Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Кафедра автоматизации обработки информации (АОИ)

Ю. Б. Гриценко

Методические указания к выполнению лабораторных работ по дисциплине «Вычислительные системы, сети и телекоммуникации»

Для студентов направления подготовки
Программная инженерия
(квалификация (степень) "бакалавр")

2015

Корректор: Осипова Е. А.

Гриценко Ю. Б.
Методические указания к выполнению лабораторных работ по дисциплине «Вычислительные системы, сети и телекоммуникации». Для студентов направления подготовки Программная инженерия (квалификация (степень) "бакалавр"). — Томск: Факультет дистанционного обучения, ТУСУР, 2015. — 35 с.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящие методические указания соответствуют требованиям Федерального Государственного образовательного стандарта высшего профессионального образования (ФГОС ВПО) третьего поколения и содержанию рабочих программ по дисциплине «Вычислительные системы, сети и телекоммуникации».

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

В рамках изучения дисциплины необходимо выполнить одну лабораторную работу.

Выбор варианта лабораторной работы осуществляется по общим правилам с использованием следующей формулы:

$$\mathbf{V = (N \times K) \ div \ 100,}$$

где V — искомый номер варианта,

N — общее количество вариантов,

div — целочисленное деление,

при V= 0 выбирается максимальный вариант,

K — код варианта.

Варианты заданий представлены в разделе 3 методических указаний.

1 ИНТЕРФЕЙС КОМАНДНОЙ СТРОКИ ОС UNIX

Большинство версий операционной системы Unix имеют графический интерфейс, подобный интерфейсу, используемому на компьютерах Macintosh и впоследствии в IBM-совместимых компьютерах с операционной системой Windows. Однако интерфейс командной строки до сих пор остается популярным среди программистов.

Изучение команд Unix-подобных операционных систем можно вести, установив систему на жесткий диск своего компьютера или загрузив Unix (Linux) с компакт-диска (CD) без инсталляции на жесткий диск. Удобно воспользоваться эмулятором Unix для операционной системы Windows. Эмулятор можно легко найти в Интернете (любая поисковая система находит различные эмуляторы по поисковому запросу: «эмулятор Unix»).

Приглашение к вводу команды в Unix может выглядеть по-разному: # — это приглашение для суперпользователя (root), вошедшего в систему; $ или [имя@localhost имя]$ — для обычных пользователей. Помощь по командам Unix можно получить, набрав man и через пробел — имя команды, например: man gnome.

Вход в систему производится в диалоге, когда система запрашивает имя пользователя и его пароль.

Выход из системы может производиться по-разному. Для выхода из системы служит команда logout, по которой прекращается сеанс работы с данным пользователем, но система не завершает свою работу. Прекратить текущий сеанс работы можно также, нажав одновременно три клавиши Ctrl+Alt+Backspace.

Для полного завершения работы нажимается Ctrl+Alt+Del.

Перечень команд. В состав Unix входит более трехсот команд. Данная операционная система обладает более богатыми методами управления ресурсами, чем ОС Windows.

При изучении системы команд Unix необходимо иметь в виду, что многие команды операционных систем MS DOS и Windows совпадают по имени и, частично, по функциям с командами Unix (табл. 1.1).

Таблица 1.1 — Список команд Unix

at	Выполнение команд и запуск программ по расписанию
cat	Вывод на экран содержимого файлов
cd	Выбор имени либо смена текущей папки
chmod	Изменение атрибутов прав доступа к файлам
cmp	Поиск различий между файлами (до первого различия)
cp	Копирование одного или нескольких файлов в другое место
date	Вывод текущей даты
df	Определение свободного пространства диска
diff	Поиск всех различий в файлах
du	Определение занятого пространства диска
ed	Вызов текстового редактора
exit	Выход из системы
kill	Послать сигнал процессу. Завершить процесс.
ls	Вывод списка файлов в каталоге
mail	Вызов почтового клиента
man	Вызов помощи по командам
mesg	Установка режима запрета или разрешения сообщений
mkdir	Создание каталога
mv	Перенос файлов
news	Запуск клиента новостей
nice	Запуск программы с пониженным приоритетом
nohup	Выполнение программы после отключения терминала
pr	Распечатка файла по 66 строк
ps	Получение списка процессов
pwd	Определение своего рабочего каталога
rm	Удаление файла
rmdir	Удаление каталога
sh	Переход в порожденный командный процессор (shell)
tail	Вывод на экран содержимого файлов с конца
time	Информация о времени выполнения команды
touch	Заменяет время модификации файла на настоящее
uname	Информация о системе
wc	Подсчет числа строк, слов и символов в файле
who	Вывод активных пользователей
who am i	Вывод собственного имени
write	Установка связи с другим пользователем

