Методические указания по выполнению лабораторных работ
и организации самостоятельной работы студентов
по дисциплине

«Системы реального времени»

Для студентов направления подготовки **09.04.04 Программная инженерия (Методы и технологии индустриального проектирования программного обеспечения)**
Уровень основной образовательной программы: **Магистратура**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

Утверждаю:
Зав. каф АОИ, профессор
_____ Ю.П. Ехлаков
«____» _____2017г.

Методические указания

по выполнению практических, лабораторных работ и организации
самостоятельной работы студентов по дисциплине

«Системы реального времени»

Для студентов направления подготовки **09.04.04 Программная инженерия (Методы и технологии индустриального проектирования программного обеспечения)**

Уровень основной образовательной программы: **Магистратура**
Очная форма обучения, план набора 2015 г.

Разработчик:
доцент каф. АОИ
_____ Ю.Б. Гриценко
«____» _____ 2017г.

Томск – 2017

Аннотация

Целью дисциплины «Системы реального времени» является обучение студентов систематизированному представлению о базовых принципах функционирования и методах разработки систем реального времени, навыкам разработки приложений в операционных системах реального времени

Задачи изучения дисциплины:

- Усвоение студентами теоретических знаний по вопросам изучения базовых принципов функционирования и методов разработки СРВ.

- Формирование устойчивых знаний и практических навыков организации вычислительных процессов в СРВ.

Дисциплина «Системы реального времени» (Б1.В.ДВ.2.2) относится к блоку 1 (вариативная часть) профессионального цикла обязательных дисциплин. Изучается в двух семестрах (1 и 2 семестр).

Процесс изучения дисциплины направлен на формирование следующих компетенций:

ОК-8 способностью к профессиональной эксплуатации современного оборудования и приборов.

ПК-5 владением существующими методами и алгоритмами решения задач цифровой обработки сигналов.

Содержание

СЕМЕСТР 1.

ПРАКТИЧЕСКАЯ РАБОТА № 1. «Управление задачами в ОС Windows»

1.1 Цель работы

Целью работы является изучение процесса управления заданиями в ОС Windows.

1.2 Информация об организации вычислительных задач

Современные операционные системы содержат встроенные средства, предоставляющие информацию о компонентах вычислительного процесса. Диспетчер задач (Task Manager) операционных систем Windows (например, Windows XP) позволяет получить обобщенную информацию об организации вычислительного процесса с детализацией до выполняющихся прикладных программ (приложений) и процессов. Однако диспетчер задач не позволяет отслеживать потоки [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. 1.1).

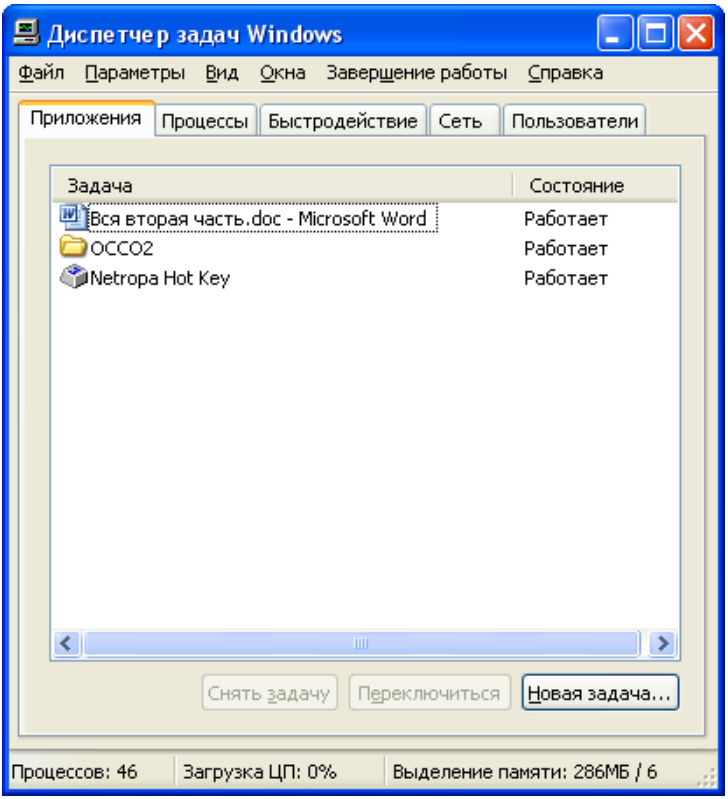

Рис. 1.1 – Окно диспетчера задач в ОС Windows XP

2. Для просмотра приложений перейти на вкладку «Приложения». Здесь можно завершить приложение (кнопка «Снять задачу»), переключиться на другое приложение (кнопка «Переключиться») и создать новую задачу (кнопка «Новая задача»). В последнем случае после нажатия кнопки «Новая задача» в появившемся окне (рис. 1.2) нужно ввести имя задачи.

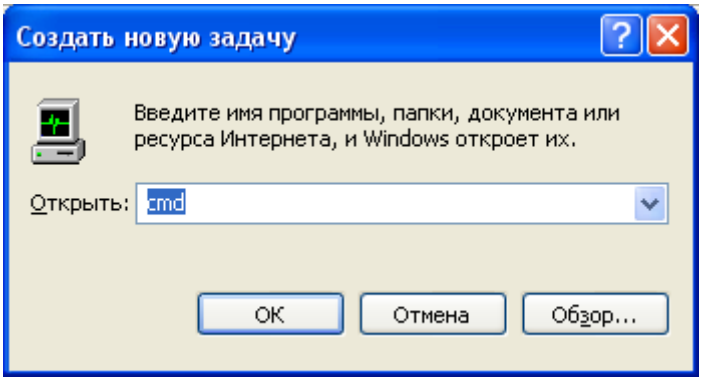

Рис. 1.2 – Окно создания новой задачи в ОС Windows XP

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.

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. 1.3).

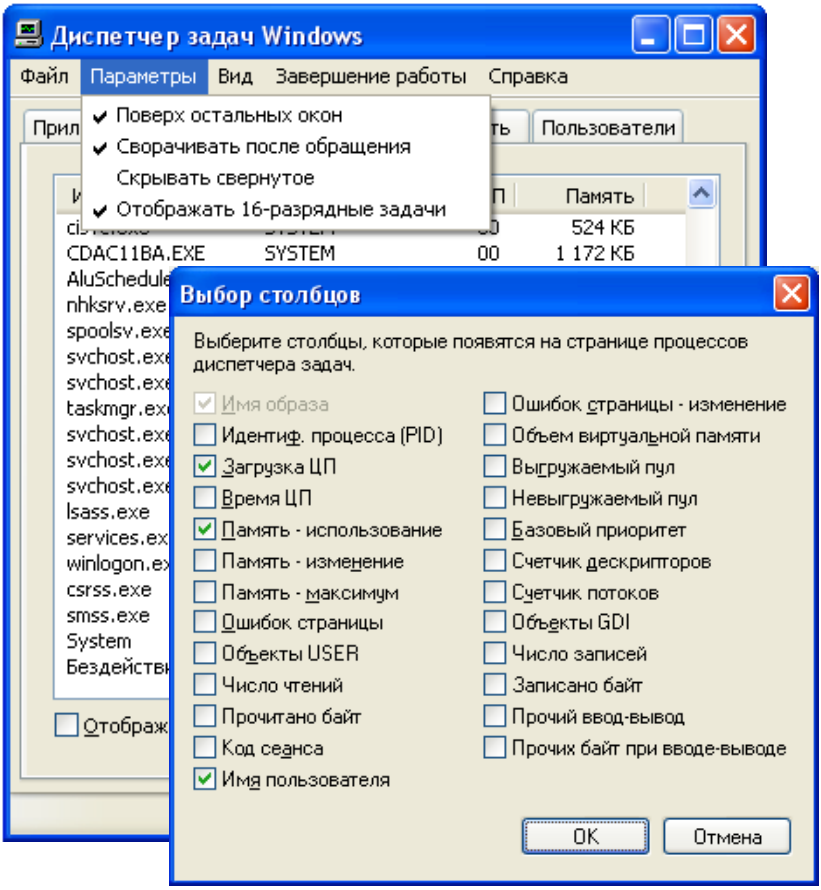

Рис. 1.3 – Окно диспетчера задач в ОС Windows XP на вкладке процессы с окном настройки отображения столбцов

В качестве примера можно рассмотреть процессы приложения MS Word. Для этого нужно выполнить следующие действия [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. 1.4). Верхние два окна показывают интегральную загрузку процессора и хронологию загрузки. Нижние два окна — те же показатели, но по использованию памяти.

2. Для просмотра использования процессора в режиме ядра (красный цвет) войти в меню «Вид» и щелкнуть на строке Вывод времени ядра.

В нижней части окна вкладки «Быстродействие» отображается информация о количестве процессов и потоков, участвующих в мультипрограммном вычислительном процессе, об общем количестве дескрипторов (описателей) объектов, созданных операционной системой, а также информация о доступной и выделенной памяти для реализации приложений. Кроме того, приводятся сведения о выделении памяти под ядро операционной системы с указанием выгружаемой и невыгружаемой памяти ядра и объеме системного кэша.

Также в диспетчере задач имеются вкладки для отображения состояния сети (вкладка «Сеть») и информации о вошедших в систему пользователях (вкладка «Пользователи»).

Ряд программ, как производителей операционных систем, так и сторонних производителей могут предоставить более детальную информацию о компонентах вычислительного процесса и механизмы управления им: Process Explorer, Process Viewer, Microsoft Spy++, CPU Stress, Scheduling Lab, Job Lab и др.

