Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра сетей и устройств телекоммуникаций

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ В СЕТЯХ ТЕЛЕКОММУНИКАЦИЙ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторно-практическим занятиям
для студентов специальности 45 01 03 «Сети телекоммуникаций»
дневной, вечерней и заочной форм обучения

В 3-х частях

Часть 1

Операционная система DRS-Unix. Основные особенности работы, работа с файлами, регистрация в системе

Минск 2003

УДК 621.391.2 (075.8)
ББК 32.811 я 73
К 82

С о с т а в и т е л и :
В.А. Мельников, Я.В. Рощупкин

Компьютерные технологии в сетях телекоммуникаций: Метод. К 82 указания к лабораторно-практическим занятиям для студ. спец. 45 01 03 «Сети телекоммуникаций» дневной, вечерней и заочной форм обучения: В 3 ч. Ч. 1. Операционная система DRS-Unix. Основные особенности работы, работа с файлами, регистрация в системе / Сост. В.А. Мельников, Я.В. Рощупкин. – Мн.: БГУИР, 2003. – 31 с.

Методические указания включают основные теоретические положения, указания к выполнению лабораторной работы и содержание отчета по ней.

УДК 621.391.2 (075.8)
ББК 32.811 я 73

ВВЕДЕНИЕ

Система UNIX – одна из наиболее известных операционных систем (ОС), изначально ориентированная на работу в сети. Наличие важных достоинств в виде расширенных возможностей резервирования и администрирования позволило этой операционной системе интенсивно развиваться, и в настоящее время она является основной в глобальных сетях.

Вначале ОС UNIX была одной из многих других, однако со временем большой интерес к ОС UNIX стали проявлять коммерческие компании – производители компьютеров и программного обеспечения. Это объясняется тем, что с развитием технологии электронных схем резко упала стоимость производства новых однокристальных процессоров. Поэтому наличие по-настоящему мобильной операционной системы, перенос которой на новую аппаратную платформу не занимал слишком много времени и средств, позволяло экономно оснастить новые компьютеры качественным базовым программным обеспечением. Появились компании, специализирующиеся на переносе UNIX на новые платформы.

Специализированных вариантов ОС UNIX множество, например, XENIX UNIX Version 7, V/32 и др. Одним из них является ОС DRS-UNIX.

1. ОБЩАЯ ОРГАНИЗАЦИЯ ТРАДИЦИОННОГО ЯДРА ОС UNIX

Одним из основных достоинств ОС UNIX является то, что система обладает свойством высокой мобильности. Смысл этого качества состоит в том, что вся операционная система, включая ее ядро, сравнительно просто переносится на различные аппаратные платформы. Все части системы, не считая ядра, являются полностью машинно-независимыми. Эти компоненты написаны на языке Си настолько удачно, что для их переноса на новую платформу (по крайней мере в классе 32-разрядных компьютеров) требуется только перекомпиляция исходных текстов в коды целевого компьютера.

Наибольшие проблемы связаны с ядром системы, которое полностью скрывает специфику используемого компьютера, но само зависит от этой специфики. В результате продуманного разделения машинно-зависимых и машинно-независимых компонентов ядра (видимо, с точки зрения разработчиков операционных систем, в этом состоит наивысшее достоинство традиционного ядра ОС UNIX) удалось добиться того, что основная часть ядра не зависит от архитектурных особенностей целевой платформы, написана полностью на языке Си и для переноса на новую платформу нуждается только в перекомпиляции.

Однако сравнительно небольшая часть ядра является машинно-зависимой и написана на смеси языка Си и языка ассемблера целевого процессора. При переносе системы на новую платформу требуется переписывание этой части ядра

с использованием языка ассемблера и учетом специфических черт целевой аппаратуры. Машинно-зависимые части ядра хорошо изолированы от основной машинно-независимой части, и при хорошем понимании назначения каждого машинно-зависимого компонента переписывание машинно-зависимой части является в основном технической задачей (хотя и требует высокой квалификации программиста).

Машинно-зависимая часть традиционного ядра ОС UNIX включает следующие компоненты:

– раскрутка и инициализация системы на низком уровне (пока это зависит от особенностей аппаратуры);

– первичная обработка внутренних и внешних прерываний;

– управление памятью (в той части, которая относится к особенностям аппаратной поддержки виртуальной памяти);

– переключение контекста процессов между режимами пользователя и ядра;

– связанные с особенностями целевой платформы части драйверов устройств.

1.1. Основные функции

К основным функциям ядра ОС UNIX принято относить следующие:

– инициализация системы – функция запуска и раскрутки. Ядро системы обеспечивает средство раскрутки (bootstrap), которое отвечает за загрузку полного ядра в память компьютера и затем запускает ядро;

– управление процессами и нитями – функция создания, завершения и отслеживания существующих процессов и «нитей» (процессов, выполняемых на общей виртуальной памяти). Поскольку ОС UNIX является мультипроцессной операционной системой, ядро обеспечивает разделение между запущенными процессами времени процессора (или процессоров в мультипроцессорных системах) и других ресурсов компьютера для создания внешнего ощущения того, что процессы реально выполняются одновременно;

– управление памятью – функция отображения практически неограниченной виртуальной памяти процессов в физическую оперативную память компьютера, которая имеет ограниченные размеры. Соответствующий компонент ядра обеспечивает разделяемое использование одних и тех же областей оперативной памяти несколькими процессами с использованием внешней памяти;

– управление файлами – функция, реализующая абстракцию файловой системы – иерархии каталогов и файлов. Файловые системы ОС UNIX поддерживают несколько типов файлов. Некоторые файлы могут содержать данные в формате ASCII, другие будут соответствовать внешним устройствам. В файловой системе хранятся объектные файлы, выполняемые файлы и т.д. Файлы обычно хранятся на устройствах внешней памяти, доступ к ним обеспечивается средствами ядра. В мире UNIX существует несколько типов организации файловых систем. Современные варианты ОС UNIX одновременно поддерживают большинство типов файловых систем;

– коммуникационные средства – функция, обеспечивающая возможности обмена данными между процессами, выполняющимися внутри одного компьютера (IPC – Inter-Process Communications), между процессами, выполняющимися в разных узлах локальной или глобальной сети передачи данных, а также между процессами и драйверами внешних устройств;

– программный интерфейс – функция, обеспечивающая доступ к возможностям ядра со стороны пользовательских процессов на основе механизма системных вызовов, оформленных в виде библиотеки функций.

