкции языка программирования высокого уровня, включая разнообразные средства организации ветвлений, циклов и ведения диалога с пользователем. Весьма удобный редактор VBA позволяет не только писать и редактировать программы, но и вести их отладку.

Можно выделить следующие разновидности макросов:

- *Командные макросы* – это наиболее распространенные макросы, обычно состоящие из операторов, эквивалентных тем или иным командам меню или параметрам диалоговых окон. Основным предназначением такого макроса является выполнение действий, аналогичных командам меню – т. е. изменение окружения и основных объектов приложения
- *Пользовательские функции* – работают аналогично встроенным функциям Excel. Отличие этих функций от командных макросов состоит в том, что они используют значения передаваемых им аргументов, производят некоторые вычисления и возвращают результат в точку вызова, но не изменяют среды приложения
- *Макрофункции* – представляют собой сочетание командных макросов и пользовательских функций. Наряду с тем, что они могут использовать аргументы и возвращать результат, подобно пользовательским функциям, они могут также изменять среду приложения, как и командные макросы. Чаще всего эти макросы вызываются из других макросов, и активно используются для модульного программирования. Если необходимо выполнить ряд одинаковых действий в различных макросах, то обычно эти действия выделяются в отдельную макрофункцию (подпрограмму), которая вызывается всякий раз, когда необходимо выполнить эти повторяющиеся действия.

Автоматическая запись макросов

Для автоматической записи макроса надо выполнить следующие действия:

1. В обычном режиме выполнить команды, которые надо сохранить в макросе. Это надо сделать, чтобы четко выполнить последовательность команд.
2. Перейти в режим записи макроса.
3. Задать имя макроса.
4. Выполнить необходимую последовательность команд.
5. Выйти из режима записи макроса.

Рассмотрим подробнее пункты со 2 по 5.

Для перехода в режим записи макроса надо на вкладке Разработчик выполнить команду **Макрос – Запись макроса**. В появившемся окне задать имя макроса.

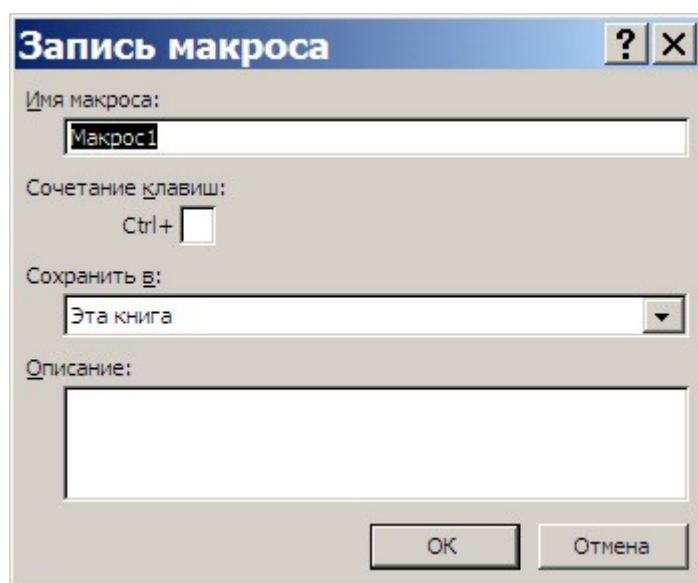


Рис.1

Имя макроса может содержать буквы и цифры и не может содержать пробелы. Можно связать выполнение макроса с комбинацией клавиш. Эта комбинация так же задается в окне записи макроса. В этом же окне можно описать, что делает данный макрос.

После щелчка по кнопке ОК. Выполняется последовательность команд, которую надо запомнить в макросе.

В конце выполняется команда Макрос – Остановить запись.

Вызвать макрос можно двумя способами.

Первый способ:

Выполнить команду **Макросы – Макросы**. В окне Макрос (Рис.2) выбрать нужный макрос и щелкнуть по кнопке Выполнить.

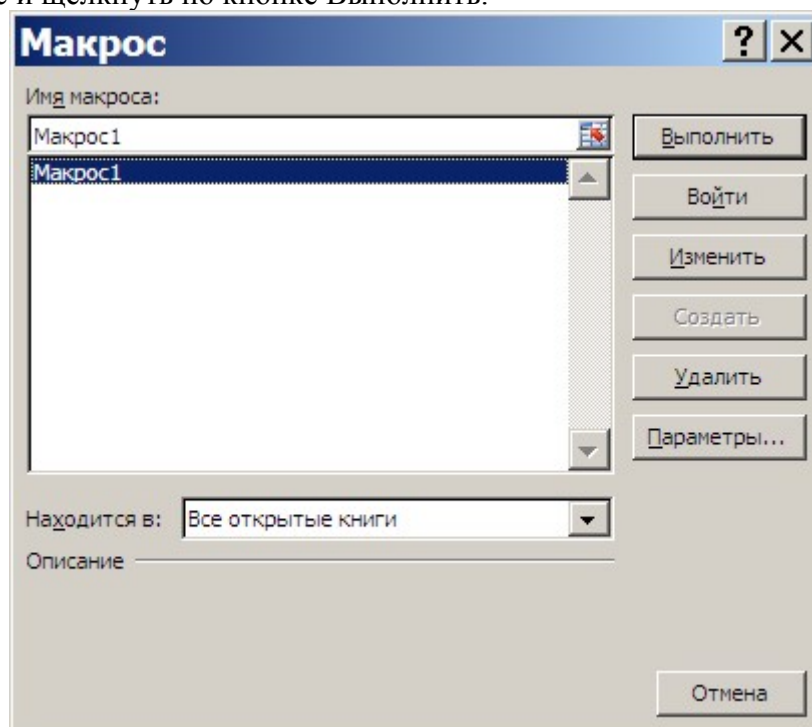


Рис. 2

В этом же окне можно удалить макрос, который работает неправильно.

1 способ

Нажать комбинацию клавиш, связанную с данным макросом.

Задание 1

Создать макрос, который выделяет диапазон таблицы A1:D5, и выполняет его оформление. Внешние границы оформления - двойная линия. Внутренние линии - одинарные. Результат:

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					

Имя макроса – *обрамление1*. Комбинация клавиш: *Ctrl+й*.

После записи макроса перейдите на новый лист и вызовите макрос любым из описанных способов. Обратите внимание, что во всех случаях выполнение макроса связано с конкретным диапазоном A1:D5, но не привязано к листу данной книги.

Можно записать командный макрос, привязанный не к конкретным ячейкам, а к положению курсора. Для этого надо записывать макрос в режиме относительной адресации.

Задание 2

Создать макрос, который выполняет оформление 4 столбцов и 5 строк. Внешние границы оформления - двойная линия. Внутренние линии - одинарные. Порядок выполнения:

1. Установите курсор в ячейку A1.
2. Выполните команду Макрос – Запись макроса.
3. Задайте имя макроса *обрамление2*. комбинация клавиш *Ctrl+у*.
4. После щелчка по кнопке ОК выполните команду **Макрос – Относительные ссылки**.
5. Выделите диапазон A1:D5.
6. Выполните заданное оформление диапазона.
7. Выполните команду Макрос – Остановить запись.

После записи макроса поставьте курсор в ячейку D8 и выполните макрос *обрамление2*.

В обоих заданиях выделение диапазона выполнялось в макросе. Это приводило к тому, что оформление выполнялось для диапазона определенных размеров – 4 столбца и 5 строк. Если выделение диапазона выполнять до записи макроса, то размер диапазона оформления будет произволен.

Задание 3

Выполните оформление диапазона A1:D5. Но сначала надо выделить данный диапазон, а потом выполнить команду **Макрос – Запись макроса**. Задайте имя макроса *обрамление3*, комбинация клавиш – *Ctrl+у*.

После записи макроса выделите ячейки B10:K12 и вызовите макрос *обрамление3*.

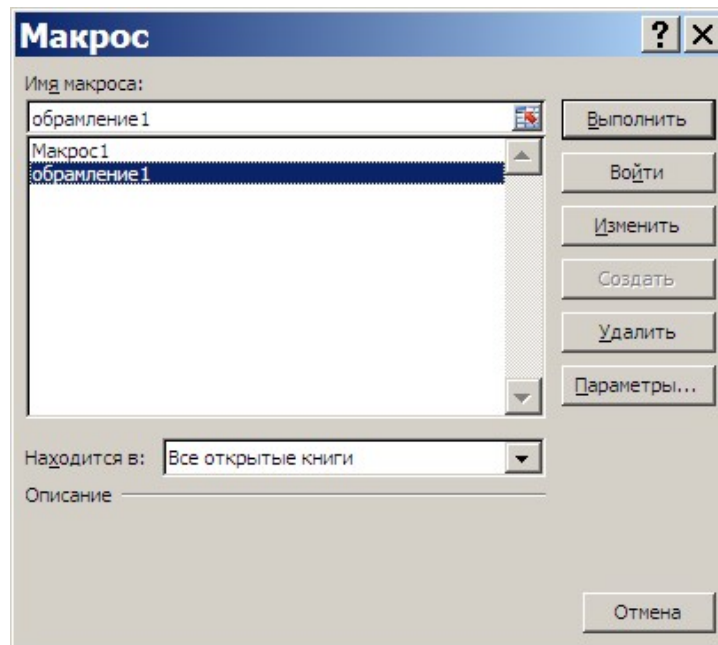
Сохраните книгу под именем Макросы.

Просмотр макросов.

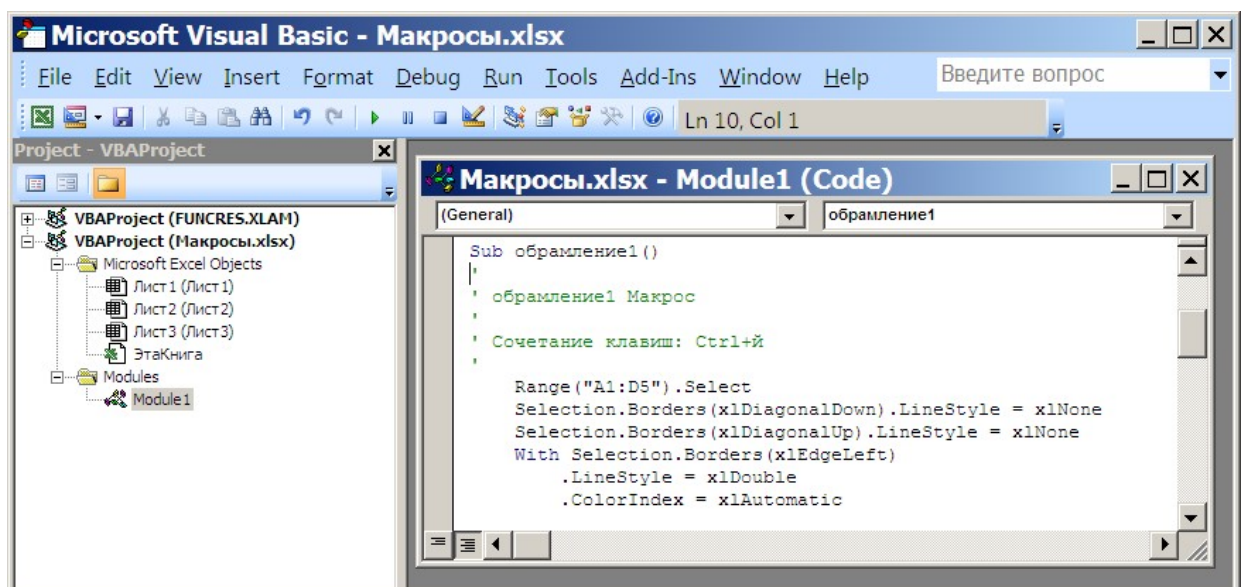
При записи макросов Макрорекодер записывает все выполненные команды в макрос. Эти макросы находятся в модулях и можно увидеть текст этих макросов и, при необходимости, изменить. Это широко используется в программировании VBA. Нет необходимости точно запоминать команды форматирования, копирования и т.д. Достаточно записать их в командный макрос, а потом изменить так как надо. И в нашем случае не надо помнить команды обрамления на языке VBA. Макрорекодер их сам запишет.

Для просмотра текста макроса надо:

1. Выполнить команду **Макрос – Макрос**.
2. Выбрать нужный макрос.
3. Щелкнуть по кнопке **Изменить**.
- 4.



После этого откроется окно Visual Basic и в окне Модуля1 будет выведен код макроса *обрамление1*.



Любой макрос имеет следующую структуру:

Sub НазваниеМакроса()

Команда VBA Команда VBA

...

End Sub

Текст зеленого цвета – это комментарии. Комментарии начинаются со знака «апостроф» и не выполняются компьютером.

Рассмотрим код макроса *обрамление1*.