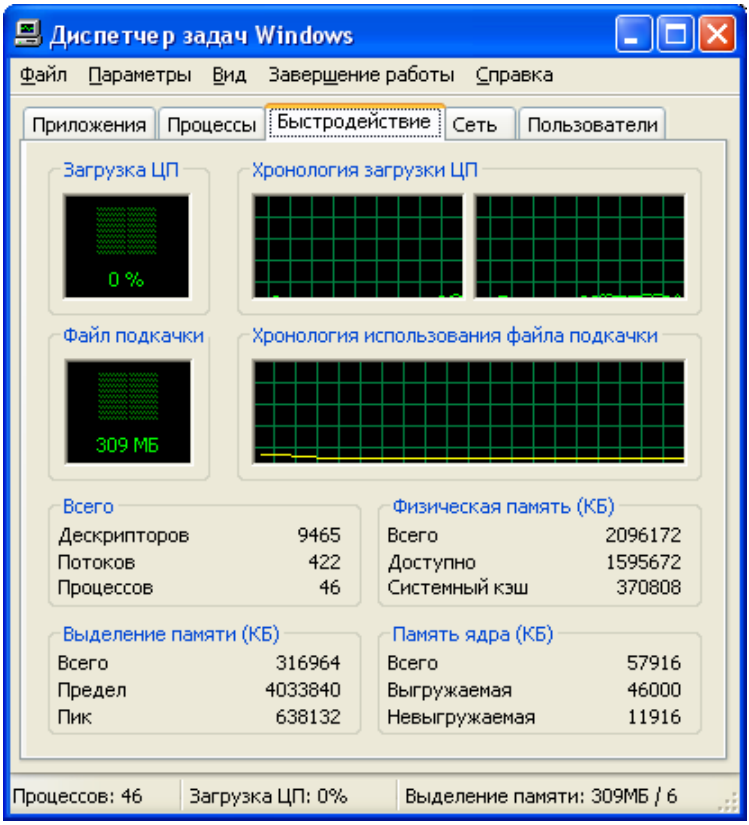

Рис. 1.4 – Окно диспетчера задач в ОС Windows XP на вкладке быстродействие

На рис. 1.5 показан окно с получением информации о потоках в программе Process Explorer. В данной программе можно получить исчерпывающую информацию о количестве и состоянии задач в операционной системе Windows.

Любой поток состоит из двух компонентов [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.

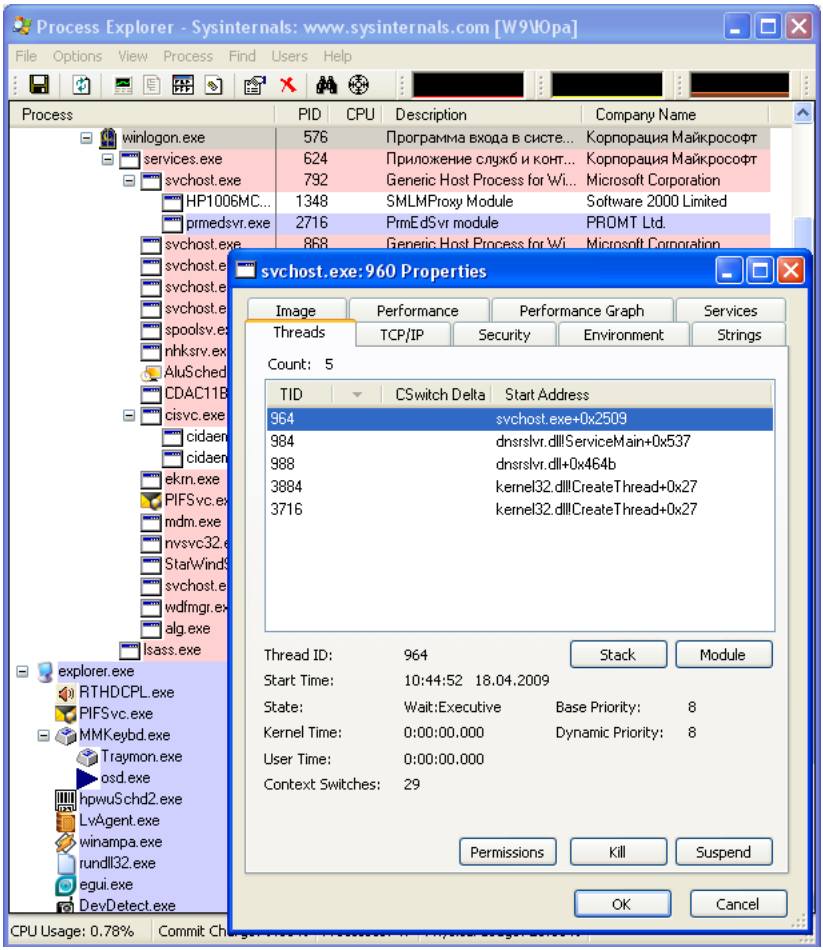

Рис. 1.5 – Окно с информацией о потоках в программе Process Explorer

Создав объект ядра «поток», система присваивает счетчику числа его пользователей начальное значение, равное двум. Затем система выделяет стеку потока память из адресного пространства процесса (по умолчанию резервирует 1 Мбайт адресного пространства процесса и передает ему всего две страницы памяти, далее память может добавляться). После этого система записывает в верхнюю часть стека два значения (стеки строятся от старших адресов памяти к младшим). Первое из них является значением параметра **pvParam**, который позволяет

передать функции потока какое-либо инициализирующее значение. Второе значение определяет адрес функции потока **pfnStartAddr**, с которой должен будет начать работу создаваемый поток.

У каждого потока собственный набор регистров процессора, называемый контекстом потока. Контекст отображает состояние регистров процессора на момент последнего исполнения потока и записывается в структуру **CONTEXT**, которая содержится в объекте ядра «ПОТОК».

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].

1.3 Исследование производительности

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. 1.6).

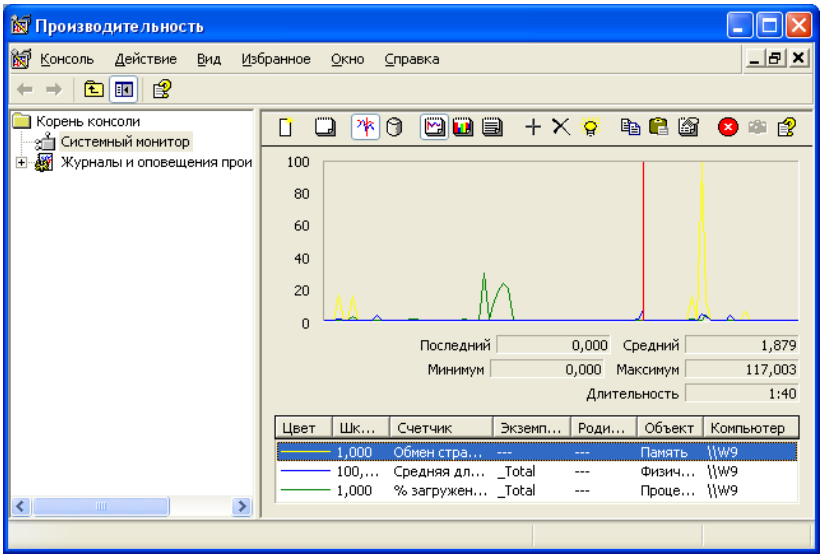

Рис. 1.6 – Окно Производительность в ОС Windows XP на вкладке быстродействие

Системный монитор позволяет анализировать вычислительный процесс, используя различные счетчики. Объектами исследования являются практически все компоненты компьютера: процессор, кэш, задание, процесс, поток, физический диск, файл подкачки, очереди сервера, протоколы и др.

Для просмотра и выбора объектов мониторинга и настройки счетчиков нужно выполнить следующие действия:

1. Открыть оснастку «Производительность». По панели результатов (правая панель) щелкнуть правой клавишей мыши и выбрать в контекстном меню строку «Добавить счетчики» или щелкнуть по кнопке «Добавить» (значок +) на панели инструментов.

2. В появившемся окне «Добавить счетчики» (рис. 1.7) выбрать объект мониторинга, например процессор, а затем выбрать нужные счетчики из списка «Выбрать счетчики из списка», например «% времени прерываний», нажимая кнопку Добавить, для потока можно определить:

– число контекстных переключений в сек.;
– состояние потока (для построения графа состояний и переходов);
– текущий приоритет (для анализа его изменения);
– базовый приоритет;

– % работы в привилегированном режиме и др.