1.2. Принципы взаимодействия с ядром

В любой операционной системе поддерживается некоторый механизм, который позволяет пользовательским программам обращаться за услугами ядра ОС. В операционных системах наиболее известной советской вычислительной машины БЭСМ-6 соответствующие средства общения с ядром назывались экстракодами, в операционных системах IBM они назывались системными макрокомандами и т.д. В ОС UNIX такие средства называются системными вызовами. Название не изменяет смысла, состоящего в том, что для обращения к функциям ядра ОС используются «специальные команды» процессора, при выполнении которых возникает особого рода внутреннее прерывание процессора, переводящее его в режим ядра (в большинстве современных ОС этот вид прерываний называется trap – ловушка). При обработке таких прерываний (дешифрации) ядро ОС распознает, что на самом деле прерывание является запросом к ядру со стороны пользовательской программы на выполнение определенных действий, выбирает параметры обращения и обрабатывает его, после чего осуществляет «возврат из прерывания», возобновляя нормальное выполнение пользовательской программы.

Понятно, что конкретные механизмы возбуждения внутренних прерываний по инициативе пользовательской программы различаются в разных аппаратных архитектурах. Поскольку ОС UNIX стремится обеспечить среду, в которой пользовательские программы могли бы быть полностью мобильны, потребовался дополнительный уровень, скрывающий особенности конкретного механизма возбуждения внутренних прерываний. Этот механизм обеспечивается так называемой библиотекой системных вызовов.

Для пользователя библиотека системных вызовов представляет собой обычную библиотеку заранее реализованных функций системы программирования языка Си. При программировании на языке Си использование любой функции из библиотеки системных вызовов ничем не отличается от использования любой собственной или библиотечной Си-функции. Однако внутри любой функции конкретной библиотеки системных вызовов содержится код, являющийся, вообще говоря, специфичным для данной аппаратной платформы.

Наиболее важные системные вызовы ОС UNIX рассматриваются в оставшихся разделах этой части курса и в следующей части.

1.3. Принципы обработки прерываний

Применяемый в операционных системах механизм обработки внутренних и внешних прерываний в основном зависит от того, какая аппаратная поддержка обработки прерываний обеспечивается конкретной аппаратной платформой. К счастью, к настоящему моменту (и уже довольно давно) основные производители компьютеров де-факто пришли к соглашению о базовых механизмах прерываний.

Если говорить не очень точно и конкретно, суть принятого на сегодня механизма состоит в том, что каждому возможному прерыванию процессора (будь то внутреннее или внешнее прерывание) соответствует некоторый фиксированный адрес физической оперативной памяти. В тот момент, когда процессору разрешается прерваться по причине наличия внутренней или внешней заявки на прерывание, происходит аппаратная передача управления на ячейку физической оперативной памяти с соответствующим адресом – обычно адрес этой ячейки называется «вектором прерывания» (как правило, заявки на внутреннее прерывание, т.е. заявки, поступающие непосредственно от процессора, удовлетворяются немедленно).

Функция операционной системы – разместить в соответствующих ячейках оперативной памяти программный код, обеспечивающий начальную обработку прерывания и инициирующий полную обработку. В основном ОС UNIX придерживается общего подхода. В векторе прерывания, соответствующем внешнему прерыванию, т.е. прерыванию от некоторого внешнего устройства, содержатся команды, устанавливающие уровень выполнения процессора (уровень выполнения определяет, на какие внешние прерывания процессор должен реагировать незамедлительно) и осуществляющие переход на программу полной обработки прерывания в соответствующем драйвере устройства. Для внутреннего прерывания (например, прерывания по инициативе программы пользователя при отсутствии в основной памяти нужной страницы виртуальной памяти, при возникновении исключительной ситуации в программе пользователя и т.д.) или прерывания от таймера в векторе прерывания содержится переход на соответствующую программу ядра ОС UNIX.

2. ЛАБОРАТОРНАЯ РАБОТА № 1
«Основные представления о работе в системе UNIX. Работа с файлами. Регистрация в системе»

2.1. Основные функции и компоненты системы UNIX

Операционная система UNIX – это набор программ, который управляет компьютером, осуществляет связь между пользователем и компьютером и обеспечивает инструментальными средствами для выполнения работы. Разработанная, чтобы обеспечить легкость, эффективность и гибкость программного обеспечения, система UNIX имеет несколько полезных функций:

– выполнение широкого спектра заданий и программ;

– интерактивное окружение, которое позволяет связываться напрямую с компьютером и получать немедленно ответы на запросы и сообщения;

– многопользовательское окружение, которое позволяет разделять ресурсы компьютера с другими пользователями без уменьшения производительности. Этот метод называется разделением времени. Система UNIX взаимодействует с пользователями поочередно, но так быстро, что кажется, что взаимодействует со всеми пользователями одновременно;

– многозадачное окружение, которое позволяет выполнять более одного задания в одно и то же время.

Система UNIX имеет 4 основных компонента:

– **ядро** – это программа, которая образует ядро операционной системы; она координирует внутренние функции компьютера, например, такую как размещение системных ресурсов. Ядро работает невидимо для пользователя;

– **shell** – это программа, которая осуществляет связь между пользователем и ядром, интерпретируя и выполняя его команды. Так как она читает вводную информацию пользователя и посылает ему сообщения, то определяется как интерактивная;

– **commands** – это имена программ, которые компьютер должен выполнить. Пакеты программ называются инструментальными средствами. Система UNIX обеспечивает инструментальными средствами для таких задач, как создание и изменение текста, написание программ, развитие инструментария программного обеспечения, обмен информацией с другими посредством компьютера;

– **file system** – файловая система – это набор всех файлов, возможных для конкретного компьютера. Она помогает легко сохранять и отыскивать информацию.

Функции основного пакета программ

Внешний круг системы UNIX образуют программы и инструментальные средства системы, разделенные на категории функционально. Эти функции включают:

– **программное окружение** – несколько программ системы UNIX, устанавливающих дружественное программное окружение, обеспечивающее интерфейсы между системой и языками программирования и использование обслуживающих программ;

– **обработку текстов** – система обеспечивает программы, такие как строковый и экранный редакторы, для создания и изменения текстов, орфографическую программу проверки для обнаружения ошибок орфографии, и необязательный форматер текста для создания высококачественных копий, которые подходят для публикаций;

– **организацию информации** – система предоставляет много программ, которые позволяют создавать, организовывать и удалять файлы и каталоги;

– **обслуживающие программы** – инструментальные средства, создающие графику и выполняющие вычисления;

– **электронную связь** – несколько программ (например **mail**) предоставляют возможность передавать информацию другим пользователям и в другие системы UNIX.