Окончание табл. 1.6

Файлы и процессы, являются центральными понятиями операционной системы UNIX. Файловая подсистема управляет файлами, размещает записи файлов в отведенные для них места, управляет свободным пространством, доступом к файлам и поиском данных для пользователей.

Работа с файлами ведется с помощью команд. Команда представляет собой имя исполняемого файла (двоичного или текстового, так называемого скрипта, написанного на одном из специальных командных языков) или имя внутренней команды самого процессора. При активизации каждой такой команды операционная система создает процесс. Процессы взаимодействуют с подсистемой управления файлами и с аппаратными средствами, используя для этого совокупность специальных команд, таких как ***open*** (для того, чтобы открыть файл на чтение или запись), ***close***, ***read***, ***write***, ***stat*** (запросить атрибуты файла).

Подсистема управления процессами ядра ОС отвечает за синхронизацию процессов, их взаимодействие, распределение памяти и планирование выполнения процессов, По характеру выполнения процессы могут быть фоновыми и привилегированными. Любой запускаемый процесс по умолчанию будет выполняться как привилегированный (***foreground***). Это значит, что такой процесс постоянно связан с терминалом ЭВМ и делает невозможным выполнение еще каких-либо действий с системой, пока не завершится.

Фоновый процесс (***background***) после запуска освобождает терминал и позволяет перейти к другой задаче, не дожидаясь его завершения. Фоновая обработка наиболее пригодна для процессов, которые долго выполняются. Программы, выполняющиеся в виде фоновых процессов, называются демонами (***daemon***). В любой момент времени в системе существуют десятки процессов, которые были запущены при старте операционной систе-

мы, вызваны ядром для обслуживания каких-либо событий, добавлены пользователем при запуске какой-либо задачи.

Обычно большинство процессов находится в состоянии ожидания — сна (sleep), не мешая остальным и дожидаясь сигнала для активизации. Кроме того, в системе можно найти процессы, закончившие работу, но еще не получившие разрешения на выгрузку из основной памяти, — эти процессы называются зомби. В ядре операционной системы находится таблица процессов, каждая запись которой описывает состояние одного из процессов.

Основное отличие **файловой системы Unix** от файловой системы Windows заключается в том, что в Unix отсутствует такое понятие, как логическое устройство. При указании пути к файлу в Unix имя устройства не упоминается. Дерево каталогов Unix «растет» из одного корня. Корневой каталог имеет предопределенное имя / (слэш). Этот же символ применяется и для разделения подкаталогов. Полный путь к файлу в Unix выглядит следующим образом:

/каталог1/каталог2/каталог3/ ... /файл.

Физически разные компоненты дерева каталогов Unix могут размещаться на разных дисках, но логически они принадлежат одной древовидной структуре с одним корневым узлом. Для объединения файловых систем различных устройств в одну структуру используется операция монтирования.

Сущность этой операции заключается в том, что каждое физическое устройство можно рассматривать, как свою собственную файловую систему (файловую систему устройства) с корневым каталогом /. Если этот раздел диска объявлен в операционной системе как корневой раздел (root), его каталог становится корневым каталогом всей файловой системы (файловой

системы ЭВМ). Файловые системы остальных устройств должны быть смонтированы в каталогах файловой системы ЭВМ.

Операция монтирования связывает корневой каталог монтируемого раздела (устройства) с выбранным каталогом файловой системы ЭВМ — точкой монтирования. В результате монтирования корневой каталог файловой системы устройства получает имя каталога, являющегося точкой монтирования, благодаря чему файловая система устройства «привязывается» к файловой системе ЭВМ в точке монтирования.

