**Министерство образования и науки Российской Федерации**
Федеральное государственное бюджетное образовательное
учреждение высшего образования
**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра автоматизации обработки информации (АОИ)

# ОПЕРАЦИОННЫЕ СИСТЕМЫ И СЕТИ

Методические указания к лабораторным работам
и организации самостоятельной работы для студентов
направления подготовки
«Программная инженерия»
(уровень бакалавриата)

**Томск – 2018**

**Гриценко Юрий Борисович**

Операционные системы и сети: Методические указания к лабораторным работам и организации самостоятельной работы для студентов направления подготовки «Программная инженерия» (уровень бакалавриата) / Ю.Б. Гриценко – Томск, 2018. – 188 с.

# Оглавление

# 1 Введение

Целью дисциплины «Операционные системы и сети» является формирование у студента профессиональных знаний по теоретическим основам построения и функционирования компьютеров вычислительных систем, телекоммуникационных вычислительных сетей и коммуникаций, их структурной и функциональной организации, программному обеспечению, эффективности и перспективам развития.

Задачи изучения дисциплины:

1) Изучение принципов построения, функционирования и внутренней архитектуры операционных систем и сетей, функциональность всех составных компонентов и механизмы их взаимодействия в одно- и многопроцессорных системах, методы работы с внешними интерфейсами операционных систем.

2) Изучение способов написания системных процедур, механизмов их функционирования в операционных системах и сетях, взаимодействия с системными функциями и инструментарием.

3) Изучение классификаций и архитектурных решений в области построения операционных систем.

4) Изучение механизмов функционирования отдельных функциональных составляющих операционных систем.

5) Изучение принципов функционирования системных и пользовательских процессов в операционных системах и сетях.

# 2 Методические указания по проведению лабораторных работ (часть 1)

## 2.1 Лабораторная работа «Управление задачами в ОС Windows»

**Цель работы**

Целью работы является изучение процесса управления заданиями в ОС Windows.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита отчета с описанием хода выполнения задания и ответы на теоретические вопросы.

**Теоретические основы**

Современные операционные системы содержат встроенные средства, предоставляющие информацию о компонентах вычислительного процесса. Диспетчер задач (Task Manager) операционных систем Windows (например, Windows) позволяет получить обобщенную информацию об организации вычислительного процесса с детализацией до выполняющихся прикладных программ (приложений) и процессов. Однако диспетчер задач не позволяет отслеживать потоки [1].

***Для запуска диспетчера задач и просмотра компонентов вычислительного процесса нужно выполнить следующие действия*** [2]:

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. 2.1).

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. 2.2) нужно ввести имя задачи.

3. 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.

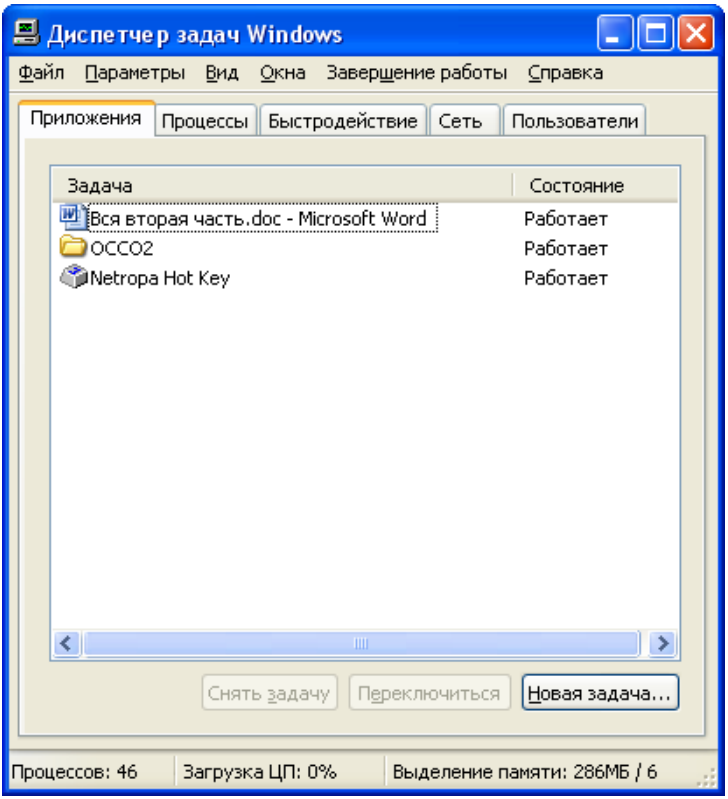



Рис. 2.1 – Окно диспетчера задач в ОС Windows

4. Если требуется просмотреть 16-разрядные процессы, то в меню «Параметры» необходимо выбрать команду «Отображать 16-разрядные задачи».

5. Для выбора просматриваемых показателей (характеристик) с помощью команды «Выбрать столбцы» (меню «Вид») необходимо установить флажки рядом с показателями, которые требуется отображать (рис. 2.3).

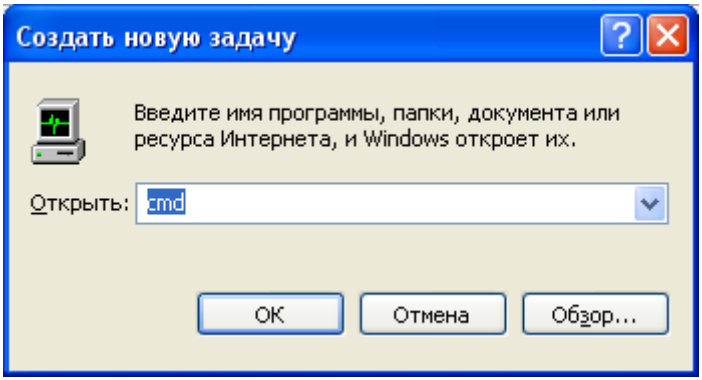



Рис. 2.2 – Окно создания новой задачи в ОС Windows

***В качестве примера можно рассмотреть процессы приложения MS Word.*** Для этого нужно выполнить следующие действия [2]:

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 [2]:

1. Перейти на вкладку «Быстродействие» (рис. 2.4). Верхние два окна показывают интегральную загрузку процессора и хронологию

загрузки. Нижние два окна — те же показатели, но по использованию памяти.

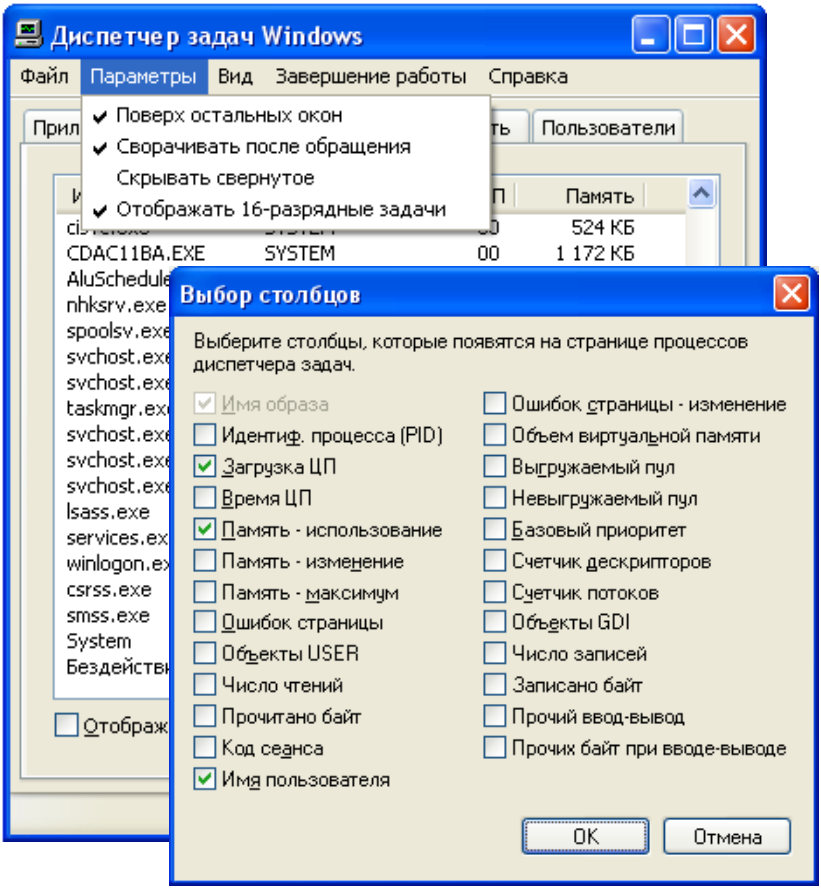



Рис. 2.3 – Окно диспетчера задач в ОС Windows на вкладке процессы с окном настройки отображения столбцов

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.

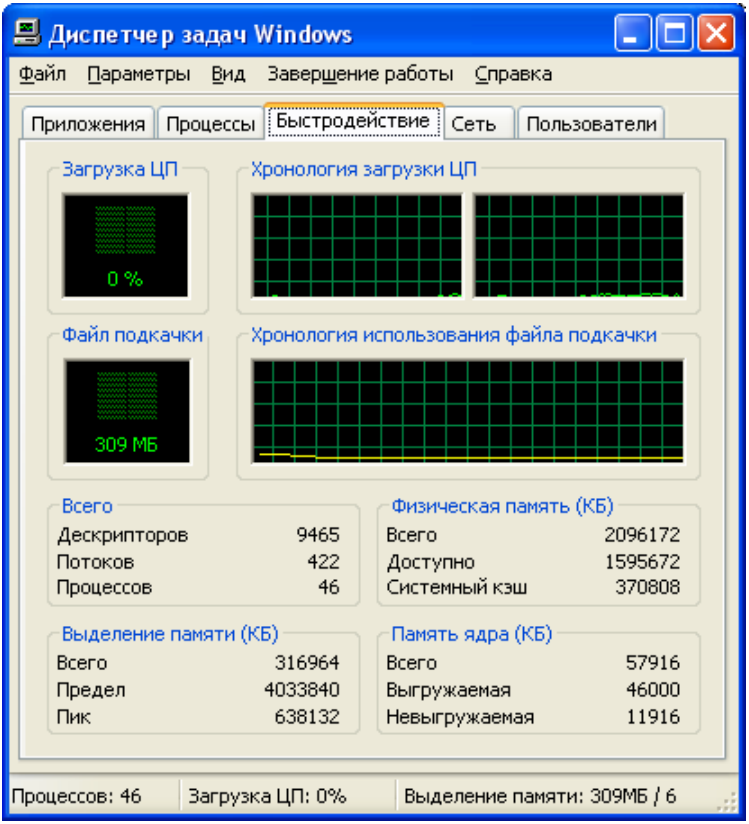



Рис. 2.4 – Окно диспетчера задач в ОС Windows на вкладке быстродействие

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.

На рис. 2.5 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 [2]:

– 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.