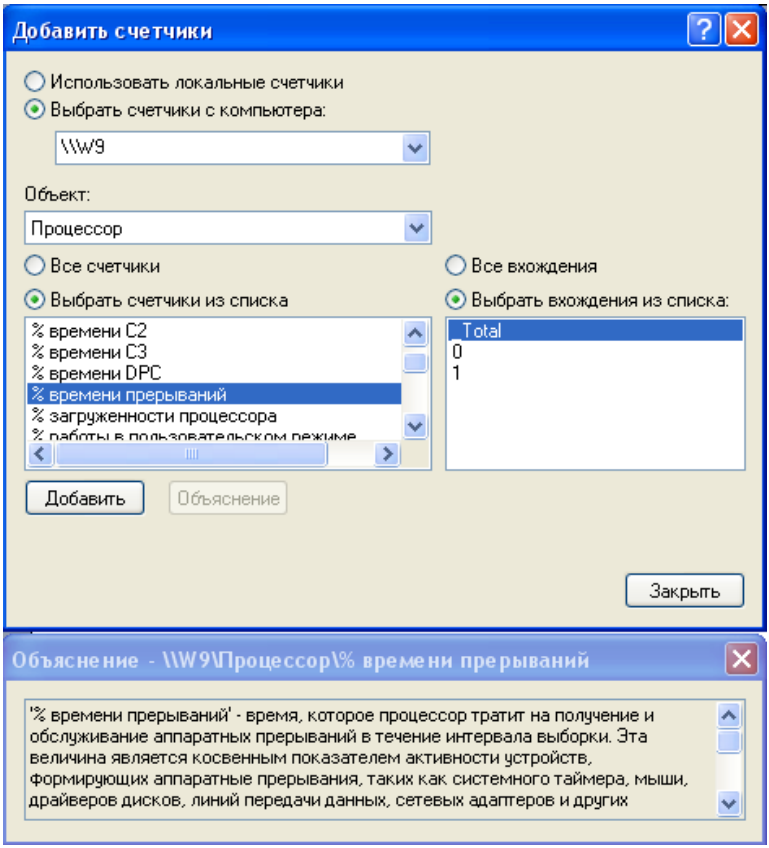

Рис. 1.7 – Окно Добавить счетчики в программе оценки производительности в ОС Windows XP

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.

1.4 Задание на выполнение

Часть 1.

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Исследовать мультипрограммный вычислительный процесс на примере выполнения самостоятельно разработанных трех задач (например, заданий по курсу программирования).

2. Для одной из задач определить PID, загрузку ЦП, время ЦП, базовый приоритет процесса, использование памяти. Изменить приоритет процесса и установить, влияет ли это на время выполнения приложения.

3. Монопольно выполнить каждую из трех задач, определить время их выполнения.

4. Запустить одновременно (друг за другом) три задачи, определить время выполнения пакета.

Ответьте на вопросы:

1. В каком случае суммарное время выполнения задач больше? При последовательном выполнении или одновременном выполнении?

2. Как изменилось время выполнения каждой отдельной задачи?

3. Как изменится время выполнения отдельной задачи при изменении ее приоритета?

4. Окажет ли влияние изменение приоритета одной задачи на время выполнения другой задачи? Объяснить результаты.

Часть 2.

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Запустить некоторое количество программ. Используя возможности оснастки Производительность, получить диаграммы, характеризующие использование процессора при его нагрузке различным количеством потоков, меняя их активность и уровни приоритета.

2. Исследовать свои задачи (например, задания по курсу программирования). Определить характеристики процессов: % загрузки процессора (в пользовательском и привилегированном режиме), % времени прерываний, количество прерываний, базовый приоритет, обращения к диску, время выполнения процесса.

3. Исследовать свои приложения с записью результатов в Журнал счетчиков, выбрав следующие счетчики: % загруженности, работы процессора в привилегированном и пользовательском режимах, % времени прерываний, % использования выделенной памяти, частота обращений к диску, скорость обмена с диском.

4. Выполнить следующие действия:

– Запустить журнал (частота съема данных 10 сек., файл типа CVS).
– Запустить исследуемую программу.

– Через 2 - 3 мин. остановить журнал.
– Просмотреть Результаты, открыв файл журнала в Excel. Объяснить полученные результаты.
– Исследовать программу еще раз, указав тип журнала —двоичный (чтобы потом можно было просмотреть диаграммы).

5. Создать журнал трассировки для исследования своего приложения. Создать Оповещения по выбранным счетчикам для своего приложения. Просмотреть журнал событий. Объяснить полученные результаты.

Ответьте на вопросы:

1. Что можно просматривать, используя счетчики в системном мониторе?
2. В каких видах можно просматривать информацию о производительности?

Часть 3. Дополнительное задание

1. Найти в Интернет бесплатную программу Process Explorer для Microsoft Windows.
2. Установить ее на компьютер.
3. Произвести исследование ее работы.

Ответьте на вопрос:

1. Сколько потоков содержит запущенная на вашем компьютере программа Microsoft Word?

Трудоемкость практической работы: 9 часов.

ПРАКТИЧЕСКАЯ РАБОТА № 2. «Исследование блоков управления памятью»

2.1 Цель работы

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

2.2 Организация хранения байтов в памяти

При просмотре памяти имейте в виду, что двухбайтовые слова хранятся в виде {младший байт}{старший байт} – т.е. порядке обратном естественному представлению 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:

```
78 56 34 12
 \/    \/
  |     |___ старшее слово с переставленными байтами
  |
  |_________ младшее слово с переставленными байтами
```

2.3 Информация о структурах памяти

Это список указателей, каждый из которых представляет собой двойное слово (4 байта). Старшее слово – это сегментный адрес, младшее – смещение в сегменте. Например, для указателя, у которого сегментный адрес=1234H, а смещение 5678H, абсолютный физический адрес ячейки памяти образуется, как сумма сегментного адреса * 16 + смещение (т.е. сегментный адрес сдвинут влево на 1 шестнадцатеричный разряд):

```
 1234 H       0110 H     0112 H
+ 5678H      + 0026H    + 0006H
--------     --------   ---------
=179B8H      =01126H    =01126H
```

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

2.4 Структура таблицы таблиц

Данная структура является НЕДОКУМЕНТИРОВАННОЙ и используется для изучения низкоуровневой информации о структурах памяти.

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$)
…	…	…

2.5 Структура блока управления памятью (MCB)

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

Смещение	Длина	Содержимое
+0	1	'M' (4dH) – за этим блоком есть еще блоки '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:

- блоки памяти всегда выровнены на границу параграфа («сегмент блока»);
- блоки M-типа: следующий блок находится по (сегмент блока + Размер):0000;
- блоки Z-типа: (сегмент блока + Размер):0000 = конец памяти (a000H=640K).

В любом MCB указан его владелец – сегментный адрес PSP (префикс программного сегмента) программы владельца данного блока памяти. А в PSP есть ссылка на окружение данной программы, в котором можно найти имя программы – 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!

2.6 Задание на выполнение

Часть 1.

1. Подготовиться к работе, используя материалы лекций, данное пособие, справочник TEACH-HELP.

2. Познакомиться с работой одной из программ, позволяющих просмотреть содержимое ОЗУ в виде шестнадцатеричного дампа – например, PEEK.COM (во время работы доступен HELP – F1, карта памяти – F8 и информация о блоке памяти – F6).

3. Найти в памяти таблицу таблиц (для получения ее адреса – запусти lol.com), познакомиться с ее содержимым и посмотреть указатель на 1 MCB (упр. блок памяти).

4. Проследить в памяти цепочку блоков, определяя их принадлежность и сравнивая с информацией из карты памяти (F8).

5. Написать отчет о найденной цепочке блоков памяти с их адресами и размерами.

Часть 2. Дополнительное задание

1. Найти в Интернет бесплатную программу Process Explorer для Microsoft Windows.

2. Установить ее на компьютер.

3. Произвести исследование ее работы.

Ответьте на вопрос:

1. Сколько потоков содержит запущенная на вашем компьютере программа Microsoft Word?

Трудоемкость практической работы: 9 часов.

ПРАКТИЧЕСКАЯ РАБОТА № 3. «Диагностика IP-протокола»

3.1 Цель работы

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

3.2 Просмотр свойств сетевого окружения

Получить информацию о свойствах сетевого окружения возможно с использованием следующих действий: Нажмите кнопку «Пуск» и в появившемся окне щелкните правой кнопкой мыши по пункту «Сетевое окружение». В появившемся контекстном меню выберите пункт «Свойства». Перед вами появится окно, показанное на рис. 3.1.

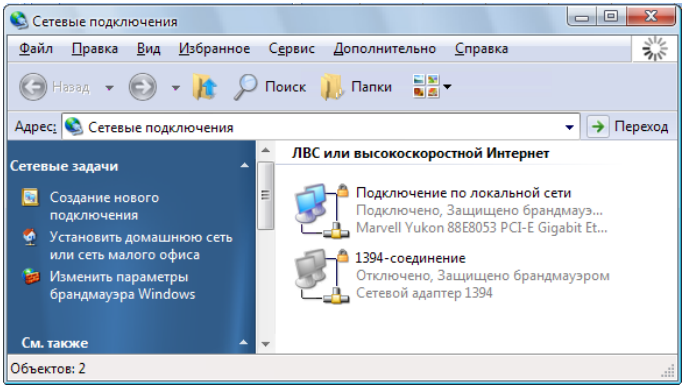

Рис. 3.1. Свойства сетевого окружения

Чтобы получить информацию о свойствах подключения по локальной сети, щелкните по надписи «Подключение по локальной сети» правой кнопкой мыши и также в появившемся меню выберите свойства. В появившемся окне (рис. 3.2) вы можете настраивать протоколы сетевых взаимодействий.