Таким образом, для монтирования файловой системы устройства в файловой системе ЭВМ необходимо сначала в файловой системе ЭВМ создать каталог, который будет точкой монтирования, а затем соединить две файловые системы командой ***mount.***

После монтирования сменных носителей их нельзя извлекать из устройства без демонтирования файловой системы. Для этой цели служит команда ***umount.***

При просмотре содержимого каталога можно увидеть, что информация о файле начинается с кода, содержащего 10 символов:

-rwxrwxrwx, или **drwxrwxrwx**, и **lrwxrwxrwx.**

Первый символ кода указывает тип файла:

– ***символ —*** означает, что это обычный файл, текстовый или двоичный, содержащий данные или программу;

– ***символ d*** указыет, что данный файл является каталогом;

– ***символ l*** указывает на то, что данный файл является символьной ссылкой.

Символы ***rwx*** определяют права доступа к файлу. Доступ к файлу могут иметь три категории пользователей:

– владелец файла;

– выделенная группа пользователей;

– остальные пользователи (не являющиеся владельцами файла и не входящие в выделенную группу).

Всем им могут быть установлены следующие права:

– ***символ r*** — разрешено чтение файла;

– ***символ w*** — разрешена запись в файл;

– ***символ x*** — разрешен запуск файла на исполнение.

Эти права всегда перечисляются подряд в порядке: ***rwx***. Если какое-либо право не предоставлено, вместо соответствующего символа ставится -. Например, ***r - -*** означает, что разрешено только чтение; ***-wx*** означает, что разрешены запись в файл и его исполнение.

Поскольку права определяются для трех видов пользователей, указанная триада повторяется трижды и образует запись из 9 символов, первые три из которых относятся к владельцу, вторые — к группе и третьи — к остальным пользователям. Например, запись rwx--xr-- означает, что владельцу файла разрешено все, выделенной группе — только запуск на исполнение, остальным пользователям — только чтение.

Иногда указанные девять символов кодируются числом. Ключ к расшифровке цифрового кода приведен в виде таблицы 1.2.

Таблица 1.2 — Ключ расшифровки цифрового кода прав доступа

4	2	1	4	2	1	4	2	1
r	w	x	r	w	x	r	w	x
1 цифра			2 цифра			3 цифра		
4+2+1=7			4+2+1=7			4+2+1=7		

Установка и изменение режима доступа к файлу производятся с помощью команды ***chmod***.

Формат команды chmod (change mode) для установки режима: ***chmod <режим> <файлы>***

Пример использования команды: $ chmod 644 f1 f2 f3,

где 644 соответствует rw-r--r--

Формат команды chmod для изменения режима:

chmod <изменения> <файлы>

В изменениях используются обозначения:

– r — read — права на чтение;

– w — write — права на запись;

– x — execute — права на выполнение;

– u — user — права для владельца файла;

– g — group — права для группы пользователей;

– o — other — права для остальных пользователей;

– a — all — прва для всех пользователей;

– = — назначить права;

– + — добавить права;

– - — отнять права.

Пример использования команды:

```
$ ls -l
-r-------- ... f1
-r-------- ... f2
-r-------- ... f3
$ chmod  a = r, u + w    f1 f2 f3
```

или (эквивалентный вариант изменения прав доступа)

```
$ chmod  u = rw, go = r  f1 f2 f3
$ ls -l
-rw-r--r-- ... f1
-rw-r--r-- ... f2
-rw-r--r-- ... f3
$ chmod o-r f1 f2 f3
```

$ ls -l

-rw-r----- ... f1

-rw-r----- ... f2

-rw-r----- ... f3

Другие пользователи, не входящие в группу, потеряли право читать файлы.

В Unix операционных системах есть также возможность использовать **метасимволы**. Метасимволы служат для подстановки любых строк и символов. В именах файлов в командах языка заданий Shell:

– * — представляет произвольную строку (возможно, пустую);

– ? — любой одиночный знак;

– [C1 - C2] — любая литера из диапазона C1–C2 (в стандарте ASCII).