Sub обрaмление1()

,

' обрaмление1 Макрос

,

' Сочетание клавиш: Ctrl+й

,

Range("A1:D5").Select „Выделяется диапазон A1:D5

„Все команды выполняются для выделенного диапазона

„ диагонального обрaмления нет

Selection.Borders(xlDiagonalDown).LineStyle = xlNone

Selection.Borders(xlDiagonalUp).LineStyle = xlNone

„ задается стиль левой границы

With Selection.Borders(xlEdgeLeft)

.LineStyle = xlDouble „двойная линия

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

„ задается стиль верхней границы

With Selection.Borders(xlEdgeTop)

.LineStyle = xlDouble „двойная линия

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

„ задается стиль нижней границы

With Selection.Borders(xlEdgeBottom)

.LineStyle = xlDouble „двойная линия

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

„ задается стиль правой границы

With Selection.Borders(xlEdgeRight)

.LineStyle = xlDouble „двойная линия

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

„ задается стиль внутренней вертикальной

With Selection.Borders(xlInsideVertical)

.LineStyle = xlContinuous „одинарная сплошная линия

.ColorIndex = xlAutomatic

```

        .TintAndShade = 0
        .Weight = xlThin
    End With
    „ задается стиль внутренней горизонтальной
    With Selection.Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous „одинарная сплошная линия
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThin
    End With
End Sub
Рассмотрим код макроса обрамление2.
Sub обрамление2()
'
' обрамление2 Макрос
'
' Сочетание клавиш: Ctrl+ц
'

    ActiveCell.Range("A1:D5").Select
    _от активной ячейки выделяется диапазон A1:D5 Остальной текст программы
    совпадает с обрамление1.
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThin

```

```

End With
With Selection.Borders(xlInsideHorizontal)
    .LineStyle = xlContinuous
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThin
End With
End Sub
Код макроса обрамление 3
Sub обрамление3()
'
' обрамление3 Макрос
'
' Сочетание клавиш: Ctrl+y
'Все команды выполняются для выделенного диапазона
Selection.Borders(xlDiagonalDown).LineStyle = xlNone
Selection.Borders(xlDiagonalUp).LineStyle = xlNone
With Selection.Borders(xlEdgeLeft)
    .LineStyle = xlDouble
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThick
End With
With Selection.Borders(xlEdgeTop)
    .LineStyle = xlDouble
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThick
End With
With Selection.Borders(xlEdgeBottom)
    .LineStyle = xlDouble
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThick
End With
With Selection.Borders(xlEdgeRight)
    .LineStyle = xlDouble
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThick
End With
With Selection.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlInsideHorizontal)
    .LineStyle = xlContinuous
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThin

```

End With
End Sub

Задания для самостоятельной работы

1. Создать макрос, который объединяет выделенные ячейки и форматирует текст в этой объединенной ячейке по центру по горизонтали и по вертикали.
2. Создать макрос, который защищает лист с паролем.
3. Создать макрос, который снимает защиту листа с паролем.

Практическое занятие №40

Циклы и ветвления в языке VBA

Цель и содержание: изучить основные конструкции языка Visual Basic for Applications.

Теоретические сведения

Свойство Range возвращает объект Range, определяемый аргументами.

Используются два разных способа записи свойства Range.

Первый способ object.Range(Cell1)

Второй способ object.Range(Cell1 [,Cell2])

- object - ссылка на объект, например, на рабочий лист или на интервал ячеек.

Ссылка необязательна. По умолчанию используется активный лист;

- Cell1, Cell2 - аргументы для задания интервала ячеек. Cell1 - указание обязательно при обоих способах записи свойства Range.

Первый способ

Аргумент Cell1 задает интервал ячеек произвольного размера.

ВАЖНО

- Могут использоваться имена, определенные в таблице, или координаты ячеек, столбцов, строк или интервалов.
- Координаты задаются в стиле A1.
- Координаты и имена заключаются в кавычки.
- При задании интервалов координаты левого верхнего угла и правого нижнего угла интервала разделяются двоеточием.
- Для задания несмежных интервалов используется запятая.
- Для задания пересечения интервалов используется пробел.

Примеры записи оператора Range (1 способ)

Запись	Возвращаемый объект
ActiveSheet.Range("A1:A10")	интервал ячеек A1:A10 на активном листе
Range("A:B")	столбцы A:B
Range("налог")	интервал с именем налог
Range("1:3")	строки с первой по третью
Range("A1:C2, B10:D24")	объединение двух несмежных интервалов A1:C2 и B10:D24
Range("A1:C10 B10:D24")	пересечение двух интервалов A1:C10 и B10:D24, т.е. интервал B10:C10

Второй способ

Аргументы задают координаты интервала:

- Cell1 - единственная ячейка (строка или столбец), задающая левый верхний угол интервала;
- Cell2 - единственная ячейка (строка или столбец), задающая правый нижний угол интервала. Необязательный аргумент.

Допустимо задание аргументов переменными, выражениями, свойствами или методами, представляющими объект Range - одну ячейку, одну строку или один столбец рабочего листа.

Примеры записи оператора Range (2 способ)

Запись	Возвращаемый объект
Range("A5","D18")	интервал A5:D18
Range(Columns(1), Columns(5))	интервал, содержащий первые пять столбцов рабочего листа

ЗАПОМНИТЕ

- Если свойство Range применяется к объекту Range, то ссылка на интервал ячеек считается относительной и возвращается смещенный объект Range.

Например, если выделен интервал **C1:D5**, то запись Selection.Range("B2") возвратит ячейку **D2**.

Свойство Cells

Свойство Cells возвращает единственную ячейку рабочего листа, которая находится на пересечении строки и столбца, задаваемых целыми числами. Синтаксис object.Cells (RowIndex,ColumnIndex) object - ссылка на объект. Ссылка необязательна. По умолчанию используется активный лист; RowIndex - индекс строки; ColumnIndex - индекс столбца.

ЗАМЕЧАНИЯ

- В свойстве Cells индекс строки является первым аргументом, а индекс столбца - вторым аргументом, тогда как при задании адреса ячейки в стиле A1 сначала указывается столбец, а затем строка.
- Понятие "индекс" (Index, ColumnIndex, RowIndex) всегда подразумевает целое число, целочисленную переменную или выражение, результат вычисления которого есть целое число или может быть преобразован в целое число.

Примеры записи свойства Cells

Запись	Комментарий	Возвращаемый объект
ActiveSheet.Cells	Свойство Cells без аргументов	все ячейки активного рабочего листа
Range("C5:C10").Cells(1,1)	Свойство Cells применяется к объекту Range (относительная ссылка)	ячейка C5
Range(Cells(7,3),Cells(10,4))	Свойство Cells используется в качестве аргументов свойства Range	интервал ячеек C7:D10

Свойство Offset

Свойство Offset позволяет задавать ячейки или интервалы при помощи числа строк и колонок, которые отделяют нужную ячейку от исходной ячейки, т.е. указывая смещение относительно выбранной ячейки. Например, Range("A5").Offset(-2,1) возвращает ячейку **B3**.

Синтаксис object.Offset([RowOffset],[ColumnOffset])

- object - ссылка на объект Range. Ссылка обязательна и определяет объект, относительно которого задается смещение;
- RowOffset - смещение строки искомой ячейки относительно исходной ячейки;
- ColumnOffset - смещение столбца искомой ячейки относительно исходной ячейки.

Необязательные аргументы RowOffset и ColumnOffset - числовые выражения. Если какой-то аргумент не задан, то соответствующее смещение равно нулю.

Например, если выделен интервал **C1:D5**, то запись Selection.Offset(2,1).Select выделяет интервал **D3:E7**.

Метод Union и свойство Areas

Метод Union используется для объединения двух и более объектов Range, заданных ссылками на непересекающиеся интервалы, в один объект Range. Синтаксис Object.Union (arg1,arg2,...) object - всегда объект Application. Ссылка необязательна; arg1,arg2 - интервалы ячеек. Количество аргументов произвольно.

Обязательно наличие хотя бы двух аргументов.

Например, оператор Union(Range("A1:C5"),Range("B10:D12")).Select выделяет несмежные интервалы **A1:C5** и **B10:D12**.

Свойство Areas выполняет обратное действие, разделяя объединенные интервалы на несколько объектов Range. Синтаксис Object.Areas(index)

■ object - ссылка на объект Range, состоящий из нескольких интервалов; ■ index - номер интервала в объекте. Аргумент необязателен. Примеры

Оператор	Комментарий	Результат
p=Union (Range("A1:C5"), Range("B10:D12")).Areas(2).Count	Если аргумент задан, то свойство Areas возвращает интервал - объект Range, определенный индексом интервала	равен девяти, так как во втором интервале ровно 9 ячеек
p=Union(Range("A1:C5"), Range("B10:D12")).Areas.Count	Свойство Areas без аргументов рассматривает каждый из несмежных интервалов как элемент коллекции объектов Range	равен двум, так как объект, определенный методом Union, состоит из двух областей коллекции из двух элементов
p=Range("B10:D12").Areas.Count		равен единице, так как объект Range представляет один элемент коллекции

Свойства Column и Row (R/O Integer)

Свойства возвращают целое число, показывающее индекс первого столбца или первой строки соответственно для заданного объекта. Синтаксис свойств object.Column object Row object - обязательная ссылка на объект Range.

Например, запись Range("C5").Column возвращает число 3, а запись Range("C5").Row возвращает число 5.

Свойства Columns и Rows

Свойство Columns (не путайте со свойством Column!) возвращает объект Range, представляющий колонку или коллекцию колонок в объекте, к которому это свойство было применено. Синтаксис Object.Columns(index)

object - ссылка на объект. Указание необязательно, по умолчанию используется активный рабочий лист; index - индекс колонки в объекте.

Например, запись Columns(1) возвращает колонку **A** активного рабочего листа, а запись Range("C1:D5").Columns(1) возвращает колонку **C** заданного интервала, а именно, ячейки **C1:C5**.

ВАЖНО

- Если не указан индекс колонки, то возвращаются все колонки объекта в виде объекта Range.
- Индекс колонки можно указывать числом или буквой, при этом буква заключается в кавычки. Ссылки Columns(2) и Columns("B") указывают на одну и ту же колонку **B**.

Свойство Rows (не путайте со свойством Row!) возвращает объект Range, представляющий строку или коллекцию строк в объекте, к которому это свойство было применено.

Синтаксис Object.Rows(index)

- object - ссылка на объект. Указание необязательно, по умолчанию используется активный рабочий лист; index - индекс строки в объекте.

ВАЖНО

Если не указан номер строки, то возвращаются все строки объекта в виде объекта Range.

Например, оператор nr=Selection.Rows(Selection.Rows.Count).Row позволяет получить номер последней строки в выделенном интервале ячеек.

Свойство CurrentRegion

Текущий регион (CurrentRegion) - это диапазон ячеек, ограниченный пустыми строками и колонками или сочетанием пустых строк, колонок и границ рабочего листа. Пример

В процедуре сравниваются значения первой ячейки первой строки и первой ячейки каждой следующей строки заполненного данными интервала, включающего первую ячейку. Если значения совпадают, то очередная строка удаляется.

Предполагается, что данные начинаются с ячейки **A1** и занимают несколько строк и столбцов, при этом расположены не плотно, т.е. внутри интервала с данными могут находиться пустые строки или пустые столбцы. Анализируются только строки заполненного данными интервала ячеек вокруг ячейки **A1**, не содержащего пустых строк и столбцов.

Свойства, связанные с шириной и высотой ячейки

Свойства	Примеры и комментарии
ColumnWidth (R/W Variant)	Возвращает или изменяет ширину колонки в единицах, эквивалентных одному символу в стиле Обычный (Normal). Шрифт стиля по умолчанию Arial Cyr и размер шрифта 10.
	Range("A1").ColumnWidth=15 устанавливает ширину колонки A в 15 символов

Width (R/O Variant)	Возвращает ширину интервала ячеек в пунктах. Range("A1").Width возвращает значение 93.75, если ширина колонки 15 символов, шрифт Times New Roman, размер шрифта 12 пунктов (72 пункта равны 1 дюйму или приблизительно 2,54 см). Debug.Print Range("A1:C3").ColumnWidth распечатает значение 8.43, а оператор Debug.Print Range("A1:C3").Width распечатает значение 144, если для колонок установлена стандартная ширина, шрифт Arial Cyr и размер шрифта 10
RowHeight (R/W Variant)	Возвращает или изменяет высоту строк интервала в пунктах. ActiveCell.RowHeight = 14 устанавливает высоту строки, в которой находится активная ячейка, в 14 пунктов
Height (R/O Variant)	Возвращает суммарную высоту интервала строк, зависящую от названия и размера шрифта. Если шрифт Arial Cyr и размер шрифта 10, то Debug.Print Range("A1").Height распечатает 12,75 и Debug.Print Range("A1:C3").Height распечатает 38,25
WrapText (R/W Boolean)	Range("A1").WrapText=True Значение True разбивает текст ячейки на несколько строк, если ширина столбца недостаточна для размещения текста целиком

ЗАМЕЧАНИЕ

■ Свойства Width и Height имеют статус Read-Only для объектов Range, но для других объектов, например, для объекта Window, они имеют статус Read-Write.

Свойство Address

Address – позволяет вернуть адрес текущего диапазона. Этому свойству можно передать много параметров – для определения стиля ссылки, абсолютного или относительного адреса для столбцов и строк, по отношению к чему этот адрес будет относительным и т.п. Свойство доступно только для чтения. AddressLocal – то же самое, но с поправкой на особенности локализованных версий Excel.

На практике встречается множество ситуаций, когда адрес ячейки нужно разобрать на части и вернуть из него имя столбца или номер строки. Это очень просто сделать при помощи строковых функций

Методы Select и Activate

Метод Select выделяет интервал ячеек.

Синтаксис object.Select(Replace)

- object - выделяемый объект типа Range. Ссылка на объект обязательна; ■
- Replace - для расширения выделения аргумент устанавливается в False. Если аргумент не задан или принимает значение True, то вместо старой области выделения создается новая область выделения.

