Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Томский государственный университет систем управления и радиоэлектроники»

Кафедра электронных приборов

ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ И КОМПЬЮТЕРНЫЕ СЕТИ

Методические указания к лабораторным работам
для студентов направления «Электроника и микроэлектроника»
(специальность «Электронные приборы и устройства»

2012

Шандаров Евгений Станиславович
Лысенко Иван Владимирович

Персональные компьютеры и компьютерные сети: методические указания к лабораторным работам для студентов направлений «Электроника и микроэлектроника» (специальность «Электронные приборы и устройства»/ Е.С. Шандаров, И.В. Лысенко; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Томский государственный университет систем управления и радиоэлектроники, Кафедра электронных приборов. - Томск: ТУСУР, 2012. - 89 с.

Данный курс лабораторных работ посвящен дисциплине «Персональные компьютерные средства» и включает в себя описание 16 лабораторных работ.

Предназначено для студентов очной и заочной форм, обучающихся по направлению «Электроника и микроэлектроника» (специальность «Электронные приборы и устройства») по дисциплине «Персональные компьютерные средства».

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Томский государственный университет систем управления и радиоэлектроники»

Кафедра электронных приборов

УТВЕРЖДАЮ
Зав.кафедрой ЭП
_____С.М. Шандаров
«___» _____ 2012 г.

ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ И КОМПЬЮТЕРНЫЕ СЕТИ

Методические указания к лабораторным работам
для студентов направления «Электроника и микроэлектроника»
(специальность «Электронные приборы и устройства»

Разработчик
ст.преподаватель каф.ЭП
_____Е.С. Шандаров
«____»_____2012 г

ассистент каф.ЭП
_____И.В. Лысенко
«____»_____2012 г

2012

Содержание

Лабораторная работа №1. Работа с файловой системой Linux с помощью консоли

1.1 Цель работы

Целью настоящей лабораторной работы является изучение базовых возможностей ОС Linux.

1.2 Теоретическая часть

1.2.1 Интерфейс командной строки

Интерфейс командной строки — разновидность текстового интерфейса человека и компьютера, в котором инструкции компьютеру даются только путём ввода с клавиатуры текстовых строк (команд). Также известен под названием консоль.

Интерфейс командной строки противопоставляется системам управления программой на основе меню, а также различным реализациям графического интерфейса.

Формат вывода информации в интерфейсе командной строки не регламентируется; обычно это также простой текстовый вывод, но может быть и графическим, звуковым и т.д.

1.2.2 Назначение

На устройстве-консоли, которое печатало текст на бумаге, интерфейс командной строки был единственным возможным. На видеотерминалах интерфейс командной строки применяется по таким причинам:

– небольшой расход памяти по сравнению с системой меню;

– в современном программном обеспечении имеется большое число команд, многие из которых нужны крайне редко. Поэтому даже в некоторых программах с графическим интерфейсом применяется командная строка: набор команды (при условии, что пользователь знает эту команду) осуществляется гораздо быстрее, чем, например, навигация по меню;

– естественное расширение интерфейса командной строки — пакетный интерфейс. Его суть в том, что в файл обычного текстового формата записывается последовательность команд, после чего этот файл можно выполнить в программе, что возымеет такой же (не меньший) эффект, как если бы эти команды были по очереди введены в командную строку.

Если программа полностью или почти полностью может управляться командами интерфейса командной строки и поддерживает пакетный интерфейс, умелое сочетание интерфейса командной строки с графическим предоставляет пользователю очень мощные возможности.

1.2.3 Формат команды

Наиболее общий формат команд (в квадратные скобки помещены необязательные части):

```
[символ_начала_команды]имя_команды [параметр_1 [параметр_2 [...]]]
```

Символ начала команды может быть самым разным, однако чаще всего для этой цели используется косая черта (/). Если строка вводится без этого символа, выполняется некоторая базовая команда: например, строка «Привет» в IRC эквивалентна вводу «/msg Привет». Если же такой базовой команды нет, символ начала команды отсутствует вообще (как, например, в DOS).

Параметры команд могут иметь самый разный формат. В основном применяются следующие правила:

– параметры разделяются пробелами (и отделяются от названия команды пробелом);

– параметры, содержащие пробелы, обрамляются кавычками-апострофами (') или двойными кавычками (");

– если параметр используется для обозначения включения какой-либо опции, выключенной по умолчанию, он начинается с косой черты (/) или дефиса (-);

– если параметр используется для включения/выключения какой-либо опции, он начинается (или заканчивается) знаком плюс или минус (для включения и выключения, соответственно);

– если параметр указывает действие из группы действий, назначенных команде, он не начинается со специальных символов;

– если параметр указывает объект, к которому применяется действие команды, он не начинается со специальных символов;

– если параметр указывает дополнительный параметр какой-либо опции, то он имеет формат /опция:дополнительный_параметр (вместо косой черты также может употребляться дефис).

1.2.4 Достоинства и недостатки

Достоинства:

– любую команду можно вызвать небольшим количеством нажатий;

– пакетные файлы — это, по сути, простейшая программируемость;

– можно управлять программами, не имеющими графического интерфейса (например, выделенным сервером);

– просмотрев содержимое консоли, можно повторно увидеть промелькнувшее сообщение, которое вы не успели прочитать.

Недостатки:

– интерфейс командной строки не является дружественным для начинающих;

– искать неизвестную команду по справочникам не менее сложно, чем отыскивать в меню нужную команду;

– ввод некоторых параметров с клавиатуры может быть затруднительным.

Несколько простейших команд Unix

Общие замечания:

– система Unix различает БОЛЬШИЕ и малые буквы;

– если вы уничтожили какой-нибудь файл, то никаких проблем с его восста-

новлением у вас не будет. Потому что восстановить уничтоженный файл в системе Unix НЕВОЗМОЖНО. В Unix отсутствует команда unerase, и к этому тоже нужно привыкнуть. Вирусы и антивирусы в Unix тоже отсутствуют;

– В имени файла директории отделяются от последующей части символом "/". Если имя начинается со слэша - значит, это полное маршрутное имя.

– Простое имя файла может состоять из ЛЮБЫХ символов. Длина простого имени не более 256 символов. Длина полного маршрутного имени файла не более 32000 символов.

– Для задания шаблона имен используются символы "*" (произвольная последовательность символов) и "?" (один произвольный символ). Имя из одной точки "." обозначает текущую директорию, имя из двух точек ".." - вышележащую (родительский каталог).

– Чтобы запустить программу на выполнение, достаточно набрать ее имя и, если нужно, другие аргументы командной строки. Имя программы - это маршрутное имя файла, в котором эта программа находится. Аргументы разделяются одним или несколькими пробелами и табуляторами. Ключи команды обычно (но не всегда) выделяются знаком "-" (`команда -ключи -ключи ...`).

– Если командная строка кончается знаком &, то команда запустится параллельно (фоном). На терминале печатается номер, который получает запущенный процесс. После чего можно продолжать работу, не дожидаясь завершения фоновой задачи.

```
команда -всякие разные аргументы ... &
```

– Команда имеет три предопределенных направления ввода-вывода - стандартный ввод, стандартный вывод и стандартный протокол. Как правило, команда берет исходные данные из стандартного ввода и печатает результаты в стандартный вывод. В стандартный протокол печатаются сообщения об ошибках и диагностика. Первоначально стандартные ввод, вывод и протокол назначены на терминал, однако их можно переназначить, используя следующие конструкции:

```
команда > имя_файла
```
(для стандартного вывода),
```
команда < имя_файла
```
(для переназначения стандартного ввода),
```
команда 2> имя_файла
```
(для стандартного протокола - системной диагностики).

Пример - записать в файл содержимое текущей директории:

```
ls > infdir
```

Если вывод назначен в файл, то перед началом выполнения команды создается пустой файл с соответствующим именем (если файл уже существовал, он опустошается), а затем в него помещается информация. Если информацию нужно дописать в конец существующего файла, следует воспользоваться конструкцией

```
команда >> имя файла
команда 2>> имя файла
```

– Пользователи системы Unix объединяются в группы, и каждая из групп обладает определенным набором прав доступа к файлам.