Примеры:

1) $ ls c?

c1 c2 c3 cs cz

2) $ ls c*

c1 c12 c2 c23 c3 cs cs1 cxy cz

3) $ ls ?1*

c1 c12

4) $ ls *1*

c1 c12 cs1

5) $ ls c [12 x y z]

c1 c2 cz

6) ls c [12 x y z *]

c1 c2 c12 c25 cz cxy

Стандартные файлы. Многие команды работают по умолчанию со стандартными файлами:

– Standard Input (S.I.) — стандартный ввод;

– Standard Output (S.O.) — стандартный вывод;

– Diagnostic Output (D.O.) — диагностический вывод.

Однако есть средства изменения умолчания, т. е. возможность указать другие файлы вместо стандартных. Можно также в качестве диагностического вывода использовать стандартный вывод. Эти средства называются перенаправлением (редирекцией) ввода и вывода.

Примеры:

1. Перенаправления стандартного ввода:

$ cat < this_file

2. Одновременные перенаправления ввода и вывода:

$ cat < left > right

3. Перенаправления стандартного вывода:

$cat > newfile

1. Соединение команд каналами (***pipeline***)

$ who | wc –l — создание списка активных пользователей и подсчет их числа (count); 19 — ответ, то есть 19 пользователей.

$ ls -l /tmp | grep vladimir | sort +3nr | lpr

листинг каталога — поиск записей, содержащих (по 4-му по- — сортировка — печать упорядоченного списка

/tmp жащих стро- лю) найден-
ку vladimir ных записей

5. Одновременный стандартный вывод и перенаправление вывода

$ ls -l | ***tee*** dirconts — команда одновременно выводит содержимое текущего каталога на экран и в файл dirconts.

Сообщения об ошибках, возникающих при выполнении команд, выводятся на диагностический вывод, по умолчанию это (как и стандартный вывод) — на экран.

Диагностический вывод тоже может быть перенаправлен в любой файл. Для этого используется дескриптор файла (целое), который для стандартных файлов равен:

0 — Standard input;

1 — Standard output;

2 — Diagnostic output.

Если вы хотите, чтобы сообщения об ошибках нигде не проявлялись, направьте их на /dev/null.

Пример:

$ cat somefile > outfile 2> errfile,

где знак > эквивалентен 1>.

Разработка командных файлов. Для того чтобы текстовый файл можно было использовать как командный, существует несколько возможностей. Можно вызвать оболочку ***shell*** (интерпретатор команд, подаваемых с терминала или из командного файла, — это обычная программа, которая не входит в ядро операционной системы UNIX) как команду, обозначаемую ***sh***, и передать ей файл ***f1*** как аргумент или как перенаправленный вход: $ sh fl или $ sh < f1.

Файл можно выполнить и в текущем экземпляре shell. Для этого существует специфическая команда. (точка). Пример: ***.f1***

Еще один способ, это сделать текстовый файл исполняемым с помощью команды chmod. Пример: chmod 711 f1.

Shell имеет в своем составе функциональные возможности, благодаря которым его можно смело назвать языком программирования. К этим функциональным возможностям относятся:

- переменные;
- управляющие структуры (типа if);
- подпрограммы (в том числе командные файлы);
- передача параметров;
- обработка прерываний.

Переменные Shell

В языке Shell версии 7 определение переменной содержит имя и значение: var = value.

Доступ к переменной — по имени со знаком $ спереди:

fruit = apple (определение);

echo $fruit (доступ);

apple (результат echo).

Таким образом, переменная — это строка. Возможна конкатенация строк:

$ fruit = apple

$ fruit = pine$fruit

$ echo $fruit

```
pineapple
$ fruite = apple
$ wine = ${fruite}jack
$ echo $wine
applejack
$
```

Другие способы установки значения переменной — ввод из файла или вывод из команды, а также присваивание значений переменной — параметру цикла for из списка значений, заданного явно или по умолчанию.

Переменная может быть:

– частью полного имени файла: $d/filename, где $d — переменная (например, d = /usr/bin);

– частью команды:

```
$ S = "sort + 2n + 1 - 2" (наличие пробелов требует кавычек "")
$ $S tennis/lpr
$ $S basketball/lpr
$ $S pingpong/lpr
$
```

Однако внутри значения для команды не могут быть символы |, >, <, & (обозначающие канал, перенаправления и фоновый режим).