Принципы ввода команд

Чтобы запрос был понятен системе UNIX, необходимо ввести каждую команду в корректном формате или синтаксисе командной строки. Необходимо расположить все составные части командной строки в требуемом синтаксисом порядке, иначе **shell** не сможет интерпретировать запрос.

Пример синтаксиса командной строки:

```
$ command option(s) argument(s) <CR>
```

Для каждой команды системы UNIX требуется ввести как минимум два компонента: имя команды и клавишу <ENTER>. (Далее по тексту информация, вводимая пользователем, будет обозначаться жирным начертанием, а выводимая системой – обычным. Обозначение <CR> в дальнейшем будет использоваться как инструкция для нажатия клавиши ENTER). Командная строка может также содержать ключи и аргументы. В указанном примере синтаксиса командной строки:

– command – это имя программы, которую необходимо выполнить;

– option – ключи, которые указывают как запустить команду;

– argument – указывает на данные, которые эта команда обрабатывает, обычно это имя каталога или файла.

В командной строке, которая включает ключи и/или аргументы, каждый компонент отделяется друг от друга по крайней мере одним пробелом. Если аргумент содержит пробел, его надо заключить в двойные кавычки. Например, если аргумент sample 1, то вы должны указать его в командной строке как «sample 1». Если пользователь позабудет поставить двойные кавычки, то **shell** будет интерпретировать sample и 1 как два отдельных аргумента.

Некоторые команды позволяют указывать несколько аргументов в одной командной строке, например:

```
$ ls    -l  -l     file1  file2  file3
```

В этом примере команда ls использует два ключа -l и -i и три аргумента file1, file2 и file3. Ключ -l обеспечивает информацию в длинном формате, включая режим, владельца и размер, а ключ -i печатает номер inode. Система UNIX обычно позволяет группировать ключи, например -li, и выводить их в любом порядке. Этого нельзя делать с аргументами.

Чтобы выполнить команду, введите командную строку, когда на экране появится подсказка (например, символ # для администратора и символ $ для простого пользователя). **Shell** рассмотрит команду как ввод, просмотрит один или

более каталогов, чтобы вызвать программу, и перенаправит запрос вместе с затребованными программами ядру. Ядро последует инструкциям в программе и выполнит команду.

После завершения выполнения программы, **shell** сигнализирует, что готов выполнить следующую команду, напечатав подсказку.

2.2. Файловая система

Файловая система является краеугольным камнем операционной системы UNIX. Она обеспечивает логический метод организации, восстановления и управления информацией. Файловая система имеет иерархическую структуру. Файл, который является основной единицей системы UNIX, может быть обыкновенным файлом, каталогом (папкой, директорией), специальным файлом или символическим каналом связи.

Обыкновенные файлы являются набором символов и используются для хранения любой информации. Они могут содержать тексты для писем или отчетов, коды программ, написанных пользователем, либо команды для запуска программ. Однажды создав обыкновенный файл, пользователь может добавить нужный материал в него, удалить материал из него либо удалить файл целиком.

Каталоги (папки, директории) являются суперфайлами, которые могут содержать файлы или другие директории. Обычно файлы, содержащиеся в них, устанавливают отношения каким-либо способом. Например, каталог, названный sales, может хранить файлы, содержащие цифры ежемесячных продаж, названные jan, feb, mar, и т.д. Вы можете создать каталоги, добавить или удалить файлы из них или удалить директорию.

Все каталоги, которые вы создаете, будут размещены в вашем собственном, «домашнем» каталоге. Он назначается вам системой во время входа в систему. Никто, кроме привилегированных пользователей, не может читать или записывать файлы в этот каталог без вашего на то разрешения, и только вы определяете структуру этого каталога.

Система UNIX также содержит несколько директорий для собственного использования. Структура этих каталогов аналогична во всех системах UNIX. Каталог root (обозначенный /) является исходным в файловой структуре UNIX. Все директории и файлы иерархически располагаются ниже.

Специальные файлы соответствуют физическим устройствам, таким как терминал, дисковое устройство, магнитная лента или канал связи. Система читает, записывает или считывает специальные файлы так же, как и в обыкновенные файлы. Однако запросы системы на чтение и запись не приводят в действие нормальный механизм доступа к файлу. Вместо этого они активизируют драйвер устройства, связанный с файлом, приводя, например, в действие головки диска или магнитной ленты.

Символические каналы связи (линки) – это файлы, которые указывают на другие файлы.

Структура системы

Некоторые операционные системы требуют от пользователя определить тип файла и использовать его определенным способом. От этого будет зависеть то, как будут файлы сохраняться, так как файлы могут быть последовательными, двоичными или произвольной выборки.

Для системы UNIX все файлы одинаковы, что делает файловую структуру UNIX легкой в использовании. Если в процессе работы программы необходим доступ к определенному устройству (например принтеру), то указывается устройство, так же как и любой из файлов. В системе UNIX существует только один интерфейс для всего ввода и вывода, что упрощает взаимодействие пользователя с системой.

Каталог «/» root имеет несколько важных системных директорий, которые содержат:

/stand – ядро системы и файлы данных, используемые в процессе загрузки. Иногда файл ядра находится прямо в корневом каталоге /;

/sbin – основные исполняемые программы, используемые в процессе загрузки и при восстановлении системы;

/dev – специальные файлы, которые представляют периферийные устройства (например, консоль, построчно-печатное устройство, терминалы пользователя и диски);

/etc – файлы конфигурации и базы данных организации системы;

/home – корневой каталог для каталогов пользователей;

/tmp – временные файлы;

/var – корневой каталог для часто изменяющихся файлов (например, файлы log);

/usr – другие каталоги, включая lib и bin.

2.3. Основные сведения для пользователей UNIX

Чтобы установить контакт с системой UNIX, вам необходимо иметь:

– терминал;

– регистрационное имя, которое идентифицирует вас как полномочного пользователя;

– пароль, который проверяет вас на идентичность;

– инструкции для диалога и доступа к системе UNIX, если ваш терминал напрямую не связан с компьютером.

Терминал. *Терминал является устройством ввода/вывода: пользователь использует его для ввода запросов системе UNIX, а система – для выдачи ответов.* Существует два основных вида терминалов: видеотерминал и печатающий терминал.