– Программы могут распознавать значения переменных окружения, значения которых были заданы перед их запуском.

```
TERM=vt220
HOME=/home/lysenko
PATH=:/bin/:dss/rk:/home/lysenko/bin
```

– Выдачу одной команды можно перенаправлять на вход другой команды. Для этого используется "|" - конвейер.

Пример: посчитать суммарный размер и число строк всех файлов в каталоге /tmp, имена которых начинаются с символов "text" ...

```
cat /tmp/text* | wc -c
```

Команда cat сливает вместе все файлы вида /tmp/text*,
wc -c -l считает количество символов и число строк во входном
потоке.

Или просмотреть с листанием длинную выдачу команды:

```
ps -ef | more
```

Того же результата можно достичь и без конвейера таким образом:

```
ps -ef > temp-file
more temp-file
rm temp-file
```

Некоторые наиболее употребительные команды.

`pwd` - получить имя текущей директории
`cd` - изменить текущую директорию
`ls` - распечатать информацию о файлах или директориях
`cat` - слить или вывести файлы-аргументы на стандартный вывод
`cp` - копировать файлы
`mv` - переместить (переименовать) файлы
`ln` - создать ссылку на файл
`rm` - удалить файлы
`rmdir` - удалить директорию
`mkdir` - создать директорию
`echo` - вывести аргументы командной строки на стандартный вывод
`ps` - распечатать информацию и номера выполняемых процессов
`kill` - "убить" выполняемый процес по его номеру
`man` - выдать справку об использовании и ключах команды

Примеры использования команд

pwd Выдать имя текущей директории.

```
pwd
```

cd Сменить текущую директорию.

```
cd [ директория]
```

```
cd /usr/spool/lp/adm
cd ..
```

Если директория не указана, вы попадаете в свой "домашний" каталог $HOME

`ls` Распечатать каталог.

Формат команды: `ls [ключи] [имена]`

`имена` - имена файлов или директорий; для директорий распечатывается список входящих в нее файлов, для файлов - выводится его имя и дополнительная информация. Имена файлов сортируются по алфавиту. Без аргументов `ls` выдает содержимое текущей директории.

`ls -al` - вывод в длинном полном формате:

```
-rwxr-xr-x       1 lysenko sys             17 Oct 18 16:13 ../jean
drwxr-xr-x       1 lysenko sys             12 Oct 18 16:11 ../jelly
-rwxr-xr-x       1 lysenko sys              0 Oct 13 14:14 ../j.bu.txt
```

Права доступа:

- r - файл доступен для чтения,
- w - файл доступен для записи,
- x - файл является выполнимым,
- - - данное право доступа отсутствует.

Ключи команды:

`-a` - вывести все файлы (даже если имена начинаются с точки);

`-x` - вывод в 4 колонки;

`-t` - имена файлов сортируются не по алфавиту, а по времени последнего изменения

-R - рекурсивно пройти по всем подкаталогам

-CF - оглавление каталога в несколько столбиков

-al - оглавление в полном формате

`cat` Слить и вывести файлы на стандартный вывод.

Формат команды:

```
cat файл1 [файл2... ]
```

Cat выводит содержимое перечисленных файлов на стандартный вывод.

Пример:

```
cat файл
```

распечатывает содержимое файла, а

```
cat файл1 файл2 > файл3
```

сливает первые два файла и помещает результат в третий. Чтобы добавить содержимое файла1 к файлу2, надо выполнить команду

```
cat файл1 >> файл2
```

`more, pg` Просмотреть файл постранично.

```
more file_name ...
pg     file_name ...
less file_name ...
```

Все эти команды позволяют просматривать файл, листая его поэкранно. Просмотром можно управлять с помощью клавиш. Самое удобное управление у команды "less":

q - завершить просмотр
ПРОБЕЛ - показать следующую страницу
ENTER - сдвинуться на одну строку
b - показать предыдущую страницу
/ - поиск
h - Help - посмотреть список всех возможных команд

more,less,pg используют, чтобы посмотреть "длинную" выдачу команды, ставя его конвейером "ей на хвост"

```
ls -al | more    - просмотреть оглавление
ps -e | pg      - просмотреть список работающих процессов
```

cp Копировать файлы.

Формат команды:

```
cp файл1 файл2
cp файл1 [файл2 ...] каталог
```

Эта команда копирует файл1, ... , в ФАЙЛ. Если ФАЙЛ – это директория, то файл1 и др. копируются в нее под своими именами. Если ФАЙЛ не существовал, то он создается, если существовал, его старое содержимое теряется.

mv Переместить (переименовать) файлы.

Формат команды:

```
mv файл1 файл2
mv файл1 [файл2 ...] каталог
```

Команда аналогична команде cp, но исходный файл уничтожается. Ее основная роль - переименование файлов и перенос файлов из одной директории в другую). Пример:

```
mv /dss/*/rk_*.help /dss/delo
```

ln Связать файлы.

Формат команды:

```
ln файл1 файл2
# создается "жесткий" линк
ln -s файл1 файл2
# создается "символический" линк
```

Эта команда создает файлу файл1 еще одно имя. В итоге файл1 и файл2 на самом деле физически являются одним и тем же файлом. Если вы создаете так называемый "символический линк" (используя ключ "-s"), то файл file1 при этом не обязан существовать. Имя "файл2" при этом все равно будет создано.

rm Удалить файл или директорию.

Формат команды:

```
rm [-fri] файл ...
```

-i - просить подтверждения на каждое удаление
-r - рекурсивно удалить вместе с подкаталогами

-f - не просить подтверждения, а сразу удалять

Для удаления пустой директории можно воспользоваться командой

```
rmdir директория ...
```

Чтоб удалить непустую директорию, нужно выполнить команду

```
rm -r директория
```

ВНИМАНИЕ: Ввиду того, что под шаблон ".*" подходит каталог ".." НИКОГДА НЕ ДЕЛАЙТЕ КОМАНДЫ `# rm -r .*`

mkdir Создать директорию.

Формат команды:

```
mkdir    имя_директории ...
```

chmod Изменение прав доступа к файлам.

Команда chmod меняет атрибуты (права доступа) файла.

Проставить файлам право на выполнение:

```
chmod u+x file1
```

Разрешить остальным пользователям исправлять файл

```
chmod a+w file1
```

echo Эхо.

Команда выводит на стандартный вывод свои аргументы

```
echo "\017"
```

Вывести на терминал символ Ctrl-O, он же 017 восьмеричное.

ps Какие программы выполняются.

Сообщает номера процессов, которые выполняются в системе.

`ps` - запущенные только с этого терминала

`ps -ef` - все, в "полном" формате (в SYSV Unix)

`ps -ax` - все, в "полном" формате (в BSD* и Linux)

kill Прервать процесс.

Команда kill посылает указанному процессу сигнал немедленной остановки.

Прервать процесс номер 1078 (номер можно узнать командой ps -e)

```
kill -9 1078
```

man Если вы забыли, как пользоваться командой.

Как правило, при запуске команды без аргументов она выдает формат своего вызова. Но для большинства команд достаточно полную информацию вы можете получить так:

```
man имя_интересующей_вас_команды
man -k ключевое_слово     # список команд, относящихся к ...
```

Для того, чтобы срабатывала команда man -k, файлы документации должны быть предварительно проиндексированы. Достигается это командой

```
catman -w
```

who Кто работает в системе.

`who` - этими командами можно узнать, кто вы такой
`finger`
man -k who - сообщит все возможные команды, которыми можно узнать, кто и что делает в системе.

Как создать файл

Для создания файла можно воспользоваться следующими способами:

```
touch primer       #  создает пустой файл primer;
cat > primer       #  создает файл primer и пишет в него
#  со стандартного ввода. Запись в файл закончится
#  после нажатия CTRL+D
```

1.3 Задание на лабораторную работу

Выполнить набор операций согласно выданному варианту задания. Вариант определяется преподавателем. Все операции ведутся в домашнем каталоге.

По результатам работы оформить отчет.