Предопределенные переменные Shell. Некоторые из них можно только читать. Наиболее употребительные:

HOME — «домашний» каталог пользователя; служит аргументом по умолчанию для cd;

PATH — множество каталогов, в которых UNIX ищет команды;

Изменение PATH:

$ echo $PATH	- посмотреть;
:/bin:/usr/bin	- значение PATH;
$ cd	- «домой»;
$ mkdir bin	- новый каталог;
$ echo $HOME	- посмотреть;
/users/maryann	- текущий каталог;
$ PATH = :$HOME/bin:$PATH	- изменение PATH;
$ echo $PATH	- посмотреть;
:/users/maryann/bin:/bin:/usr/bin	- новое значение PATH.

Пример 1 (установка переменной Shell выводом из команды):

$ now = `date` (где `` - обратные кавычки)

$ echo $now

Sun Feb 14 12:00:01 PST 1985

$

Пример 2 (получение значения переменной из файла):

$ menu = `cat food`

$ echo $menu

apples cheddar chardonnay (символы возврата каретки заменяются на пробелы).

Переменные Shell — аргументы процедур

Это особый тип переменных, именуемых цифрами.

Пример:

$ dothis grapes apples pears (процедура).

Тогда позиционные параметры (аргументы) этой команды доступны по именам:

$1 = `grapes`

$2 = `apples`

$3 = `pears`

и т. д. до $9. Однако здесь также есть команда ***shift***, которая сдвигает имена на остальные аргументы, если их больше 9 (окно шириной 9).

Другой способ получить все аргументы (даже если их больше 9): $*, что эквивалентно $1$2... Количество аргументов присваивается другой переменной: $# (диез). Наконец, имя процедуры — это $0; переменная $0 не учитывается при подсчете $#.

Структурные операторы Shell

Оператор цикла **for**

Пусть имеется командный файл makelist (процедура)

```
$ cat makelist
sort +1 -2 people | tr -d -9 | pr -h Distribution | lpr.
```

Если вместо одного файла people имеется несколько, например: adminpeople, hardpeople, softpeople,..., то необходимо повторить выполнение процедуры с различными файлами. Это возможно с помощью for-оператора. Синтаксис:

```
for <переменная> in <список значений>
do <список команд>
done
```

Ключевые слова for, do, done пишутся с начала строки.

Пример (изменим процедуру makelist):

```
for file in adminpeople, hardpeople, softpeople
do
    Sort +1 -2 $file | tr ... | lpr
done.
```

Можно использовать метасимволы Shell в списке значений.

Пример:

```
for file in *people (для всех имен, кончающихся на people)
do
    ...
done.
```

Если in опущено, то по умолчанию в качестве списка значений берется список аргументов процедуры, в которой содержится цикл, а если цикл не в процедуре, то список параметров командной строки (то есть в качестве процедуры выступает команда).

Пример:

```
for file
do
    ...
done
```

Для вызова makelist adminpeople hardpeople softpeople будет сделано то же самое.

Условный оператор ***if***

Используем имена переменных, представляющие значения параметров процедуры:

```
sort +1 -2 $1 | tr ... | lpr
```

Пример неверного вызова:

makelist (без параметров), где $1 неопределен. Исправить ошибку можно, проверяя количество аргументов — значение переменной $# посредством if-оператора.

Пример: (измененной процедуры makelist):

```
if test $# -eq 0
then
```

```
        echo "Вы должны указать имя файла"
        exit 1
else
        sort +1 -2 $1 | tr ... | lpr
fi
```

Здесь ***test*** и ***exit*** — команды проверки и выхода. Таким образом, синтаксис оператора if:

if <если эта команда выполняется успешно, то>;

then <выполнить все следующие команды до else или, если его нет, до fi>;

Ключевые слова ***if***, ***then***, ***else*** и ***fi*** пишутся с начала строки.

Успешное выполнение процедуры означает, что она возвращает значение true = 0 (zero) (неуспех — возвращаемое значение не равно 0).

Оператор exit 1 задает возвращаемое значение 1 для неудачного выполнения makelist и завершает процедуру.

Возможны вложенные if. Для else if есть сокращение ***elif***, которое одновременно сокращает fi.

Команда ***test***

Не является частью Shell, но применяется внутри Shell-процедур.

Имеется три типа проверок:

– оценка числовых значений;

– оценка типа файла;

– оценка строк.