Необязательный параметр.

Метод Activate активизирует единственную ячейку. Синтаксис object.Activate

- object - активизируемая ячейка. Ссылка на объект обязательна. Примеры

Оператор

Активная
ячейка

Range("C7:E9").Select	C7
Range("C7:E9").Offset(1,1).Activate	D8
Range("C7:E9").Activate	C7
Range("C7:E9").Cells(2,1).Activate	C8

ЗАМЕЧАНИЯ

Активная ячейка выделяется фоном среди всех выделенных ячеек.

- Метод Select выделяет интервал ячеек, тогда как метод Activate активизирует только одну ячейку.
- При использовании метода Select первая ячейка интервала становится активной.
- Если выделена только одна ячейка, то она является активной и свойства ActiveCell и Selection возвращают одну и ту же ячейку (объект Range).

Метод Clear

Очищает интервал ячеек, изменяя, таким образом, свойство Value каждой ячейки интервала.

Задание 1 Создать 2 макроса.

1. Макрос **оформление**. Выполнить оформление для созданной таблицы. Залить первую строку и задать для нее жирный шрифт.
2. Макрос **ряд**. Для выделенного диапазона (столбца) получить натуральный ряд.

Первоначальный вариант макроса оформление.

Sub оформление()

Selection.Borders(xlDiagonalDown).LineStyle = xlNone

Selection.Borders(xlDiagonalUp).LineStyle = xlNone

With Selection.Borders(xlEdgeLeft)

.LineStyle = xlDouble

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

With Selection.Borders(xlEdgeTop)

.LineStyle = xlDouble

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

With Selection.Borders(xlEdgeBottom)

.LineStyle = xlDouble

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

With Selection.Borders(xlEdgeRight)

.LineStyle = xlDouble

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThick

End With

```

With Selection.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThin
End With
With Selection.Borders(xlInsideHorizontal)
    .LineStyle = xlContinuous
    .ColorIndex = xlAutomatic
    .TintAndShade = 0
    .Weight = xlThin
End With
Range("E6:H6").Select
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 65535
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
Selection.Font.Bold = True
End Sub

```

Заливка строки в данной макросе будет выполняться неправильно для произвольной таблицы, т.к. выделение привязывается к конкретному диапазону. (выделенная строка)

Для решения этой задачи надо запомнить первоначально выделенный диапазон в объектной переменной в начале макроса.

```

Dim r As Range
Set r = Selection

```

Перед заливкой надо выделить первую строку этого диапазона, а затем выполнить заливку. Для этого надо вместо выделенной строки в тексте макроса выполнить команду:

```

r.rows(1).Select

```

Окончательный вариант макроса ***оформление***.

```

Sub оформление()
    Dim r As Range
    Set r = Selection
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlDouble
    End With

```

```

        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlDouble
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThick
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThin
    End With
    r.Rows(1).Select    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 65535
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
    Selection.Font.Bold = True
End Sub

```

Для получения макроса **ряд** натуральные числа следует получить через команду прогрессия.

Первоначальный вариант макроса таблица

```

Sub ряд()
    ActiveCell.FormulaR1C1 = "1"
    Range("E6:E24").Select
        Selection.DataSeries    Rowcol:=xlColumns, Type:=xlLinear,
Date:=xlDay, _
        Step:=1, Trend:=False
End Sub

```

Для создания макроса для произвольного диапазона следует исключить конкретные адреса.

Окончательный вариант макроса ряд

```

Sub ряд()
    Dim r As Range
    Set r = Selection „Сохранение выделенного диапазона    n = r.Row 'Номер верхнего
ряда    m = r.Column 'Номер левого столбца
    k = r.Rows.Count 'Количество выделенных рядов
    Cells(n, m).Select
    ActiveCell.FormulaR1C1 = "1"
    Range(Cells(n, m), Cells(n + k, m)).Select

```

```

        Selection.DataSeries Rowcol:=xlColumns, Type:=xlLinear,
Date:=xlDay, _
        Step:=1, Trend:=False
End Sub

```

Задание 2

В некоторых командах надо указывать конкретный адрес диапазона. Для этого используется свойства Address.

Создать макрос **сумма**, который суммирует числа в выделенном диапазоне и помещает результат под этим диапазоном в последний столбец.

Заполним диапазон какими-нибудь числами. В макросе вызовем функцию суммирования и укажем диапазон суммирования.

В результате макрос будет иметь вид:

```

Sub Сумма ()
    Range("K20").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-14]C[-5]:R[-1]C[-1])"
    Range("K21").Select
End Sub

```

Из текста макроса видно, что преобразовать данный макрос к нужной форме достаточно сложно.

Для создания данного макроса надо использовать функцию СУММ(диапазон суммирования). Для определения диапазона суммирования необходимо выделить этот диапазон перед началом записи макроса, а затем в макросе определить адрес этого диапазона, его местоположение и размеры, для определения местоположения ячейки, где будет находится сумма.

Макрос будет иметь вид:

```

Sub Сумма ()
    s = Selection.Address „определение адреса диапазона    n = Selection.Row „номер
ряда начальной ячейки диапазона    m = Selection.Column „номер колонки начальной
ячейки диапазона    a = Selection.Rows.Count „количество рядов диапазоне    b =
Selection.Columns.Count „количество столбцов диапазоне
    Cells(n + a, m + b).FormulaLocal = "=СУММ(" + s + ")" End Sub

```

В макросе используется свойство FormulaLocal, потому, что используется функция на русском языке.

Еще один пример использования свойства Address.

Создать макрос **диаграмма**, в котором построить диаграмму по выделенному диапазону и разместить эту диаграмму на отдельном листе.

Создаем диапазон с числовыми данными.

До записи макроса выделяем диапазон. В макросе указываем тип диаграммы (объемная гистограмма), убираем легенду и размещаем диаграмму на отдельном листе.

Записанный макрос имеет вид:

```

Sub Диаграмма()
    ActiveSheet.Shapes.AddChart.Select
    ActiveChart.SetSourceData Source:=Range("'Лист1'!$A$1:$A$7")
    ActiveChart.ChartType = xl3DColumnClustered
    ActiveChart.SetElement (msoElementLegendNone)
    ActiveChart.Location Where:=xlLocationAsNewSheet End Sub

```

В данном макросе конкретный адрес диапазона данных находится в выделенной строке. Поэтому надо определить адрес выделенного диапазона и подставить его в команду.

```

Sub Диаграмма()
    s = Selection.Address

```

```
ActiveSheet.Shapes.AddChart.Select  
ActiveChart.SetSourceData Source:=Range(s)  
ActiveChart.ChartType = xl3DColumnClustered  
ActiveChart.SetElement (msoElementLegendNone)  
ActiveChart.Location Where:=xlLocationAsNewSheet
```

End Sub

Задания для самостоятельной работы

1. Создать макрос итог, которым для созданной таблицы выполнить форматирование:

1. Для столбцов выполнить автоподбор ширины.
2. Выполнить обрамление таблицы.
3. Для шапки выполнить заливку и задать полужирный шрифт.
4. для строки над таблицей задать объединение ячеек и центрирование по вертикали и по горизонтали.

5. Для последнего столбца задать денежный формат.

6. Под последней колонкой поместить функцию суммирования и выделить ее жирным шрифтом.

7. Объединить ячейки перед суммой на написать туда слово ИТОГО жирным шрифтом.

2. Создать макрос, который строит график функции по заданным данным. На графике изобразить линии вертикальной и горизонтальной сетки. Убрать легенду.

Практическое занятие №41

Работа с символьными данными на языке VBA

Цель и содержание: изучить принципы организации взаимодействия Visual Basic for Applications с содержимым документа.

Теоретические сведения

При записи строковой константы ее надо заключать в кавычки. В выражениях, наряду с константами могут встречаться и функции.

Функция (function) – это встроенная формула, выполняющая действия над выражениями и генерирующая значение. Функция всегда возвращает значение, которое VBA вставляет в программу в том месте, где появляется имя функции. Функции VBA делятся на несколько групп в зависимости от типа операции или вычисления, которое они выполняют.

Чтобы использовать функцию, надо просто ввести имя функции в оператор VBA вместе с любыми аргументами, которые требуются для этой функции, в том месте в операторе, где необходимо использовать результат функции. [Помещение имени функции в оператор VBA для активизации функции называют *вызовом (calling)* функции.] При использовании функций в выражениях существуют следующие правила:

- Можно использовать результат функции как часть выражения.
- Можно присваивать результат функции какой-либо переменной.
- Можно использовать результат функции для предоставления значения в список аргументов другой процедуры или функции.
- Функции имеют списки аргументов, заключенные в круглые скобки.

В основном функцию можно использовать для предоставления значения в любом месте в любом операторе VBA, где может быть оправданно использование значения константы или переменной. Тип данных значения, возвращаемого функцией, зависит от этой конкретной функции.

Большинство функций возвращают значения типа Variant, хотя некоторые функции возвращают данные определенных типов, таких как String, Double и Integer. VBA во многих случаях может автоматически преобразовывать результат какой-либо функции в данные типа, совместимого с другими типами значений в выражении, содержащем эту функцию, точно, как VBA преобразует типы данных в присваивания переменных и вычислениях выражений.

Встроенные функции VBA делятся на несколько категорий на основе общего назначения функций (математические, преобразования данных, даты и времени, строковые и работы с диском). Далее обсуждаются категории функций и описываются их действия. Большинство функций VBA, такие как математические функции, являются довольно ясными из их названия и не требуют подробного объяснения. Другие функции, такие как функции преобразования типа данных и обработки строк, описаны более подробно. VBA-функции обработки строк имеют важное значение, поэтому рассмотрены способы их использования.

Математические функции

VBA предоставляет стандартный набор математических функций. В табл. 2 приведены математические функции, имеющиеся в VBA. В этой таблице N означает любое численное выражение; все аргументы функций являются обязательными, если только не указано иначе.

Таблица 2

Функции (аргументы)	Возвращает /действие
------------------------	----------------------

Abs(N)	Возвращает абсолютное значение N.
Atn(N)	Возвращает арктангенс N как угол в радианах.
Cos(N)	Косинус угла N, где N – это угол, измеренный в радианах.
Exp(N)	Возвращает константу e , возведенную в степень N. (e – это основание натуральных логарифмов и она (приблизительно) равна 2,718282).
Fix(N)	Возвращает целую часть N. Fix не округляет число, а отбрасывает любую дробную часть. Если N является отрицательным, Fix возвращает ближайшее отрицательное целое большее, чем или равное N.
Int(N)	Возвращает целую часть N. Int не округляет число, а отбрасывает любую дробную часть. Если N является отрицательным, Int возвращает ближайшее отрицательное целое меньшее, чем или равное N.
Log(N)	Возвращает натуральный логарифм N.
Rnd(N)	Возвращает случайное число; аргумент является необязательным. Используйте функцию Rnd только после инициализации VBA-генератора случайных чисел оператором Randomize.
Sgn(N)	Возвращает знак числа: -1, если N – отрицательное; 1, если N – положительное; 0, если N равно 0.
Sin(N)	Возвращает синус угла; N – это угол, измеренный в радианах.
Sqr(N)	Возвращает корень квадратный из N. VBA отображает ошибку времени исполнения, если N – отрицательное.
Tan(N)	Возвращает тангенс угла; N – угол в радианах.

Функции **Fix** и **Int** укорачивают целые, то есть они отбрасывают дробную часть числа без округления. Единственное различие между функциями **Fix** и **Int** – это то, как они обрабатывают отрицательные числа.

Дополнительные тригонометрические функции можно выводить из базовых математических функций VBA. Например, если необходимо вычислить котангенс угла, для его нахождения можно использовать формулу **1/Tan(x)**.

Примеры использования математических функций:

```
gipot=Sqr(kat2^2+kat2^2) chislo=Int(Rnd(1)*100)
```

Функции преобразования данных

Visual Basic предоставляет несколько функций для преобразования одного типа данных в другой. Надо использовать эти функции для устранения ошибок несовпадения типов и обеспечения явного контроля за типами данных в выражениях.

Например, при получении сообщения об ошибке несовпадения типов в определенном выражении можно преобразовать значения в выражении в типы, совместимые друг с другом, используя функции преобразования. Или же можно сохранять результат выражения в диапазоне численного типа **Single** (большинство численных выражений имеют результатом значение типа **Double**); в таком случае следует использовать функцию **CSng** для преобразования результата выражения в число типа **Single**.

В табл. 3 приведены функции преобразования данных в VBA. В этой таблице N – это любое численное, S – любое строковое, а E – выражение любого типа. Аргументы каждой функции являются обязательными, если не указано иначе.