Видеотерминал отображает ввод и вывод на экране дисплея; печатающий терминал постоянно подает сигнал на бумагу. Во многих случаях эти различия никак не влияют на действия пользователя или на ответы системы.

Получение регистрационного имени. *Регистрационное имя – это имя, с помощью которого система UNIX проверяет полномочия пользователя системы во время запроса доступа к ней.* Регистрационное имя необходимо вводить каждый раз при входе в систему.

Чтобы получить регистрационное имя, необходимо обратиться к администратору системы UNIX. Существует несколько правил выбора регистрационного имени. Обычно длина имени составляет от 3 до 8 символов. Оно может состоять из больших или маленьких букв, цифр, символа подчеркивания, но не может начинаться с цифры.

Часто регистрационное имя определяется конкретным применением. Примеры допустимых имен: startship, mary2, jmrs.

Процедура регистрации. *Появление подсказки* login*: после включения это говорит о необходимости ввода регистрационного имени.*

Необходимо помнить, что система различает регистр букв, и это необходимо учесть при вводе данных, полученных у администратора. Регистрационные имена «startship», «Startship», «StartShip» на самом деле являются совершенно различными именами. Тот же принцип относится и к вводу пароля.

Пароль. *В процессе регистрации система выдает подсказку для ввода пароля. Необходимо ввести пароль и нажать клавишу <ENTER>*. Система UNIX не отображает пароль на экране в целях безопасности.

Если регистрационное имя и пароль допустимы в системе UNIX, то система напечатает текущую информацию и затем подсказку команды.

Ниже приведен пример регистрации в системе пользователя startship:

```
login:  startship
password:
UNIX  system  room506
$
```

Если пользователь при входе в систему сделает ошибку, то UNIX напечатает сообщение:

```
login  incorrect
```

Затем она предоставит второй шанс войти в систему, выдав подсказку login: Экран будет выглядеть следующим образом:

```
login:  startship
password:
login  incorrect
login:
```

Процедура регистрации в системе нового пользователя может отличаться от описанной выше. Это может произойти в том случае, если администратор системы предусмотрел процедуру назначения временных паролей новым пользователям. Если пользователь имеет временный пароль, то система заставит выбрать новый пароль прежде, чем предоставит доступ к системе.

Процедура входа в систему будет примерно выглядеть следующим образом:

1. Пользователь включает терминал, и система UNIX отображает подсказку login:. Необходимо ввести регистрационное имя и нажать клавишу <Enter>.

2. UNIX напечатает подсказку password:. Необходимо ввести временный пароль и нажать клавишу <Enter>.

3. Система сообщит, что временный пароль больше не действителен и необходимо выбрать новый пароль.

4. UNIX предложит ввести старый пароль.

5. Система предложит ввести новый пароль.

Пароль должен соответствовать следующим требованиям:

– должен иметь по крайней мере 6 символов. Только первые 8 символов являются значимыми;

– должен содержать по крайней мере 2 буквенных символа и одну цифру или специальный символ. Буквенный символ может быть набран либо на регистре больших символов, либо малых;

– должен отличаться от вашего регистрационного имени. Большие буквы и соответствующие им маленькие буквы эквивалентны;

– новый пароль должен отличаться от старого по крайней мере на три символа.

Примеры допустимых паролей:

mar84ch
Jonath0n
BRAV3S

6. Для предотвращения ошибки ввода система просит заново ввести пароль.

7. Если пользователь введет новый пароль второй раз не так, как в первый раз, то система сообщит, что пароль не совпадает, и предложит повторить процедуру регистрации снова. Когда пароли совпадут, система отобразит подсказку.

Следующий пример отображает вышеописанную процедуру:

```
login: startship
password:
Your password has expired
Choose a new one
Old password:
New password:
```

```
Re-enter new password:
$
```

Появление на экране подсказки означает, что система UNIX распознала пользователя как полномочного и ждет ввода команды. Например, для запуска команды **date** (команда **date** печатает дату и время) необходимо ввести команду и нажать клавишу <ENTER>. Система UNIX обратится к программе, называемой **date**, выполнит ее и напечатает результат на экране:

```
$ date
Tues Jul 18 14:49:07 1991
$
```

В следующем примере продемонстрировано использование команды who:

```
$ who
startship  term/00  Jul18  8:53
mlf        term/02  Jul18  8:53
jro        term/05  Jul18  8:53
ral        term/06  Jul18  8:53
$
```

Команда who печатает список регистрационных имен пользователей, работающих в настоящее время в системе. Обозначение tty (вторая колонка) делает ссылку на специальные файлы, которые соответствуют каждому пользовательскому терминалу. В третьей и четвертой колонках указываются дата и время регистрации каждого пользователя.

Выход из системы. Чтобы завершить работу с системой UNIX, необходимо ввести команду exit или нажать <^d>. Через несколько секунд система UNIX должна отобразить подсказку login: вновь:

```
$ exit
login:
```

Это означает, что завершение сеанса работы с системой было успешным и система готова зарегистрировать нового пользователя. Прежде чем вы отключите терминал, обязательно завершите работу с системой.

3. ОБЗОР ОСНОВНЫХ КОМАНД ОС

3.1. Shell – интерпретатор командного языка

В этом разделе описаны команды и символы, имеющие специальное значение, которые позволяют:

– находить с помощью шаблона и манипулировать группами файлов;
– запускать команду в фоновом режиме или в определенное время;
– выполнять последовательно группу команд;
– перенаправлять стандартные ввод и вывод;
– завершать работающие программы.

3.1.1. Метасимволы

Ниже приведены метасимволы, используемые **shell**:

*** ? [ ]** – позволяют указывать сокращенные имена файлов при поиске по шаблону;
& – означает, что команда будет выполняться в фоновом режиме;
; – разделяет команды в командной строке;
**** – отменяет специальное значение символов, таких как *, ?, [,], &, ;, <, >, | ;
'...' – отменяют значение пробела как разделителя и специальное значение всех символов;
"..." – отменяют значение пробела как разделителя и специальное значение всех символов, за исключением **$** и \ ;
> – перенаправляет вывод команды в файл;
< – перенаправляет ввод команды из файла;
>> – перенаправляет вывод команды, который должен быть добавлен в конец существующего файла;
| – создает канал, направляющий вывод одной команды во ввод другой команды;
`...` – используется в паре; позволяет использовать вывод команды как аргументы в командной строке;
$ – используется с позиционными параметрами и определенными пользователем переменными, также используется по умолчанию в качестве подсказки **shell**.