Для каждого типа свои примитивы (операции op).

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

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

2. Для файла синтаксис такой: op filename, где op принимает значения:

– -s (файл существует и не пуст);

– -f (файл, а не каталог);

– -d (файл-директория (каталог);

– -w (файл для записи);

– -r (файл для чтения).

3. Для строк синтаксис такой: S op R, где S, R — строки или строковые переменные или op1 S, где op принимает значения:

– = (эквивалентность);

– != (не эквивалентность);

– op1 принимает значения:

– -z (строка нулевой длины);

– -n (ненулевая длина строки).

Несколько проверок разных типов могут быть объединены логическими операциями -a (AND) и -o (OR).

Примеры:

```
$ if test -w $2 -a -r S1
> then cat $1 >> $2
> else echo "невозможно добавить"
> fi
$
```

В некоторых вариантах ОС UNIX вместо команды ***test*** используются квадратные скобки, т. е. if [...] вместо if test ... .

Оператор цикла ***while***

Синтаксис:

```
while <команда>
do
<команды>
```

done

Если «команда» выполняется успешно, то выполнить «команды», завершаемые ключевым словом done.

Пример:

```
if test $# -eq 0
then
    echo "Usage: $0 file ..." > &2
    exit
fi
while test $# -gt 0
    do
if test -s $1
    then
    echo "no file $1" > &2
    else
    sort + 1 - 2 $1 | tr -d ... (процедуры)
fi
    shift (* перенумеровать аргументы *)
done
```

Процедуры выполняются над всеми аргументами.

Оператор цикла ***until***

Инвертирует условие повторения по сравнению с while

Синтаксис:

```
until <команда>
do
<команды>
```

done

Пока «команда» не выполнится успешно, выполнять команды, завершаемые словом done.

Пример:

```
if test S# -eq 0
then
        echo "Usage $0 file..." > &2
        exit
fi
        until test S# -eq 0
        do
        if test -s $1
        then
        echo "no file $1" > &2
else
        sort +1 -2 $1 | tr -d ... (процедура)
fi
        shift (сдвиг аргументов)
 done
```

Исполняется аналогично предыдущему.

Оператор выбора ***case***

Синтаксис:

```
case <string> in
string1) <если string = string1, то выполнить все следующие команды до ;; >
;;
```

string2) <если string = string2, то выполнить все следующие команды до ;; >
;;
string3) ... и т. д. ...
esac

Пример:

Пусть процедура имеет опцию -t, которая может быть подана как первый параметр:

.....
together = no
case $1 in
-t) together = yes
shift ;;
-?) echo "$0: no option $1"
exit ;;
esac
if test $together = yes
then
sort ...
fi

где ? — метасимвол (если -?, т. е. «другая» опция, отличная от -t, то ошибка). Можно употреблять все метасимволы языка Shell, включая ?, *, [-].

Использование временных файлов в каталоге /tmp

Это специальный каталог, в котором все файлы доступны на запись всем пользователям.

Если некоторая процедура, создающая временный файл, используется несколькими пользователями, то необходимо обеспечить уникальность имен создаваемых файлов. Стандартный прием — имя временного файла

$0$$, где $0 — имя процедуры, а $$ — стандартная переменная, равная уникальному идентификационному номеру процесса, выполняющего текущую команду.

Хотя администратор периодически удаляет временные файлы в /tmp, хорошей практикой является их явное удаление после использования.

Комментарии в процедурах

Они начинаются с двоеточия :, которое считается нуль-командой, а текст комментария — ее аргументом. Чтобы Shell не интерпретировал метасимволы ($, * и т. д.), рекомендуется заключать текст комментария в одиночные кавычки.

В некоторых вариантах ОС UNIX примечание начинается со знака #.