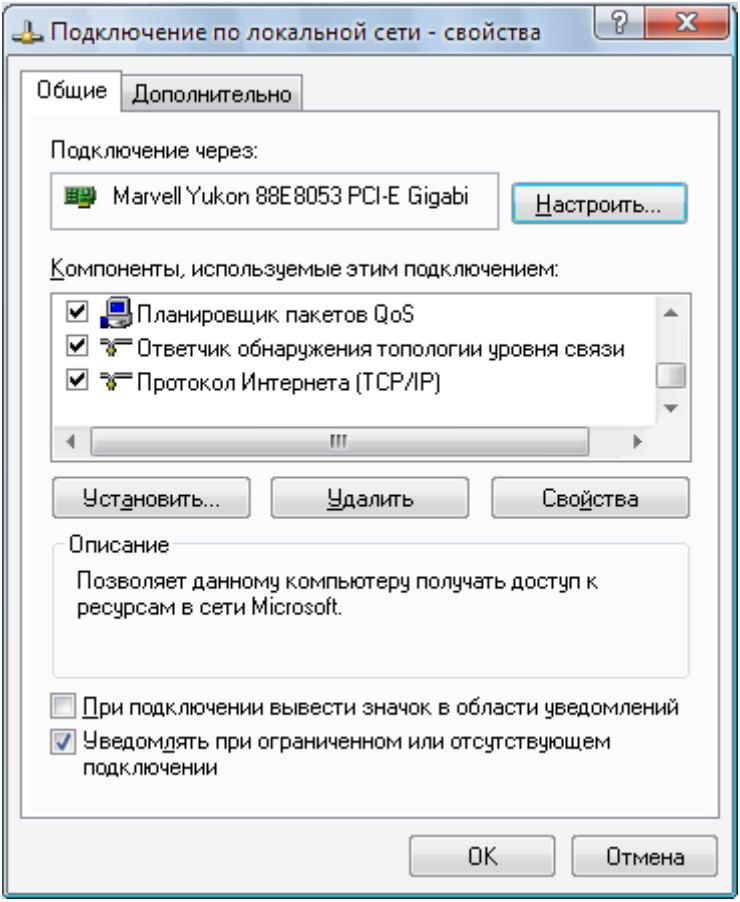

Рис. 3.2 – Свойства подключения по локальной сети

Важным элементом в свойствах подключения по локальной сети, является протокол Интернета TCP/IP [3]. Выбрав это компонент и нажав кнопу «Свойства» откроется окно (рис. 3.3) где можно устанавливать настройки сетевого подключения по протоколу TCP/IP.

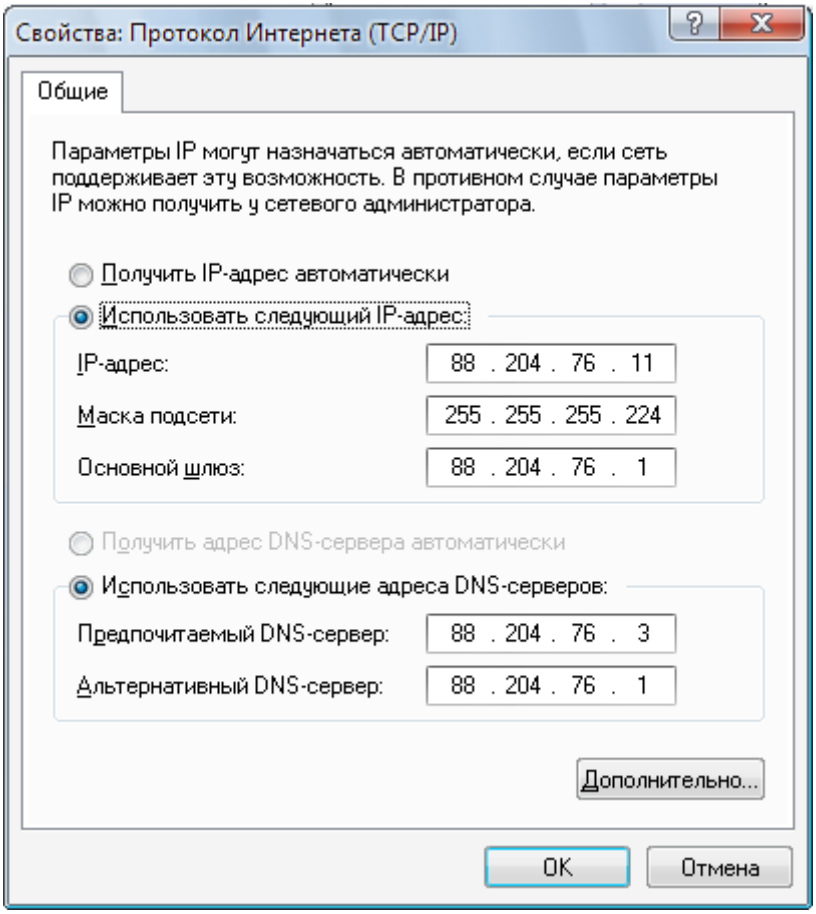

Рис. 3.3 Свойства протокола Интернета (TCP/IP)

3.3 Утилита диагностики сети

Существуют различные утилиты, позволяющие быстро протестировать IP-подключение. Однако большинство операций легко может быть выполнено с использованием команд самой операционной системы.

Пользователи Windows XP для диагностики сетевого подключения могут воспользоваться специальным мастером. Эта программа вызывается из меню задачи «Сведения о системе». Произведите следующие действия (Пуск > Все программы > Стандартные > Служебные > Сведения о системе > меню Сервис > Диагностика сети). На рисунке 3.4 показан процесс работы утилиты «Диагностики сети». На

рис. 3.5. результат работы утилиты по диагностики сетевого подключения.

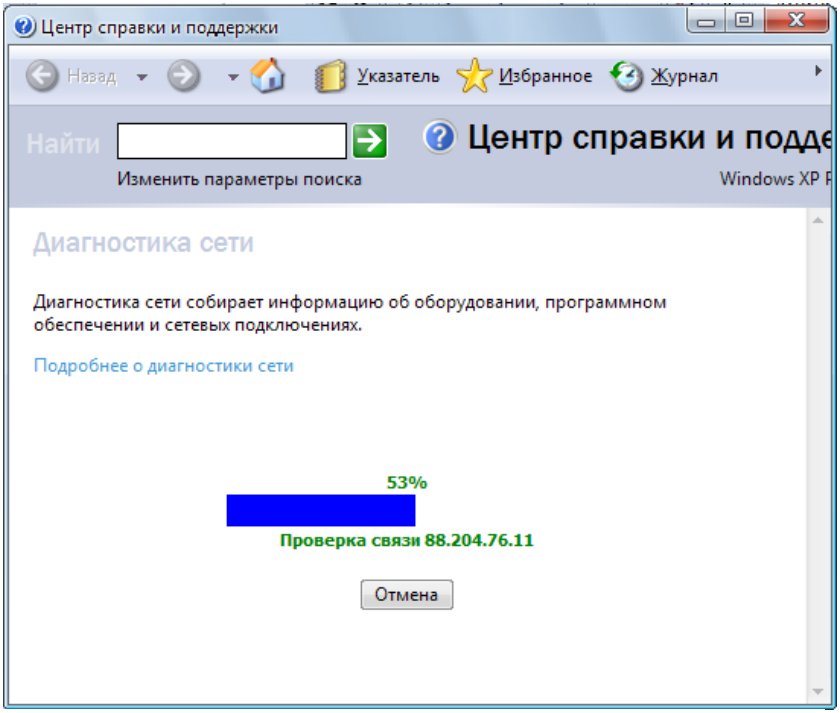

Рис. 3.4 Ход работы утилиты «Диагностика сети»

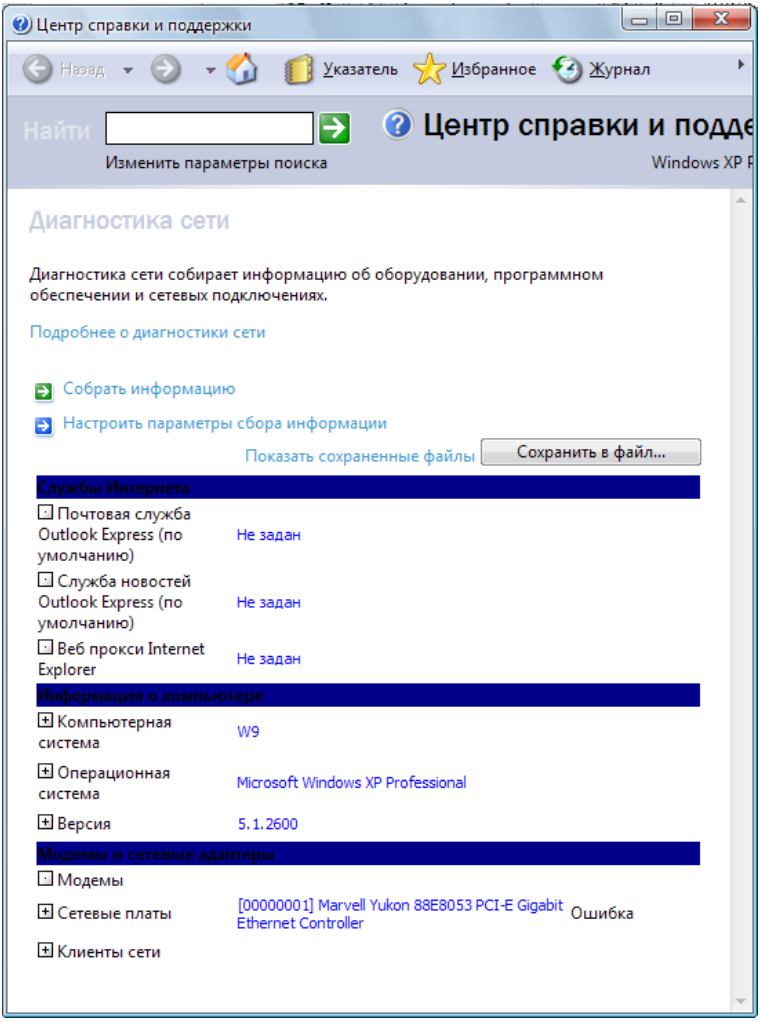

Рис. 3.5 Ход работы утилиты «Диагностика сети»

3.4 Утилита «Ipconfig»

Для отображения параметров IP-протокола в ОС на платформе Windows NT используются утилиты ipconfig. Эта утилита выводит на экран основные параметры настройки протокола TCP/IP: значения адреса, маски, шлюза [3].