Таблица 3

Функция (аргументы)	Возвращает/действие
Asc(S)	Возвращает число кода символа, соответствующее первой букве строки S. Буква "А", например, имеет код символа 65.
Chr(N)	Возвращает строку из одного символа, соответствующего коду символа N, который должен быть числом между 0 и 255, включительно. Код символа 65, например, возвращает букву "А".
Format(E, S)	Возвращает строку, содержащую значение, представленное выражением E, в формате в соответствии с инструкциями, содержащимися в S.
Hex(N)	Возвращает строку, содержащую шестнадцатичное представление N.
Oct(N)	Возвращает строку, содержащую восьмичное представление N.
RGB(N, N, N)	Возвращает целое типа Long , представляющее значение основных цветов изображения. N в каждом аргументе должно быть целым в диапазоне 0 – 255, включительно. Аргументы (слева направо) – это значения для красного, зеленого и синего цвета.
Str(N)	Возвращает строку, эквивалентную численному выражению N.
Val(S)	Возвращает численное значение, соответствующее числу, представленному строкой S, которая должна содержать только цифры и одну десятичную точку, иначе VBA не может преобразовать ее в число. Если VBA не может преобразовать строку в S, то функция Val возвращает 0.
CBool(N)	Возвращает Boolean-эквивалент численного выражения N.
CByte(E)	Возвращает численное значение типа Byte (от 0 до 255); E – любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
CCur(E)	Возвращает численное значение типа Currency ; E – любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
CDate(E)	Возвращает значение типа Date . E может быть любым допустимым выражением (строкой или числом), представляющим дату в диапазоне 1/1/100 – 12/31/9999, включительно.
CDbl(E)	Возвращает численное значение типа Double ; E – любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
CInt(E)	Возвращает численное значение типа Integer ; E – любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
CLng(E)	Возвращает численное значение типа Long ; E – любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
CSng(E)	Возвращает численное значение типа Single ; E – любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
CStr(E)	Возвращает значение типа String ; E – любое допустимое численное или строковое выражение.
CVar(E)	Возвращает значение типа Variant ; E – любое допустимое численное или строковое выражение.

Наиболее часто используемые функции – это функции (объединенные в конце табл. 3 в группу), начинающиеся с буквы C (от слова *conversion*), за которыми следует сокращение имени типа: **CStr**, **CSng**, **Cdbl** и так далее.

Примеры использования функций преобразования данных: `d=CDat("1.9.2006")`
`text=CStr(36*3)`

Функции даты и времени

Язык VBA содержит специальный тип данных **Date**, предназначенный для представления значений даты и времени. Внутри программ значения типа **Date** представлены как числа с плавающей запятой:

- целая часть значения отображает дату как количество дней, прошедшее с 30 декабря 1899 года;
- дробная часть значения представляет время в виде доли 24-часового дня (например, .25 – это 6 часов утра, 5 – это полдень и т.д.);  дробное число без целой части отображает время без даты.

Для пользователя или программиста представленные таким образом значения даты и времени являются бесполезными. К счастью, в языке VBA предусмотрены инструменты, позволяющие преобразовывать эти значения к более знакомому пользователям виду. Исходное "сырое" представление значений даты и времени в виде чисел с плавающей запятой скрыто от посторонних глаз и используется программой только в процессе проведения внутренних вычислений.

Язык VBA способен распознавать значения даты и времени, представленные практически в любом стандартном формате. Чтобы указать на текстовые константы как на значения даты/времени, необходимо заключить их в символы #. Используйте обычный оператор присваивания, чтобы присвоить переменной типа **Date** значение даты/времени:

```
dt = #01.05.2003#
```

VBA-функции даты и времени обычно используются для получения текущей даты и времени, разбиения значения даты на ее составляющие части или для преобразования строк и чисел в значения типа **Date**. В таблице 4 приведены VBA-функции даты и времени и их действие. В этой таблице N – любое допустимое численное выражение, а D – любое допустимое выражения типа **Date** (включая значения типа **Date**, числа или строки, которые VBA может преобразовать в дату). Все аргументы функций в этой таблице являются обязательными, если не указано иначе.

Таблица 4

Функции(аргументы)	Возвращает/действие
Date	Возвращает системную дату. Можно также использовать эту функцию как процедуру для установки системных часов компьютера. Более подробно можно узнать из справочной системы VBA.
Time	Возвращает системное время компьютера как значение типа Date . Можно также использовать эту функцию как процедуру для установки системных часов. Более подробно можно узнать из справочной системы VBA.
Now	Возвращает системную дату и время.
Year(D)	Возвращает целое, являющееся частью выражения типа Date и содержащее год. Год возвращается как число между 100 и 9999.
Month(D)	Возвращает целое, являющееся частью выражения типа Date , содержащее месяц. Месяц возвращается как число между 1 и 12, включительно.

Day(D)	Возвращает целое, являющееся частью выражения типа Date и содержащее день. День возвращается как число между 1 и 31, включительно.
Weekday(D)	Возвращает целое, содержащее день недели для выражения типа Date . День недели возвращается как число между 1 и 7, включительно; 1 – это воскресенье, 2 – понедельник и так далее.
WeekdayName(N1,B, N)	Возвращает строку с наименованием дня недели, номер которого задается параметром N1.
Hour(D)	Возвращает целое, содержащее часы как часть времени, содержащегося в выражении типа Date . Часы возвращаются как число между 0 и 23, включительно. Если выражение D не содержит значения времени, то Hour возвращает 0.
Minute(D)	Возвращает целое, содержащее минуты как часть времени в выражении типа Date . Минуты возвращаются как число между 0 и 59, включительно. Если выражение D не содержит значения времени, Minute возвращает 0.
Second(D)	Возвращает целое, содержащее секунды как часть времени в выражении типа Date . Секунды возвращаются как число между 0 и 59, включительно. Если выражение D не содержит значения времени, Second возвращает 0.
DateAdd(S, N, D)	Возвращает значение [тип Variant (Date)], содержащее дату, к которой добавлен заданный интервал времени.
DateDiff(S, D1, D2[, N1 [, N2]])	Возвращает значение [тип Variant (Long)] числа временных интервалов между двумя определенными датами.
DatePart(S, D,[,N1 [, N2]])	Возвращает определенную часть [тип Variant (Integer)] заданной даты.
DateSerial(N, N,N)	Возвращает значение последовательной даты для заданной даты. Слева направо аргументы представляют год, месяц и день. Аргумент года должен быть целым числом между 100 и 9999, месяца – между 1 и 12, дня – между 1 и 31 (все диапазоны являются включающими).
TimeSerial(N, N,N)	Возвращает значение последовательного времени. Слева направо аргументы представляют часы, минуты и секунды. Аргумент часов должен быть целым числом между 0 и 23, аргументы минут и секунд должны оба быть числами 0 и 59 (все диапазоны являются включающими).
DateValue(E)	Возвращает значение типа Date , эквивалентное дате, заданной аргументом E, который должен быть строкой, числом или константой, представляющей дату.
TimeValue(E)	Возвращает значение типа Date , содержащее время, заданное аргументом E, который может быть строкой, числом или константой, представляющей время.
Timer	Возвращает число, представляющее количество секунд от полуночи в соответствии с системным временем компьютера.

Примеры использования функций даты и времени:

god=year(Date()) nomer_den=WeekDay(#25/12/2006#)-1

Строковые функции

Строковые функции VBA часто применяются для нахождения заданных строк внутри других строк, для сравнения одной строки с другой и копирования выбранных частей строк. Строковые функции VBA используются довольно часто, потому что строковые данные очень важны и встречаются в каждом приложении VBA – Word, Excel,

Access или другом host-приложении VBA. Часто необходимо манипулировать строковыми данными, полученными как пользовательский ввод функцией **InputBox**. В других случаях строковые данные появляются в коде VBA как имена файлов для документов Word, рабочих книг Excel, баз данных Access и других типов данных, сохраняемых в файлах на дисках.

В Word важность манипулирования строковыми данными очевидна: данные, сохраняемые в документах Word, в основном являются ничем иным, как строковыми данными. Строковые данные также важны в Excel в качестве имен рабочих листов, именованных диапазонов данных и так далее. В этом разделе приводятся имеющиеся в VBA строковые функции; в последующем разделе более подробно описывается, как использовать наиболее важные и полезные строковые функции.

В табл. 5, где приведены основные строковые функции VBA, N – это любое допустимое численное выражение, а S – это любое допустимое строковое выражение. Все аргументы функций являются обязательными, если не указано иначе.

Таблица 6

Функция(аргумент)	Возвращает/действие
InStr([N1,] S1, S2 [, N2])	Возвращает положение S2 в S1. N1 – начальное положение для поиска; N2 определяет тип сравнения. N1 и N2 необязательны. Если N2 опускается, то для поиска используется текущая установка Option Compare .
InStrRev(S1, S2 [, N1[, N2]])	Возвращает позицию появления строки S2 внутри S1, в направлении от конца (или N1) к началу строки. N2 определяет тип сравнения. Если N2 опускается, то для поиска используется текущая установка Option Compare .
LCase(S)	Возвращает строку (тип String), содержащую копию S со всеми символами верхнего регистра, преобразованными в символы нижнего регистра.
Left(S, N)	Возвращает строку; копирует N символов из S, начиная с левого крайнего символа S.
Len(S)	Возвращает число символов в S, включая начальные и конечные пробелы.
LTrim(S)	Возвращает копию строки S после удаления символов пробела из левой части строки (начальные пробелы).
Mid(S, N1, N2)	Возвращает строку; копирует N2 символов из S, начиная с позиции символа в S, заданной аргументом N1. N2 является необязательным; если N2 опущен, то Mid возвращает все символы в строке S от позиции N1 до конца строки.
Right(S, N)	Возвращает значение типа String ; копирует N символов из S, начиная с правого крайнего символа S. Например, Right(outright, 5) возвращает строку right .
RTrim(S)	Возвращает копию строки S после удаления символов пробела из правой части строки (конечные символы).
Space(N)	Возвращает строку пробелов длиной N символов.
StrComp(S1, S2, N)	Сравнивает S1 с S2 и возвращает число, обозначающее результат сравнения: -1, если S1 < S2; 0, если S1 = S2; и 1, если S1 > S2. N является необязательным и указывает, следует ли выполнять сравнение с учетом регистра. Если N опускается, строки сравниваются с использованием текущей установки Option Compare .

StrConv(S, N)	Возвращает строку, преобразованную в новую форму в зависимости от числового кода, заданного аргументом N. VBA предоставляет внутренние константы для использования с функцией StrConv ; наиболее полезными являются: vbProperCase (преобразует строку так, что каждая буква, начинающая слово, становится заглавной), vbLowerCase (преобразует строку в буквы нижнего регистра) и vbUpperCase (преобразует строку в буквы верхнего регистра).
String(N, S)	Возвращает строку длиной N символов, состоящую из символа, заданного первым символом в S. Например, String(5, "x") возвращает строку "xxxxx".
Trim(S)	Возвращает копию строки S после удаления начальных и конечных символов пробела из этой строки.
UCase(S)	Возвращает S со всеми символами нижнего регистра, преобразованными в символы верхнего регистра.

Несколько перечисленных в табл. 5 функций преобразования типа данных относятся также к манипулированию строками: **Chr**, **Format**, **CStr**, в частности.

Примеры использования строковых функций: n_probel=Instr(—Найти пробелll, ll —) perv_simv=Left(—Выделение текстаll, 1)

Пользовательские функции

Создание пользовательских функций или, как их иногда еще называют, UDF-функций (User Defined Functions) принципиально не отличается от создания макроса в обычном программном модуле. Разница только в том, что макрос выполняет последовательность действий с объектами книги (ячейками, формулами и значениями, листами, диаграммами и т.д.), а пользовательская функция - только с теми значениями, которые мы передадим ей как аргументы (исходные данные для расчета).

При написании функций-процедур для использования в качестве UDF в рабочих листах Excel необходимо знать несколько фактов, помимо общих требований для определенных пользователем функций:

- Определенные пользователем функции, которые будут использоваться в Excel, не должны иметь имена, похожие на записи ссылок на ячейку (например, A2; C1B5);
- Любые строковые данные, возвращаемые из VBA в Excel, не должны иметь более 255 символов в длину. Если UDF возвращает строку, имеющую больше 255 символов в длину, в ячейку рабочего листа, Excel укорачивает строку до максимальной длины 255 символов перед вставкой в ячейку;
- При написании UDF, возвращающей значение даты для Excel убедитесь, что задаете тип результата функции как Date. Excel применяет формат Date для результата функции в ячейке рабочего листа только, если результат имеет VBA-тип Date.