Пример процедуры

```
:'Эта процедура работает с файлами, содержащими имена'
: 'и номера телефонов,'
:'сортирует их вместе или порознь и печатает результат на'
:'экране или на принтере'
:'Ключи процедуры:'
:'-t (together) - слить и сортировать все файлы вместе'
:'-p (printer) - печатать файлы на принтере'
if test $# - eq 0
then
        echo "Usage: $ 0 file ... " > & 2
        exit
fi
together = no
print = no
while test $# -gt 0
do case $1 in
```

```
-t)     together = yes
        shift ;;
-p)     print = yes
        shift ;;
-?)     echo "$0: no option $1"
        exit ;;
*) if test $together = yes
then
        sort -u +1 -2 $1 | tr ... > /tmp/$0$$
        if $print = no
then
        cat /tmp/$0$$
        else
        lpr -c /tmp/$0$$
fi
        rm /tmp/$0$$
        exit
else if test -s $1
then    echo "no file $1" > &2
        else    sort +1 -2 $1 | tr...> /tmp/$0$$
if $print = no
then    cat /tmp/$0$$
else    lpr -c /tmp/$0$$
fi
                rm /tmp/$0$$
        fi
shift
fi;;
```

esac

done.

Процедура проверяет число параметров $# и, если оно равно нулю, завершается. В противном случае она обрабатывает параметры (оператор case). В качестве параметра может выступать либо ключ (символ, предваряемый минусом), либо имя файла (строка, представленная метасимволом *). Если ключ отличен от допустимого (метасимвол ? отличен от t и p), процедура завершается. Иначе в зависимости от наличия ключей t и p выполняются действия, заявленные в комментарии в начале процедуры.

Обработка прерываний в процедурах

Если при выполнении процедуры получен сигнал прерывания (от клавиши BREAK или DEL, например), то все созданные временные файлы останутся неудаленными (пока это не сделает администратор) ввиду немедленного прекращения процесса.

Лучшим решением является обработка прерываний внутри процедуры оператором ***trap***. Синтаксис:

trap 'command arguments' signals...

Кавычки формируют первый аргумент из нескольких команд, разделенных точкой с запятой. Они будут выполнены, если возникнет прерывание, указанное аргументами signals (целые):

– 2 — когда вы прерываете процесс;

– 1 — если вы "зависли" (отключены от системы) и др.

Пример (развитие предыдущего):

case $1 in

.....

) trap 'rm /tmp/; exit' 2 1 (удаление временных файлов)

if test -s $1

.....

```
rm /tmp/*
```

Лучше было бы:

```
trap 'rm /tmp/* > /dev/null; exit' 2 1
```

так как прерывание может случиться до того, как файл /tmp/$0$$ создан и аварийное сообщение об этом случае перенаправляется на null-устройство.

Выполнение арифметических операций: expr

Команда expr вычисляет значение выражения, поданного в качестве аргумента, и посылает результат на стандартный вывод. Наиболее интересным применением является выполнение операций над переменными языка Shell.

Пример суммирования 3 чисел:

```
$ cat sum3
expr $1 + $2 + $3
$ chmod 755 sum3
$ sum3 13 49 2
64
$
```

Пример непосредственного использования команды:

```
$ expr 13 + 49 + 2 + 64 + 1
129
$
```

В expr можно применять следующие арифметические операторы: +, -, *, /, % (остаток). Все операнды и операции должны быть разделены пробелами.

Заметим, что знак умножения следует заключать в кавычки (одинарные или двойные), например: '*', так как символ * имеет в Shell специальный смысл.

Более сложный пример expr в процедуре (фрагмент):

```
num = 'wc -l < $1'
tot = 100
count = $num
avint = 'expr $tot / $num'
avdec = 'expr $tot % $num'
while test $count -gt 0
do ...
```

Здесь wc -l осуществляет подсчет числа строк в файле, а далее это число используется в выражениях.

Отладка процедур Shell

Имеются три средства, позволяющие вести отладку процедур.

1. Размещение в теле процедуры команд echo для выдачи сообщений, являющихся трассой выполнения процедуры.

2. Опция -v (verbose = многословный) в команде Shell приводит к печати команды на экране перед ее выполнением.

3. Опция -x (execute) в команде Shell приводит к печати команды на экране по мере ее выполнения с заменой всех переменных их значениями; это наиболее мощное средство.

2 ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа по курсу «Вычислительные системы, сети телекоммуникации» состоит из двух частей: практическая часть и теоритическая часть.

В ***практической части*** студенты должны разработать командные файлы.

При разработке необходимо учесть возможность обработки различных ошибок ввода данных, например неправильного запуска ваших программ (с недостаточным количеством параметров или с неправильными параметрами), и предусмотреть вывод сообщения об ошибке и подсказки.