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:

```
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).

3.5. Утилита «Ping»

Утилита «Ping» используется для проверки протокола TCP/IP и достижимости удаленного компьютера. Она выводит на экран время, за которое пакеты данных достигают заданного в ее параметрах компьютера.

1. Проверка правильности установки протокола TCP/IP. Откройте командную строку и выполните команду:

```
ping 127.0.0.1
```

Адрес 127.0.0.1 — это личный адрес любого компьютера. Таким образом, эта команда проверяет прохождение сигнала «на самого себя». Она может быть выполнена без наличия какого-либо сетевого подключения. Вы должны увидеть приблизительно следующие строки:

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

По умолчанию команда посылает пакет 32 байта. Размер пакета может быть увеличен до 65 Кбайт. Так можно обнаружить ошибки при пересылке пакетов больших размеров. За размером тестового пакета отображается время отклика удаленной системы (в нашем случае — меньше 1 миллисекунды). Потом показывается еще один параметр протокола — значение TTL. TTL — «время жизни» пакета. На практике это число маршрутизаторов, через которые может пройти пакет. Каждый маршрутизатор уменьшает значение TTL на единицу. При достижении нулевого значения пакет уничтожается. Такой механизм введен для исключения случаев 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.

3.6 Утилита «Tracert»

При работе в сети одни информационные серверы откликаются быстрее, другие медленнее, бывают случаи недостижимости желаемого хоста. Для выяснения причин подобных ситуаций можно использовать специальные утилиты.

Например, команда tracert, которая обычно используется для показа пути прохождения сигнала до желаемого хоста. Зачастую это позволяет выяснить причины плохой работоспособности канала. Точка, после которой время отклика резко увеличено, свидетельствует о наличии в этом месте узла, не справляющегося с нагрузкой.

В командной строке введите команду:

```
tracert 192.168.0.19
```

Вы должны увидеть примерно такой листинг:

```
Tracing route to 192.168.0.19
over a maximum of 30 hops:
  1    <1 ms    <1 ms    <1 ms  192.168.0.19