Структура пользовательской функции

Function имя_функции(arg1 As тип, arg2 As тип ...) As тип

Тело функции

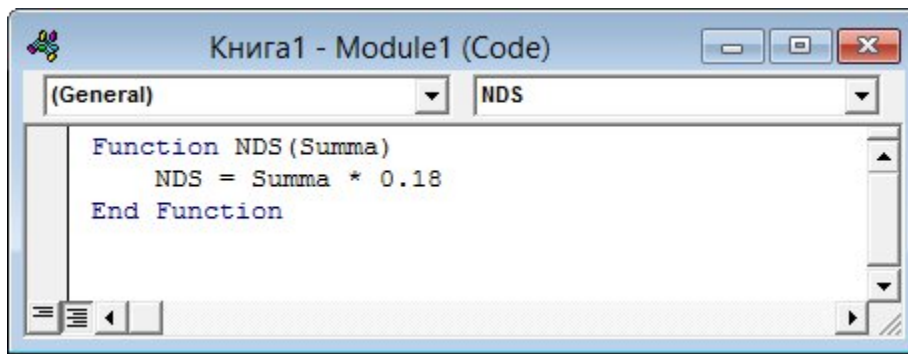
имя_функции = выражение End Function

Досрочный выход из функции выполняется с помощью оператора Exit

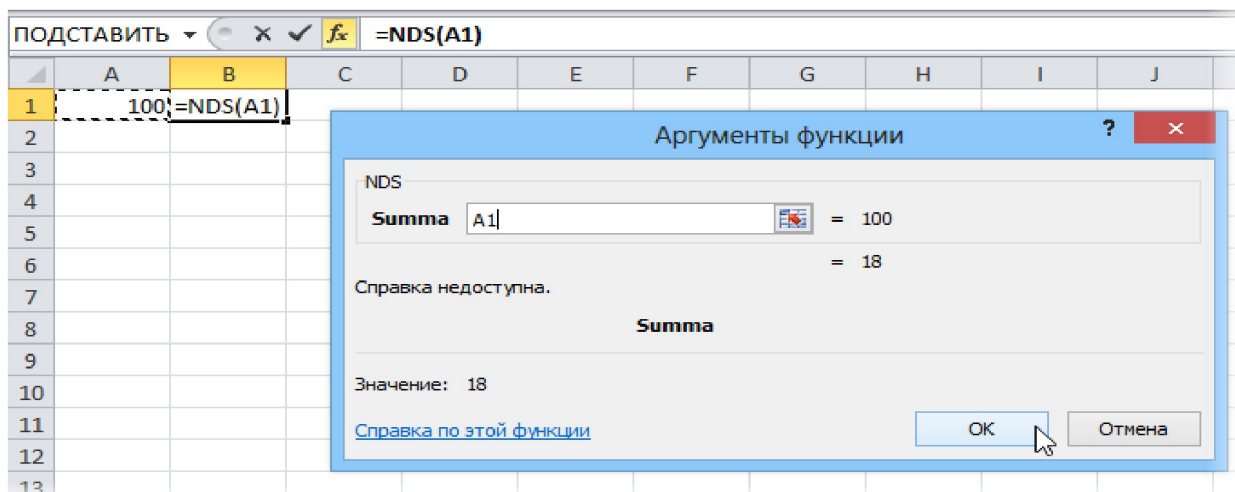
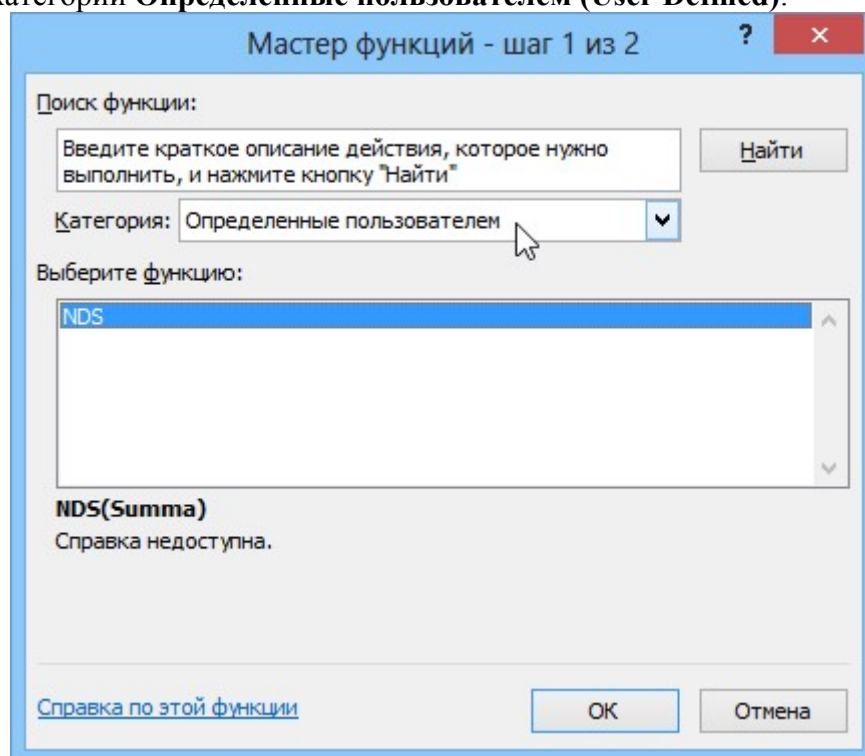
Function

Порядок создания пользовательской функции

Чтобы создать пользовательскую функцию для расчета, например, налога на добавленную стоимость (НДС) откроем редактор VBA, добавим новый модуль через меню **Insert - Module** и введем туда текст нашей функции:



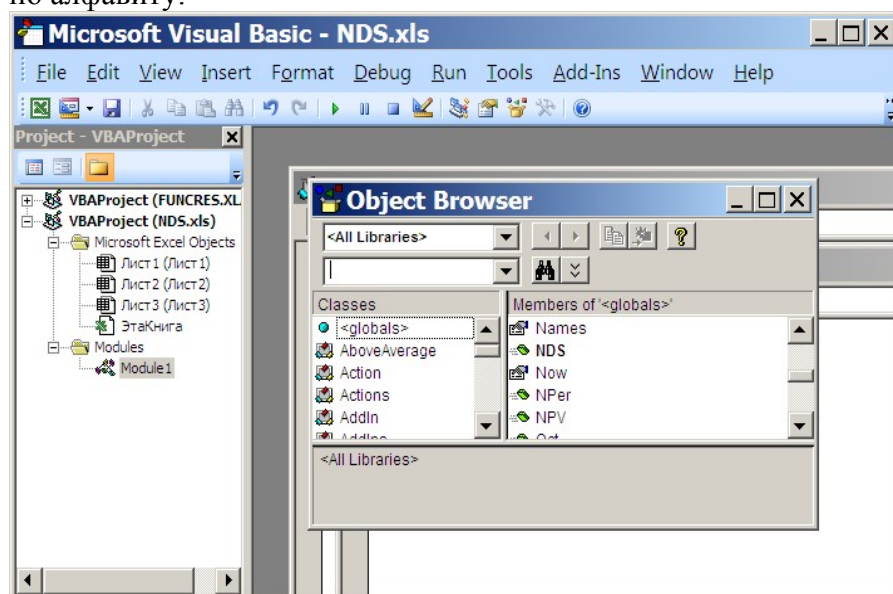
Обратите внимание, что в отличие от макросов функции имеют заголовок **Function** вместо **Sub** и непустой список аргументов (в нашем случае это *Summa*). Если тип переменных и функции не задан, то тип интерпретируется как **Variant**. После ввода кода наша функция становится доступна в обычном окне Мастера функций (**Вставка - Функция**) в категории **Определенные пользователем (User Defined)**:



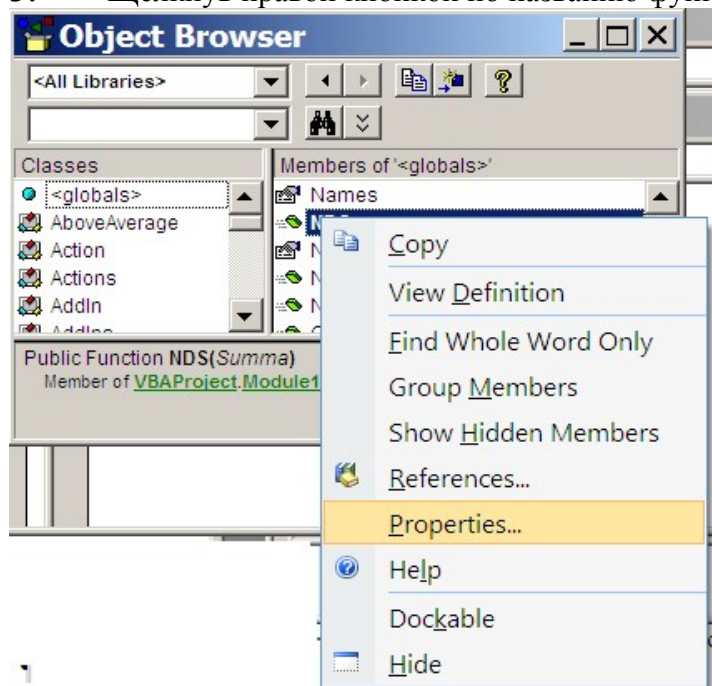
После выбора функции выделяем ячейки с аргументами (с суммой, для которой надо посчитать НДС) как в случае с обычной функцией:

Обратите внимание, что в диалоговом окне выводится : «Справка недоступна». Для того, чтобы отображать справку для пользовательской функции надо:

1. В редакторе VBA открыть окно Object Browser **View → Object Browser**.
2. В списке выбрать пользовательскую функцию. Эта функция располагается по алфавиту.



3. Щелкнув правой кнопкой по названию функции, выбрать Properties ...



4. В диалоговом окне Member Options ввести пояснения в текстовое окно Description.

Member Options

Name:

Description:

Help File:

Help Context ID:

OK Cancel Help

ВАЖНО

Чтобы пояснения отображались в окне Мастер функций, надо Перед пользовательской функции указать доступ Public:

```
Public Function NDS(Summa)
    NDS = Summa * 0.18
End Function
```

При вызове функции в окне будет отображаться:

Мастер функций - шаг 1 из 2

Поиск функции:

Найти

Категория:

Выберите функцию:

NDS

NDS(Summa)
Вычисление НДС от суммы.

[Справка по этой функции](#)

OK Отмена

Рассмотрим примеры пользовательских функций.

Задание 1

Создать новую книгу Функции. Набрать и отладить функции Vozr и ИНИЦИАЛЫ/ Функция, вычисляющая возраст по дате рождения.

```
Function Vozr(d As Date) As Integer
    Dim k As Integer
    k = Date - d
    Vozr = Int(k / 365.25)
End Function
```

Функция получения фамилии и инициалов

```

Function ИНИЦИАЛЫ(ФИО As String) As String
Dim fam As String, im As String, ot As String
Dim n1_pr As Integer, n2_pr As Integer
ФИО = StrConv(ФИО, vbProperCase) 'сделать первые буквы в словах
                                'прописными
n1_pr = InStr(ФИО, " ") 'найти первый пробел
fam = Left(ФИО, n1_pr - 1) 'выделить фамилию
n2_pr = InStr(n1_pr + 1, ФИО, " ") 'найти второй пробел
im = Mid(ФИО, n1_pr + 1, 1) 'выделить первую букву имени
ot = Mid(ФИО, n2_pr + 1, 1) 'выделить первую букву отчества
ИНИЦИАЛЫ = fam + " " + im + "." + ot + "."
End Function

```

Обеспечить вывод справки по данным функциям.

Продемонстрировать работу функций преподавателям.

Задания для самостоятельной работы

Вариант 1

1. Даны длина катета, найти длину гипотенузы
2. Найти количество дней между датами.
3. Написать функцию вставляющую подстрока в строку с заданной позицией.

Вариант 2

1. Дана гипотенуза и катет, найти второй катет
2. Найти количество месяцев между датами.
3. Дана строка, содержащая название книги, а затем фамилия и инициалы автора.

Получить фамилию автора.

Вариант 3

1. Дан радиус окружности найти ее площадь
2. Найти количество недель между датами.
3. Дана строка, содержащая фамилию и группу. Получить строку фамилия и курс.

Вариант 4

1. Дан радиус найти длину окружности
2. Найти количество кварталов между датами.
3. Написать функцию, удаляющую из строки заданную подстроку.

Вариант 5

1. Даны стороны прямоугольника найти площадь прямоугольника
2. Сколько дней прошло с начала года
3. Дана строка с информацией о фирме содержащей расчетный счет. Расчетный счет начинается со знака №. Выделить расчетный счет фирмы.

Вариант 6

1. Даны стороны прямоугольника. Найти периметр прямоугольника
2. Сколько дней осталось до конца года
3. Дано вещественное число получить из него цену в виде целая часть руб дробная часть из двух цифр коп.

Вариант 7

1. Даны длины сторон равнобедренной трапеции. Найти высоту трапеции.
2. Определить день недели первого дня заданного года.
3. Дана строка с ценой числа руб. число коп. Преобразовать ее в вещественное число.

Вариант 8

1. Даны координаты начала и конца прямой линии.
2. Определить день недели сотого дня заданного года.

3. Дана дата в виде строки в американском формате месяц/день/год. Переделать ее в русский формат день.месяц.год.

Вариант 9

1. Найти координаты середины линии.
2. Определить день недели 8 марта в заданном году.
3. Из строки Фамилия Имя Отчество дата рождения в виде день.месяц.год получить:

Фамилия Имя Отчество возраст.

Вариант 10

1. Даны стороны равнобедренного треугольника. Найти его высоту.
2. По дню рождения определить 10000 день жизни.
3. Из города улицы дома и квартиры получить адрес в виде:
г. Город, ул. Улица, д.дом, кв. квартира

Вариант 11

1. Даны стороны параллелограмма. Найти площадь параллелограмма.
2. Определить день недели для 25-летнего юбилея.
3. Функцию выделяющую из строки подстроку по начальному и конечному символу.

Вариант 12

1. Даны длины сторон равнобедренной трапеции. Найти ее площадь.
2. По введенной дате определить век.
3. Даны отдельно фамилия, имя отчество получить фамилию и инициалы.

Контрольные вопросы

1. Где пишется код пользовательской функции.
 2. Что такое аргументы функции.
 3. Для чего надо указывать Public в заголовке функции.
 4. Как задать справку для пользовательской функции.
 5. Для чего указывается тип для функции.
- Чем отличаются стандартные функции от пользовательских функций.

Практическое занятие №42

Создание форм VBA

Цель и содержание: изучить принципы создания оконных приложений средствами Visual Basic for Applications.