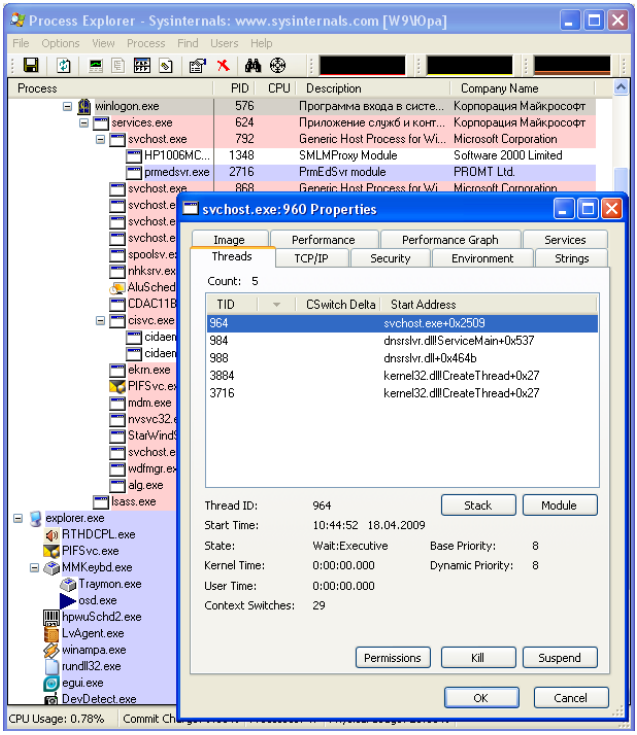



Рис. 2.5 – Окно с информацией о потоках в программе Process Explorer

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».

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 Kerne132.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** [2].

В операционной 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. 2.6).

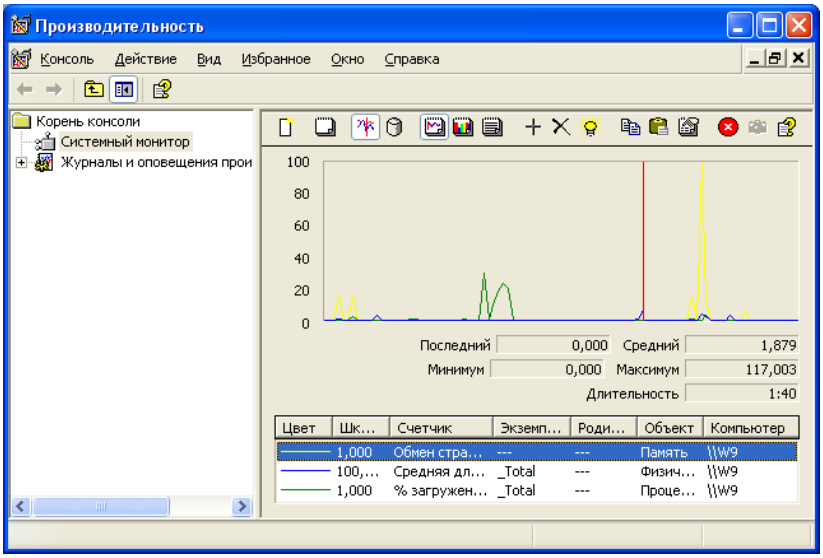



Рис. 2.6 – Окно Производительность в ОС Windows на вкладке быстродействие

**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. 2.7) 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:

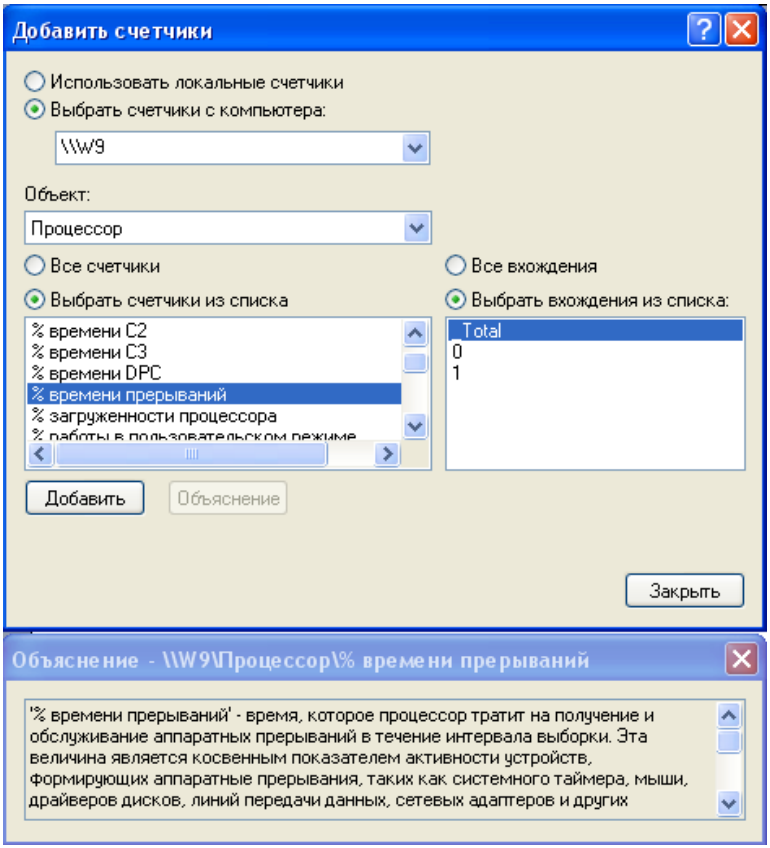



Рис. 2.7 – Окно Добавить счетчики в программе оценки производительности в ОС Windows

– число контекстных переключений в сек.;

– состояние потока (для построения графа состояний и переходов);

– 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.

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 [2]:

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;

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.

**План выполнения**

Выполнение работы состоит из двух этапов.

*I. 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.

5. 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.

*II. 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.

6. Ответьте на вопросы:

1. Что можно просматривать, используя счетчики в системном мониторе?

2. В каких видах можно просматривать информацию о производительности?

## 2.2 Лабораторная работа «Исследование блоков управления памятью»

**Цель работы**

Изучение структуры системных таблиц реального режима Windows и организации цепочек блоков памяти.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

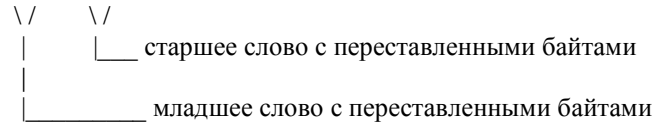
Защита отчета с описанием хода выполнения задания и ответы на теоретические вопросы.

**Теоретические основы**

*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 многоразрядного 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:




*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):

```
 1234 H     0110 H          0112 H
+ 5678H   + 0026H          + 0006H
```

-------- -------- ---------
=179B8H =01126H =01126H

Таким образом 0110:0026 – это тоже, что и 0112:0006 !

*Структура таблицы таблиц*

Данная структура (таблица 2.1) является НЕДОКУМЕНТИРОВАННОЙ и используется для изучения низкоуровневой информации о структурах памяти.

Таблица 2.1. 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$) |
| … | … | … |

*Структура блока управления памятью (MCB)*

MCB – Это НЕДОКУМЕНТИРОВАННЫЙ управляющий блок (таблица 2.2), который используется при распределении, модификации и освобождении блоков системной памяти.

Таблица 2.2. Структура блока управления памятью

| Смещение | Длина | Содержимое |
|---|---|---|
| +0 | 1 | 'M' (4dH) – за этим блоком есть еще блоки<br>'Z' (5aH) – данный блок является последним |
| +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:

1) блоки памяти всегда выровнены на границу параграфа («сегмент блока»);

2) блоки M-типа: следующий блок находится по (сегмент блока + Размер):0000;

3) 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 – путь ее запуска.

Следует помнить, что сама программа (и PSP в том числе) и ее окружение сами располагаются в блоках памяти. Поэтому, в MCB блока памяти самой программы в качестве хозяина указан собственный адрес самого себя.

Когда программа в реальном режиме начинает выполнение, DS:0000 и ES:0000 указывают на начало PSP этой программы. Информация PSP позволяет выделить имена файлов и опции из строки команд, узнать объем доступной памяти, определить окружение и т.д.

***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 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.

Более подробную информацию о структурах памяти можно получить из справочника TECH Help!

**План выполнения**

1. Познакомиться с работой одной из программ, позволяющих просмотреть содержимое ОЗУ в виртуальном режиме DOS. Если ипользуете шестидесяти четырех разрядную версию операционной системы, то необходимо воспользоваться какой-либо свободно распространяемой виртуальной машиной, например, DOS-Box.
2. Найти в памяти таблицу таблиц, познакомиться с ее содержимым и посмотреть указатель на 1 MCB (упр. блок памяти).
3. Проследить в памяти цепочку блоков, определяя их принадлежность и сравнивая с информацией из карты памяти.
4. Написать отчет о найденной цепочке блоков памяти с их адресами и размерами.
5. Ответьте на вопросы:
   1. Что ооозначает L-H-порядок следования байт?
   2. Как строится адресация памяти в реальном режиме?
   3. Опишите структуру таблици-таблиц.
   4. Опишите структуру блока управления памятью.

## 2.3 Лабораторная работа «Диагностика IP-протокола»

**Цель работы**

Целью работы является проверка работоспособности сетевого подключения в ОС Windows, через диагностику IP-протокола.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита отчета с описанием хода выполнения задания и ответы на теоретические вопросы.

**Теоретические основы**

*Свойства 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. 2.8.

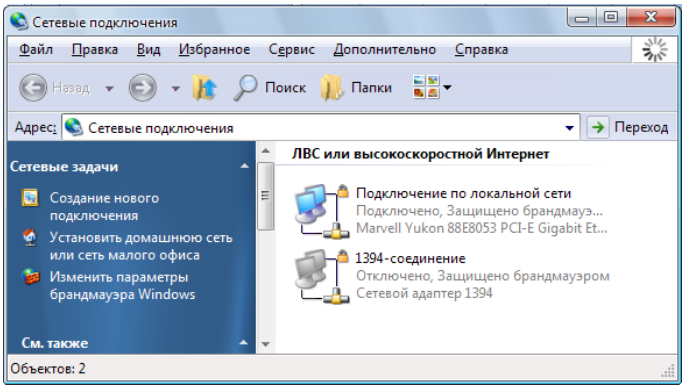



Рис. 2.8 – Свойства сетевого окружения

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. 2.9) вы можете настраивать протоколы сетевых взаимодействий.

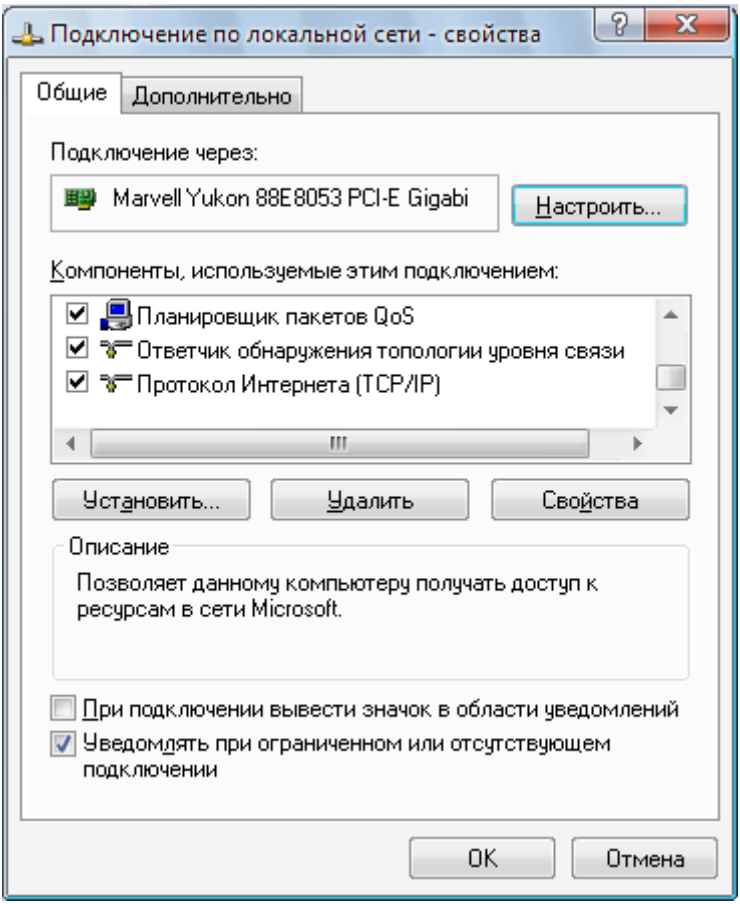



Рис. 2.9 – Свойства подключения по локальной сети

Важным элементом в свойствах подключения по локальной сети, является протокол Интернета TCP/IP. Выбрав это компонент и нажав кнопу «Свойства» откроется окно (рис. 2.10) где можно устанавливать настройки сетевого подключения по протоколу TCP/IP.