Вариант 1

1. Создать папки da и db
2. Создать файл fa с любым содержимым
3. Просмотреть файл fa
4. Копировать файл fa в папку da присвоив ему новое имя fb
5. Удалить файл fa
6. Переместить файл fb в папку db
7. Выставить права для файла fb (-rwxr-xr-x)
8. Удалить файл fb
9. Удалить папки da и db
10. Вывести полную информацию о всех файлах вашего домашнего каталога

Вариант 2

1. Создать файл fa и fb с любым содержимым
2. Выставить права для файла fa (-rwxrwxr--)
3. Выставить права для файла fb (-rwxrwxrwx)
4. Создать папки da и db
5. Копировать файл fa в папку da
6. Переместить файл fb в папку db
7. Вывести содержимое папки da
8. Вывести содержимое папки db
9. Получить информацию по команде ls
10. Удалить папку db

Вариант 3

1. Создать папку da
2. Создать файлы fa с любым содержимым
3. Выставить права для файла fa (-rwx------)
4. Создать в папке da папки db и dc
5. Копировать файл fa в папки db и dc
6. Создать в папке dc файл fb с любым содержанием
7. Получить информацию по команде cp
8. Переместить файл fb в папку db
9. Удалить папку da
10. Вывести полную информацию о всех файлах вашего домашнего каталога

Вариант 4

1. Создать файлы fa с любым содержимым
2. Изменить файл fa с помощью vi и сохранить результат
3. Создать папку da
4. Переместить файл fa в папку da
5. Выставить права для файла fa (-r--r--r--)
6. Вывести содержимое файла fa на экран
7. Получить информацию по команде rm
8. Создать в папке da файл fb с любым содержимым
9. Вывести полную информацию о всех файлах папки da
10. Удалить папку da

Вариант 5

1. Создать папки da и db
2. Выставить права для папки da (-rwx--xr-x)
3. Выставить права для папки db (-rwxr-xr-x)
4. Создать файл fa с любым содержимым
5. Изменить файл fa с помощью vi не сохраняя результат
6. Копировать файл fa в da
7. Копировать файл fa в db
8. Получить информацию по команде mkdir
9. Создать в папке da папку dc
10. Вывести содержимое папки da

Вариант 6

1. Создать файл fa с помощью vi с любым содержимым
2. Создать папки da и db
3. Копировать файл fa в da присвоив ему новое имя fb
4. Вывести содержимое файла fa
5. Удалить файл fa
6. Получить информацию по текстовому редактору vi

7. Создать файл fc в папке db
8. Выставить права для файла fc (-r-xr-xr-x)
9. Вывести полную информацию о всех файлах вашего домашнего каталога
10. Удалить папку da

Вариант 7

1. Создать файлы fa и fb с любым содержимым
2. Выставить права для обоих файлов (-rwxrwxrwx)
3. Вывести содержимое файла fa
4. Получить информацию по команде man
5. Создать папку da
6. Выставить права для папки da (-rwxrwx---)
7. Копировать файл fa в da
8. Вывести содержимое папки da
9. Удалить папку da
10. Показать с помощью mc размер домашнего каталога

Вариант 8

1. Создать папки da и db
2. Создать файл fa с помощью vi
3. Копировать файл fa в da присвоив ему новое имя fb
4. Вывести содержимое fa
5. Удалить fa
6. Создать папку dc в db
7. Переместить файл fb в папку dc
8. Удалить папку db
9. Вывести полную информацию о всех файлах вашего домашнего каталога
10. Получить информацию по команде less

Вариант 9

1. Создать файлы fa, fb и fc с любым содержимым
2. Создать папку da
3. Создать папку db в папке da
4. Переместить файлы fa, fb, fc в каталог db
5. Выставить права для файлов fa, fb, fc (-rwxrwxr-x)
6. Удалить файл fc
7. Получить информацию по команде chmod
8. Вывести содержимое файла fb
9. Удалить папку da
10. Поменять свой пароль для входа в систему (Достаточно рассказать о команде и объяснить принцип работы)

Лабораторная работа №2. Разработка интерфейса командной строки.

2.1 Цель работы

Цель работы: разработать на любом языке программирования интерфейс командной строки.

2.2 Теоретическая часть

2.2.1 Создание shell-скриптов в Linux

Традиционный скрипт "hello world"

```
#!/bin/bash
echo Hello World!
```

Данный скрипт содержит только две строки. Первая сообщает системе о том, какая программа используется для запуска файла. Вторая строка - это единственное действие, выполняемое данным скриптом, печатающее 'Hello world' на терминале. Если Вы получите что-то типа ./hello.sh: Command not found. , то, возможно, первая строка '#!/bin/bash' неправильная; запустите whereis bash или посмотрите finding bash, чтобы выяснить, какой должна быть эта строка.

Простой скрипт резервного копирования

```
#!/bin/bash
tar -cZf /var/my-backup.tgz /home/me/
```

В данном скрипте вместо печати сообщения на терминале мы создаём tar-архив пользовательского домашнего каталога. Скрипт НЕ предназначен для практического применения. Далее в данном документе будет представлен более эффективный скрипт резервного копирования.

2.2.2 Всё о перенаправлении

Теория и быстрый просмотр

Существуют 3 файловых дескриптора: `stdin` - стандартный ввод, `stdout` – стандартный вывод и `stderr` - стандартный поток ошибок. Ваши основные возможности:

1. перенаправлять stdout в файл
2. перенаправлять stderr в файл
3. перенаправлять stdout в stderr
4. перенаправлять stderr в stdout
5. перенаправлять stderr и stdout в файл
6. перенаправлять stderr и stdout в stdout
7. перенаправлять stderr и stdout в stderr

Это действие записывает стандартный вывод программы в файл.

```
ls -l > ls-l.txt
```

Здесь создаётся файл с именем 'ls-l.txt'. В нём будет содержаться всё то, что Вы бы увидели, если бы просто выполнили команду 'ls -l'.

Это действие записывает стандартный поток ошибок программы в файл.

```
grep da * 2> grep-errors.txt
```

Здесь создаётся файл, названный 'grep-errors.txt'. В нём будет содержаться часть вывода команды 'grep da *', относящаяся к стандартному потоку ошибок.

Это действие записывает стандартный вывод программы в тот же файл, что и стандартный поток ошибок.

```
grep da * 1>&2
```

Здесь стандартный вывод команды отправляется в стандартный поток ошибок. Вы можете увидеть это разными способами.

Это действие записывает стандартный поток ошибок программы туда же, куда и стандартный вывод.

```
grep * 2>&1
```

Здесь стандартный поток ошибок команды отправляется на стандартный вывод; если Вы перешлёте результат через конвейер (|) в less, то увидите, что строки, которые обычно пропадают (как записанные в стандартный поток ошибок), в этом случае сохраняются (так как они находятся на стандартном выводе).

Это действие помещает весь вывод программы в файл. Это является подходящим вариантом для заданий cron: если Вы хотите, чтобы команда выполнялась абсолютно незаметно.

```
rm -f $(find / -name core) &> /dev/null
```

Это (предположим, для cron) удаляет любой файл с названием 'core' в любом каталоге. Помните, что Вам следует полностью быть уверенным в том, что выполняет команда, если возникает желание затереть её вывод.

2.2.3 Конвейеры

Данный раздел объясняет достаточно простым и практичным способом, каким образом следует использовать конвейеры и для чего Вам это может потребоваться.

Конвейеры предоставляют Вам возможность использовать вывод одной программы в качестве входа другой.

Это очень простой способ использования конвейеров.

```
ls -l | sed -e "s/[aeio]/u/g"
```

Здесь происходит следующее: первоначально выполняется команда ls -l, и её вывод, вместо отображения на экране, отправляется в программу sed, которая, в свою очередь, выводит на экран то, что должна.

2.2.4 Переменные

Вы можете использовать переменные таким же образом, что и в любом языке программирования. Типы данных отсутствуют. Переменная в bash может представлять собой число, символ или строку символов. Вам не следует объявлять переменную. В действительности, присвоение значения на её указатель уже создаёт её.

```
#!/bin/bash
STR="Hello World!"
```

```
echo $STR
```

Вторая строка создаёт переменную, которая называется STR, и присваивает ей строчное значение "Hello World!". Затем ЗНАЧЕНИЕ этой переменной извлекается добавлением в начале знака '$'. Пожалуйста, запомните (постарайтесь), что если Вы не используете знак '$', вывод программы может быть другим. Вероятно, не таким, который Вам требуется.