Метасимволы используются для поиска имен файлов, посредством их упрощается задача указания файлов или групп файлов как аргументов команды.

Поиск всех символов (метасимвол *). Осуществляет поиск любой строки символов, включая нулевую (пустую) строку. Символ «*» можно использовать для обозначения полного или частичного имени файла. Просто символ «*» ищет все имена файлов и каталогов в текущем каталоге, за исключением тех, которые начинаются с точки. Чтобы посмотреть метасимвол «*» в действии, введите его

как аргумент в команде echo:

```
$ echo *
```

В ответ система распечатает перечень всех имен файлов в текущем каталоге.

Символ «*» может представлять символы в любой части имени файла. Например, если известно, что несколько файлов имеют одинаковые первые и последние буквы, то можно дать запрос, основываясь на этом факте. Если в каталоге находятся файлы FATE, FE, FADED_LINE, FIG3.4E, FINE_LINE, FAST_LINE, то для их отображения на экране введите команду:

```
$ ls F*E
```

Поиск по одному символу (метасимвол ?). Осуществляет поиск любого одного символа в имени файла, за исключением лидирующей точки. Предположим, имеются файлы, в которых содержится 12 глав книги, и необходимо получить список глав до 9-й главы. Если текущий каталог содержит следующие файлы:

```
Chapter1
Chapter2
Chapter5
Chapter9
Chapter11
```

то необходимо ввести:

```
$ ls Chapter?
Chapter1
Chapter2
Chapter5
Chapter9
$
```

Хотя метасимвол осуществляет поиск одного символа, можно использовать его для поиска более одного символа в имени файла. Например, если ввести следующую команду, то система отобразит перечень всех остальных глав в текущем каталоге:

```
$ ls Chapter??
```

И, конечно, для получения списка всех глав в текущем каталоге, необходимо использовать метасимвол *:

```
$ ls Chapter*
```

Поиск одного набора (метасимвол []). Если необходимо, чтобы **shell** нашел любой символ из перечисленных символов, то необходимо заключить эти символы в квадратные скобки. Предположим, текущий каталог содержит следующие файлы: cat, fat, mat, rat. Если воспользоваться в качестве части имени файла шаблоном [crf], то **shell** будет искать имена файлов, в которые входят либо буква «c», либо буква «r», либо буква «f» в указанной позиции.

Пример:

```
$ ls [crf]at
cat
fat
rat
$
```

Символы, которые могут быть сгруппированы в скобки, называются классом символов. Скобки могут также использоваться для обозначения диапазона символов, цифр или букв. Предположим, в каталоге содержатся следующие файлы: chapter1, chapter2, chapter3, chapter4, chapter5, chapter6. Если указать:

```
$ ls chapter[1-5]
```

то **shell** найдет все файлы от chapter1 до chapter5.

Класс символов можно также указать с помощью диапазона букв. Если указать [A-Z], то **shell** будет искать только большие буквы, если [a-z], то малые буквы.

3.1.2. Специальные символы

Shell имеет также и другие символы, которые выполняют различные полезные функции.

Запуск команды в фоновом режиме (символ &)

Некоторые команды **shell** занимают много времени при выполнении. Эти команды можно запустить в фоновом режиме с использованием &, освобождая тем самым терминал для других задач. Общий формат для запуска команд в фоновом режиме следующий:

```
command &
```

Есть одно исключение – интерактивные команды **shell** (например **read**) нельзя запускать в фоновом режиме.

Когда пользователь запускает команду в фоновом режиме, то система UNIX выводит номер процесса. Этот номер можно использовать для завершения выполняющейся в фоновом режиме команды. Появившаяся подсказка означает,

что терминал свободен и ожидает команду. Запустить команду в фоновом режиме можно только в том случае, если терминал предоставляет такую возможность.

Последовательное выполнение команд (символ ;)

В одной командной строке можно указать несколько команд. Эти команды должны быть разделены символом ; (точка с запятой) или символом & (амперсанд):

```
command1; command2; command3
```

Система UNIX выполняет команды в том порядке, в котором они стоят в командной строке, и печатает вывод этих команд в том же порядке. Этот процесс называется последовательным выполнением.

Например, введите:

```
$ cd ~/; pwd; ls
```

shell выполнит эти команды последовательно:

1) cd изменит местоположение пользователя, переместив в домашний каталог;

2) pwd распечатает полное имя пути текущего каталога;

3) ls перечислит файлы в текущем каталоге.

Отмена специального значения (метасимвол \)

Символ \ позволяет отменить специальное значение следующего за ним символа. Например, есть файл trail, который содержит следующий текст:

```
The all * game
was held in Summit.
```

Чтобы найти символ звездочка (*) в файле, необходимо воспользоваться командой grep:

```
$ grep \* trail
The all * game
$
```

Команда grep найдет символ * в тексте и отобразит строку, в которой она появилась. Без символа \, символ звездочка будет интерпретироваться **shell** как метасимвол.

Отмена специального значения (метасимвол «кавычки»)

Отменить специальное значение символа также можно с помощью метасимвола «кавычки». Одиночные кавычки ('...') отменяют специальное значение всех символов, за исключением самих одиночных кавычек. Двойные

кавычки ("...") отменяют специальное значение всех символов, за исключением символов двойные кавычки, $ и ` (слабое ударение). Использование кавычек удобно для цифровых специальных символов.

Например, файл trail содержит строку:

```
He really wondered why? Why???
```

Чтобы найти строку, содержащую три вопросительных знака, воспользуйтесь командой grep:

```
$ grep '???' trail
He really wondered why? Why???
$
```

Использование кавычек для отмены значения символа «пробел»

Кавычки аналогично обратной косой черте часто используются для отмены специального значения пробела. **Shell** интерпретирует пробел в командной строке как разделитель между аргументами команды. Одиночные и двойные кавычки и обратная косая черта позволяют отменить это значение.

Например, чтобы в тексте разместить два или более слова, необходимо сделать эти два слова одним аргументом, заключив их в кавычки. Чтобы найти два слова «The all» в файле trail, необходимо ввести следующую команду:

```
$ grep 'The all' trail
The all * game
$
```

Особенно полезно применение отмены специального значения пробела для функции banner, которая использует пробел как разделитель аргументов и печатает аргументы на отдельных строках.