Теоретические сведения

Практически во всех приложениях Office используются пользовательские диалоговые окна. Диалоговые окна в VBA называются формами (объект UserForms). Каждому объекту UserForm присущи определенные свойства, методы и события, которые он наследует от класса объектов UserForms. Диалоговые окна (формы) и элементы управления составляют основу современного визуального интерфейса. Все элементы управления и технология работы с ними в основном стандартизованы и похожи для разных платформ и программных сред. Эти объекты помещены в специальную библиотеку MSForms.

Выделим основные моменты, которые следует иметь в виду при создании визуального интерфейса.

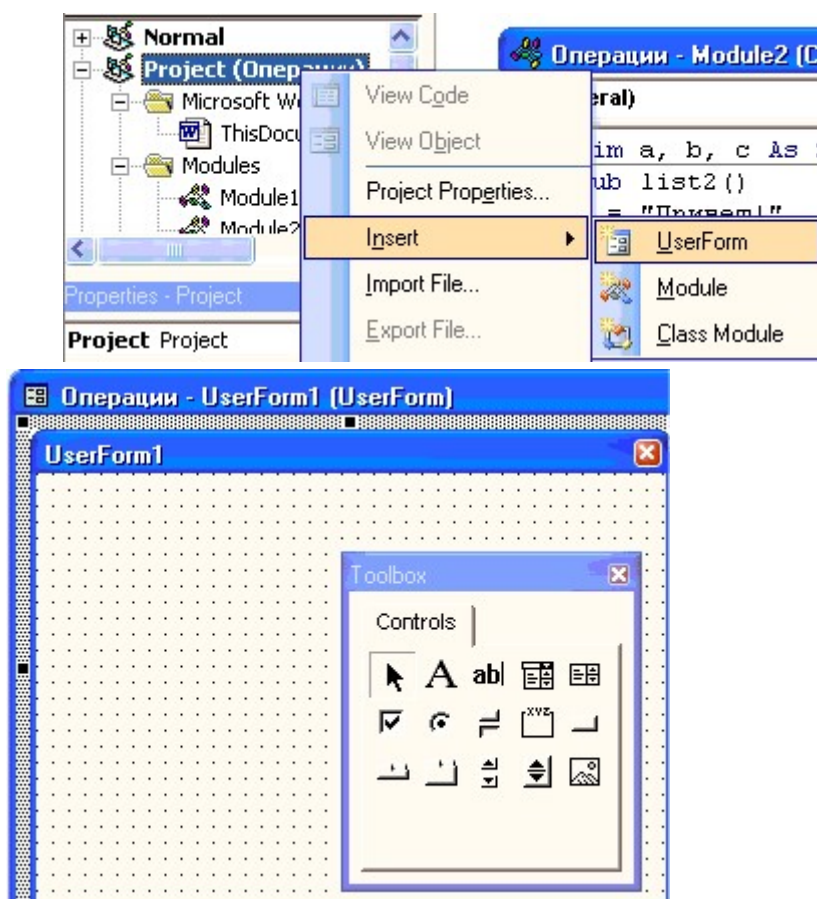
- Все загруженные диалоговые окна представляют коллекцию UserForms со стандартными методами и свойствами. Элемент коллекции – объект класса UserForm – задает отдельное окно.
- Для каждого типа элементов управления в библиотеке msforms имеется класс объектов, имя которого совпадает с именем элемента управления (его типа). Например, есть классы SpinButton и TextBox.
- Диалоговые окна создаются, как правило, не программно, а визуально. Вначале создается само окно, а затем оно наполняется элементами управления при помощи соответствующей панели элементов. Этот этап называется этапом проектирования, и его следует отличать от этапа выполнения, когда приложение выполняется и конечный пользователь взаимодействует с приложением, в частности через диалоговые окна и их элементы управления. Как только создается диалоговое окно и помещается в него тот или иной элемент управления, в этот же самый момент автоматически в программе появляется объект соответствующего класса, с которым можно работать, вызывая его методы и изменяя его свойства.

На этапе проектирования, используя окно свойств, можно задать большинство свойств как самого диалогового окна, так и всех элементов управления, помещенных в него, кроме этого, программно необходимо прописать все обработчики событий.

- Последний момент – отладка. Для ведения отладки нужно предварительно откомпилировать приложение и затем перейти в режим выполнения приложения.

Для того чтобы в разрабатываемое приложение можно было добавить форму, необходимо выполнить следующие действия:

- запустить редактор VBA;
- выделить правой кнопкой мыши объект Project, выполнить команду Insert + UserForm, после чего появляются новая форма и панель элементов Toolbox.



Добавление формы

Форма как объект имеет некоторые встроенные свойства, и их можно устанавливать или программным образом, или в Properties Window (окне свойств) редактора VBA (таблица. Наиболее часто используемые свойства объектов UserForm).

Наиболее часто используемые свойства объектов UserForm

Свойство	Описание
ActiveControl	Возвращает объектную ссылку на элемент управления, находящийся в фокусе в данный момент. Только для чтения
BackColor	Целое типа Long определяет цвет фона формы
Caption	Текст, выводимый в качестве заголовка формы
Controls	Возвращает коллекцию всех элементов управления формы
Cycle	Определяет, должно ли нажатие клавиши табуляции вызывать последовательный выбор всех элементов управления во всех группах и на каждой странице многостраничных элементов управления или только в пределах текущей группы или страницы. Может содержать одну из двух встроенных констант: fmCycleAllForms или fmCycleCurrentForm
Enabled	Содержит значение типа Boolean, указывающее, доступна ли форма. Если его значение равно False, ни один из элементов управления формы не доступен
Font	Возвращает ссылку на объект Font, посредством которого можно выбрать параметры шрифта формы или элемента управления
ForeColor	То же самое, что и свойство BackColor, но устанавливает цвет, используемый для переднего плана (обычно это цвет текста) объекта формы

Методы объекта UserForm

Всякий раз, создавая в проекте новый объект UserForm, одновременно создается новый подкласс объекта UserForm. Любые процедуры или функции, написанные в разделе General (общий) модуля класса, относящегося к форме, становятся дополнительными методами для отдельного подкласса объекта (таблица. Наиболее часто используемые методы для объектов UserForm).

Наиболее часто используемые методы для объектов UserForm

Метод	Назначение
Copy	Копирует выделенный в элементе управления текст в буфер обмена Windows
Cut	Вырезает выделенный в элементе управления текст и помещает его в буфер обмена Windows
Hide	Скрывает UserForm, не выгружая ее из памяти, сохраняя значения элементов управления формы и всех переменных, объявленных в модуле класса формы
Paste	Вставляет содержимое буфера обмена Windows в текущий элемент управления
PrintForm	Выводит на используемый в Windows по умолчанию принтер изображение формы, включая все данные, введенные в элементы управления
Repaint	Перерисовывает форму, выведенную на экран. Используется этот метод, если необходимо перерисовать форму, не ожидая, когда она будет перерисована через обычный период времени
Show	Выводит форму на экран. Если форма еще не загружена в память, то данный метод сначала ее загружает. Синтаксис метода Show.FormName.Show

События объекта UserForm

Событие - это что-то, что может произойти с диалоговым окном или элементом управления диалогового окна (таблица. События объектов UserForm).

Событийные процедуры следует записывать в модуль класса, который является частью User Form. При этом такие процедуры должны иметь имена в виде

ObjectName_EventName,

где ObjectName - имя формы или элемента управления, а EventName - имя события, с которым идет работа. Такой формат имени позволяет VBA сопоставлять заданному событию требуемую процедуру.

События объектов UserForm

Событие	Синтаксис заголовка процедуры	Описание
Activate	Private Sub object_Activate()	Иницируется всякий раз, когда окно формы становится активным. Используйте это событие для обновления содержимого диалоговых элементов управления, чтобы отразить любые изменения, которые произошли, пока окно было неактивным
Click	Private Sub object_Click()	Иницируется всякий раз, когда по форме (любой ее части, не занятой элементами управления) щелкают мышью
DbClick	Private Sub object_DbClick()	Иницируется всякий раз, когда по форме (любой ее части, не занятой элементами управления) дважды щелкают мышью
Deactivate	Private Sub object_Deactivate()	Иницируется всякий раз, когда форма перестает быть активной

Initialize	Private Sub object_Initialize()	Иницируется всякий раз, когда форма впервые загружается в память посредством выполнения оператора Load или с помощью метода Show. Используйте это событие для инициализации элементов управления формы при ее появлении на экране
Resize	Private Sub object_Resize()	Иницируется при изменении размеров формы
Terminate	Private Sub object_Terminate()	Иницируется всякий раз, когда форма выгружается из памяти. Используйте это событие для осуществления любых специальных служебных задач, которые необходимо выполнить прежде, чем переменные формы будут выгружены

Элементы управления

Объект UserForm может содержать те же элементы управления, что и находящиеся в диалоговых окнах Word, Excel или других приложений Windows (таблица. Стандартные элементы управления, включенные в VBA). Элементы управления - это элементы диалогового окна, позволяющие пользователю взаимодействовать с программой. Они включают в себя кнопки-переключатели, текстовые поля, линейки прокрутки, командные кнопки и так далее

Стандартные элементы управления, включенные в VBA

Элемент управления	Назначение
Label (надпись, метка)	Позволяет создавать заголовки элементов управления, которые не имеют собственных встроенных заголовков
TextBox (текстовое поле)	Окно редактируемого текста свободной формы для ввода данных. Может быть одно- или многострочным
ComboBox (поле со списком)	Этот элемент управления объединяет окно редактирования и окно списка. Используйте, когда хотите предложить пользователю выбрать значение, но при этом дать ему возможность ввести данные, отсутствующие в списке
ListBox (список)	Отображает список значений, из которых пользователь может сделать выбор. Окна списка можно использовать, чтобы дать возможность пользователю выбрать только одно значение или же несколько
CheckBox (флажок)	Стандартный флажок (квадратное окно, содержащее, если элемент выбран, галочку). Используйте флажки для выбора вариантов, которые не являются взаимоисключающими
OptionButton (переключатель)	Стандартная кнопка-переключатель (круглое окно, при выборе в центре него находится черная точка). Используйте OptionButton, когда пользователю необходимо сделать выбор между положениями «включено/выключено» или «истина/ложь». Кнопки переключатели, как правило, объединяются вместе при помощи рамки для создания группы переключателей
ToggleButton (выключатель)	Выключатели служат для той же цели, что и флажки, но выводят установки в виде кнопки, находящейся в «нажатом» или «отжатом» состоянии
Frame (рамка)	Визуально и логически объединяет некоторые элементы управления (особенно флажки, переключатели и выключатели)
CommandButton (кнопка)	Используйте кнопки для выполнения таких действий, как Cancel (Отмена), Save (Сохранить), OK и так далее. Когда пользователь щелкает по кнопке, выполняется VBA-процедура, закрепленная за данным элементом управления

TabStrip (набор вкладок)	Этот элемент управления состоит из области, в которую вы помещаете другие элементы управления (такие как текстовые поля, флажки и так далее) и полосы кнопок табуляции. Используйте элемент управления TabStrip для создания диалоговых вкладок, отображающих одни и те же данные в различных категориях
MultiPage (набор страниц)	Этот элемент управления состоит из нескольких страниц. Вы можете выбрать любую из них, щелкнув по соответствующей вкладке. Используйте элемент управления MultiPage для создания диалоговых окон с вкладками
ScrollBar (полоса прокрутки) и SpinButton (счетчик)	Элемент управления ScrollBar позволяет выбирать линейное значение аналогично тому, как это можно сделать при помощи счетчика. Элемент управления SpinButton является специальной разновидностью текстового поля
Image (рисунок)	Элемент управления Image позволяет вывести на форме графическое изображение. Используйте Image для вывода графических изображений в любом из следующих форматов: *.bmp, *.cur, *.gif, *.ico, *.jpg или *.wmf

Обращение к элементам управления происходит в основном через их свойства и с помощью процедур обработки событий, написанных для каждого элемента (таблица. Свойства стандартных элементов управления).