Очень простой скрипт резервного копирования (более эффективный)

```
#!/bin/bash
OF=/var/my-backup-$(date +%Y%m%d).tgz     #OF - Output File - выходной файл
tar -cZf $OF /home/me/
```

Данный скрипт вводит ещё одно понятие. Прежде всего, Вам следует разобраться со второй строкой. Обратите внимание на выражение '$(date +%Y%m%d)'. Если Вы запустите этот скрипт, то заметите, что он выполняет команду внутри скобок, перехватывая её вывод. Обратите внимание, что в этом скрипте имя выходного файла будет ежедневно изменяться, исходя из формата ключа к команде date (+%Y%m%d). Вы можете поменять это заданием другого формата.

Локальные переменные

Локальные переменные могут быть созданы при использовании ключевого слова `local`.

```
#!/bin/bash
HELLO=Hello
function hello {
     local HELLO=World
     echo $HELLO
}
echo $HELLO
hello
echo $HELLO
```

Данного примера должно быть достаточно для отображения способов использования локальных переменных.

2.2.5 Условные операторы

Условные операторы предоставляют Вам возможность решить, выполнять действие или нет; решение принимается при вычислении значения выражения. Существует большое количество форм условных операторов. Элементарная форма - это if выражение then оператор, где 'оператор' выполняется только в том случае, если 'выражение' имеет значение "истина". '2<1' - это выражение, имеющее значение "ложь", в то время как '2>1' – "истина". Существуют другие формы условных операторов, такие как: if выражение then оператор1 else оператор2. Здесь 'оператор1' выполняется, если 'выражение'- истина; в противном случае, выполняется 'оператор2'. Ещё одной формой условных операторов является: if выражение1 then оператор1 else if выражение2 then оператор2 else оператор3. В данной форме добавляется только последовательность "ELSE IF 'выражение2' THEN 'оператор2'", заставляющая 'оператор2' выполняться, если 'выражение2' имеет значение "истина". Всё остальное соответствует Вашему представлению об этом (см. предыдущие формы).

Элементарная конструкция оператора 'if' в bash выглядит следующим образом:

```
if [выражение];
then
     code if 'выражение' is true.
fi
```

Элементарный образец условного оператора if .. then

```
#!/bin/bash
if [ "foo" = "foo" ]; then
     echo-выражение вычислилось как истина
fi
```

Если выражением внутри квадратных скобок является истина, то выполняемый код находится после слова 'then' и перед словом 'fi', которое обозначает конец исполняемого при выполнении условия кода.

Элементарный пример условного оператора if .. then ... else

```
#!/bin/bash
if [ "foo" = "foo" ]; then
     echo-выражение вычислилось как истина
else
     echo-выражение вычислилось как ложь
fi
```

Условные операторы с переменными

```
#!/bin/bash
T1="foo"
T2="bar"
if [ "$T1" = "$T2" ]; then
     echo-выражение вычислилось как истина
else
     echo-выражение вычислилось как ложь
fi
```

2.2.6 Циклы for, while и until

Цикл for немного отличается от аналогов в других языках программирования. Прежде всего, он предоставляет Вам возможность выполнять последовательные действия над "словами" в строке.

Цикл while выполняет кусок кода, если тестируемым выражением является истина; и останавливается при условии, если им является ложь (или внутри исполняемого кода встречается явно заданное прерывание цикла).

Цикл until практически идентичен циклу while. Отличие заключается только в том, что код выполняется при условии, если проверяемым выражением является ложь.

Пример цикла for

```
#!/bin/bash
for i in $( ls ); do
     echo item: $i
done
```

Во второй строке мы представляем i в качестве переменной, которая получает различные значения, содержащиеся в $(ls). При необходимости третья строка могла бы быть длиннее; или там могло бы находиться несколько строк перед done (4-я строка) показывает, что код, в котором используется значение $i, заканчивается и $i получает новое значение. Данный скрипт не предполагает большой важности. Более полезным применением цикла for было бы использование его для отбора только каких-то определённых файлов в предыдущем примере.

Цикл for, наиболее похожий на for в языках C, Perl и т.п.

```
#!/bin/bash
for i in `seq 1 10`;
do
     echo $i
done
```

Пример цикла while

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
     echo The counter is $COUNTER
     let COUNTER=COUNTER+1
done
```

Данный скрипт "эмулирует" широко известную (в языках C, Pascal, perl и т.д.) структуру 'for'.

Пример цикла until

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]; do
     echo COUNTER $COUNTER
     let COUNTER-=1
done
```

2.2.7 Функции

Аналогично любому другому языку программирования, Вы можете использовать функции для группировки кусков кода более логичным способом, а также для практического применения волшебного искусства рекурсии.

Объявление функции - это только лишь запись function my_func { my_code }. Вызов функции осуществляется аналогичным образом, что и вызов других программ. Вы просто пишете её имя.

Пример функций

```
#!/bin/bash
function quit {
     exit
}
function hello {
     echo Hello!
```

```
}
hello
quit
```

Следует заметить, что совсем необязательно объявлять функции в каком-то определённом порядке.

Пример функций с параметрами

```
#!/bin/bash
function quit {
     exit
}
function e {
     echo $1
}
e Hello
e World
quit
```

Данный скрипт практически идентичен предыдущему. Главное отличие - это функция 'e'. Она выводит самый первый получаемый аргумент. Аргументы в функциях обрабатываются таким же образом, что и аргументы, переданные скрипту.

2.2.8 Интерфейсы пользователя

Использование select для создания простых меню

```
#!/bin/bash
OPTIONS="Hello Quit"
select opt in $OPTIONS; do
        if [ "$opt" = "Quit" ]; then
                echo done
                exit
        elif [ "$opt" = "Hello" ]; then
                 echo Hello World
        else
                clear
                echo bad option
        fi
done
```

Если Вы запустите этот скрипт, то увидите, что он является мечтой программиста о меню на текстовой основе. Вероятно, Вы заметите, что это очень похоже на конструкцию 'for', только вместо циклической обработки каждого "слова" в $OPTIONS программа опрашивает пользователя.

Использование командной строки

```
#!/bin/bash
if [ -z "$1" ]; then
        echo используйте: $0 каталог
        exit
```

```
fi
SRCD=$1     #SRCD - SouRCe Directory - исходный каталог
TGTD="/var/backups/"      #TGTD - TarGeT Directory – конечный каталог
OF=home-$(date +%Y%m%d).tgz          #OF - Output File - выходной файл
tar -cZf $TGTD$OF $SRCD
```

Вам должно быть понятно, что выполняет этот скрипт. Выражение в первом условном операторе проверяет, получила ли программа аргумент ($1). Если – нет, оно завершает работу программы, предоставляя пользователю небольшое сообщение об ошибке. Оставшаяся на данный момент часть скрипта, очевидно, является понятной.

2.2.9 Разное

Чтение пользовательского ввода с помощью read

В некоторых случаях, возможно, возникнет необходимость попросить пользователя что-нибудь ввести. Существуют различные способы выполнения этого. Одним из способов является следующий:

```
#!/bin/bash
echo Введите, пожалуйста, Ваше имя
read NAME
echo "Привет, $NAME!"
```

В качестве варианта Вы можете получать сразу несколько значений с помощью read. Следующий пример поясняет это:

```
#!/bin/bash
echo "Введите, пожалуйста, Ваше имя и фамилию"
read FN LN     #FN - First Name - имя; LN - Last Name - фамилия
echo "Hi! $LN, $FN !"
```

Арифметические вычисления

В командной строке (или оболочке) попробуйте ввести следующее:

```
echo 1 + 1
```

Если Вы рассчитываете увидеть '2', то будете разочарованы. Что следует выполнить, если возникает необходимость, чтобы BASH произвёл вычисления над Вашими числами? Решение заключается в следующем:

```
echo $((1+1))
```

В результате этого вывод будет более "логичным". Такая запись используется для вычисления арифметических выражений. Вы также можете выполнить это следующим образом:

```
echo $[1+1]
```

Если Вам необходимо использовать дроби или более сложную математику, то можно использовать bc для вычисления арифметических выражений.