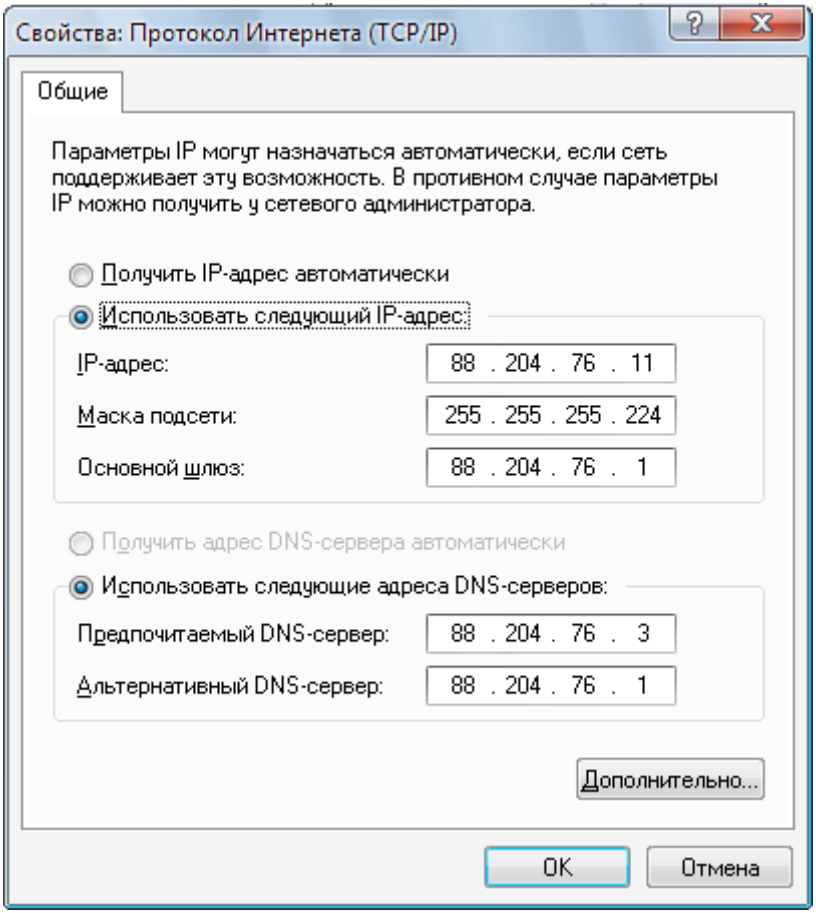



Рис. 2.10 – Свойства протокола Интернета (TCP/IP)

*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 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 2.11 показан процесс работы утилиты «Диагностики сети». На рис. 2.12 результат работы утилиты по диагностики сетевого подключения.

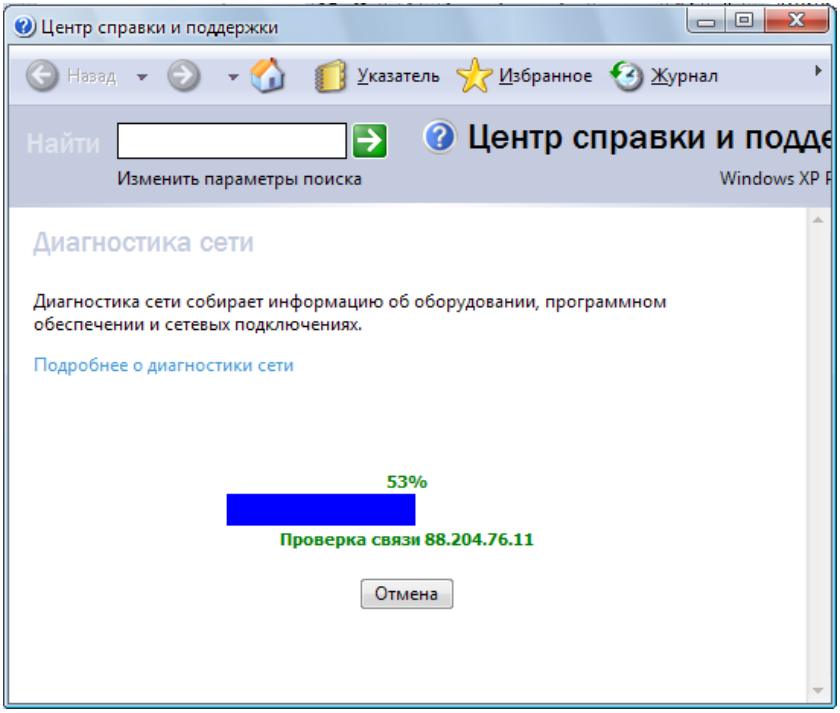



Рис. 2.11 – Ход работы утилиты «Диагностика сети»

*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 вывестись примерно такой листинг:

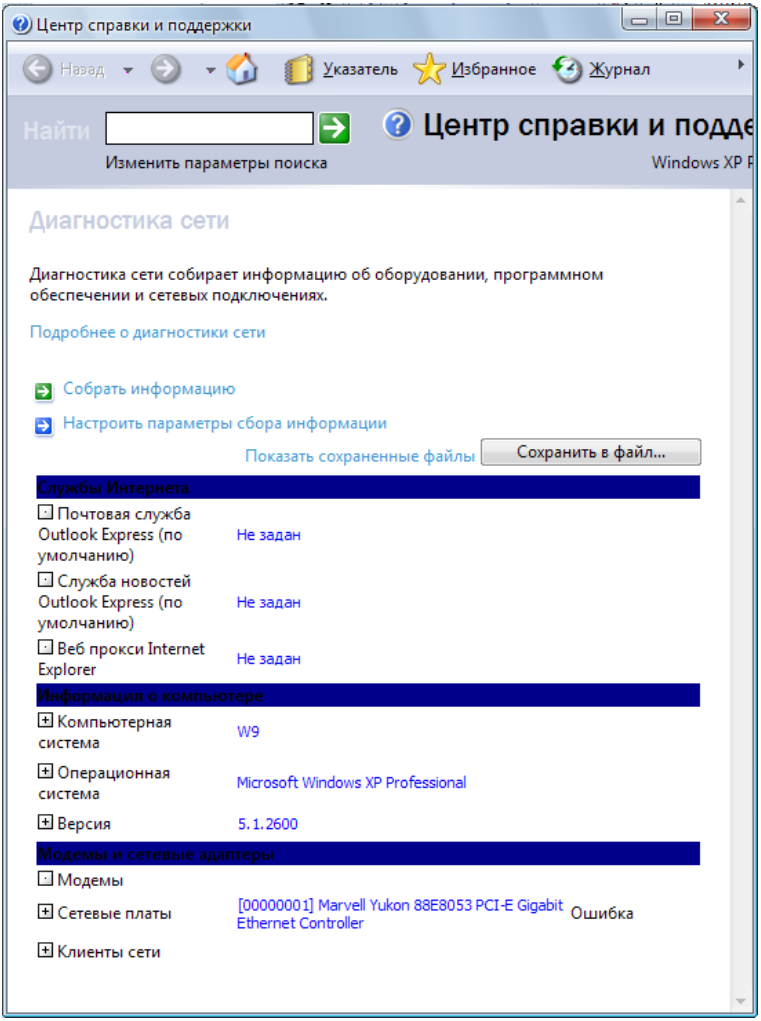



Рис. 2.12 – Ход работы утилиты «Диагностика сети»

Windows IP Configuration

Host Name . . . . . : w9
Primary Dns Suffix . . . . . : aoi.tusur.ru
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No

DNS Suffix Search List. . . . . : aoi.tusur.ru
tomsk.ru
Ethernet adapter Local Area Connection:
Connection-specific DNS Suffix . : aoi.tusur.ru
Description . . . . . : Intel(R) PRO/100 S Desktop
Adapter
Physical Address. . . . . : 00-03-BA-8D-42-5B
Dhcp Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
IP Address. . . . . : 83.192.12.54
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 83.192.12.254
DHCP Server . . . . . : 83.192.12.2
DNS Servers . . . . . : 192.168.0.1
83.192.12.2
Primary WINS Server . . . . . : 83.192.12.2
Secondary WINS Server . . . . . : 213.183.109.8
Lease Obtained. . . . . : 27 августа 2012 г. 19:20:22
Lease Expires . . . . . : 13 октября 2012 г. 19:20:22

Отключите сетевое подключение, повторите команду. При отсутствующем соединении на экран выводится примерно такой листинг:

Windows IP Configuration
Host Name . . . . . : w9
Primary Dns Suffix . . . . . : aoi.tusur.ru
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No
DNS Suffix Search List. . . . . : aoi.tusur.ru
tusur.ru
Ethernet adapter Local Area Connection:
Media State . . . . . : Media disconnected
Description . . . . . : Intel(R) PRO/100 S Desktop
Adapter
Physical Address. . . . . : 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).

*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 зацикливания 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.

*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.

*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.

*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

*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 получат все пользователи рабочей группы.

**План выполнения**

1. Просмотрите через оконный интерфейс ОС Windows свойства протокола TCP/IP. Выпишите IP-адрес.

2. Осуществите диагностику сети.

3. Последовательно исследуйте все возможности сетевых утилит.

4. 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?

## 2.4 Лабораторная работа «Управление устройствами ввода-вывода и файловыми системами в ОС Windows»

**Цель работы**

Целью работы является изучение процесса управления устройствами ввода-вывода, файловыми системами.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита отчета с описанием хода выполнения задания и ответы на теоретические вопросы.

**Теоретические основы**

*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 [2].

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 [2].

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 [2].

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) [2].

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. 2.13) 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 [2].

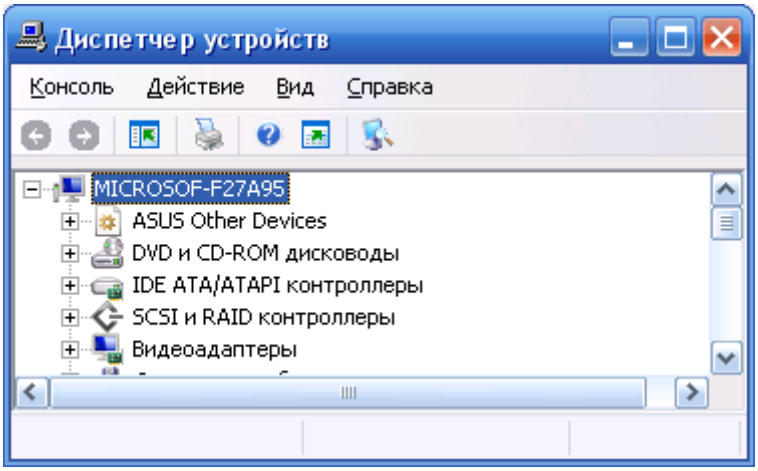



Рис. 2.13 – Окно программы «Диспетчер устройств»

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 (1/0);

– 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 [2].