Trace complete.
```

3.7 Утилита «Route»

Команда Route позволяет просматривать маршруты прохождения сетевых пакетов при передаче информации.

Выведите на экран таблицу маршрутов TCP/IP, для этого в командной строке введите команду `route print`.

3.8 Утилита «Net view»

Выводит список доменов, компьютеров или общих ресурсов на данном компьютере. Вызванная без параметров, команда `net view` выводит список компьютеров в текущем домене.

1. `net view` и вы увидите список компьютеров своей рабочей группы.

2. `net view \192.165.0.12` для просмотра общих ресурсов расположенных на компьютере 192.165.0.12

3.9 Утилита «Net send»

Служит для отправки сообщений другому пользователю, компьютеру или псевдониму, доступному в сети.

1. Введите `net send 192.168.0.1 Привет. Проверка связи.`

Ваше сообщение получит пользователь 192.168.0.1

2. Введите `net send * Привет. Проверка связи.`

Ваше сообщение получат все пользователи рабочей группы.

3.10 Задание на выполнение

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

2. Осуществите диагностику сети.

3. Последовательно исследуйте все возможности сетевых утилит.

Ответьте на вопросы:

1. Какие сетевые протоколы установлены на вашем компьютере?

2. Чему равно «время жизни» пакета посылаемого с вашего компьютера?

3. Сколько компьютеров в вашей рабочей группе?

4. Чему равна длина маршрута пакета отправляемого вами на соседний компьютер?

Трудоемкость практической работы: 9 часов.

ПРАКТИЧЕСКАЯ РАБОТА № 4. «Управление устройствами ввода-вывода и файловыми системами в ОС Windows»

4.1 Цель работы

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

4.2 Диспетчер устройств и драйвера устройств

Задача системы ввода-вывода ОС Windows заключается в предоставлении основных средств (каркаса) для эффективного управления широким спектром устройств ввода-вывода. Основу этих средств образует набор независимых от устройств процедур для определенных аспектов ввода-вывода и набор загруженных драйверов для общения с устройствами. Формирует этот каркас ***менеджер ввода-вывода***, который предоставляет остальной операционной системе независимый от устройств ввод-вывод, вызывая для выполнения физического ввода-вывода соответствующий драйвер [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. 4.1) можно, раскрывая соответствующие узлы, видеть устройства, которые либо подключены и работают, либо отключены. Диспетчер устройств обычно используется для проверки состояния оборудования, подключения-отключения оборудования и обновления драйверов устройств, установленных на компьютере. Кроме того, возможности диагностики диспетчера устройств могут использоваться опытными пользователями, обладающими глубокими знаниями о компьютерном оборудовании, для разрешения конфликтов устройств и изменения параметров ресурсов, однако при этом следует соблюдать большую осторожность [2].

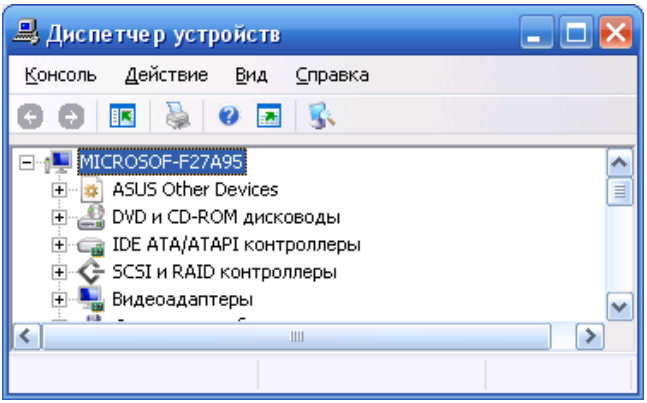

Рис. 4.1 – Окно программы «Диспетчер устройств»

При установке устройства ***Plug and Play*** Windows автоматически настраивает его, обеспечивая его правильную работу с другими установленными на компьютере устройствами. В ходе процесса настройки Windows назначает устанавливаемому устройству уникальный набор системных ресурсов. Эти ресурсы могут включать в себя один или несколько из следующих параметров:

- номера строк запросов на прерывание (IRQ);
- каналы прямого доступа к памяти (DMA);
- адреса портов ввода/вывода (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.

4.3 Диски и файловая система

Для получения доступа к просмотру состояния и управлению дисками нужно щелкнуть правой клавишей мыши по значку «Мой

компьютер», выбрать строку «Управление» и щелкнуть по ней. В открывшемся окне щелкнуть по строке «Управление дисками» (рис. 4.2). В правой части окна будут отображены все дисковые устройства компьютера и основные параметры их состояния.

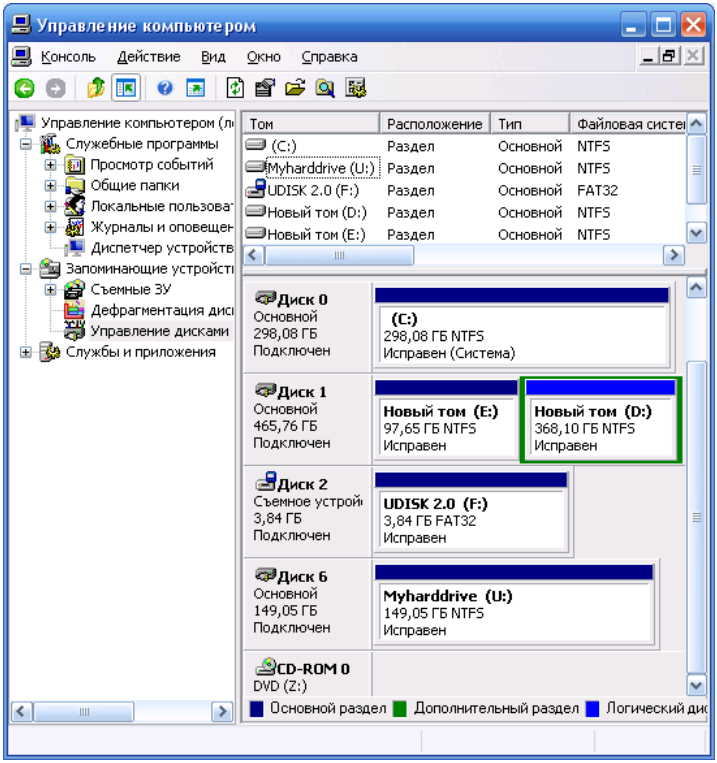

Рис. 4.2 – Вид окна «Управление компьютером» на вкладке «Управление дисками»

В окне можно управлять разделами дисковых устройств. Можно создать или удалить раздел или логический диск, можно сделать первичный раздел активным, чтобы при перезагрузке операционной системы обращение к загрузочной записи осуществлялось с указанного раздела. Активный раздел может быть только один. Здесь же можно отформатировать диск и изменить букву или путь диска. Все эти действия вызываются щелчком правой кнопки мыши по выбранному разделу в окне, представленном на рисунке 4.2.

При работе с жестким диском всегда имеет место фрагментация. С течением времени после установки программ диск заполняется, а после их удаления файлы фрагментируются и операционной системе приходится искать свободные фрагменты на диске для размещения файлов. Это может привести к заметному снижению быстродействия компьютера. Негативный эффект фрагментации устраняется с помощью встроенной в Windows программы дефрагментации, запустить которую можно, указав предварительно имя диска, в левой панели оснастки «Управление компьютером» (рис. 4.3) [2].

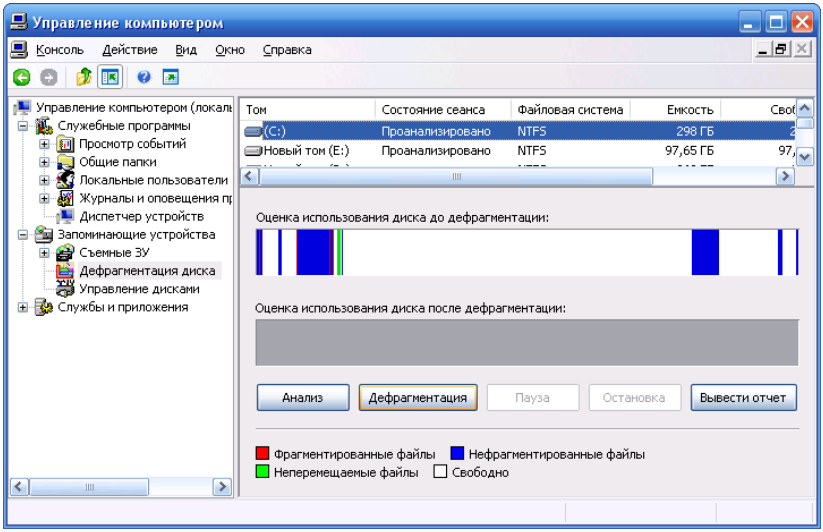

Рис. 4.3 – Вид окна «Управление компьютером» на вкладке «Дефрагментация диска»

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

4.4 Дисковые квоты

При совместном использовании дисковой памяти несколькими пользователями, работающими на одном компьютере, необходим контроль расходования дискового пространства. В Windows на платформе NT эта проблема решается квотированием дискового пространства по каждому тому (независимо от количества физических дисков) и для каждого пользователя.

После установки квот дискового пространства пользователь сможет хранить на томе ограниченный объем данных, в то время как на этом томе может оставаться свободное пространство. Если пользователь превышает выданную ему квоту, в журнал событий вносится соответствующая запись. Затем, в зависимости от конфигурации системы, пользователь либо сможет записать информацию на том (более мягкий режим), либо ему будет отказано в записи.

Устанавливать и просматривать квоты на диске можно только в разделе NTFS 5.0 и при наличии необходимых полномочий (задаваемых с помощью локальных или доменных групповых политик) у пользователя, устанавливающего квоты.

Чтобы установить квоты, нужно выполнить следующие действия:

1. Щелкнуть правой кнопкой мыши по конфигурируемому тому и выбрать в контекстном меню команду «Свойства». В появившемся окне перейти на вкладку Квота (рис. 4.4).

2. Установить флажок «Включить управление квотами». В этом случае будет установлен мягкий режим контроля используемого дискового пространства. Для задания жесткого режима контроля нужно установить флажок «Не выделять место на диске при превышении квоты». На этой же вкладке устанавливается размер выделяемой квоты и порог, превышение которого вызовет запись предупреждений в журнале событий.

Чтобы узнать, какие пользователи превысили выделенную им квоту (в мягком режиме), нужно нажать кнопку «Записи квот», где будет отражен список пользователей с параметрами квот и объемом используемого ими пространства диска.

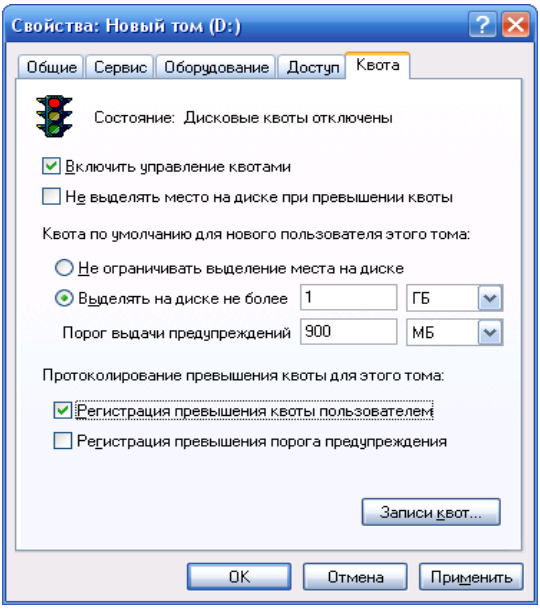

Рис. 4.4 – Вид окна по просмотру свойств диска на вкладке «Квота»

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

Устанавливая пользователям определенные **разрешения для файлов и каталогов (папок)**, администраторы системы могут защищать конфиденциальную информацию от несанкционированного доступа[1]. Каждый пользователь имеет определенный набор разрешений на доступ к конкретному объекту файловой системы (рис. 4.5). Администратор может назначить себя владельцем любого объекта файловой системы.

Действующие разрешения в отношении конкретного файла или каталога образуются из всех прямых и косвенных разрешений, назначенных пользователю для данного объекта с помощью логической функции ИЛИ.

Пользователь может назначить себя владельцем какого-либо объекта файловой системы, если у него есть необходимые права, а также передать права владельца другому пользователю.

[1] Для практического исследования данных возможностей необходимо использовать Windows Server на платформе 2000 или выше.

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[2] или выше. Точки соединений прозрачны для приложений, это означает, что приложение или пользователь, осуществляющий доступ к локальной папке NTFS, автоматически перенаправляется к другой папке.

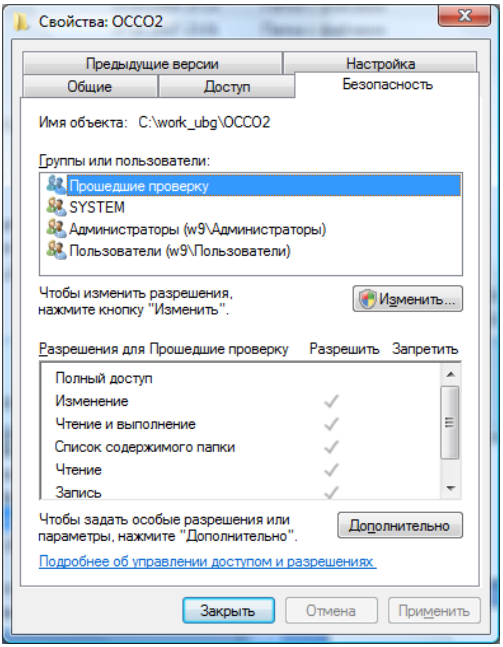

Рис. 4.5 – Вид окна установки разрешений на доступ к конкретному объекту файловой системы

Для работы с точками соединения на уровне томов можно использовать стандартные средства системы — ***утилиту Mountvol*** (рис. 4.6) и оснастку «Управление дисками». Для монтирования папок нужна утилита Linkd (из Windows 2000 Resource Kit).

[2] Поддерживаются только в NTFS 5.0

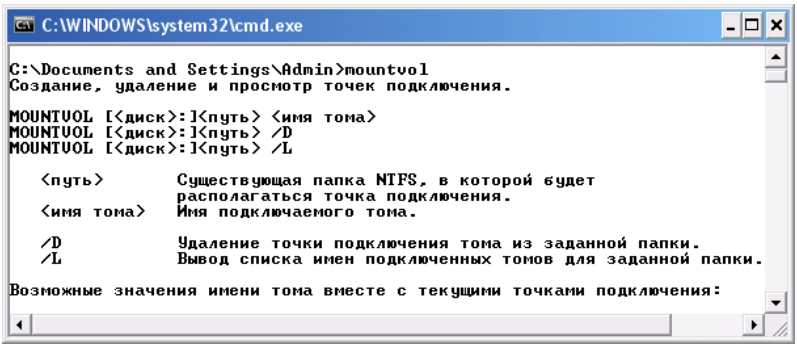

Рис. 4.6 – Вызов утилиты mountvol

С помощью утилиты Mountvol можно выполнить следующие действия:

– отобразить корневую папку локального тома в некоторую целевую папку NTFS, т.е. подключить или монтировать том;

– вывести на экран информацию о целевой папке точки соединения NTFS, использованной при подключении тома;

– просмотреть список доступных для использования томов файловой системы;

– уничтожить точки подключения томов.

Оснастка «Управление дисками» позволяет также создать соединения для дисков компьютера.

Шифрующая файловая система EFS (Encrypting File System). Поскольку шифрование и дешифрование выполняются автоматически, пользователь может работать с файлом так же, как и до установки его криптозащиты. Все остальные пользователи, которые попытаются получить доступ к зашифрованному файлу, 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 ОК. Далее следует нажать кнопку ОК 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 пап-

ке 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].

4.6 Задание на выполнение

1. Исследуйте работу диспетчера устройств.
2. Опишите структуру дисков и файловых систем на вашем компьютере.
3. Исследуйте механизм раздачи дисковых квот.
4. Исследуйте механизм надежности хранения информации.

Ответьте письменно на вопросы по самостоятельной подготовке:

1. Опишите структуру файловой системы NTFS.
2. Опишите преимущества и недостатки файловой системы NTFS.

Трудоемкость практической работы: 9 часов.

СЕМЕСТР 2.

1. ЛАБОРАТОРНАЯ РАБОТА №1 «Процессы в ОС QNX»

1.1. Цель работы

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

1.2. Создание процессов

На самом высоком уровне абстракции система состоит из множества процессов. Каждый процесс ответственен за обеспечение служебных функций определенного характера.

Разделение объектов на множество процессов дает ряд преимуществ:

1. возможность декомпозиции задачи и модульной организации решения;
2. удобство сопровождения;
3. надежность.

Любой поток может осуществить запуск процесса. Однако необходимо учитывать ограничения, вытекающие из основных принципов защиты.

Из курса «Операционные системы» вы уже должны быть знакомы с возможностями запуска процессов из командного интерпретатора *(shell).*

Например:

$ program1 — запуск приложения в режиме переднего плана;
$ program2 & — запуск приложения в режиме заднего плана.
$ nice program3 — запуск приложения с заниженным приоритетом.

Обычно разработчиков программного обеспечения не заботит тот факт, что командный интерпретатор создает процессы — это просто подразумевается. Однако в большой мультипроцессорной системе вы можете пожелать, чтобы одна главная программа выполняла запуск всех других процессов вашего приложения.

Рассмотрим некоторые функции, которые ОС QNX использует для запуска других процессов:

- *system();*

- *fork();*
- *vfork();*
- *exec();*
- *spawn().*

Какую из этих функций применять, зависит от двух требований: переносимости и функциональности.

system() — самая простая функция; она получает на вход одну командную строку, такую же, которую вы набрали бы в ответ на подсказку командного интерпретатора, и выполняет ее.

Фактически, для обработки команды функция system() запускает копию командного интерпретатора.

fork() — порождает процесс, являющийся его точной копией. Новый процесс выполняется в том же адресном пространстве и наследует все данные порождающего процесса.

Между тем, родительский и дочерний процесс имеют различные идентификаторы процессов, так как в системе не может быть двух процессов с одинаковыми идентификаторами. Есть и еще одно отличие, это значение, возвращаемое функцией fork(). В дочернем процессе функция возвращает ноль, а в родительском процессе идентификатор дочернего процесса.

Пример, использования функции fork():