Когда автор запустил "echo $[3/4]" в командной оболочке, она вернула значение 0. Это связано с тем, что если bash отвечает, он использует только целые значения. Если Вы запустите "echo 3/4|bc -l", оболочка вернёт правильное значение 0.75.

Операторы сравнения строк

s1 = s2	s1 совпадает с s2
s1 != s2	s1 не совпадает с s2
s1 < s2	s1 в алфавитном порядке предшествует s2 (в соответствии с текущей локалью)
s1 > s2	s1 в алфавитном порядке следует после s2 (в соответствии с текущей локалью)
-n s1	s1 имеет ненулевое значение (содержит один символ или более)
-z s1	s1 имеет нулевое значение

Примеры сравнения строк

Сравнение двух строк.

```
#!/bin/bash
S1='string'
S2='String'
if [ $S1=$S2  ];
then
        echo "S1('$S1') не равна to S2('$S2')"
fi
if [ $S1=$S1  ];
then
        echo "S1('$S1') равна to S1('$S1')"
fi
```

Арифметические операторы

+
-
*
/
% (remainder)

Арифметические операторы сравнения

-lt (<)
-gt (>)
-le (<=)
-ge (>=)
-eq (==)
-ne (!=)

Программистам на C необходимо просто выбрать оператор, соответствующий выбранному оператору в скобках.

Полезные команды

Некоторые из этих команд практически содержат полноценные командные языки. Здесь объясняются только основы таких команд. Для более подробной информации внимательно просмотрите man-страницы каждой команды.

awk (манипулирование файлами данных, выборка и обработка текста)

Существует большое количество реализаций языка программирования AWK (наиболее распространенными интерпретаторами являются gawk из проекта GNU и "новый awk" mawk.) Принцип достаточно прост: AWK находится в поиске шаблона; для каждого подходящего шаблона выполняется какое-нибудь действие.

Автор повторно создал файл dummy, содержащий следующие строки:
"test123
test
tteesstt"

```
$awk '/test/ {print}' /tmp/dummy
test123
test
```

Шаблон, искомый AWK, это 'test', а действие, выполняемое AWK при обнаружении строки в /tmp/dummy с подстрокой 'test', это 'print'.

```
$awk '/test/ {i=i+1} END {print i}' /tmp/dummy
3
```

Если Вы находитесь в поиске нескольких шаблонов, замените текст между кавычками на '-f file.awk'. В этом случае, Вы можете записать все шаблоны и действия в файле 'file.awk'.

grep (выводит строки, соответствующие искомому шаблону)

Мы рассматривали несколько команд grep в предыдущих главах, которые отображали строки, соответствующие шаблону. Однако grep способен выполнять значительно большее.

```
$grep "look for this" /var/log/messages -c
12
```

Строка "look for this" была обнаружена 12 раз в файле /var/log/messages.
[ok, данный пример был фикцией, /var/log/messages был переделан :-)]

wc (считает строки, слова и байты)

В следующем примере можно заметить, что выводится не то, что мы ожидаем. В этом случае, файл dummy содержит следующий текст:
"bash introduction
howto test file"

```
$wc --words --lines --bytes /tmp/dummy
2 5 34 /tmp/dummy
```

wc не заботится о порядке параметров. Он всегда выводит их в стандартном порядке: <число строк><число слов><число байтов><имя файла>.

sort (сортирует строки текстового файла)

В этом случае, файл dummy содержит следующий текст:
"b
c

a"
$sort /tmp/dummy

Вывод выглядит следующим образом:
a
b
c
Команды не должны быть такими простыми :-)
bc (вычислительный язык программирования)
bc производит вычисления с командной строки (ввод из файла, но не через перенаправление или конвейер), а также из пользовательского интерфейса. Следующий пример показывает некоторые команды. Обратите внимание, что автор использовал bc с параметром -q, чтобы отказаться от вывода сообщения с приглашением.

```
$bc -q
1 == 5
0
0.05 == 0.05
1
5 != 5
0
2^8
256
sqrt(9)
3
while (i != 9) {
i = i + 1;
print i
}
123456789
quit
```

tput (инициализирует терминал или запрашивает базу данных terminfo)
Небольшая иллюстрация возможностей tput:
$tput cup 10 4
Приглашение командной строки появится в координатах (y10,x4).
$tput reset
Экран очищается и приглашение появляется в (y1,x1). Обратите внимание, что (y0,x0) - это левый верхний угол.
$tput cols

80
Отображает возможное количество символов в направлении по оси x.
Настоятельно рекомендуется быть с этими программами на "ты" (как минимум). Существует огромное количество небольших программ, которые предоставляют Вам возможность заняться настоящей магией в командной строке.
[Некоторые примеры были заимствованы из man-страниц или FAQ.]

12. Ещё скрипты
12.1 Применение команды ко всем файлам в каталоге.
12.2 Пример: очень простой скрипт резервного
копирования (более эффективный)

```
#!/bin/bash
SRCD="/home/"               #SRCD - SouRCe Directory - исходный
каталог
TGTD="/var/backups/"        #TGTD - TarGeT Directory - конечный
каталог
OF=home-$(date +%Y%m%d).tgz  #OF - Output File - выходной файл
tar -cZf $TGTD$OF $SRCD
```

12.3 Программа переименования файлов

```
#!/bin/sh
# renna: переименование нескольких файлов по специальным правилам
# Автор - felix hudson Jan - 2000
#Прежде всего, посмотрите на различные "режимы", которые имеются у
этой программы.
#Если первый аргумент ($1) является подходящим, мы выполняем эту часть
#программы и выходим.
# Проверка на возможность добавления префикса.
if [ $1 = p ]; then
#Теперь переходим от переменной режима ($1) и префикса ($2)
prefix=$2 ; shift ; shift
# Необходимо проверить, задан ли, по крайней мере, хотя бы один файл.
# В противном случае, лучше ничего не предпринимать, чем
переименовывать несуществующие
# файлы!!
  if [$1 = ]; then
     echo "не задано ни одного файла"
     exit 0
  fi
# Этот цикл for обрабатывает все файлы, которые мы задали
# программе.
# Он осуществляет одно переименование на файл.
   for file in $*

   do
   mv ${file} $prefix$file
 done
#После этого выполняется выход из программы.
 exit 0
fi
# Проверка на условие добавления суффикса.
# В остальном, данная часть фактически идентична предыдущему разделу;
# пожалуйста, смотрите комментарии, содержащиеся в нем.
if [ $1 = s ]; then
```

```
   suffix=$2 ; shift ; shift
   if [$1 = ]; then
    echo "не задано ни одного файла"
    exit 0
   fi
 for file in $*
   do
    mv ${file} $file$suffix
 done
 exit 0
fi
# Проверка на условие переименования с заменой.
if [ $1 = r ]; then
   shift
# Из соображений безопасности автор включил эту часть, чтобы не
повредить ни один файл, если пользователь
# не определил, что следует выполнить:
   if [ $# -lt 3 ] ; then
      echo "Ошибка; правильный ввод: renna r [выражение] [замена]
файлы... "
      exit 0
   fi
# Рассмотрим другую информацию
   OLD=$1 ; NEW=$2 ; shift ; shift
# Данный цикл for последовательно проходит через все файлы, которые мы
# задали программе.
# Он совершает одно переименование на файл, используя программу 'sed'.
# Это простая программа с командной строки, которая анализирует
стандартный
# ввод и заменяет регулярное выражение на заданную строку.
# Здесь мы задаём для sed имя файла (в качестве стандартного ввода) и
заменяем
# необходимый текст.
   for file in $*
   do
      new=`echo ${file} | sed s/${OLD}/${NEW}/g`
      mv ${file} $new
   done
exit 0
fi
# Если мы достигли этой строки, это означает, что программе были
заданы
# неправильные параметры. В связи с этим, следует объяснить
пользователю, как её
     # использовать
     echo "используйте:"
```

```
    echo " renna p [префикс] файлы.."
    echo " renna s [суффикс] файлы.."
    echo " renna r [выражение] [замена] файлы.."
    exit 0
    # done!
```