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.

*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. 2.14). В правой части окна будут отображены все дисковые устройства компьютера и основные параметры их состояния.

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.

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. 2.15) [2].

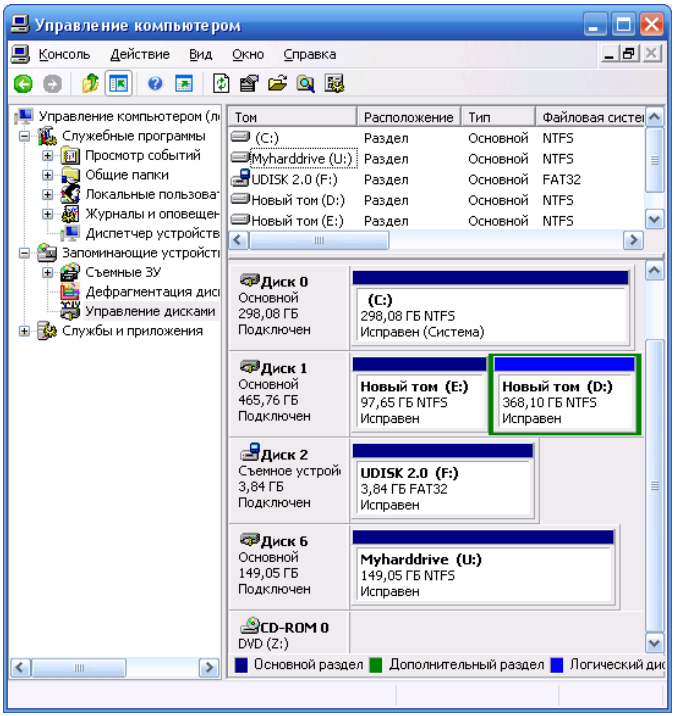



Рис. 2.14 – Вид окна «Управление компьютером» на вкладке «Управление дисками»

Результаты дефрагментации можно просмотреть, нажав на кнопку «Вывести отчет», которая становится доступной после завершения дефрагментации.

*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.

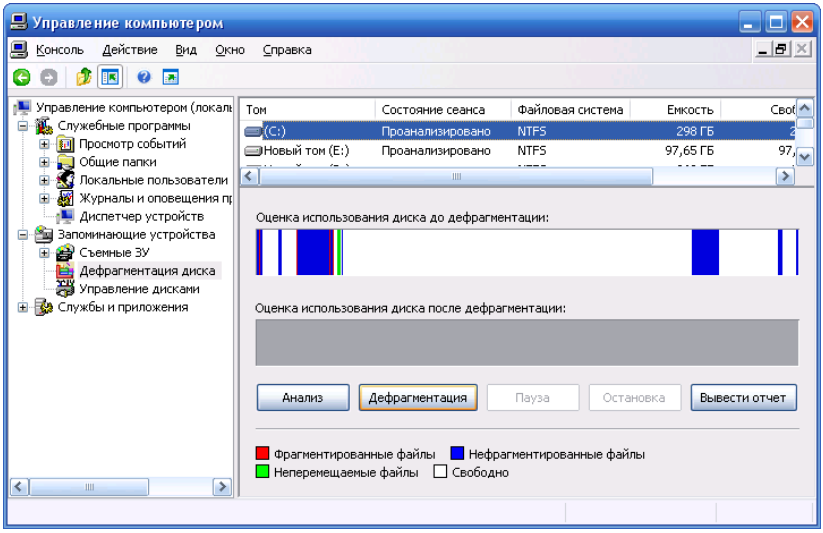



Рис. 2.15 – Вид окна «Управление компьютером» на вкладке «Дефрагментация диска»

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. 2.16).

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.

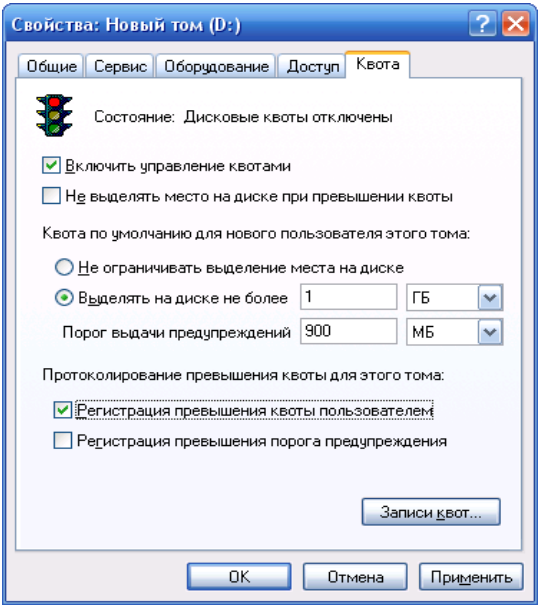



Рис. 2.16 – Вид окна по просмотру свойств диска на вкладке «Квота»

*Обеспечение надежности хранения данных на дисковых накопителях с файловой системой NTF 5.0*

Устанавливая пользователям определенные **разрешения для файлов и каталогов (папок)**, администраторы системы могут защищать конфиденциальную информацию от несанкционированного доступа. Каждый пользователь имеет определенный набор разрешений на доступ к конкретному объекту файловой системы (рис. 2.17). 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 2000 или выше. Точки соединений прозрачны для приложений, это означает, что приложение или пользователь, осуществляющий доступ к локальной папке NTFS, автоматически перенаправляется к другой папке.

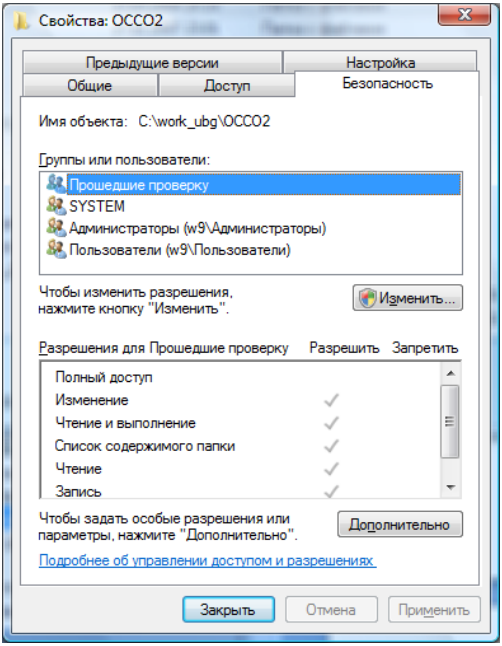



Рис. 2.17 – Вид окна установки разрешений на доступ к конкретному объекту файловой системы

Для работы с точками соединения на уровне томов можно использовать стандартные средства системы — ***утилиту Mountvol*** (рис. 2.18) и оснастку «Управление дисками». Для монтирования папок нужна утилита Linkd (из Windows 2000 Resource Kit).

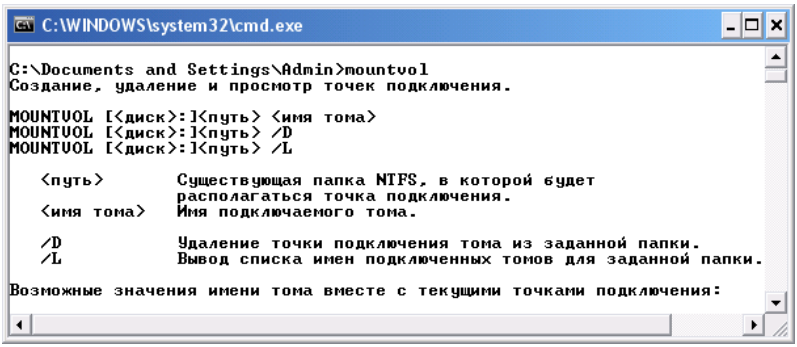



Рис. 2.18 – Вызов утилиты mountvol

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, получат сообщение об ошибке доступа, поскольку они не владеют необходимым личным ключом, позволяющим им расшифровать файл [2].

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 OK. Далее следует нажать кнопку OK в окне свойств зашифровываемого файла или папки. Появится окно, в котором надо указать режим шифрования [2].

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» [2].

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 [2].

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) [2].

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 [2].

**План выполнения**

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.
5. Ответьте на вопросы:
    1. Опишите структуру файловой системы NTFS.
    2. Опишите преимущества и недостатки файловой системы NTFS.

# 3 Методические указания по проведению лабораторных работ (часть 2)

## 3.1 Лабораторная работа «Файлы пакетной обработки в ОС Windows»

**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

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 «HIHO%.TXT» в командной строке должно быть представлено как «HIHO%%.TXT».

*Некоторые команды DOS (Windows)*

Для получения полного списка команд DOS, поддерживаемых вашей операционной системой Windows, построенной на платформе NT, необходимо ввести команду[1]:

**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.

[1] Синтаксис представлен для ОС Windows построенной на базе технологии NT, в ОС MS-DOS и Windows 9x количество аргументов и команд несколько меньше.

CHKNTFS— Отображение или изменение выполнения проверки диска во время загрузки.

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 выполнения команд в пакетном файле.

LABEL — Создание, изменение и удаление меток тома для дисков.

MD — Создание папки.

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 предыдущего значения текущей активной папки, сохраненного с помощью команды PUSHD.

PRINT — Вывод на печать содержимого текстовых файлов.

PROMPT — Изменение приглашения в командной строке Windows.

PUSHD — 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 получение 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 (filenameset) 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, разбивки 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» служит 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 filenameset, 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.

**CHOICE**[2] — ожидает ответа пользователя.

CHOICE [/C[:]варианты] [/N] [/S] [/T[:]c,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[:]c,nn Ответ «c» 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][диск1:][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».

[2] CHOICE — это внешняя команда.

/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

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 Задает 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[OCALE] язык Перекрывает установленные в системе по умолчанию язык и раскладку заданными. Пока существует возможность только одного выбора: ««C»« – наиболее быстрый способ упорядочивания последовательности.

Сортировка всегда идет без учета регистра.

/M[EMORY] килобайтов Задает 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.

**План выполнения**

1. Согласуйте с преподавателем вариант выполнения задания.
2. Согласно варианту, разработайте программный файл. При разработке учтите возможность неправильного запуска ваших программ (например, с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки.

*Варианты заданий на выполнение*

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 токая 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.