Программа должна быть реализована в виде скриптов Shell ОС Unix. Файлы должны быть самостоятельными, а не в тексте отчета (документе Microsoft Word). Каждая строчка командного файла должна сопровождаться подробными комментариями.

Теоретическая часть содержит ряд вопросов по учебному пособию, на которые студент должен дать развернутый и аргументированный ответ.

3 ВАРИАНТЫ ЗАДАНИЙ НА ВЫПОЛНЕНИЕ

Вариант 1

Практическая часть

Разработать командный файл, создающий, копирующий или удаляющий каталог, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Теоретическая часть

1. Какие режимы работы имеют микропроцессоры семейства x86-64 и что они собой представляют?

2. Что Вы знаете о Microsoft Active Directory?

Вариант 2

Практическая часть

Разработать командный файл, который бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования в текущем каталоге.

Теоретическая часть

1. Опишите существующие единицы информации и их представление.

2. В чем состоит преимущество использования доменов?

Вариант 3

Практическая часть

Разработать командный файл, который бы получал в качестве аргумента имя текстового файла и выводил на экран информацию о том, сколько символов, слов и строк в текстовом файле. Количество символов равно размеру файла.

Теоретическая часть

1. Дайте определения видам памяти и их месту в иерархии.

2. Приведите обзор семейства протоколов TCP/IP.

Вариант 4

Практическая часть

Разработать командный файл, копирующий произвольное число файлов, заданных аргументами из текущего каталога в указываемый каталог. Используйте проверку на пустые параметры и команду SHIFT.

Теоретическая часть

1. Что собой представляет адресация и распределение памяти в реальном режиме работы микропроцессора Intel x86?

2. Приведите примеры протоколов транспортного и прикладного уровней TCP/IP.

Вариант 5

Практическая часть

Разработать командный файл, который получал бы в качестве параметра какой-либо символ и в зависимости от второго параметра вырезал или сохранял в заданном файле все строки, начинающиеся на этот символ. Можно выполнить с помощью команды FOR (под ОС Windows).

Теоретическая часть

1. Что собой представляет адресация и распределение памяти в защищенном режиме работы микропроцессора Intel x86?

2. Как строится адресация TCP/IP четвертой версии?

Вариант 6

Практическая часть

Разработать командный файл (аналог команды tail в Unix). Командный файл печатает конец файла. По умолчанию — 10 последних строк. Явно можно задать номер строки, от которой печатать до конца. Если задание будет выполняться под ОС Unix, команду tail использовать нельзя.

Теоретическая часть

1. Что собой представляет адресация и распределение памяти в архитектуре AMD64?

2. Назовите номер сети и номер хоста, если у компьютера IP адрес 192.32.09.220 и маска подсети 255.255.255.0.

Вариант 7

Практическая часть

Разработать командный файл, который бы склеивал текстовые файлы, заданные в качестве аргументов, и сортировал бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Теоретическая часть

1. Как в ОС Windows на платформе NT можно узнать такие параметры, как общий размер физической памяти и размер памяти ядра ОС?

2. Как строится процесс разрешения DNS?

Вариант 8

Практическая часть

Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы измененные). Под ОС Windows можно воспользоваться анализом атрибутов файлов.

Теоретическая часть

1. Что такое файл подкачки и как им можно управлять в ОС Windows на платформе NT?

2. Какие утилиты диагностики сети Вы знаете?

Вариант 9

Практическая часть

Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные). Необходимо хранить список файлов в файле истории.

Теоритическая часть

1. Приведите классификацию устройств ввода-вывода.

2. Опишите протоколы HTTP и HTTPS.

Вариант 10

Практическая часть

Выполняющий в зависимости от ключа один из 3–х вариантов работы:

– с ключом /n дописывает в начало указанных текстовых файлов строку с именем текущего файла;

– с ключом /b создает резервные копии указанных файлов;

– с ключом /d удаляет указанные файлы после предупреждения.

Количество обрабатываемых файлов может быть переменным и задаваться в качестве параметров.

Теоретическая часть

1. Опишите основные характеристики устройств внешней памяти.

2. Опишите протоколы SMTP и POP3.