12.4 Программа переименования файлов (простая)

```
#!/bin/bash
# renames.sh
# простая программа переименования
criteria=$1
re_match=$2
replace=$3
for i in $( ls *$criteria* );
do
    src=$i
    tgt=$(echo $i | sed -e "s/$re_match/$replace/")
    mv $src $tgt
done
```

2.3 Задание на лабораторную работу

Разработать на любом языке программирования интерфейс командной строки. Командная строка должна уметь выполнять следующие команды:

1. Вывести помощь о себе
2. Вывести текущее время, дату
3. Вывести текущий каталог
4. Перейти в другой каталог
5. Создать файл
6. Вывести файл
7. Удалить файл
8. Показать размер домашнего каталога или любого другого
9. Показать запущен тот и иной процесс в системе (получить информацию о нём)
10. Найти файл в любой директории
11. Завершение своей работы
12. Корректно реагировать на незнакомую команду

В качестве примеров можно воспользоваться двумя программами-прототипами написанными на языке shell и pascal:

```
#!/bin/sh

while true
do
    echo -n "$USER@$HOSTNAME:$PWD> "
    read comm
    case "$comm" in
```

```
     [Hh][Ee][Ll][Pp]) echo "Simple command line";;
     [Dd][Aa][Tt][Ee]) date;;
        [Ee][Xx][Ii][Tt]) exit;;
     *         ) echo "Command not found";
    esac
done

program Comand_line;
 uses Crt,dos;
 Var
  input: string;
  comand: integer;
 procedure help;
  begin;
   writeln('This is a short help on comand line 1.0');
   writeln('Enabled comands are: help, time and exit');
  end;
 procedure time;
  var h,m,s,s100 :word;
  begin;
   gettime(h,m,s,s100);
   writeln('Current time is ',h,' hour, ',m,' minutes, ',s,' sec-
onds.');
  end;
 begin
  clrscr;
  writeln('Comand line ready. Please, enter your comand.');
   repeat
    comand:=0;
    write('->'); readln(input);
     if input=('help') then comand:=1;
     if input=('time') then comand:=2;
     if input=('exit') then comand:=3;
     case comand of
      1: help;
      2: time;
      3: break;
      else
      writeln('comand not found');
     end
   until input='exit';
  writeln('Press any key and good-bye');
  readkey;
 end.
```

2.4 Содержание отчета

По результатам выполнения работы необходимо подготовить отчет, в котором привести текст программы и процесс выполнения команд.

Лабораторная работа №3. Работа с процессами

3.1 Цель работы

Целью настоящей лабораторной работы является изучение процессов в ОС Linux.

3.2 Теоретическая часть

Процессы в Linux

Термин "процесс" впервые появился при разработке операционной системы Multix и имеет несколько определений, которые используются в зависимости от контекста. Процесс - это:

1. программа на стадии выполнения
2. "объект", которому выделено процессорное время
3. асинхронная работа

Для описания состояний процессов используется несколько моделей. Самая простая модель - это модель трех состояний. Модель состоит из:

1. состояния выполнения
2. состояния ожидания
3. состояния готовности

Выполнение - это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

Ожидание - это пассивное состояние, во время которого процесс заблокирован, он не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

Готовность - это тоже пассивное состояние, процесс тоже заблокирован, но в отличие от состояния ожидания, он заблокирован не по внутренним причинам (ведь ожидание ввода данных - это внутренняя, "личная" проблема процесса - он может ведь и не ожидать ввода данных и свободно выполняться - никто ему не мешает), а по внешним, независящим от процесса, причинам. Когда процесс может перейти в состояние готовности? Предположим, что наш процесс выполнялся до ввода данных. До этого момента он был в состоянии выполнения, потом перешел в состояние ожидания - ему нужно подождать, пока мы введем нужную для работы процесса информацию. Затем процесс хотел уже перейти в состояние выполнения, так как все необходимые ему данные уже введены, но не тут-то было: так как он не единственный процесс в системе, пока он был в состоянии ожидания, его "место под солнцем" занято - процессор выполняет другой процесс. Тогда нашему процессу ничего не остается как перейти в состояние готовности: ждать ему нечего, а выполняться он тоже не может. Из состояния готовности процесс может перейти только в состояние выполнения.

В состоянии выполнения может находится только один процесс на один процессор. Если у вас n-процессорная машина, у вас одновременно в состоянии выполнения могут быть n процессов. Из состояния выполнения процесс может перейти либо в состояние ожидания или состояние готовности. Почему процесс может оказаться

в состоянии ожидания, мы уже знаем - ему просто нужны дополнительные данные или он ожидает освобождения какого-нибудь ресурса, например, устройства или файла. В состояние готовности процесс может перейти, если во время его выполнения, квант времени выполнения "вышел". Другими словами, в операционной системе есть специальная программа - планировщик, которая следит за тем, чтобы все процессы выполнялись отведенное им время. Например, у нас есть три процесса. Один из них находится в состоянии выполнения. Два других - в состоянии готовности. Планировщик следит за временем выполнения первого процесса, если "время вышло", планировщик переводит процесс 1 в состояние готовности, а процесс 2 - в состояние выполнения. Затем, когда, время отведенное, на выполнение процесса 2, закончится, процесс 2 перейдет в состояние готовности, а процесс 3 - в состояние выполнения. Диаграмма модели трех состояний представлена на рисунке 1.

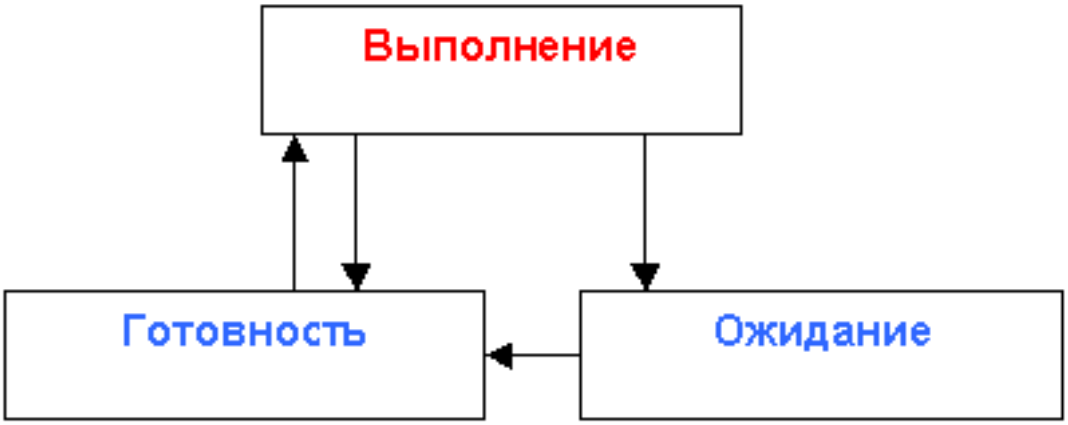

Рисунок 1. Модель трех состояний

Более сложная модель - это модель, состоящая из пяти состояний. В этой модели появилось два дополнительных состояния: рождение процесса и смерть процесса. Рождение процесса - это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса. Как говорится в афоризме: "Мало найти хорошее место, надо его еще застолбить", так вот во время рождения как раз и происходит "застолбление" этого места. Смерть процесса - самого процесса уже нет, но может случиться, что его "место", то есть структура, осталась в списке процессов. Такие процессы называются зобми и о них мы еще поговорим. Диаграмма модели пяти состояний представлена на рисунке 2.