## 3.2 Лабораторная работа «Программирование на языке SHELL в ОС Unix»

**Цель работы**

Изучение языка Shell, использование переменных среды, переменных Shell и предопределенных переменных.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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
fi
```

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) (неуспех - 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 op).

1. Для чисел синтаксис такой: N op M, где N, M - числа или числовые переменные;

op принимает значения: -eq, -ne, gt, -lt, -ge, -le.

2. Для файла синтаксис такой: op filename, где op 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 S1
> 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
  else
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 S# -eq 0
then
echo "Usage $0 file..." > &2
exit
fi
  until test S# -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 p), процедура завершается. Иначе в зависимости от наличия ключей t и p выполняются действия, заявленные в комментарии 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 grep. 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 grep. 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 "^I" $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 "^I" $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}
^D
$ 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 С. 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 | = += -= *= /= %= |
| 2 | \|\| |

| | |
|---|---|
| 3 | && |
| 4 | > >= < <= == != ~ !~ |
| 5 | Строка конкатенации «x» «y» становится «xy» |
| 6 | + - |
| 7 | * / % |
| 8 | ++ -- |

**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).

*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 (cat>имя — 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;
cmp — 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;
`...` — инициирует выполнение команды;
( ) — инициирует выполнение команды в порожденном shell;
{ } — инициирует 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;
# — остальная 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.

**План выполнения**

1. Согласуйте с преподавателем вариант выполнения задания.
2. Согласно варианту, разработайте программный файл. При разработке учтите возможность неправильного запуска ваших

программ (например, с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки.

*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 run.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.

## 3.3 Лабораторная работа «Управление процессами в ОС QNX»

**Цель работы**

Познакомиться с визуальным интерфейсом ОС QNX, возможностями различных функций управления процессами и изучить принципы работы с компилятором С в среде ОС QNX.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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("Это родитель, PID сына %d\n", child_pid);
} else {
printf("Это 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.

**План выполнения**

1. Познакомиться с интерфейсом ОС QNX.

Изучить процедуру компиляции (компилятор командной строки gcc). Повторить стандартный ввод – вывод, разбор аргументов и переменных среды. Исследовать работу функций по работе с файлами языка С.

2. 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).

3. Подготовиться к ответам на теоретическую часть лабораторной работы.

Посмотреть задание на следующую лабораторную работу, с целью вспомнить численные методы, чтобы быть готовым к выполнению лабораторной работы.

## 3.4 Лабораторная работа «Управление потоками в ОС QNX»

**Цель работы**

Познакомиться возможностями функций управления потоками в ОС QNX.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 моногопоточночной 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 имеет программу, выполняющую алгоритм трассировки луча. Каждая строка растра не зависит от остальных. Это обстоятельство (независимость строк растра) автоматически приводит к программированию данной задачи как многопоточной.

```
int main ( int argc, char **argv)
{
int x1;
… // Выполнить инициализации
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; x < 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);
}


… // 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);
}
}
```

**План выполнения**

1. Выполнить задание согласно варианту.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

*Варианты заданий на выполнение*

**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 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.

## 3.5 Лабораторная работа «Организация обмена сообщениями в ОС QNX»

**Цель работы**
Познакомиться с механизмами обмена сообщениями в ОС QNX.

**Форма проведения**
Выполнение индивидуального задания.

**Форма отчетности**
Защита программного кода командного файла.

**Теоретические основы**

*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_bn, 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);
- 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.*

*Примеры обмена сообщениями*

*Клиент*

Передача сообщения со стороны клиента осуществляется применением какой-либо функции из семейства *MsgSend()*.

Мы рассмотрим это на примере простейшей из них — *MsgSend():*

```
include <sys/neutrino.h>
int MsgSend(int cold, 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 возвратить 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).

*Сервер*

**Создание канала.** Сервер должен создать канал — то, к чему присоединялся клиент, когда вызывал функцию *ConnectAttach().*

Обычно сервер, однажды создав канал, приберегает его «впрок». Канал создается с помощью 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 rcvid, 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  rcvid;
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).

*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.

**План выполнения**

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.

## 3.6 Лабораторная работа «Управление таймером и периодическими уведомлениями в ОС QNX»

**Цель работы**

Познакомиться с механизмом управления временем ОС QNX — системным таймером.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 proxy. Proxy 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 — 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
// использовать &event.

// SIGEV_SIGNAL_INIT(&event, SIGALRM);
```

```
timer.it_value.tv_sec=  3; //Взвести 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_iriterval. 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,
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_notyfy (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 (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.

**План выполнения**

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.

## 3.7 Лабораторная работа «Использование среды визуальной разработки программ в ОС QNX»

**Цель работы**

Ознакомление со средой визуальной разработки программ и построение пробного приложения в Phab (Photon Application Builder).

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 делаются 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.

#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.

**План выполнения**

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.

## 3.8 Лабораторная работа «Улучшение навыков программирования в ОС QNX»

**Цель работы**

Осмысление знаний о механизмах и ресурсах ОС QNX и с получение дополнительного опыта программирования в среде ОС QNX.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

В зависимости от варианта задания дополнительные теоретические основы можно получить, используя систему помощи QNX, которая включает как описание функционала операционной системы, так и описание функционала среды разработки.

**План выполнения**

Выполнить задание согласно Вашего варианта (Вариант должен быть согласован с преподавателем).

Защита программного кода, выполненного в ходе лабораторной работы.

*Варианты заданий на выполнение*

**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..65535К.

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 клиент.

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

**Вариант 2.** Базовая станция сотовой связи

Описание: Процесса 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 сообщит 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 5х5. 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 делаю 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.

**Вариант 4.** Сетевые крестики-нолики

Описание: Для игры в крестики-нолики, запускаются две программы, которые представляют собой графические окна с матрицей 3х3. В окне есть кнопки, нажатие на которые реализуют функции: подключиться к серверу, начать игру, сдаться. После начала игры пользователи поочередно делаю 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 0), 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 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.

**Вариант 6.** Информационная система «Выборы»

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.

**Вариант 8.** Реализовать ЧАТ для пользователей.

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

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 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 1-30%, то он отображается красным цветом. Если длина датчика достигла 15%, то на кран 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.

# 4 Методические указания по проведению лабораторных работ (часть 3)

## 4.1 Лабораторная работа «Изучение структуры программы на ассемблере»

**Цель работы**

Целью данной работы является изучение структуры программы на ассемблере, использование различных директив сегментации и создание примитивных программ, типа «Hello Word».

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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.

*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 ассемблера:

Оператор директивы [;текст комментария]

Оператор команды [;текст комментария]

Оператор макрокоманды [; текст комментария]

Формат директив:

[Имя] директива [операнд1…операндN][;комментарий]

Формат команд и макрокоманд

[Имя метки:] КОП [операнд1…операндN][;комментарий]

Здесь:

- *имя метки* – идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает;

- *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**;

- 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: **0**ef15**h.**

*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;

- AT 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.

```
masm ;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 4.1.

Таблица 4.1. Упрощенные директивы определения сегмента

| Формат | Формат | Назначение |
|---|---|---|

| директивы (режим MASM) | директивы (режим IDEAL) | |
|---|---|---|
| .CODE [имя] | CODESEG[имя] | 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 4.3). Перечислим эти идентификаторы и их значения (табл. 4.2).

Таблица 4.2. 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. 4.3).

Таблица 4.3. 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.

*Создание COM-программ*

Все вышеприведенные директивы сегментации и примеры программ предназначены для создания программ в EXE-формате[3]. Компоновщик LINK автоматически генерирует особый формат для EXE-файлов, в котором присутствует специальный начальный блок (заголовок) размером не менее 512 байт.

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

[3] За исключением модели памяти TINY при использовании упрощенных директив сегментации.

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
```

*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.

**План выполнения**

1. Ознакомиться со структурой программы на ассемблере.
2. Написать на языке Ассемблера и скомпилировать программу «Hello World» с использованием стандартных директив компиляции в exe-формате.
3. Написать на языке Ассемблера и скомпилировать программу «Hello World» с использованием упрощенных директив компиляции в exe-формате.
4. Написать на языке Ассемблера и скомпилировать программу «Hello World» с использованием стандартных директив компиляции в com-формате.
5. Написать на языке Ассемблера и скомпилировать программу «Hello World» с использованием упрощенных директив компиляции в com-формате.
6. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

## 4.2 Лабораторная работа «Изучение функций ввода/вывода»

**Цель работы**

Целью данной работы является изучение функций ввода/вывода.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 AH 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. 4.1-4.4).