Свойства стандартных элементов управления

Свойство	Где применяется	Описание
Accelerator	CheckBox, Tab, CommandButton, Label, Page, OptionButton, ToggleButton	Содержит символ, используемый в качестве быстрой клавиши вызова, элемента управления, при нажатии Alt + <клавиша быстрого вызова> происходит выбор элемента управления
BackColor	Все элементы	Число, представляющее определенный цвет фона элемента управления
Caption	CheckBox, CommandButton,	Для надписи - текст, отображаемый элементом управления. Для других
	Frame, Label, OptionButton, ToggleButton, Page, Tab, UserForm	элементов управления - надпись, которая появляется на кнопке или вкладке или рядом с рамкой, флажком или переключателем
Cancel	CommandButton	Задает кнопку отмены диалогового окна. При нажатии на эту кнопку или клавишу Esc диалоговое окно исчезает. Только одна кнопка формы может иметь данное свойство
ControlTipText	Все элементы управления	Устанавливает текст, который отображается в виде всплывающей подсказки (ControlTip, называемой также ToolTip), когда указатель мыши помещается на элемент управления
Default	CommandButton	Определяет заданную по умолчанию кнопку. Когда пользователь нажимает в процессе диалога клавишу Enter, эта кнопка ведет себя так, как если бы по ней щелкнули мышью

Enabled	Все элементы управления	Хранит значение типа Boolean, определяющее, доступен или нет элемент управления. Если Enabled имеет значение False, то элемент управления продолжает отображаться в диалоговом окне, но не может быть выбран
ForeColor	Все элементы управления	То же самое, что и BackColor, но устанавливает цвет для переднего плана элемента управления, как правило, символов текста
List	ComboBox	Массив типа variant (одно- или многомерный), представляет список, содержащийся в элементе управления
Max	ScrollBar, SpinButton	Переменная типа Long, определяющая максимальное значение счетчика, или значение, при котором полоса прокрутки находится в самом верху (для вертикальной полосы) или справа (для горизонтальной)
Min	ScrollBar, SpinButton	Переменная типа Long, определяющая минимальное значение счетчика, или значение, при котором полоса прокрутки находится в самом низу (для вертикальной полосы) или слева (для горизонтальной)
Name	Все элементы управления	Содержит имя элемента управления. Вы можете установить данное свойство только с помощью Properties Window
RowSource	ComboBox	Задаёт источник, из которого ListBox берет список объекта. В Excel VBA RowSource обычно использует диапазон рабочего листа
Selected	ListBox	Возвращает массив значений типа Boolean для списка, который допускает множественный выбор. Каждый элемент массива содержит по одному элементу, соответствующему каждому пункту списка. Если значение элемента в массиве selected равно True, то соответствующий пункт списка выбран
TabIndex	Все элементы управления	Число, указывающее положение элемента управления в порядке табуляции (может иметь значение от 0 до значения, равного количеству элементов управления на форме)
TabStop	Все элементы управления	Значение типа Boolean, указывающее, может ли элемент управления быть выбран клавишей Tab. Если значение TabStop равно False, вы тем не менее можете щелкнуть на элементе и таким образом его выбрать
Value	Все элементы управления	Значение текущих установок элемента управления: текст в текстовом поле, какие выбраны флажки и переключатели, индекс выбранного раздела списка или число, указывающее текущее положение полосы прокрутки или счетчика
Visible	Все элементы управления	Значение типа Boolean, указывающее, является ли элемент управления видимым

Задание 1

Создать приложение Меню для выбора блюд из списка и вывода результата выбора на лист Excel. Можно одновременно выбрать несколько блюд. Предусмотрено наличие

скидки 5%. Данные о блюдах (название блюда и его цена) находятся на листе Excel с именем «списки».

	A	B	C
1	Салат Оливье	30	
2	Винегрет	15	
3	Салат Витаминный	10	
4	Борщ	28	
5	Рассольник	36	
6	Пельмени	40	
7	Перец фаршированный	43	
8	Гуляш	50	
9	Рис	18	
10	Картофельное пюре	16	
11	Желе	26	
12	Кофе	30	
13	Чай	15	
14	Морс	24	
15			

Пользовательская форма имеет вид:

Обед

Меню

Салат Оливье	30
Винегрет	15
Салат Витаминны	10
Борщ	28
Рассольник	36
Пельмени	40
Перец фаршировз	43
Гуляш	50
Рис	18
Картофельное пк	16
Желе	26
Кофе	30
Чай	15
Морс	24

☐ Скидка 5%

Рассчитать

Свойства элемента ListBox1:

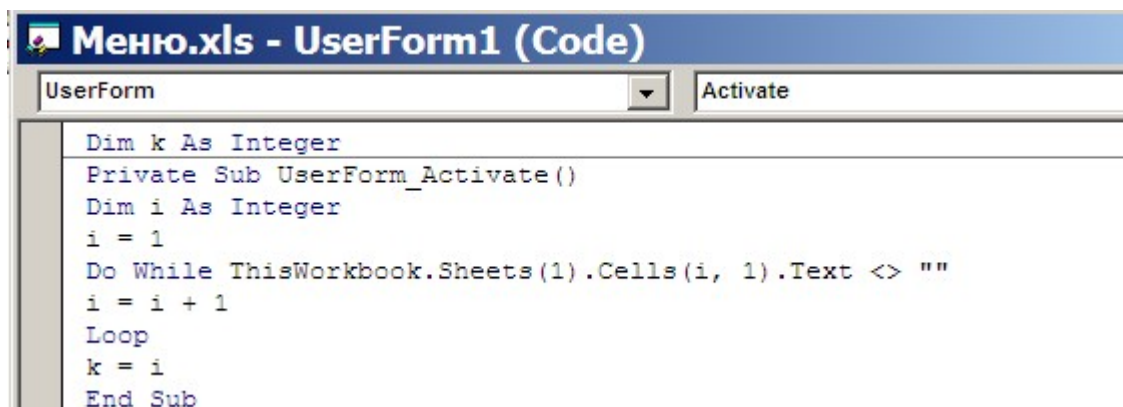
Properties - ListBox1

ListBox1 ListBox

Alphabetic | Categorized

Locked	False
MatchEntry	0 - fmMatchEntryFirstLetter
MouseIcon	(None)
MousePointer	0 - fmMousePointerDefault
MultiSelect	2 - fmMultiSelectExtended
RowSource	списки!A1:B14
SpecialEffect	2 - fmSpecialEffectSunken
TabIndex	0
TabStop	True

На Лист1 записываются результаты выбора. Но при каждом сеансе данные должны заноситься в конец списка. Поэтому каждый раз необходимо определять последнюю заполненную строку на листе. Это делается при активации формы.



```
Dim k As Integer
Private Sub UserForm_Activate()
Dim i As Integer
i = 1
Do While ThisWorkbook.Sheets(1).Cells(i, 1).Text <> ""
i = i + 1
Loop
k = i
End Sub
```

Номер первой пустой строки записывается в переменную k. Так как переменная k будет использоваться в других обработчиках событий, эта переменная описывается в первой строчке модуля формы.

При щелчке по кнопке Рассчитать будут выполняться следующие действия:

```
Private Sub CommandButton1_Click()
Dim s As Integer, i As Integer
Dim itog As Currency, skidka As Single
s = 0
skidka = 0
'Суммирование цены в выделенных строках
For i = 0 To ListBox1.ListCount - 1
If ListBox1.Selected(i) = True Then
s = s + Val(ListBox1.List(i, 1))
'вывод цены и названия выделенных блюд в k строку
ThisWorkbook.Sheets(1).Cells(k, 1) = ListBox1.List(i, 0)
ThisWorkbook.Sheets(1).Cells(k, 2) = ListBox1.List(i, 1)
k = k + 1
End If
Next
'Вывод суммы на Лист1
ThisWorkbook.Sheets(1).Cells(k, 1) = "Сумма"
ThisWorkbook.Sheets(1).Cells(k, 2) = s
k = k + 1
'Вычисление скидки
If CheckBox1.Value = True Then
skidka = s * 0.05
End If
'Вывод итоговых результатов и форматирование чисел
ThisWorkbook.Sheets(1).Activate
Cells(k, 1) = "Скидка"
Cells(k, 2) = skidka
Cells(k, 2).Style = "Currency"
k = k + 1
Cells(k, 1) = "Итого"
Cells(k, 2) = s - skidka
Cells(k, 2).Style = "Currency"
```

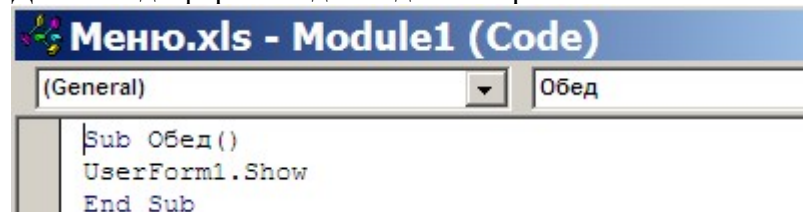
После вывода очередного сеанса надо отделить результаты от новых данных.

```

'Вывод итоговых результатов и форматирование чисел
ThisWorkbook.Sheets(1).Activate
Cells(k, 1) = "Скидка"
Cells(k, 2) = skidka
Cells(k, 2).Style = "Currency"
k = k + 1
Cells(k, 1) = "Итого"
Cells(k, 2) = s - skidka
Cells(k, 2).Style = "Currency"
'отделение чертой результата от следующего
Range(Cells(k, 1), Cells(k, 2)).Select
Selection.Borders(xlDiagonalDown).LineStyle = xlNone
Selection.Borders(xlDiagonalUp).LineStyle = xlNone
Selection.Borders(xlEdgeLeft).LineStyle = xlNone
Selection.Borders(xlEdgeTop).LineStyle = xlNone
With Selection.Borders(xlEdgeBottom)
.LineStyle = xlContinuous
.Weight = xlThick
.ColorIndex = xlAutomatic
End With
Selection.Borders(xlEdgeRight).LineStyle = xlNone
Selection.Borders(xlInsideVertical).LineStyle = xlNone
k = k + 1
End Sub

```

Для вывода формы надо создать макрос



Результаты будут иметь вид:

Первый сеанс

The form titled 'Обед' (Dinner) contains a menu list and a calculation button. The menu items and their prices are as follows:

Салат Оливье	30
Винегрет	15
Салат Витаминны	10
Борщ	28
Рассольник	36
Пельмени	40
Перец фаршировз	43
Гуляш	50
Рис	18
Картофельное пк	16
Желе	26
Кофе	30
Чай	15
Морс	24

Below the menu list, there is a checkbox labeled 'Скидка 5%' (5% discount) and a button labeled 'Рассчитать' (Calculate).

Второй сеанс

Обед

Меню

Салат Оливье	30
Винегрет	15
Салат Витаминны	10
Борщ	28
Россольник	36
Пельмени	40
Перец фарширов	43
Гуляш	50
Рис	18
Картофельное пк	16
Желе	26
Кофе	30
Чай	15
Морс	24

☒ Скидка 5%

Рассчитать

	А	В	С
1	Блюдо	Цена	
2	Салат Оливье	30	
3	Борщ	28	
4	Пельмени	40	
5	Морс	24	
6	Сумма	122	
7	Скидка	- р.	
8	Итого	122,00р.	
9	Винегрет	15	
10	Россольник	36	
11	Гуляш	50	
12	Рис	18	
13	Кофе	30	
14	Сумма	149	
15	Скидка	7,45р.	
16	Итого	141,55р.	
17			

В приведенном задании список listbox заполняется с помощью свойства RowSource и разрешается выбрать одновременно несколько строк с помощью клавиши Ctrl.

В следующем задании можно выбрать из списка только один элемент.

Список заполняется во время работы программы

Задание 2

Создать приложение рассчитывающее стоимость железнодорожного билета в зависимости от направления вида вагона и сезона. Летом стоимость увеличивается на 20%, зимой - уменьшается на 10%.

Данные о стоимости билетов по-прежнему располагаются на листе Excel.

	A	B	C	D	E	F
1	№	Направление	Плацкарт	Купе	СВ	
2	128	Екатеринбург-Челябинск	2000	1500	1000	
3	130	Екатеринбург-Пермь	2500	1800	1300	
4	21	Екатеринбург-Москва	5000	3500	3000	
5	36	Екатеринбург-Москва	4500	3200	2800	
6	40	Екатеринбург-Москва	4800	3300	3000	
7	45	Екатеринбург-С.Петербург	5200	3800	3200	
8	56	Екатеринбург-С.Петербург	5100	4000	3500	
9	150	Екатеринбург-Уфа	2400	1700	1200	
10	160	Екатеринбург-Тюмень	1500	1000	800	
11						

Пользовательская форма имеет вид:

У элемента ListBox1 свойство

Список заполняется при инициализации формы. Количество элементов в списке определяются во время работы программы, а не во время создания формы, как это было в предыдущем задании.

```

Private Sub UserForm_Initialize()
Dim i As Integer
i = 1
ListBox1.ColumnWidths = "20;140"
With ThisWorkbook.Sheets("список")
Do While .Cells(i, 1).Text <> ""
ListBox1.AddItem .Cells(i, 1).Value
ListBox1.List(i - 1, 1) = .Cells(i, 2).Value
i = i + 1
Loop
End With
End Sub

```

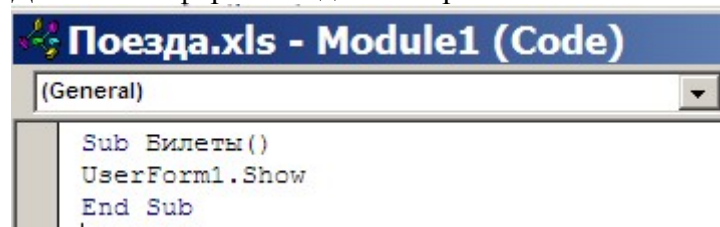
При щелчке по кнопке Оплата выполняются операторы:

```

Private Sub CommandButton1_Click()
Dim s As Integer, cena As Single
Dim i As Integer, k As Integer
Dim m As Integer, pr As Single
'Определение номера выделенной строки
i = ListBox1.ListIndex
'Определение номера столбца с категорией вагона
If OptionButton1.Value = True Then k = 3
If OptionButton2.Value = True Then k = 4
If OptionButton3.Value = True Then k = 5
'Определение цены билета
s = ThisWorkbook.Sheets("список").Cells(i, k).Value
'Определение месяца поездки
m = Calendar1.Month
'Определение процента скидки/надбавки
Select Case m
Case 1, 2, 12:
pr = 0.9
Case 6, 7, 8:
pr = 1.2
Case Else
pr = 1
End Select
'Вычисление и вывод цены.
cena = s * pr
MsgBox "цена билета = " + CStr(cena)
End Sub

```

Для вызова формы создаем макрос Билеты:



Результат работы программы:

Билеты

Поезда

№	Направление
128	Екатеринбург-Челябинск
130	Екатеринбург-Пермь
21	Екатеринбург-Москва
36	Екатеринбург-Москва
40	Екатеринбург-Москва
45	Екатеринбург-С.Петербург
56	Екатеринбург-С.Петербург
150	Екатеринбург-Уфа
160	Екатеринбург-Тюмень

Дата отъезда

ноя 2013

Вс	Пн	Вт	Ср	Чт	Пт	Сб
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

Тип вагона

☒ Плацкарт

☐ Купе

☐ СВ

Оплата

Microsoft Excel

цена билета = 5000

OK

Задания для самостоятельной работы

В приведенных заданиях используются списки. Необходимо определить какой режим для работы со списками надо задать: выпор нескольких вариантов или один вариант.