Чтобы напечатать более одного аргумента на одной строке, следует заключить слова в двойные кавычки. Например, если ввести

```
$ banner happy birthday to you
```

то команда banner напечатает сообщение на 4 строках. Если ввести

```
$ banner happy birthday "to you"
```

то команда banner напечатает сообщение на 3 строках, причем слова «to» и «you» напечатает на одной строке.

Команда banner печатает сообщения на экране терминала большими буквами плакатного размера.

3.1.3. Перенаправление ввода и вывода

В системе UNIX некоторые команды принимают ввод только с клавиатуры

(стандартный ввод), а большинство команд отображают свой вывод на экране терминала (стандартный вывод). Однако система UNIX позволяет перенаправлять ввод и вывод в файлы и программы, т.е. можно сказать **shell**:

– взять ввод из файла, а не с клавиатуры;

– послать вывод в файл, а не на терминал;

– использовать программу как исходные данные для другой программы.

Перенаправить ввод: знак <

Чтобы перенаправить ввод, следует указать в командной строке после знака «меньше чем» (<) имя файла:

```
$ command < имя_файла
```

Перенаправить вывод: знак >

Чтобы перенаправить вывод, необходимо указать в командной строке после знака «больше чем» (>) имя файла:

```
$ command > имя_файла
```

Внимание! Если пользователь перенаправляет вывод в уже существующий файл, то вывод команды заменит содержимое существующего файла.

Перед тем, как перенаправить вывод команды в конкретный файл, необходимо убедиться, что этот файл не существует. **Shell** не предупреждает, что выполняет перезапись существующего файла.

Чтобы убедиться, что файл с запланированным именем не существует, воспользуйтесь командой ls с аргументом «имя_файла». Если этот файл не существует, то ls выдаст сообщение, что файл не найден в текущем каталоге. Например, проверка существования файлов temp и junk даст следующий результат:

```
$ ls temp
temp
$ ls junk
junk: no such file or directiry
$
```

Это означает, что можно назвать свой файл junk, но нельзя использовать в качестве имени temp, если нельзя потерять содержимое существующего файла.

Добавить вывод в существующий файл

Чтобы добавить вывод в существующий файл и не разрушить его, можно воспользоваться символом >>:

```
$ command >> имя_файла
```

В результате выполнения команды вывод будет добавлен в конец существующего файла. Если файл не существует, то он будет создан. Например, рассмотрим, как добавить вывод с помощью команды cat. Команда cat печатает содержимое файлов, имена которых являются ее аргументами, в стандартный вывод. Если нет аргументов, то она печатает стандартный ввод в стандартный вывод. Сначала выполните команду cat без перенаправления вывода. Затем содержимое файла trial2 добавляем после последней строки в файл trial1 при выполнении команды cat над файлом trial2, перенаправив вывод в файл trial1:

```
$ cat trial1
This is the first line of trial1.
Hello.
This is the last line of trial1.
$
$ cat trial2
This is the beginning of trial2.
Hello.
This is the end of trial2.
$ cat trial2 >> trial1
$ cat trial1
This is the first line of trial1.
Hello.
This is the last line of trial1.
This is the beginning of trial2.
Hello.
This is the end of trial2.
$
```

Применение перенаправления вывода

Перенаправление вывода очень удобно в том случае, если необходимо, чтобы вывод появлялся на экране не немедленно, или необходимо сохранить его.

Рассмотрим применение команд: spell и sort.

Команда spell

Команда spell сравнивает каждое слово в файле со своим словарем и печатает список всех потенциальных орфографических ошибок на экране. Если в словаре spell нет какого-либо слова (например персональное имя), то она также выдает его как орфографическую ошибку. Если подать на ввод spell большой файл, то его обработка займет много времени и список ошибок может быть очень большим. Команда spell распечатывает весь список ошибок сразу, поэтому лучше всего перенаправить вывод spell в файл. Например, spell осуществляет поиск файла memo и помещает список орфографических ошибок в файл misspell:

```
$ spell memo > misspell
```

***Команда* sort**

Команда sort размещает строки указанного файла в алфавитном или цифровом порядке. Прежде чем перенаправить вывод команды в файл, следует убедиться, что имя этого файла не существует. Команда sort сначала очищает файл, который будет содержать вывод, затем выполняет сортировку и помещает вывод в пустой файл.

Комбинирование фонового режима и перенаправления вывода

Когда команда запущена в фоновом режиме, то вывод ее печатается на экране терминала. И если терминал в то же время используется для выполнения других задач, то вывод фоновой задачи будет прерывать работу. Для нормальной работы следует перенаправить вывод в файл.

Предположим, что необходимо найти все появления слова «test» в файле schedule. Следует запустить команду grep в фоновом режиме и перенаправить вывод в файл testfile:

```
$ grep test schedule > testfile
```

Теперь можно использовать терминал для других работ и просмотреть файл testfile позднее.

Перенаправление вывода на вход другой команды

Символ | называется каналом. Он является мощным средством, которое позволяет брать вывод одной команды и использовать его в качестве ввода для другой команды без создания временных файлов. Таким образом, построенная последовательность команд называется конвейером. Общий формат конвейера:

```
$ command1 | command2 | command3 ...
```

Вывод command1 используется как ввод для command2. Вывод command2 используется как ввод для command3.

Чтобы понять, насколько эффективен конвейер, рассмотрим два способа, которые дают одинаковый результат:

– использование метода перенаправления ввода/вывода. Запустим одну команду и перенаправим ее вывод во временный файл, затем запустим вторую команду, которая берет содержимое временного файла как ввод, и в конце удалим временный файл;

– использование метода конвейера. Например, предположим, что необходимо послать сообщение happy birthday с помощью команды banner пользователю david.

По первому методу последовательность команд будет следующей:

1. Ввод команды banner и перенаправление ее вывода во временный файл:

```
$ banner happy birhday > message.tmp
```

2. Ввод команды mail и в качестве ввода использование файла message.tmp:

```
$ mail david < message.tmp
```

3. Удаление временного файла:

```
$ rm message.tmp
```

Вторым методом это можно сделать быстрее:

```
$ banner happy birthday | mail david
```

Подстановка вывода в качестве аргумента

Вывод большинства команд может использоваться как аргумент в командной строке. Для этого команду необходимо заключить между знаками «слабое ударение» (`...`) и поместить ее в командной строке в том месте, где вывод будет трактоваться как аргумент.

Например, подстановка вывода конвейера команд date и cut в качестве аргумента в команде banner:

```
$ banner `date | cut -c12-19`
```

В результате этой комбинации система печатает banner с текущим временем.