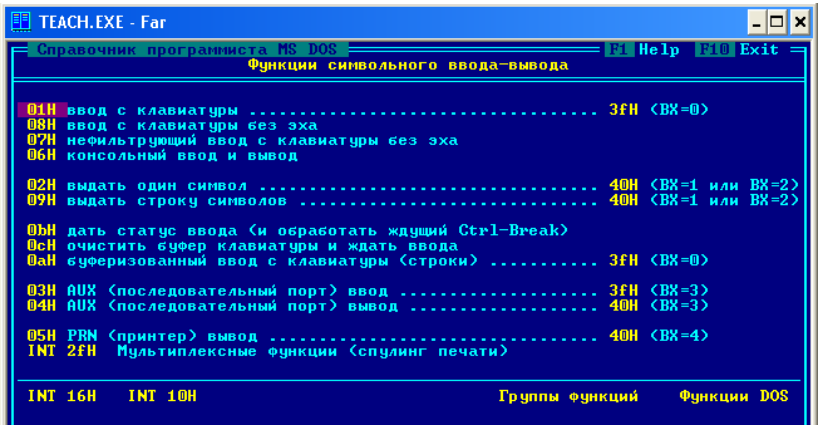



Рис. 4.1 – Функции символьного ввода/вывода

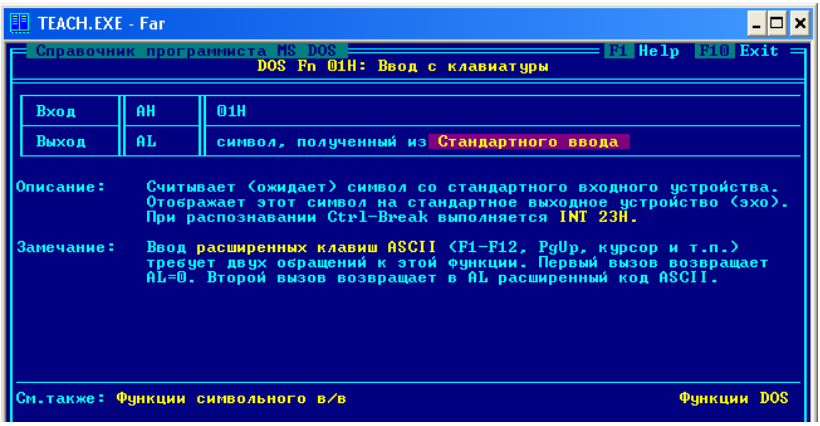



Рис. 4.2 – Функция ввод с клавиатуры

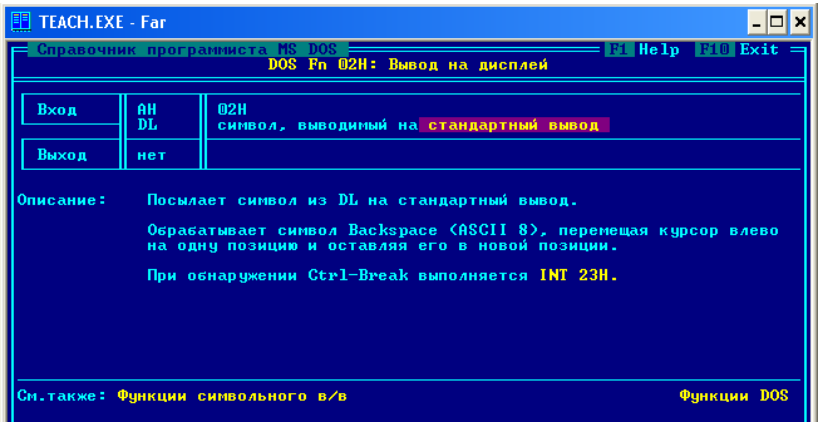



Рис. 4.3 – Функция вывода символа на дисплей

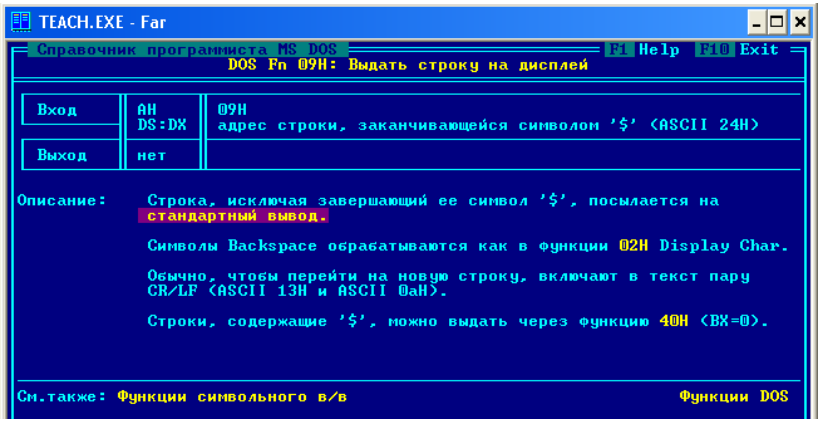



Рис. 4.4 – Функция вывода строки на дисплей

*Примеры использования функций ввода/вывода*
Пример программы ввода шестнадцатеричного числа:

```
data    segment para public 'data'  ;сегмент данных
message db      'Введите две шестнадцатеричные цифры,$'
data    ends
stk     segment stack
  db    256 dup ('?')       ;сегмент 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
xor       ax,ax    ;очистить регистр ax
  mov     ah,1h    ;1h в регистр ah
  int     21h      ;генерация прерывания с номером 21h
  mov     dl,al    ;содержимое регистра al в регистр dl
  sub     dl,30h   ;вычитание: (dl)=(dl)-30h
  cmp     dl,9h    ;сравнить (dl) с 9h
  jle     M1       ;перейти на метку M1 если dl<9h или dl=9h
  sub     dl,7h    ;вычитание: (dl)=(dl)-7h
M1:       mov     cl,4h     ;пересылка 4h в регистр cl
  shl     dl,cl    ;сдвиг содержимого dl на 4 разряда влево
  int     21h      ;вызов прерывания с номером 21h
  sub     al,30h   ;вычитание: (dl)=(dl)-30h
  cmp     al,9h    ;сравнить (al) с 9h          28
  jle     M2       ;перейти на метку M2 если al<9h или al=9h
  sub     al,7h    ;вычитание: (al)=(al)-7h
M2:       add     dl,al     ;сложение: (dl)=(dl)+(al)
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
```

Пример кода программы вывода шестнадцатеричного двухзначного числа:

```
mov bl,16 ; делитель
  xor ax,ax          ; обнуляем ax
  mov al,rez         ; заносим в al число на которое нужно
напечатать
  div bl             ; ax делим на bl, в al частное, ah остаток
  push ax            ; сохраняем ax в стек
  mov dl,al          ; готовим к печати первую цифру числа
  add dl,30h
  mov ah,02          ; номер функции вывода символа
```

```
int 21h          ; печатаем первую цифру числа
pop ax           ; достаем ax из стека
mov dl,ah        ; готовим к печати вторую цифру числа
add dl,30h
mov ah,02        ; номер функции вывода символа
int 21h          ; печатаем вторую цифру числа
```

Пример кода программы вывода двоичного числа:

```
mov     cx,16
m1:                     ; число которое необходимо вывести на экран
sal bx,1         ; арифм. сдвиг влево на 1 бит, значение бита
                 ; попадает в флаг cf
adc dl,30h       ; dl=dl+30h+cf
xor dx,dx        ; обнуляем dx (dl – младшая часть dx)   mov ah,02   ; номер функции вывода символа
int 21h
loop m1          ; если cx<>0 то cx=cx-1 и переход на метку m1
```

**План выполнения**

1. Написать на языке Ассемблера и скомпилировать программу ввода двух шестнадцатеричных чисел и вывода на экран этих чисел в двоичном виде.

2. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

## 4.3 Лабораторная работа «Изучение арифметических и логических команд»

**Цель работы**

Целью данной работы является изучение арифметических и логических команд микропроцессора.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 m1;если нет переноса, то перейти
 ;на m1
<16> adc ah,0;в 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_cf

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 4.4.

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:

Таблица 4.4. Расположение операндов и результата при умножении

| сомножитель_1 | сомножитель_2 | Результат |
|---|---|---|
| Байт | al | 16 бит в ax:<br>al – младшая часть результата;<br>ah – старшая часть результата |
| Слово | ax | 32 бит в паре dx:ax:<br>ax – младшая часть результата;<br>dx – старшая часть результата |
| Двойное слово | eax | 64 бит в паре edx:eax:<br>eax – младшая часть результата;<br>edx – старшая часть результата |

- 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 4.5.

Таблица 4.5. Расположение операндов и результата при делении

| Делимое | Делитель | Частное | Остаток |
|---|---|---|---|
| 16 бит<br>в регистре ax | Байт<br>регистр или ячейка памяти | Байт<br>в регистре al | Байт<br>в регистре ah |
| 32 бит<br>dx – старшая часть<br>ax – младшая часть | Слово 16 бит<br>регистр или ячейка памяти | Слово 16 бит в регистре ax | Слово 16 бит в регистре dx |
| 64 бит<br>edx – старшая часть<br>eax – младшая часть | Двойное слово 32 бит<br>регистр или ячейка памяти | Двойное слово<br>32 бит в регистре eax | Двойное слово<br>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).

*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.

*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.

**План выполнения**

1. Написать на языке Ассемблера и скомпилировать программу согласно полученному от преподавателя варианту.

2. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

*Варианты заданий на выполнение*

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 бит 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 A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A-B)*2+1Ah.

2. Вывести на экран 1 если шестой бит числа C установлен и 0 в противном случае.

Вариант 15

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A-7h)*4-B.
2. Сбросить третий бит числа C, если он установлен.
3. Вывести на экран число C в двоичном виде.

Вариант 16

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A+4*B.
2. Выполнить циклический сдвиг числа C на 3 бита вправо.
3. Вывести на экран число C в двоичном виде.

Вариант 17

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A*3+B*2.
2. Выполнить арифметический сдвиг числа C на 2 бита вправо.
3. Вывести на экран число C в двоичном виде.

Вариант 18

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A*3+B*2.
2. Выполнить арифметический сдвиг числа C на 2 бита вправо.
3. Вывести на экран число C в двоичном виде.

Вариант 19

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 20

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A+17h)/2+B.
2. Обнулить все четные биты C.
3. Вывести на экран число C в двоичном виде.

## 4.4 Лабораторная работа «Модульное программирование»

**Цель работы**

Целью данной работы является изучение написания процедур и макросов с использованием модульной структуры программы.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*Процедуры на языке ассемблера*

Для оформления процедур существуют специальные директивы proc/endp 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 eip/ip;
- для процедур дальнего типа – последовательно восстановить из стека содержимое eip/ip 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+число), то есть указатель стека сдвигается на число байт, равное значению число;
- если use32, то sp=(sp+2*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 eip/ip и в этот же регистр загружается новое значение адреса, соответствующее метке;

При дальней адресации – в стек заносится содержимое указателя команд eip/ip и cs. Затем в эти же регистры загружаются новые значения адресов, соответствующие дальней метке;

*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:

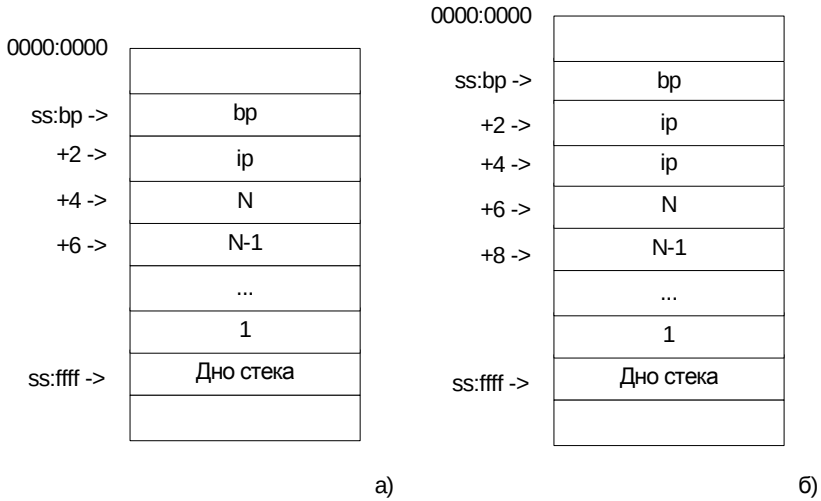
- если имя – это переменная, то тип может принимать значения byte, word, dword, pword, fword, qword и tbyte;
- если имя – это процедура, то тип может принимать значения near или far;
- если имя – это константа, то тип должен быть abs.

Остановимся боле 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. 4.5 показана структура стека при ближней (а) и дальней (б) адресации в нутрии вызываемой процедуры.




а) б)

Рис. 4.5 – 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-2-элементу
```

*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.

*Макросредства языка ассемблера*

*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.

*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 макробиблиотеке. Если у вас есть универсальные макрокоманды, которые используются практически во всех ваших программах, то их целесообразно записать в так называемую макробиблиотеку. 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.

**План выполнения**

1. Модифицировать программу из предыдущей лабораторной работы согласно полученному от преподавателя варианту.
2. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

*Варианты заданий на выполнение*

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
2. Написать макрос для ввода чисел
3. Написать макрос для расчета
Передача параметров через общую память

Вариант 10
1. Написать процедуру для вывода результата
2. Написать макрос для ввода чисел
3. Написать макрос для расчета
Передача параметров через общую память

Вариант 11

1. Написать макрос для вывода результата
2. Написать процедуру для ввода чисел
3. Написать макрос для расчета
Передача параметров через общую память

Вариант 12
1. Написать макрос для вывода результата
2. Написать макрос для ввода чисел

3. Написать процедуру для расчета
Передача параметров через общую память

Вариант 13
1. Написать процедуру для вывода результата
2. Написать процедуру для ввода чисел
3. Написать макрос для расчета
Передача параметров через общую память

Вариант 14
1. Написать макрос для вывода результата
2. Написать процедуру для ввода чисел
3. Написать процедуру для расчета
Передача параметров через общую память

Вариант 15

1. Написать процедуру для вывода результата
2. Написать макрос для ввода чисел
3. Написать процедуру для расчета
Передача параметров через общую память

Вариант 16
1. Написать процедуру для вывода результата
2. Написать процедуру для ввода чисел
3. Написать процедуру для расчета
Передача параметров через общую память

Вариант 17
1. Написать макрос для вывода результата
2. Написать макрос для ввода чисел
3. Написать макрос для расчета
Передача параметров через регистры

Вариант 18
1. Написать процедуру для вывода результата
2. Написать макрос для ввода чисел
3. Написать макрос для расчета
Передача параметров через регистры

Вариант 19

1. Написать макрос для вывода результата
2. Написать процедуру для ввода чисел
3. Написать макрос для расчета
Передача параметров через регистры

Вариант 20
1. Написать макрос для вывода результата
2. Написать макрос для ввода чисел
3. Написать процедуру для расчета
Передача параметров через регистры

## 4.5 Лабораторная работа «Работа с массивами ассемблера»

**Цель работы**

Целью работы является совершенствование навыков работы на языке ассемблера, овладевание навыками работы с массивами данных.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

При программировании на языке ассемблера используются данные следующих типов:

– Непосредственные данные, представляющие собой числовые или символьные значения, являющиеся частью команды. Непосредственные данные формируются программистом в процессе написания программы для конкретной команды ассемблера.

– Данные простого типа, описываемые с помощью ограниченного набора директив резервирования памяти, позволяющих выполнить самые элементарные операции по размещению и инициализации числовой и символьной информации.

– Данные сложного типа, которые были введены в язык ассемблера с целью облегчения разработки программ. Сложные типы данных строятся на основе базовых типов. Введение сложных типов данных позволяет несколько сгладить различия между языками высокого уровня и ассемблером. У программиста появляется возможность сочетания преимуществ языка ассемблера и языков высокого уровня (в направлении абстракции данных), что в конечном итоге повышает эффективность конечной программы.

Данных сложного типа может быть несколько: массивы, структуры, объединения, записи.

При необходимости использовать массив в программе его нужно моделировать одним из следующих способов:

- mas dd 1,2,3,4,5 ; массив из 5 элементов. Размер каждого элемента 4 байта

- mas dw 5 dup (0) ; массив из 5 нулевых элементов. Размер каждого элемента 2 байта

- Используя директивы label и rept.