```
printf("PID родителя равен %d\n", getpid());
if (child_pid = fork()) {
printf("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.

1.3. Задание на выполнение

1. Познакомиться с интерфейсом ОС QNX.

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

3. Написать программу, которая бы запускала в памяти еще один процесс и оставляла бы его работать в бесконечном цикле. При повторном запуске программа должна убирать запущенный ранее процесс из памяти (можно использовать kill).

4. Подготовиться к ответам на теоретическую часть лабораторной работы.

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

Трудоемкость практической работы: 4 часа.

2. ЛАБОРАТОРНАЯ РАБОТА №2 «Потоки в ОС QNX»

2.1. Цель работы

Познакомиться возможностями функций управления потоками в ОС QNX.

2.2. Создание потоков

Процесс может содержать один или несколько потоков. Число потоков варьируется. Один разработчик программного обеспечения, используя только единственный поток, может реализовать те же самые функциональные возможности, что и другой, используя пять потоков. Некоторые задачи сами по себе приводят к многопоточности и дают относительно простые решения, другие в силу своей природы, являются однопоточными, и свести их к 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 бу-

дут 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);

      }
}
```

2.3. Задание на выполнение

1. Выполнить задание согласно варианту.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Вариант 1. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки обменом «пузырьком».

Вариант 2. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки простых вставок.

Вариант 3. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки выбором.

Вариант 4. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом прямоугольников.

Вариант 5. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запус-

кает 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.

Трудоемкость практической работы: 6 часов.

3. ЛАБОРАТОРНАЯ РАБОТА №3 «Обмен сообщениями»

3.1. Цель работы

Познакомиться с механизмами обмена сообщениями в ОС QNX.

3.2. Связь между процессами

3.2.1. 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 A выдает запрос на передачу сообщения процессу B. 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 B); 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 A.

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 B может принять запрос Send(), выданный процессом A, с помощью функции Receive().

pid = Receive (0, msg, msg_len)

Функция Receive() имеет следующие аргументы:

- pid — идентификатор процесса, пославшего сообщение (т.е. процесса A);
- 0 — (ноль) указывает на то, что процесс B 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 A процесс B 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 A);
- 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.

3.2.2. Связь между процессами посредством proxy

Proxy представляет собой форму неблокирующей передачи сообщений, специально предназначенную для оповещения о событиях, при которых процесс-отправитель не нуждается во взаимодействии с процессом-получателем. Единственной функцией proxy является посылка фиксированного сообщения процессу, создавшему proxy. Так же, как и сообщения, proxy работают по всей сети.

Благодаря использованию proxy, процесс или обработчик прерываний может послать сообщение другому процессу, не блокируясь и

не ожидая ответа. Ниже приведены некоторые примеры использования proxy:

- процесс оповещает другой процесс о наступлении некоторого события, не желая при этом оставаться SEND-блокированным до тех пор, пока получатель не выдаст Receive() и Reply();
- процесс посылает данные другому процессу, но не требует ни ответа, ни другого подтверждения о том, что получатель принял сообщение;
- обработчик прерываний оповещает процесс о том, что некоторые данные доступны для обработки.

Proxy создаются с помощью функции qnx_proxy_attach(). Любой процесс или обработчик прерываний, которому известен идентификатор proxy, может воспользоваться функцией Trigger() для того, чтобы выдать заранее определенное сообщение. Запросами Trigger() управляет ядро.

Процесс proxy может быть запущен несколько раз: выдача сообщения происходит каждый раз при его запуске. Процесс proxy может накопить в очереди для выдачи до 65535 сообщений.

3.2.3. Связь между процессами посредством сигналов

Связь посредством сигналов представляет собой традиционную форму асинхронного взаимодействия, используемую в различных операционных системах.

В системе QNX поддерживается большой набор POSIX-совместимых сигналов, специальные QNX-сигналы, а так же исторически сложившиеся сигналы, используемые в некоторых версиях системы UNIX.

Сигнал выдается процессу при наступлении некоторого заранее определенного для данного сигнала события. Процесс может выдать сигнал самому себе.

Если вы хотите сгенерировать сигнал из интерпретатора Shell, используйте утилиты kill() или slay().

Если вы хотите сгенерировать сигнал из процесса, используйте утилиты kill() или raise().

В зависимости от того, каким образом был определен способ обработки сигнала, возможны три варианта его приема:

- Если процессу не предписано выполнять каких-либо специальных действий по обработке сигнала, то по умолчанию поступление сигнала прекращает выполнение процесса;

- Процесс может проигнорировать сигнал. В этом случае выдача сигнала не влияет на работу процесса (обратите внимание на то, что сигналы SIGCONT, SIGKILL и SIGSTOP не могут быть проигнорированы при обычных условиях);
- Процесс может иметь обработчик сигнала, которому передается управление при поступлении сигнала. В этом случае говорят, что процесс может «ловить» сигнал. *Фактически такой процесс выполняет обработку программного прерывания.* Данные с сигналом не передаются.

Интервал времени между генерацией и выдачей сигнала называется задержкой. В данный момент времени для одного процесса могут быть задержаны несколько разных сигналов. Сигналы выдаются процессу тогда, когда планировщик ядра переводит процесс в состояние готовности к выполнению. Порядок поступления задержанных сигналов не определен.

Для задания способа обработки сигнала следует воспользоваться функцией ANSI C signal() или функцией POSIX sigaction().

Функция sigaction() предоставляет больше возможностей по управлению средой обработки сигнала.

Если процессу не требуется возврата управления от обработчика сигналов в прерванную точку, то в этом случае в обработчике сигналов может быть использована функция siglongjmp() или longjmp(). Причем siglongjmp() предпочтительнее, т.к. в случае использования longjmp() сигнал остается блокированным.

Иногда может потребоваться временно задержать выдачу сигнала, не изменяя при этом способа его обработки. В системе QNX имеется набор функций, которые позволяют блокировать выдачу сигналов. После разблокировки сигнал выдается программе.

Во время работы обработчика сигналов QNX автоматически блокирует обрабатываемый сигнал. Это означает, что не требуется организовывать вложенные вызовы обработчика сигналов. Каждый вызов обработчика сигналов не прерывается остальными сигналами данного типа. При нормальном возврате управления от обработчика, сигнал автоматически разблокируется.

Существует важная взаимосвязь между сигналами и сообщениями. Если при генерации сигнала ваш процесс окажется SEND-блокированным или RECEIVE-блокированным (причем имеется обработчик сигналов), то будут выполняться следующие действия:

- процесс разблокировывается;
- выполняется обработка сигнала;

- функции Send() или Receive() возвращают управление с кодом ошибки.

Если процесс был SEND-блокированным, то проблемы не возникает, так как получатель не получит сообщение. Но если процесс был REPLY-блокированным, то неизвестно, было, обработано отправленное сообщение или нет, а, следовательно, неизвестно, нужно ли еще раз выдавать Send().

Процесс, выполняющий функции сервера (т.е. принимающий сообщения), может запрашивать уведомления о том, когда обслуживаемый процесс выдаст сигнал, находясь в REPLY-блокированном состоянии. В этом случае обслуживаемый процесс становится SIGNAL-блокированным с задержанным сигналом, и обслуживающий процесс принимает специальное сообщение, описывающее тип сигнала. Обслуживающий процесс может выбрать одно из следующих действий:

- нормально завершить первоначальный запрос: отправитель будет уведомлен о том, что сообщение было обработано надлежащим образом;
- освободить все закрепленные ресурсы и возвратить управление с кодом ошибки, указывающим на то, что процесс был разблокирован сигналом: отправитель получит чистый код ошибки.

Когда обслуживающий процесс сообщает другому процессу, что он SIGNAL-блокирован, сигнал выдается немедленно после возврата управления функцией Send().

3.3. Примеры обмена сообщениями при помощи таймера

3.3.1. Клиент

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

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

```
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).