3.1.4. Выполнение, останов и повторный запуск процессов. Запуск команд в заданное время

Команды batch и at позволяют определять время запуска команды или последовательности команд. При помощи команды batch система определяет время запуска команды, которое можно определить с помощью команды at. Обе команды ожидают ввода со стандартного входа (терминала); список команд, вводимых с терминала, должен завершаться нажатием клавиши ^d (одновременное нажатие клавиши Ctrl и клавиши d).

Команда batch очень полезна, если запускается процесс или программа, которые занимают много системного времени. Команда batch представляет системе задание (содержащее последовательность команд для выполнения), которое ставится в очередь и запускается как только у системы появляется возможность. Это позволяет системе быстро отвечать на запросы других пользователей. Общий формат команды batch:

```
$ batch
первая команда
.
.
.
```

```
последняя команда
<^d>
$
```

Если запускается только одна команда, то ее можно ввести в одной командной строке:

```
$ batch command_line
```

В следующем примере batch используется для выполнения команды grep в согласованное время. Команда grep осуществляет поиск всех файлов в текущем каталоге и перенаправляет вывод в файл dol.file:

```
$ batch
grep dollar * > dol.file
<^d>
job 155223141.b at Sun Dec 11:14:54 1989
$
```

После того, как будет дано задание batch, система выдаст ответ, в котором даны номер задания, дата и время. Номер задания – не то же самое, что номер процесса, который система генерирует, когда пользователь запускает команду в фоновом режиме.

Команда at позволяет указывать точное время выполнения команд. Общий формат команды at:

```
$ at time
первая команда
.
.
.
последняя команда
<^d>
$
```

Аргумент time состоит из времени дня и даты, если дата не сегодняшняя.

В следующем примере показано, как использовать команду at для посылки сообщения happy birthday пользователю с регистрационным именем emily:

```
$ at 8:15am Feb 27
banner happy birthday | mail emily
<^d>
$
```

Следует обратить внимание, что команда at подобно команде batch выдает ответ с номером задания, датой и временем. Если необходимо удалить команду из очереди, следует воспользоваться опцией -r в команде at, указав ее с номером

задания. Общий формат такой команды:

```
$ at -r jobnumber
```

Например, чтобы удалить предыдущее задание, следует ввести:

```
$ at -r 453400603.a
```

Если пользователь забыл номер задания, то команда

```
$ at -l
```

распечатает список текущих заданий в очереди batch или at, как показано на следующем экране:

```
$ at -l
user mylogin 168302040.a at Sat Nov 25 13:00:00 1989
user mylogin 453400603.a at Fri Feb 24 08:15:00 1989
$
```

Таким образом команда at выполняет команды в указанное время. Можно использовать от одной до 4 цифр и буквосочетания «am» и «pm», чтобы указать время. Чтобы указать дату, следует задать имя месяца и вслед за ним число. Если задание должно быть выполнено сегодня, то дату вводить не надо.

Пример:

```
at 08:15am Feb 27
at 5:14pm Sept 24
```

Получение состояния запущенного процесса

Команда ps дает состояние всех процессов, запущенных на данный момент. Например, пользователь может использовать команду ps, чтобы просмотреть состояние всех процессов, которые запущены в фоновом режиме, применив символ &.

Далее рассмотрим вопрос, как применить номер PID (идентификатор процесса), чтобы остановить выполнение команды. PID является уникальным номером, который система UNIX назначает каждому активному процессу.

В следующем примере команда grep запускается в фоновом режиме и затем выдается команда ps. Система сообщает в ответ номер идентификации процесса (PID) и номер терминала (TTY). Она также выдает время выполнения каждого процесса (TIME) и имя команды, которая выполняется (COMMAND):

```
$ grep word * > temp
28223
$
```

```
$ ps
PID     TTY    TIME    COMD
28124   tty10  0:00    sh
28223   tty10  0:04    grep
28224   tty10  0:04    ps
$
```

Следует обратить внимание, что система распечатала номер PID для команды grep так же, как и для всех других запущенных процессов: для самой команды ps и команды sh, которая была запущена во время вашей регистрации.

Возможно приостановить и вновь запустить программу. Команда jobs выдает список текущих фоновых процессов, запущенных или приостановленных. Команда jobs дополнительно к PID распечатывает идентификатор задания (JID) и имя задания. Чтобы вновь запустить приостановленное задание либо возобновить фоновый процесс в оперативном режиме, необходимо знать JID, который распечатывается на экране каждый раз, когда пользователь вводит команду запуска или останова процесса. Если ввести

```
$ jobs
```

то на экране появится следующая информация:

```
[JID] – Stopped (signal) <имя задания>
```

или

```
[JID] + Running <имя задания>
```

Завершение активных процессов

Команда kill завершает активные процессы в фоновом режиме, а команда stop приостанавливает временно процессы. Общий формат этих команд

```
$ kill PID
```

или

```
$ stop JID
```

Следующий пример показывает, как можно завершить команду grep, которая выполняется в фоновом режиме:

```
$ kill 28223
28223 Terminated
$
```

После того, как система выдаст ответ на запрос, на экране появится подсказка $, означающая, что процесс завершен. Если система не найдет

указанный PID, то появится сообщение об ошибке:

```
kill:28223:No such process
```

Чтобы приостановить оперативный процесс (если активна функция управления заданиями), необходимо ввести:

```
[Ctrl+Z]
```

На экране появится следующее сообщение:

```
<JID> Stopped(user) <имя задания>
```

Запуск остановленного задания

Если функция управления заданиями активна, то при необходимости можно вновь запустить приостановленный процесс. Чтобы вновь запустить процесс, остановленный командой stop, необходимо сначала определить JID с помощью команды jobs. Затем использовать JID со следующими командами:

fg <JID> – возобновляет приостановленное задание или переводит задание из фонового режима в оперативный;

bg <JID> – вновь запускает приостановленное задание в фоновом режиме.

Использование команды nohup

Все процессы, за исключением at и batch, завершаются, когда пользователь выходит из системы. Если необходимо, чтобы после выхода из системы процесс в фоновом режиме продолжал выполняться, то следует использовать команду nohup. Команда nohup имеет следующий формат:

```
$ nohup command &
```

Предположим, необходимо, чтобы команда grep осуществила поиск во всех файлах в вашем текущем каталоге строки «word» и перенаправила вывод в файл word.list, но если пользователю нужно выйти из системы, не ожидая завершения, то необходимо ввести следующую строку:

```
$ nohup grep word * > word.list &
```

Завершить команду nohup можно с помощью команды kill.