```
mas_b label byte
mas_w label word
rept 4
        dw 0f1f0h
endm
```

Доступ к элементу массива организуется по формуле:
база + (индекс * размер элемента)

**План выполнения**

1. Написать на языке Ассемблера и скомпилировать программу согласно полученному от преподавателя варианту.
2. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

*Варианты заданий на выполнение*

Вариант 1.
Ввести матрицу из N*N элементов.
Выбрать все элементы главной диагонали матрицы.

Вариант 2.
Ввести матрицу из N*N элементов.
Найти максимальный и минимальный элемент матрицы.

Вариант 3.
Ввести матрицу из N*N элементов.
Получить транспонированную матрицу.

Вариант 4.
Ввести матрицу из N*N элементов.
Из всех элементов матрицы вычесть минимальный элемент.

Вариант 5.
Ввести матрицу из N*N элементов.
Обнулить в матрице все четные числа

Вариант 6.

Ввести матрицу из N*N элементов.
Обнулить в матрице все нечетные числа.

Вариант 7.
Ввести матрицу из N*N элементов.
Поменять в матрице четные столбцы с нечетными.

Вариант 8.
Ввести матрицу из N*N элементов.
Поменять в матрице четные строки с нечетными.

Вариант 9.
Ввести матрицу из N*N элементов.
Умножить элементы матрицы на минимальный элемент.

Вариант 10.
Ввести матрицу из N*N элементов.
Найти среднеарифметическое число всех элементов матрицы (целая часть, остаток)

Вариант 11.
Ввести матрицу из N*N элементов.
Найти определитель матрицы 3*3.

Вариант 12.
Ввести матрицу из N*N элементов.
Сложить все элементы матрицы построчно с элементом на главной диагонали текущей строки.

Вариант 13.
Ввести матрицу из N*N элементов.
Сложить все элементы матрицы по столбцам с элементом на главной диагонали текущего столбца.

Вариант 14.
Ввести матрицу из N*N элементов.
Посчитать сумму элементов всех строк.

Вариант 15.
Ввести матрицу из N*N элементов.
Посчитать сумму элементов всех столбцов.

Вариант 16.
Ввести матрицу из N*N элементов.
Посчитать сумму элементов главной диагонали

Вариант 17.
Ввести матрицу из N*N элементов.
Посчитать сумму элементов обратной диагонали.

Вариант 18.
Ввести два массива одинаковой размерности.
Посчитать сумму и разность соответствующих элементов массивов (с одинаковым индексом).

Вариант 19.
Ввести два массива одинаковой размерности.
Выполнить действия по умножению и делению каждого элемента с одинаковым индексом (при делении запоминать только целую часть).

Вариант 20.
Ввести два массива и найти максимальные элементы.
Сравнить их.
Вывести тот массив, у которого максимальный элемент меньше, чем максимальный элемент другого массива.

## 4.6 Лабораторная работа «Интерфейс с языками высокого уровня и обработка массивов»

**Цель работы**

Целью работы является изучение связи подпрограмм на ассемблере с программами, написанными на языках высокого уровня и обработка массивов на языке ассемблере.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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 4.6). Передача аргументов, как правило, осуществляется через стек.

*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:

Таблица 4.6. 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 |

```
extern "C" void asmpoc(char ch, unsigned x, unsigned y)
```

Функция asmpoc должна описываться на ассемблере следующим процедурным блоком:

```
public _asmproc
 _asmproc proc ...
 ...
 _asmproc 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 4.7.

Таблица 4.7. Соотношение моделей памяти и типов указателей

| Модель | Ключ BCC-компилятора | 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.

*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.

**План выполнения**

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.
5. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

## 4.7 Лабораторная работа «Использование цепочечных команд»

**Цель работы**

Целью работы является изучение работы цепочечных команд при обработке массивов на языке ассемблера.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*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:

.data

```
source    db      'Тестируемая строка','$'  ;строка-источник
dest      db      19 DUP (' ')      ;строка-приёмник
.code
  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-источника
  lea     di,dest   ;загрузка в DI смещения строки-приёмника
  mov     cx,20     ;для префикса rep – счетчик повторений
rep       movs    dest,source       ;пересылка строки

  lea     dx,dest
  mov     ah,09h    ;вывод на экран строки-приёмника
  int     21h
exit:
```

**Сравнение цепочек в памяти.**

```
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.

Пример программы сравнения цепочек в памяти:

```
.data
obl1    db      'Строка для сравнения'
obl2    db      'Строка для сравнения'
a_obl1  dd      obl1
a_obl2  dd      obl2
.code
  ...
  cld  ;просмотр цепочки в направлении возрастания адресов

  mov     cx,20   ;длина цепочки
  lds     si,a_obl1       ;адрес источника в пару ds:si
  les     di,a_obl2       ;адрес назначения в пару es:di
  repe    cmpsb           ;сравнивать, пока равны
  jnz     m1      ;если не конец цепочки, то встретились разные элементы          ...                     ;действия, если цепочки совпали ...
  m1:
  ...                     ;действия, если цепочки не совпали
```

**Сканирование цепочек в памяти**

```
scas приемник
scasb
scasw
scasd
```

*Алгоритм работы:*

- выполнить вычитание (элемент цепочки-(eax/ax/al)). Элемент цепочки локализуется парой es:edi/di. Замена сегмента es не допускается;
- по результату вычитания установить флаги;
- изменить значение регистра edi/di на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:

df=0 — величина положительная, то есть просмотр от начала цепочки к ее концу;

df=1 — величина отрицательная, то есть просмотр от конца цепочки к ее началу.

Пример программы сканирования цепочек в памяти:

```
;сосчитать число пробелов в строке str
.data
```

```
str    db     '...'
len_str=$-str
.code
  mov    ax,@data
  mov    ds,ax
  mov    es,ax
  lea    di,str
  mov    cx,len_str     ;длину строки — в cx
  mov    al,' '
  mov    bx,0    ;счетчик для подсчета пробелов в строке
  cld
cycl:
repe       scasb
  jcxz    exit    ;переход на exit, если цепочка просмотрена полностью
  inc    bx
  jmp    cycl
exit:  ...
```

**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:
```

**План выполнения**

1. Реализовать задание из лабораторной работы 4.5 полностью на языке ассемблера с применением цепочечных команд.
2. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

## 4.8 Лабораторная работа «Программирование устройства с плавающей арифметикой»

**Цель работы**

Целью работы является изучение работы команд устройства с плавающей арифметикой (FPU) на языке ассемблера.

**Форма проведения**

Выполнение индивидуального задания.

**Форма отчетности**

Защита программного кода командного файла.

**Теоретические основы**

*Организация FPU*

*Общее положение*

Устройство плавающей арифметики Intel-архитектуры предоставляет возможность высокопроизводительных вычислений. Оно поддерживает вещественные, целые и BCD целые типы данных, алгоритмы вещественной арифметики и архитектуру обработки исключение определенные в стандартах IEEE 754 и 854.

Intel-архитектура FPU развивалась параллельно с Intel-архитектурой ранних процессоров. Первые математические сопроцессоры (Intel 8087, Intel 287 и Intel 387) были дополнительными устройствами к Intel 8086/8088, Intel 286 и Intel 386 процессорам соответственно.

Начиная с Intel 486 DX процессора, устройство плавающей арифметики помещается на один чип с самим процессором.

**Формат чисел с плавающей точкой**

Для увеличения скорости и эффективности вычислений, компьютеры или FPU обычно представляют вещественные числа в двоичном формате с плавающей точкой. В этом формате вещественное число имеет три части: знак, мантиссу и порядок (рис. 4.6).

Знак – это двоичное значение, которое определяет либо число положительное (0), либо число отрицательное (1).

Мантисса имеет две части: 1 битовое целое число (известное как J-бит) и двоичная вещественная часть. J-бит обычно не присутствует, а имеет фиксированное значение.

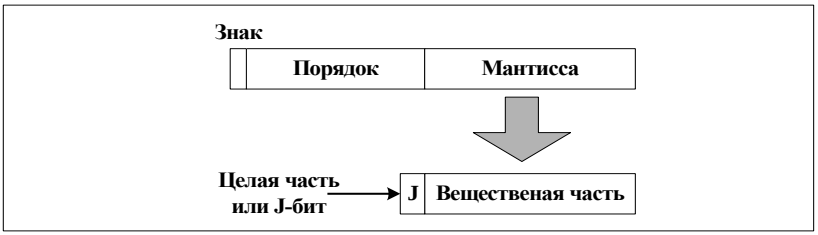



Рис. 4.6 – Формат числа сплавающей точкой

Порядок – это двоичное целое число, определяющее степень, в которую необходимо возвести 2.

Среда выполнения инструкций FPU состоит из 8 регистров данных и следующих регистров специального назначения (рис. 4.7):

- Регистр статуса.
- Регистр состояния.
- Регистр слова тега.
- Регистра указателя инструкции.
- Регистра последнего операнда.
- Регистра кода команды.
-


Рис. 4.7 – Регистры FPU

Регистры данных FPU состоят из восьми 80 битовых регистров. Значения хранятся в этих регистрах в расширенном вещественном формате. Когда вещественные, целые или BCD целые значения загружаются из памяти в любой из регистров данных FPU, значения автоматически конвертируются в расширенный вещественный формат. Когда результат вычислений перемещается назад в память, он конвертируется в один из типов данных FPU

FPU инструкции обращаются к регистрам данных как к регистрам стека. Все адресации к регистрам данных происходят относительно регистра на вершине стека. Номер регистра на вершине стека находится в поле TOP слова статуса FPU. Операция загрузки уменьшает TOP на 1 и загружает значение в новый регистр на вершине стека. Операция сохранения сохраняет содержимое регистра на вершине стека и увеличивает TOP на 1.

Если операция загрузки выполняется, когда TOP равно 0, происходит циклический возврат и новое значение TOP становится равно 7. Исключение переполнения стека плавающей арифметики происходит для индикации того, что произошел циклический возврат и не сохраненное значение может быть перезаписано.

Многие инструкции плавающей арифметики имеют несколько режимов адресации, что позволяет программисту неявно оперировать на вершине стека или явно оперировать с регистрами относительно вершины стека.