3.3.2. 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).

3.3.3. Определение идентификаторов узла, процесса и канала (ND/PID/CHID) нужного сервера

Для соединения с сервером функции ConnectAttach() необходимо указать дескриптор узла (Node Descriptor — ND), идентификатор процесса (process ID — PID), а также идентификатор канала (Channel ID — CHID).

Если один процесс создает другой процесс, тогда это просто — вызов создания процесса возвращает идентификатор вновь созданного процесса. Создающий процесс может либо передать собственные PID и CHID вновь созданному процессу в командной строке, либо вновь созданный процесс может вызвать функцию ***getppid()*** для получения идентификатора родительского процесса, и использовать некоторый «известный» идентификатор канала.

```
#include <process.h>
pid_t getpid( void );
```

Вопрос: «Как сервер объявляет о своем местонахождении?»

Существует множество способов сделать это; мы рассмотрим только три из них, в порядке возрастания «элегантности»:

1. Открыть файла с известным именем и сохранить в нем ND/PID/CHID. Такой метод является традиционным для серверов UNIX, когда сервер открывает файл (например, /etc/httpd.pid), записывает туда свой идентификатор процесса в виде строки ASCII и предполагают, что клиенты откроют этот файл, прочитают из него идентификатор.

2. Использовать для объявления идентификаторов ND/PID/CHID глобальные переменные. Такой способ обычно применяется в

многопоточных серверах, которые могут посылать сообщение сами себе. Этот вариант по самой своей природе является очень редким.

3. Занять часть пространства имен путей и стать администратором ресурсов.

3.4. Задание на выполнение

1. Необходимо создать два приложения - клиент и сервер, которые бы обменивались между собой сообщениями, используя стандартные механизмы операционной системы QNX/Neutrino2.

При реализации клиент-серверных приложений требуется предусмотреть возможность работы с сервером нескольких клиентов.

В клиенте имеется возможность консольного ввода команд, которые должны обрабатываться сервером. Сервер должен обрабатывать не менее трех команд посылаемых клиентом:

- help — помощь по командам;
- list /dir — получение содержимого указанного каталога;
- get filename — передача клиенту указанного файла.

Также сервер должен отвечать клиенту, если тот послал ему не верную команду.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Трудоемкость практической работы: 6 часов.

4. ЛАБОРАТОРНАЯ РАБОТА №4 «Таймер и периодические уведомления»

4.1. Цель работы

Познакомиться с механизмом управления временем ОС QNX — системным таймером.

4.2. Управление таймером

В QNX управление временем основано на использовании системного таймера. Этот таймер содержит текущее координатное универсальное время (UTC) относительно 0 часов 0 минут 0 секунд 1 января 1970 г. Для установки местного времени функции управления временем используют переменную среды TZ.

Процесс может создать один или несколько таймеров. Таймеры могут быть любого поддерживаемого системой типа, а их количество ограничивается максимально допустимым количеством таймеров в системе.

Функция создания таймера позволяет задавать следующие типы механизма ответа на события:

- перейти в режим ожидания до завершения. Процесс будет находиться в режиме ожидания начиная с момента установки таймера до истечения заданного интервала времени;
- оповестить с помощью 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.

4.3. Задание на выполнение

1. Модифицировать программу созданную в третьей лабораторной работе, так чтобы в клиенте работали два таймера с разной периодичностью, например. 3 и 4,44 секунды. При срабатывании каждого таймера серверу посылался бы сигнал с номером сигнала таймера.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Трудоемкость практической работы: 6 часов.

5. ЛАБОРАТОРНАЯ РАБОТА №5 «Среда визуальной разработки программ PHOTON APPLICATION BUILDER – PHAB»

5.1. Цель работы

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

5.2. Основы работы с Phab

Для запуска среды Phab необходимо выбрать Launch=>Development=>Phab.

После запуска идем в пункты меню File=>New и в окне диалога выбираем тип проекта. Далее сохраняем проект под некоторым именем в своей домашней директории

Компиляция, компоновка и запуск. Осуществляются следующим образом:

1. Application=>Generate;
2. Выбрать платформу gcc;
3. Выбрать Make – создание исполняемого модуля;
4. Выбрать Run – запуск приложения.

Запустите приложение из терминального режима независимо от Phab.

Для того, что бы узнать какие объекты, функции и сообщения необходимы для построения приложения, используйте Launch=>HelpViewer. Ключевым словом может служить «widget».

Компоновка формы:

Чтобы создать рабочую форму с полями ввода и кнопкой, нужно сделать следующее:

1. Разместить элементы на форме.
2. Задать им уникальные имена.
3. Создать действие на нажатие кнопки:
    a. Дать имя компилируемому модулю, в котором будет находиться обработчик кнопки.
    b. Применить настройки.
    c. Создать новый модуль.
4. Написание обработчика кнопки 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.

5.3. Задание на выполнение

1. Построить пробные приложения в Phab (Photon Application Builder).

Создать клиентское приложение, использующее графический интерфейс: 2 поля для ввода текста и кнопка отправки сообщения. Это

приложение должно получать от пользователя текст через поле ввода и отправлять его по нажатию кнопки серверу в виде сообщения.

Сервер должен получать сообщение и отвечать ровно через 4,75 секунды.

Ответное сообщение сервера должно приходить во второе поле формы.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Трудоемкость практической работы: 6 часов.

6. ЛАБОРАТОРНАЯ РАБОТА №6 Улучшение навыков программирования

6.1. Цель работы

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

6.2. Задания 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 5x5. Одно поле противника, другое свое. В окне есть кнопки, нажатие на которые реализуют функции: подключиться к серверу, расставить корабли перед боем (по пять кораблей на поле), начать игру, сдаться. После начала игры пользователи поочередно 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 кли-

ента. 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.

Трудоемкость практической работы: 6 часов.

Методические указания к самостоятельной работе

Семестр 1.

1. Проработка лекционного материала (18 часов):
   1. Принципы построения вычислительных систем – 4 часа.
   2. Организация памяти – 4 часа.
   3. Управление устройствами ввода-вывода – 4 часа.
   4. Принципы построения вычислительных сетей и телекоммуникаций - 6 часов.

Форма контроля: Опрос на занятии.

2. Подготовка к практическим работам из расчета 1 час на 1 час практических работ (34 часа самостоятельной работы).

Для подготовки к практическим работам следует использовать данные методические указания, в качестве дополнительной литературы следует воспользоваться литературой [1-4], приведенной в разделе «Список литературы»

Форма контроля: Опрос на занятии.

Итого самостоятельной работы – 54 часов.

Подготовка к экзамену 36 часов.

Итого 90 часов.

Семестр 2.

1. Проработка лекционного материала (22 часа):
    1. Введение в системы реального времени – 2 часа
    2. Автоматизированные системы управления технологическими процессами – 4 часа.
    3. Организация операционных систем реального времени - 2 часа.
    4. Стандарты на ОСРВ – 2 часа.
    5. Обзор ОСРВ - 2 часа.
    6. Микроядро ОС QNX Neutrino – 4 часа.
    7. Администратор процессов и управление ресурсами в ОС QNX – 6 часов.

    Форма контроля: Опрос на занятии.

2. Подготовка к лабораторным работам из расчета 1 час на 1 час лабораторных работ (34 часа самостоятельной работы).

Для подготовки к лабораторным работам следует использовать данные методические указания, в качестве дополнительной литературы следует воспользоваться литературой [5-7], приведенной в разделе «Список литературы»

Форма контроля: Опрос на занятии.

Итого самостоятельной работы – 56 часов.

4. Подготовка к экзамену 36 часов.

Итого 92 часа.

СПИСОК ЛИТЕРАТУРЫ

1. Гриценко Ю.Б. Вычислительные системы, сети и телекоммуникации: учеб. пособие / Ю.Б. Гриценко. — Томск: Томск. гос. ун-т систем управления и радиоэлектроники, 2015. — 134 с. (электронный ресурс: https://edu.tusur.ru/training/publications/5053).
2. Гриценко Ю.Б. Операционные системы : учебное пособие: в 2 ч. / Ю. Б. Гриценко. — Томск: ТМЦДО, 2009. – Ч. 2. – 230 с.
3. Назаров С.В. Операционные системы. Практикум. / С.В. Назаров, Л.П. Гудыно, А.А. Кириченко. Под ред. С.В. Назарова — М.: КУДИЦ-ПРЕСС, 2008. — 464 с.
4. Олифер В.Г. Компьютерные сети: Принципы, технологии, протоколы : учебное пособие для вузов / В. Г. Олифер, Н. А. Олифер. - 3-е изд. - СПб. : Питер, 2006. - 960 с.
5. Гриценко Ю.Б. Системы реального времени: учебное пособие. — Томск: ТМЦДО, 2005. — 117с.: ил.
6. Зыль С.Н. Операционная система реального времени QNX: от теории к практике. — СПб.: БХВ-Петербург, 2004. — 192с.: ил.
7. Роб Кёртен. Введение в QNX/Neutrino 2: Руководство по программированию приложений реального времени в QNX Realtime Platform — СПб.:ООО «Издательство “Петрополис”», 2001. — 480с.: ил.