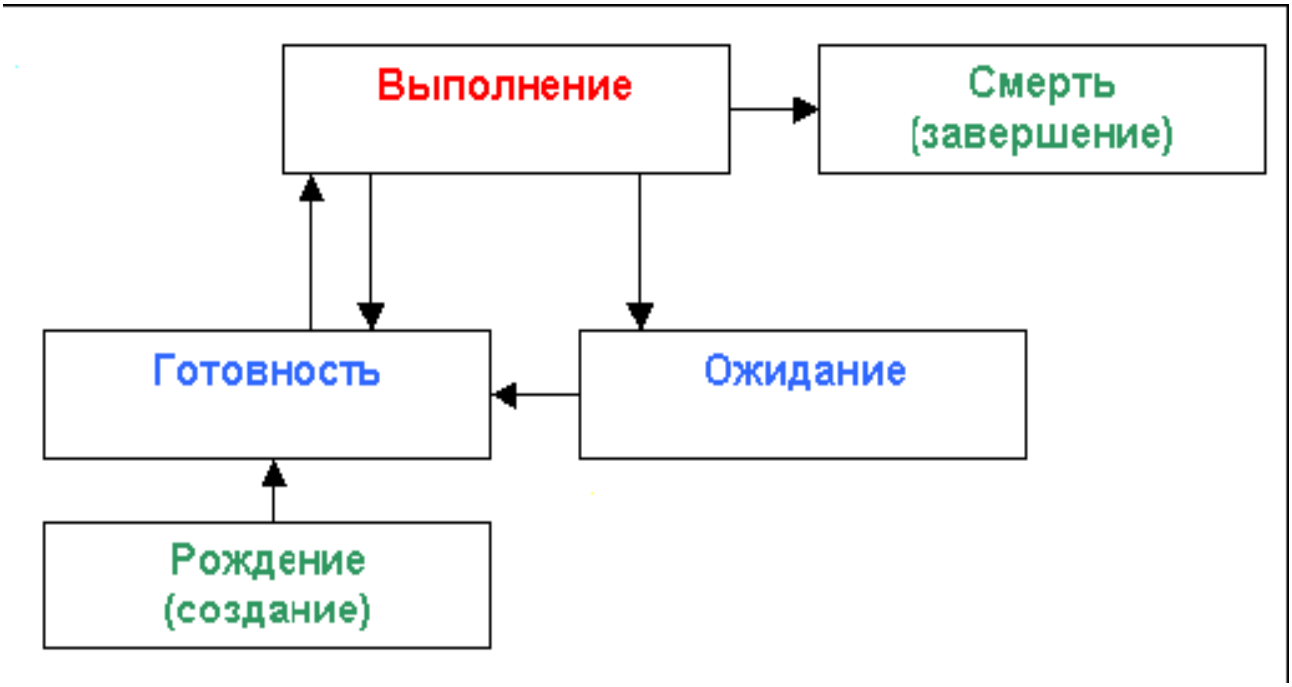

Рисунок 2. Модель пяти состояний

Над процессами можно производить следующие операции:

1. Создание процесса - это переход из состояния рождения в состояние готовности
2. Уничтожение процесса - это переход из состояния выполнения в состояние смерти
3. Восстановление процесса - переход из состояния готовности в состояние выполнения
4. Изменение приоритета процесса - переход из выполнения в готовность
5. Блокирование процесса - переход в состояние ожидания из состояния выполнения
6. Пробуждение процесса - переход из состояния ожидания в состояние готовности
7. Запуск процесса (или его выбор) - переход из состояния готовности в состояние выполнения

Для создания процесса операционной системе нужно:

1. Присвоить процессу имя
2. Добавить информацию о процессе в список процессов
3. Определить приоритет процесса
4. Сформировать блок управления процессом
5. Предоставить процессу нужные ему ресурсы

Подробнее о списке процессов, приоритете и обо всем остальном мы еще поговорим, а сейчас нужно сказать пару слов об иерархии процессов. Процесс не может взяться из ниоткуда: его обязательно должен запустить какой-то процесс. Процесс, запущенный другим процессом, называется дочерним (child) процессом или потомком. Процесс, который запустил процесс называется родительским (parent), родителем или просто - предком. У каждого процесса есть два атрибута - PID (Process ID) - идентификатор процесса и PPID (Parent Process ID) - идентификатор родительского процесса.

Процессы создают иерархию в виде дерева. Самым "главным" предком, то есть процессом, стоящим на вершине этого дерева, является процесс init (PID=1).

Команды для управления процессами

Команда ps

Предназначена для вывода информации о выполняемых процессах. Данная команда имеет много параметров, о которых вы можете прочитать в руководстве (man ps). Краткий список параметров:

-a отобразить все процессы, связанных с терминалом (отображаются процессы всех пользователей)

-e отобразить все процессы

-t список терминалов отобразить процессы, связанные с терминалами

-u идентификаторы пользователей отобразить процессы, связанные с данными идентификаторыми

-g идентификаторы групп отобразить процессы, связанные с данными идентификаторыми групп

-x отобразить все процессы, не связанные с терминалом

Программа top

Предназначена для вывода информации о процессах в реальном времени. Процессы сортируются по максимальному занимаемому процессорному времени, но вы можете изменить порядок сортировки (см. man top). Программа также сообщает о свободных системных ресурсах.

Просмотреть информацию об оперативной памяти вы можете с помощью команды free, а о дисковой – df. Информация о зарегистрированных в системе пользователей доступна по команде w

Изменение приоритета процесса – команда nice

nice [-коэффициент понижения] команда [аргумент]

Команда nice выполняет указанную команду с пониженным приоритетом, коэффициент понижения указывается в диапазоне 1..19 (по умолчанию он равен 10). Суперпользователь может повышать приоритет команды, для этого нужно указать отрицательный коэффициент, например --10. Если указать коэффициент больше 19, то он будет рассматриваться как 19

nohup – игнорирование сигналов прерывания

nohup команда [аргумент]

nohup выполняет запуск команды в режиме игнорирования сигналов. Не игнорируются только сигналы SIGHUP и SIGQUIT.

kill – принудительное завершение процесса

kill [-номер сигнала] PID

где PID – идентификатор процесса, который можно узнать с помощью команды

ps.

Команды выполнения процессов в фоновом режиме – jobs, fg, bg

Команда jobs выводит список процессов, которые выполняются в фоновом режиме, fg – переводит процесс в нормальные режим («на передний план» – foreground), а bg – в фоновый. Запустить программу в фоновом режиме можно с помощью конструкции команда &

3.3 Контрольные вопросы

1. Что такое процесс в Linux, чем он может занимаються, к чему имеет доступ?
2. Что такое PID, GID, UID?
3. Как просмотреть все свои процессы?
4. Как просмотреть все запущенные процессы в интерактивном режиме?
5. Что такое родительский процесс?
6. Для чего необходима системная команада Exec?
7. Чем отличаются процессы работающие в фоне (Background) от работающих не в фоне (Foreground)
8. Что такое приоритет? Команда изменения приоритета процесса и запуска процесса с заданным приоритетом.
9. Сигналы - что это, зачем они нужны и как работают.
10. Сигналы прекращения, приостановки и завершения работы.

3.4 Задание на лабораторную работу

Вариант задания определяется преподавателем.

Вариант 1

1. Определение процесса
2. Что такое TIME процесса
3. Записать в файл fa список всех процесов
4. Проверить, есть ли в fa процесс kwin
5. Переименовать файл fa в fb
6. Дописать в файл fb размер папки /bin
7. Запустить любую команду в фоновом режиме
8. Перевести запущенную команду в оперативный режим
9. Повторить пункт 7 и убить запущенный процесс
10. Перечислите основные атрибуты процесса

Вариант 2

1. Определение работоспособного процесса
2. Что такое STIME процесса
3. Записать в файл fa суммарный размер вашего рабочего каталога (в Мб)
4. Проверить запущен ли в данный момент процесс bash
5. Записать в файл fb список ваших собственных процессов
6. Изменить приоритет вашего процесса на -5
7. Запустить любые две команды в фоновом режиме
8. Просмотреть информацию по фоновым процессам

9. Убить одну из них
10. Команда "поиск по шаблону", синтаксис и для чего используется (примеры)

Вариант 3

1. Определение ожидающего процесса
2. Что такое ETIME процесса
3. Записать суммарный размер каталога /sbin в файл fa (в Байтах)
4. Дописать в файл fa информацию о процессе INIT в следующем формате (Идентификатор, Родительский идентификатор, Имя, Процессорное время)
5. Какие максимальное и минимальное значения атрибута NICE
6. Запустите любой процесс при приоритетом +7
7. Запустить любую команду в фоновом режиме
8. Переведите её в фоновый режим
9. Для чего предназначены "сигналы"
10. Понятие конвейера