3.2. Краткая справка по командам UNIX

Обычно человек, сидя перед монитором, знает, что он хочет сделать. Основная проблема заключается в том, чтобы объяснить это операционной системе.

Ниже приведен список наиболее часто используемых команд. Для получения более полной информации по какой-либо команде воспользуйтесь

программой справки man [команда].

Функция	Команда
Создание файлового архива	tar, cpio
Архивация файла	compress, pack
Замер времени исполнения команды	time, timex
Запуск программы в указанное время	at
Вывод файла на экран	cat, page, xedit
Постраничный вывод файла на экран	more
Вывод на экран первых десяти строк файла	head
Вывод на экран последних десяти строк файла	tail
Выполнение вычислений	be, dc
Вывод даты и времени	date
Изменение даты модификации файла на текущую	touch
Деархивация файла	unpack, uncompress
Декодирование UU-кода	uudecode
Вывод объема свободного дискового пространства	df
Завершение работы	exit
Печать заголовка	banner
Захват изображения на экране	xv, xwd
Вывод информации о системе	uname
Запуск интерпретатора командной строки	sh, csh, ksh
Запуск интерпретатора командной строки на удаленной системе	rsh
Вывод календаря	cal, cm, dtcm
Калькулятор	calctool, xcalc
Создание каталога	mkdir
Вывод размера каталога	du
Вывод списка файлов и подкаталогов данного каталога	ls
Смена текущего каталога	cd
Удаление каталога	rmdir
Вывод имени текущего каталога	pwd
Сравнение содержимого двух каталогов	dircmp
UU-кодирование файла для пересылки его по электронной почте	uuencode
Выполнение команды в указанное время	at
Ввод команды при работе с графическим интерфейсом	xterm
Компиляция С-программ	cc
Копирование файлов	cat, cp
Копирование файлов на удаленную систему и с удаленной системы	ftp, rcp, uucp
Вычисление контрольной суммы файла	sum
Объединение нескольких файлов в один	cat
Вывод или установка значений переменных окружения	env

Ожидание завершения процесса	wait
Отправка сообщения другому пользователю	write
Очистка экрана	clear
Вывод состояния службы печати	lpstat
Запуск службы печати	lpsched
Останов службы печати	lpshut
Печать заголовка	banner
Подсчет количества слов в текстовом файле	wc
Автоматическое выполнение повторяющихся задач	crontab
Поиск текстовых строк	grep, fgrep
Поиск и замена символов	tr
Поиск в файле	awk, nawk
Поиск файлов	find
Вывод списка пользователей	listusers
Вывод информации о других пользователях системы	who
Поиск информации о других пользователях системы	finger
Выполнение последовательности команд	batch
Запуск команды с пониженным приоритетом	nice
Проверка правописания	spell
Прерывание процесса	kill
Вывод списка процессов	ps
Разбиение файла на части	csplit, split
Редактирование текстового файла	vi, ed
Резервирование информации	tar, cpio
Поиск и замена символов	tr
Вывод системного идентификатора пользователя	id
Вывод системных сообщений	news
Подсчет числа слов в файле	wc
Сортировка файлов	sort
Сортировка и обработка файла	awk, nawk
Вывод списка файлов в каталоге	ls
Сравнение содержимого двух каталогов	dircmp
Сравнение содержимого двух отсортированных файлов	comm
Сравнение двух файлов и вывод отличающихся строк	diff, bdiff
Сравнение двух файлов	cmp
Создание ссылок	ln
Печать на стандартный вывод	echo
Запись стандартного вывода в файл	tee
Вывод столбца из отсортированного файла	cut
Поиск текстовых строк в бинарном файле	strings
Поиск текстовых строк	egrep, grep, fgrep
Вход в удаленную систему	telnet, riogin
Вывод имени текущего каталога	pwd
Запуск интерпретатора командной строки на удаленной системе	rsh

Копирование файлов с удаленной системы	rcp, uucp, ftp
Определение типа файла	file
Открытие окна терминала	xterm
Отправка и чтение электронной почты	mailx, mail
Удаление заданий из очереди печати	cancel
Удаление из очереди заданий, созданных командой at	atrm
Удаление каталога	rmdir
Удаление файла	rm
Удаление форматирующих символов из файла	col
Шифрование файла	crypt

4. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

4.1. Изучить методические указания.

4.2. После предварительной беседы с преподавателем получить у него регистрационное имя, конкретное задание и номер терминала.

4.3. После запуска сервера преподавателем (лаборантом) приступить к конкретному выполнению задания.

4.4. После окончания выполнения задания предъявить результаты преподавателю (в электронном либо бумажном виде).

4.5. Выйти из системы, сдать рабочее место.

5. СОДЕРЖАНИЕ ОТЧЕТА

5.1. Описание процедуры входа в систему.

5.2. Описание процедуры регистрации в системе.

5.3. Текст созданных файлов.

5.4. Описание работы с файлами – создание, копирование, уничтожение, переименование, перемещение, шифрование и т.д.

5.5. Пункты, конкретно заданные преподавателем.

ЛИТЕРАТУРА

1. DRS/NX. Desktop Handbook, 1993. The Customer Publication Manager. ICL, Bracknell, Berkshire RG12 8SN.

2. DRS/NX. Software and Documentation, 1993. The Customer Publication Manager. ICL, Bracknell, Berkshire RG12 8SN.

Св. план 2003, поз. 84

Учебное издание

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ В СЕТЯХ ТЕЛЕКОММУНИКАЦИЙ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторно-практическим занятиям
для студентов специальности 45 01 03 «Сети телекоммуникаций»
дневной, вечерней и заочной форм обучения

В 3-х частях

Часть 1
Операционная система DRS-Unix.
Основные особенности работы,
работа с файлами, регистрация в системе

С о с т а в и т е л и :
Мельников Владимир Александрович,
Рощупкин Яков Викторович

Редактор Н.А. Бебель
Корректор Е.Н. Батурчик
Компьютерная верстка Т.В. Шестакова

Подписано в печать 30.06.2003. Формат 60x84 1/16. Бумага офсетная.
Печать ризографическая. Гарнитура «Таймс». Усл. печ. л. 1,98.
Уч.-изд. л. 1,5. Тираж 50 экз. Заказ 760.

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Лицензия ЛП № 156 от 30.12.2002.
Лицензия ЛВ № 509 от 03.08.2001.
220013, Минск, П. Бровки, 6.