Данные для списков считываются из листа Excel. Результаты надо заносить в таблицу Excel/

Вариант № 1 Книжный интернет-магазин:

Вывести название книг с ценой. Указать варианты доставки: курьерская (постоянная цена 800 руб), наложенным платежом (зависит от кол-ва книг и от общей стоимости) и оплата через банк (зависит от стоимости книг).

Написать приложение для выбора книг и подсчета общей стоимости.

Вариант № 2 Ателье мод:

Вывести перечень изделий с ценой. Указать дополнительные услуги: сложность (процент от стоимости изделия), срочность (процент от стоимости и сложность), доставка на дом (конкретная сумма).

Написать приложение для заказа изделий и подсчета общей стоимости.

Вариант № 3 Хозяйственный магазин:

Вывести перечень товаров с указанием цены. Указать вид оплаты: наличные, карточка Visa (скидка 5%), карточка MasterCard (скидка 3%).

Написать приложение для покупки товара и подсчета общей стоимости.

Вариант № 4 Магазин с оплатой за валюту:

Вывести перечень товаров с указанием цены в рублях. Указать в какой валюте будет оплата с указанием курса.

Написать приложение для покупки товара и подсчета общей стоимости.

Вариант № 5 Покупка туристической путевки в Москву:

Вывести названия гостиниц для проживания с ценой.

Указать дополнительную доплату: дорога, питание, все включено (процент от стоимости гостиницы), экскурсии.

Написать приложение для покупки путевки и подсчета общей стоимости.

Вариант № 6 Гостиница:

Вывести категории номеров с ценой за день. Указать дополнительные услуги: все включено (зависит от категории номера), 2-х разовое питание (за 1 день), 3-х разовое питание (за один день), пользование бассейном (разовый взнос).

Написать приложение для покупки путевки и подсчета общей стоимости при проживании недель.

Вариант № 7 Компьютерный магазин:

Вывести список компьютеров с мониторами и указать их цену. Перечислить лицензионное программное обеспечение, которое нужно установить на компьютере.

Написать приложение для покупки компьютера и подсчета общей стоимости

Вариант № 8 Магазин стиральных машин:

Вывести список стиральных машин с указанием цены. Указать дополнительные услуги: увеличение гарантийного срока, дополнительные аксессуары, бесплатное подключение.

Написать приложение для покупки стиральной машины и подсчета общей стоимости

Вариант № 9 Магазин корпусной мебели:

Вывести список изделий из мебели (комод, шифоньер, тумбочка,...) с ценой изготовления. Указать вид материала.

Написать приложение для покупки набора мебели и подсчета общей стоимости. Общая стоимость вычисляется как проценты от цены изготовления.

Вариант № 10 Покупка кухни:

Вывести список изделий для кухни (плита, посудомоечная машина, разделочный стол, стол, пенал, навесной шкаф) с ценой. Вывести варианты материала для столов и шкафов.

Написать приложение для покупки набора для кухни и подсчета общей стоимости. Стоимость столов и шкафов зависит от выбранного материала.

Вариант № 11

Подключение телефона:

Вывести список тарифов с ценой. Указать дополнительные услуги. Написать приложение для подключения телефона.

Вариант № 12 Тренажерный зал:

Вывести список тренажеров, которые имеются в тренажерном зале с ценой. Указать время посещения зала (утром, днем, вечером, выходные дни).

Написать приложение для посещения зала с подсчетом общей стоимости. Общая стоимость зависит от времени, для утра и дня - скидки. Самое дорогое время – выходные, самое дешевое – утро.

Список литературы

Основная литература:

1. Смирнов А.А. Прикладное программное обеспечение: учебно-практическое пособие / А.А. Смирнов. – М.: Изд. Центр ЕАОИ, 2011. – 384 с.
2. Привалов И. М. Основы аппаратного и программного обеспечения: учебное пособие / И. М. Привалов. – Ставрополь: Северо-Кавказский федеральный университет, 2015. – 145 с.

Дополнительная литература:

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. – СПб.: Питер, 2015. – 1120 с.: ил. – (Серия «Классика computer science»).
2. Приходченко О.В. Прикладное программное обеспечение профессиональной деятельности: учебное пособие / О.В. Приходченко, А.А. Просолович. – Комсомольск-на-Амуре: ФГБОУ ВПО «КНАГТУ», 2012. – 158 с.
3. А.Ю. Щербаков. Современная компьютерная безопасность. Теоретические основы, практические аспекты. Учебное пособие. – М.: Книжный мир, 2009. – 352 с.
4. Пальмов, С. В. Методы и средства моделирования программного обеспечения: конспект лекций / С. В. Пальмов. – Самара: Поволжский государственный университет телекоммуникаций и информатики, 2016. – 105 с.

Методическая литература:

1. Влацкая, И. В. Проектирование и реализация прикладного программного обеспечения: учебное пособие / И. В. Влацкая, Н. А. Заельская, Н. С. Надточий. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2015. – 119 с.
2. Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие / Т. М. Зубкова. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017. – 469 с.

Интернет-ресурсы:

1. База знаний и набор вычислительных алгоритмов WolframAlpha <https://www.wolframalpha.com>
2. Лекториум – Онлайн-курсы и медиатека видеолекций на русском языке <https://lektorium.tv/>
3. Электронно-библиотечная система IPRbooks <https://iprbooks.ru/>
4. Википедия – Свободная энциклопедия <https://ru.wikipedia.org/>

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению самостоятельных работ
по дисциплине «Системное и прикладное программное обеспечение»
для студентов специальности 10.05.01 Компьютерная безопасность
специализация «Информационно-аналитическая и техническая экспертиза
компьютерных систем»

Ставрополь
2026

1 Общие положения

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

- ~ формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- ~ написание докладов;
- ~ подготовка к семинарам, практическим и лабораторным работам, их оформление;
- ~ составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);
- ~ выполнение учебно-исследовательских работ, проектная деятельность;
- ~ подготовка практических разработок и рекомендаций по решению проблемной ситуации;
- ~ выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;
- ~ компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;
- ~ выполнение курсовых работ (проектов) в рамках дисциплин;
- ~ выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- ~ подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- ~ основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- ~ заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине «Системное и прикладное программное обеспечение» направлена на формирование следующих **компетенций**:

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ОПК-2 Способен применять программные средства системного и прикладного назначения, в том числе отечественного производства, для решения задач профессиональной деятельности	ИД-2 ОПК-2 применяет программные средства системного и прикладного назначения, в том числе отечественного производства, при решении задач профессиональной деятельности	Способен применять системное и прикладное программное обеспечение для решения рабочих задач. Способен выбирать наиболее подходящие по совокупности различных критериев программные продукты для использования в профессиональной деятельности
ОПК-12 Способен администрировать операционные системы и выполнять работы по восстановлению работоспособности прикладного и системного программного обеспечения	ИД-1 ОПК-12 использует системные приложения для управления настройками операционных систем	Владеет навыками применения системных и прикладных приложений для решения задач администрирования операционных систем и обеспечения корректной работы программного и аппаратного обеспечения ЭВМ

2 Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора общенаучных, профессиональных и специальных компетенций будущего специалиста по направлению подготовки «Компьютерная безопасность».

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

3 Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
7 семестр					
ОПК-2 ИД-2 ОПК-2 ОПК-12 ИД-1 ОПК-12	Подготовка к опросам по материалам темы 1	Собеседование, проверка конспектов	9	4,5	13,5
ОПК-2 ИД-2 ОПК-2 ОПК-12 ИД-1 ОПК-12	Подготовка к защите лабораторных работ	Собеседование, проверка конспектов	9	4,5	13,5
ОПК-2 ИД-2 ОПК-2 ОПК-12 ИД-1 ОПК-12	Подготовка к практическим занятиям	Собеседование, проверка конспектов	9	4,5	13,5
Итого за 7 семестр			27	13,5	40,5
ОПК-2 ИД-2 ОПК-2 ОПК-12 ИД-1 ОПК-12	Подготовка к опросам по материалам темы 2	Собеседование, проверка конспектов	9	4	13
ОПК-2 ИД-2 ОПК-2 ОПК-12 ИД-1 ОПК-12	Подготовка к защите лабораторных работ	Собеседование, проверка конспектов	15	5	20
Итого за 8 семестр			24	9	33
Итого			51	22,5	73,5

4 Порядок выполнения самостоятельной работы студентом

4.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой – это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того, насколько осознанно читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют *четыре основные установки в чтении научного текста*:

- ~ информационно-поисковый (задача – найти, выделить искомую информацию);
- ~ усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить, как сами сведения, излагаемые автором, так и всю логику его рассуждений);

- ~ аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему);

- ~ творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

1. Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

2. Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

3. Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

4. Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

5. Конспектирование – краткое и последовательное изложение содержания прочитанного.

6. Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;

2. Выделите главное, составьте план;

3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;

4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.

5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

4.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

4.4. Методические рекомендации по написанию научных текстов (докладов, рефератов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста – это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время – важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т. п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Реферат (доклад) – это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Реферат не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в реферате должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки реферата студентом.

Выполнение реферата начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- ~ обсуждение цели и задач работы, основных моментов избранной темы;
- ~ консультирование по вопросам подбора литературы;
- ~ составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы - изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания реферата. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста реферата предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки реферат сдается на кафедру для его оценивания руководителем.

Требования к написанию реферата

Написание 1 реферата является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема реферата может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

Реферат должен быть написан научным языком.

Объем реферата должен составлять 20-25 стр.

Структура реферата:

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.

- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.

- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса

- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.

- Приложение (при необходимости).

Требования к оформлению:

- текст с одной стороны листа;
- шрифт Times New Roman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- реферат должен быть представлен в сброшюрованном виде.

Порядок защиты реферата:

Защита реферата проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту реферата отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите реферата приветствуется использование мультимедиа-презентации.

Оценка реферата

Реферат оценивается по следующим критериям:

- соблюдение требований к его оформлению;
- необходимость и достаточность для раскрытия темы, приведенной в тексте реферата информации;
- умение студента свободно излагать основные идеи, отраженные в реферате;
- способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

Описание шкалы оценивания

Максимально возможный балл за весь текущий контроль устанавливается равным 55. Текущее контрольное мероприятие считается сданным, если студент получил за него не менее 60% от установленного для этого контроля максимального балла. Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

<i>Уровень выполнения контрольного задания</i>	<i>Рейтинговый балл (в % от максимального балла за контрольное задание)</i>
Отличный	100
Хороший	80
Удовлетворительный	60
Неудовлетворительный	0

4.5. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неустойчивые занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

5 Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование по результатам выполнения лабораторных и практических работ по темам 1 и 2.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы

Основная литература:

1. Смирнов А.А. Прикладное программное обеспечение: учебно-практическое пособие / А.А. Смирнов. – М.: Изд. Центр ЕАОИ, 2011. – 384 с.
2. Привалов И. М. Основы аппаратного и программного обеспечения: учебное пособие / И. М. Привалов. – Ставрополь: Северо-Кавказский федеральный университет, 2015. – 145 с.

Дополнительная литература:

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. – СПб.: Питер, 2015. – 1120 с.: ил. – (Серия «Классика computer science»).
2. Приходченко О.В. Прикладное программное обеспечение профессиональной деятельности: учебное пособие / О.В. Приходченко, А.А. Просолович. – Комсомольск-на-Амуре: ФГБОУ ВПО «КНАГТУ», 2012. – 158 с.
3. А.Ю. Щербаков. Современная компьютерная безопасность. Теоретические основы, практические аспекты. Учебное пособие. – М.: Книжный мир, 2009. – 352 с.
4. Пальмов, С. В. Методы и средства моделирования программного обеспечения: конспект лекций / С. В. Пальмов. – Самара: Поволжский государственный университет телекоммуникаций и информатики, 2016. – 105 с.

Методическая литература:

1. Влацкая, И. В. Проектирование и реализация прикладного программного обеспечения: учебное пособие / И. В. Влацкая, Н. А. Заельская, Н. С. Надточий. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2015. – 119 с.
2. Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие / Т. М. Зубкова. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017. – 469 с.

Интернет-ресурсы:

1. База знаний и набор вычислительных алгоритмов WolframAlpha <https://www.wolframalpha.com>
2. Лекториум – Онлайн-курсы и медиатека видеолекций на русском языке <https://lektorium.tv/>
3. Электронно-библиотечная система IPRbooks <https://iprbooks.ru>
4. Википедия – Свободная энциклопедия <https://ru.wikipedia.org/>