Пример исследования команд передачи данных в FPU:

```
.586p
masm
model     use16 small
.stack    100h
.data     ;сегмент данных
ch_dt     dt      43567    ;ch_dt=00 00 00 00 00 00 00 04 35 67
x dw      3       ;x=00 03
y_real    dq      34e7     ;y_real=41 b4 43 fd 00 00 00 00
ch_dt_st  dt      0
x_st      dw      0
y_real_st dq      0
.code
main      proc    ;начало процедуры main
  mov     ax, @data
  mov     ds, ax
  fbld    ch_dt   ;st(0)=43567
  fild    x       ;st(1)=43567, st(0)=3
  fld     y_real  ;st(2)=43567, st(1)=3, st(0)=340000000
  fxch    st(2)   ;st(2)=340000000, st(1)=3, st(0)=43567
  fbstp   ch_dt_st;st(1)=340000000, st(0)=3 ch_dt_st=00 00 00 00
00 00 00 04 35 67
  fistp   x_st    ;st(0)=340000000, x_st=00 03
  fstp    y_real_st        ;y_real_st=41 b4 43 fd 00 00 00 00
```

```
exit:   mov   ax, 4c00h
  int   21h
main    endp
end     main
```

Пример вычисление выражения z=(sqrt(|x|)-y)^2:

```
.586p
masm
model   use16 small
.stack  100h
.data   ;сегмент данных
;исходный данные:
x dd    -29e-4
y dq    4.6
z dd    0
.code
main    proc
  mov   ax,@data
  mov   ds,ax
  finit ;приведение сопроцессора в начальное состояние
  fld   x       ;st(0)=x
  fabs          ;st(0)=|x|
  fsqrt
  fsub  y       ;st(0)=sqrt|x|-y
  fst   st(1)
  fmul
  fst   z
exit:   mov   ax,4c00h
  int   21h
main    endp
end     main
```

**План выполнения**

1. Написать на языке Ассемблера и скомпилировать программу, которая вычисляет выражение согласно варианту, полученному от преподавателя.

2. Подготовиться к сдаче лабораторной работы по теоретической части лабораторной работы.

*Варианты заданий на выполнение*

Вариант 1.

Z=5.3*X^2+7.2*Y+2.8

Вариант 2.
Z=5.3+ sqrt(|X|)/Y

Вариант 3.
Z=sqrt(|X|)+Y^3

Вариант 4.
Z=X*Y/3.4

Вариант 5.
Z=X+Y/(|X-Y|)

Вариант 6.
Z=2.1*X^Y

Вариант 7.
Z=4.4*(X+Y)

Вариант 8.
Z=sqtr(|X-Y|)*3.3

Вариант 9.
Z=X^3-Y^2

Вариант 10.
Z=|X*Y/4.3|

Вариант 11.
Z=sqtr(X)-sqrt(Y)

Вариант 12.
Z=X*Y-X/Y

Вариант 13.
Z=(X-1.4)*(Y-3.1)

Вариант 14.
Z=(X+2.5)/(Y+0.3)

Вариант 15.
Z=|X|/|Y|+X*Y

Вариант 16.
Z=|X/Y|+|X*Y|

Вариант 17.
Z=sqrt(|X*Y|)

Вариант 18.
Z=sqrt(|X/Y|)

Вариант 19.
Z=sqrt(|X|)+Y^2

Вариант 20.
Z=X^Y+Y^X

# 5 Методические указания к самостоятельной работе

## 5.1 Общие положения

Целями самостоятельной работы является систематизация, расширение и закрепление теоретических знаний, приобретение навыков - научно-исследовательской и производственно-технологической деятельности.

Самостоятельная работа по дисциплине «Операционные системы и сети» включает следующие виды активности студента:

- проработка лекционного материала;
- подготовка к лабораторным работам;
- подготовка к экзамену.

## 5.2 Проработка лекционного материала

Для проработки лекционного материала студентам рекомендуется воспользоваться конспектом, сопоставить записи конспекта с соответствующими разделами методического пособия [1]. Целесообразно ознакомиться с информацией, представленной в файлах, содержащих презентации лекций, предоставляемых преподавателем. Для проработки лекционного материала студентам, помимо конспектов лекций, рекомендуются следующие главы учебных пособий [1-5] по разделам курса:

Часть 1.

Глава 1 [1]: Принципы построения вычислительных систем (Общее представление о вычислительной системе. История развития вычислительных систем. Электронные вычислительные машины. Архитектура ЭВМ. Архитектуры процессоров).

Глава 2 [1]: Организация памяти (Единицы измерения информации и их представление в ЭВМ. Иерархия памяти. Адресация и распределение памяти в реальном режиме работы микропроцессора Intel x86. Адресация и распределение памяти в защищенном режиме работы микропроцессора Intel x86. Адресация и распределение памяти в архитектуре AMD64. Управление памятью в ОС Windows).

Глава 3 [1]: Управление устройствами ввода-вывода (Описание устройств ввода-вывода. Организация дисковых устройств/ Обзор файловых систем. Управление устройствами ввода-вывода и файловыми системами в ОС Windows).

Глава 4 [1]: Принципы построения вычислительных сетей и телекоммуникаций (Сетевая модель OSI. Физическая инфраструктура сети. Логическая организация сети. Основы TCP/IPv4. Диагностика сети).

Часть 2.

Глава 1 [2]: Введение в операционные среды, системы и оболочки (Основные понятия. Классификация операционных систем. Классификация построений ядер операционных систем. Представление об интерфейсах прикладного программирования. Платформенно-независимый интерфейс POSIX. Основные принципы построения операционных систем).

Глава 1 [3]: Организация вычислительных задач (Процессы. Ресурсы. Режим мультипрограммирования. Потоки. Волокна. Планирование процессов и диспетчеризация задач. Взаимодействие и синхронизация задач. Прерывания. Управление задачами в ОС Windows).

Глава 3 [2]: Интерфейсы операционных систем (Интерфейс командной строки ОС Windows. Интерфейс командной строки ОС Unix).

Глава 3 [4]: Организация операционных систем реального времени (Функциональные требования ОСРВ. Архитектуры построения ОСРВ. Разделение ОСРВ по способу разработки).

Глава 4 [4]: Стандарты на ОСРВ (SCEPTRE. POSIX. DO-178B. ARINC-653. OSEK).

Глава 5 [4]: Обзор ОСРВ (Классификация ОСРВ в зависимости от происхождения. Системы на основе обычных ОС. Самостоятельные ОСРВ. Специализированные ОСРВ).

Глава 6 [4]: Микроядро ОС QNX Neutrino (Потоки и процессы. Механизмы синхронизации. Межзадачное взаимодействие. Управление таймером. Сетевое взаимодействие. Первичная обработка прерываний. Диагностическая версия микроядра).

Глава 7 [4]: Администратор процессов и управление ресурсами в ОС QNX (Управление процессами. Обработка прерываний. Администраторы ресурсов. Файловые системы. Инсталляционные пакеты. Символьные устройства. Сетевая подсистема. Технология JumpGate. Графический интерфейс пользователя).

Часть 3.

Глава 2 [3]: Программная модель микропроцессора Intel Pentium (Состав программной модели. Регистры общего назначения. Сегментные регистры. Регистры состояния и управления. Системные регистры).

Глава 3 [5]: Программирование на языке Ассемблера Intel 80x86 (Структура программы на ассемблере. Способы адресации. Функции ввода/вывода, арифметические и логические команды. Модульное программирование. Интерфейс с языками высокого уровня).

При изучении учебно-методического пособия [1] студенту рекомендуется самостоятельно ответить на вопросы, приводимые в конце каждой главы. Рекомендуется сформулировать вопросы преподавателю и задать их либо посредством электронной образовательной среды вуза, либо перед началом следующей лекции.

## 5.3 Подготовка к лабораторным работам

Для подготовки к лабораторным работам «Управление задачами в ОС Windows» студентам необходимо изучить главу 1 учебного пособия [3] и пункт 2.1 данных методических указаний.

Для подготовки к лабораторным работам «Исследование блоков управления памятью» студентам необходимо изучить главу 2 учебного пособия [1] и пункт 2.2 данных методических указаний.

Для подготовки к лабораторным работам «Диагностика IP-протокола» студентам необходимо изучить главу 4 учебного пособия [1] и пункт 2.3 данных методических указаний.

Для подготовки к лабораторным работам «Управление устройствами ввода-вывода и файловыми системами в ОС Windows» студентам необходимо изучить главу 3 учебного пособия [1] и пункт 2.4 данных методических указаний.

Для подготовки к лабораторным работам «Файлы пакетной обработки в ОС Windows» студентам необходимо изучить раздел 3.1 учебного пособия [2] и пункт 3.1 данных методических указаний.

Для подготовки к лабораторным работам «Программирование на языке SHELL в ОС Unix» студентам необходимо изучить раздел 3.2 учебного пособия [2] и пункт 3.2 данных методических указаний.

Для подготовки к лабораторным работам «Управление процессами в ОС QNX» студентам необходимо изучить главу 7 учебного пособия [4] и пункт 3.3 данных методических указаний.

Для подготовки к лабораторным работам «Управление потоками в ОС QNX» студентам необходимо изучить главу 6 учебного пособия [4] и пункт 3.4 данных методических указаний.

Для подготовки к лабораторным работам «Организация обмена сообщениями в ОС QNX» студентам необходимо изучить главу 6 учебного пособия [4] и пункт 3.5 данных методических указаний.

Для подготовки к лабораторным работам «Управление таймером и периодическими уведомлениями в ОС QNX» студентам необходимо изучить главу 6 учебного пособия [4] и пункт 3.6 данных методических указаний.

Для подготовки к лабораторным работам «Использование среды визуальной разработки программ в ОС QNX» студентам необходимо изучить главу 7 учебного пособия [4] и пункт 3.7 данных методических указаний.

Для подготовки к лабораторным работам «Улучшение навыков программирования в ОС QNX» студентам необходимо изучить главу 7 учебного пособия [4] и ознакомиться с пунктом 3.8 данных методических указаний.

Для подготовки к лабораторным работам «Изучение структуры программы на ассемблере» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.1 данных методических указаний.

Для подготовки к лабораторным работам «Изучение функций ввода/вывода» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.2 данных методических указаний.

Для подготовки к лабораторным работам «Изучение арифметических и логических команд» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.3 данных методических указаний.

Для подготовки к лабораторным работам «Модульное программирование» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.4 данных методических указаний.

Для подготовки к лабораторным работам «Работа с массивами ассемблера» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.5 данных методических указаний.

Для подготовки к лабораторным работам «Интерфейс с языками высокого уровня и обработка массивов» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.6 данных методических указаний.

Для подготовки к лабораторным работам «Использование цепочечных команд» студентам необходимо изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.7 данных методических указаний.

Для подготовки к лабораторным работам «Программирование устройства с плавающей арифметикой» студентам необходимо

изучить главу 2 учебного пособия [3] и ознакомиться с пунктом 4.8 данных методических указаний.

## 5.4 Подготовка к экзамену

Для подготовки к экзамену рекомендуется повторить соответствующие тематике разделы учебных пособий [1-5]. Экзаменационные вопросы представлены в рабочей программе изучаемой дисциплине, размещенной на образовательном портале ТУСУРа: https://edu.tusur.ru/.

# Список литературы

1. Гриценко, Ю. Б. Вычислительные системы, сети и телекоммуникации: Учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2015. — 134 с. — Режим доступа: https://edu.tusur.ru/publications/5053.

2. Гриценко, Ю. Б. Операционные системы. Ч.1.: учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2009. — 187 с. — Режим доступа: https://edu.tusur.ru/publications/25.

3. Гриценко, Ю. Б. Операционные системы. Ч.2.: Учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2009. — 230 с. — Режим доступа: https://edu.tusur.ru/publications/31.

4. Гриценко, Ю. Б. Системы реального времени: Учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2017. — 253 с. — Режим доступа: https://edu.tusur.ru/publications/6816Зыль С.Н. Операционная система реального времени QNX: от теории к практике. — СПб.: БХВ-Петербург, 2004. — 192с.: ил.

5. Гриценко, Ю. Б. Системное программное обеспечение: Учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2006. — 174 с. — Режим доступа: https://edu.tusur.ru/publications/635