Вариант 4

1. Определение остановленного процесса
2. Оперативный режим работы процесса
3. Записать в файл fa информацию о всех процессах в следующем формате (Имя, Владелец, Время запуска процесса)
4. Получить суммарный размер вашего домашнего каталога (в Кб)
5. Запустить любую команду в фоновом режиме
6. Просмотрите информацию о фоновых процессах
7. Измените приоритет у запущенного процесса на +10
8. Переведите процесс в оперативный режим
9. Запустите ещё одну команду
10. Пошлите запущенному процессу любой сигнал

Вариант 5

1. Определение завершившегося процесса
2. Фоновый режим работы процесса
3. Записать в файл fa информацию о процессах пользователя root
4. Записать в файл fb размер вашего домашнего каталога (в Мб)
5. Проверить, есть ли в файле fa информация о процессе init
6. Запустить две команду в фоновом режиме
7. Просмотреть информацию о фоновых процессах
8. Убить один из запущенных процессов
9. Перевести процесс в оперативный режим
10. Отличие команды nice от renice

Вариант 6

1. Что такое PID процесса
2. Сигналы управления процессами
3. Записать в файл fa информацию о процессе с PID равным 1

4. Дописать в файл fa суммарный размер каталога /bin (в Байтах)
5. Запустить процесс с приоритетом +5
6. Изменить приоритет у запущенного процесса на -5
7. Запустить любую команду в фоновом режиме
8. Просмотреть информацию о фоновых задачах
9. Послать запущенному процессу любой сигнал
10. В чём отличие команд PS и TOP

3.5 Содержание отчета

Студенты должны привести в отчете ответы на поставленные в задании вопросы, а также скриншоты экранов, демонстрирующих выполненные ими операции.

Лабораторная работа №4 Определение технических параметров компьютера

4.1 Цель работы

Научиться определять технические параметры компьютера с помощью служебных программ операционной системы Linux.

4.2 Теоретическая часть

Linux – свободно распространяемая многозадачная, многопользовательская операционная система.

Ядро Linux разработано Линусом Торвальдсом в 1991 г. Файлы первая версия Linux (версия 0.01) были опубликованы в Интернете 17 сентября 1991 года.

Если быть более точными, то Linux — это только ядро, когда же речь заходит об операционной системе, то более правильно говорить «Операционная система на основе ядра Linux». Ядро ОС Linux разрабатывается под общим руководством Линуса Торвальдса и распространяется свободно (на основе лицензии GPL)

К основным характеристикам Linux можно отнести многозадачность, многопользовательский доступ и разграничение прав доступа к файлам , поддержка различных форматов файловых систем .

В силу того, что исходные коды Linux распространяются свободно и общедоступны, к развитию системы с самого начала подключилось большое число независимых разработчиков. Благодаря этому на сегодняшний момент Linux — самая современная, устойчивая и быстроразвивающаяся система, почти мгновенно вбирающая в себя самые последние технологические новшества. Она обладает всеми возможностями, которые присущи современным полнофункциональным операционным системам типа UNIX.

4.2.1 Реальная многозадачность

Все процессы независимы; ни один из них не должен мешать выполнению других задач. Для этого ядро осуществляет режим разделения времени центрального процессора, поочередно выделяя каждому процессу интервалы времени для выполнения.

4.2.2 Многопользовательский доступ

Linux — не только многозадачная ОС, она поддерживает возможность од-

новременной работы многих пользователей. При этом Linux может предоставлять все системные ресурсы пользователям, работающим с хостом через различные удаленные терминалы.

4.2.3 Свопирование оперативной памяти на диск

Свопирование оперативной памяти на диск позволяет работать при ограниченном объеме физической оперативной памяти; для этого содержимое некоторых частей (страниц) оперативной памяти записываются в выделенную область на жестком диске, которая трактуется как дополнительная оперативная память. Это несколько снижает скорость работы, но позволяет организовать работу программ, требующих большего объема ОЗУ, чем фактически имеется в компьютере.

4.2.4 Страничная организация памяти

Системная память Linux организована в виде страниц объемом 4K. Если оперативная память полностью исчерпана, ОС будет искать давно не использованные страницы памяти для их перемещения из памяти на жесткий диск. Если какие-либо из этих страниц становятся нужны, Linux восстанавливает их с диска. Некоторые старые Unix-системы и некоторые современные платформы (включая Microsoft Windows) переносят на диск все содержимое ОП, относящееся к неработающему в данный момент приложению, (т.е. ВСЕ страницы памяти, относящиеся к приложению, сохраняются на диске при нехватке памяти) что менее эффективно.

4.2.5 Загрузка выполняемых модулей "по требованию"

Ядро Linux поддерживает выделение страниц памяти по требованию, при котором только необходимая часть кода исполняемой программы находится в оперативной памяти, а не используемые в данный момент части остаются на диске.

4.2.6 Совместное использование исполняемых программ

Если необходимо запустить одновременно несколько копий какого-то приложения (либо один пользователь запускает несколько идентичных задач, либо разные пользователи запускают одну и ту же задачу), то в память загружается только одна копия исполняемого кода этого приложения, которая используется всеми одновременно исполняющимися идентичными задачами.

4.2.7 Общие библиотеки

Библиотеки — наборы процедур, используемых программами для обработки данных. Существует некоторое количество стандартных библиотек, используемых одновременно более чем одним процессом. В старых системах такие библиотеки включались в каждый исполняемый файл, одновременное выполнение которых приводило к непродуктивному использованию памяти. В новых системах (в частности, в Linux), обеспечивается работа с динамически и статически разделяемыми библиотеками, что позволяет сократить размер отдельных приложений.

4.2.8 Динамическое кеширование диска

Кеширование диска — это использование части оперативной памяти для хранения часто используемых данных с диска, что существенно ускоряет доступ к часто используемым программам и задачам. Пользователи MS-DOS работают со SmartDrive, который резервирует фиксированные области системной памяти для кеширования диска. Linux использует более динамичную систему кеширования: память, зарезервированная под кеш, увеличивается, когда память не используется, и уменьшается, если системе или процессу пользователя требуется больше памяти.

4.2.9 Поддержка различных форматов файловых систем

Linux поддерживает большое число форматов файловых систем, включая файловые системы DOS и OS/2, а также современные журналируемые файловые системы. При этом и собственная файловая система Linux, которая называется Second Extended File System (ext2fs), позволяет эффективно использовать дисковое пространство.

4.2.10 Сетевые возможности

Linux можно интегрировать в любую локальную сеть. Поддерживаются все службы Unix, включая Networked File System (NFS), удаленный доступ (telnet, rlogin), работа в TCP/IP сетях, dial-up-доступ по протоколам SLIP и PPP, и т. д.. Также поддерживается включение Linux-машины как сервера или клиента для другой сети, в частности, работает общее использование (sharing) файлов и удаленная печать в Macintosh, NetWare и Windows.

4.2.11 Работа на разных аппаратных платформах

Хотя ОС Linux первоначально была разработана для ПК на базе Intel 386/486, сейчас она может работать на всех версиях Intel-овских микропроцессоров, начиная с 386 и кончая многопроцессорными системами на Pentium. Так же успешно Linux работает на различных клонах Intel от других производите-

лей. Кроме того, разработаны версии для других типов процессоров — ARM, DEC Alpha, SUN Sparc, M68000 (Atari и Amiga), MIPS, PowerPC и других.

4.2.12 Дистрибутивы Linux

В любой операционной системе можно выделить 4 основных части: ядро, файловую структуру, интерпретатор команд пользователя и утилиты. Ядро — это основная, определяющая часть ОС, которая управляет аппаратными средствами и выполнением программ. Файловая структура — это система хранения файлов на запоминающих устройствах. Интерпретатор команд или оболочка — это программа, организующая взаимодействие пользователя с компьютером. И, наконец, утилиты — это просто отдельные программы, которые, вообще говоря, ничем принципиально не отличаются от других программ, запускаемых пользователем, разве только своим основным назначением — они выполняют служебные функции.

Как уже говорилось выше, если быть точным, то слово "Linux" обозначает только ядро. Поэтому, когда речь идет об операционной системе, правильнее было бы говорить "операционная система, основанная на ядре Linux". Ядро ОС Linux разрабатывается под общим руководством Линуса Торвальдса и распространяется свободно (на основе лицензии GPL), как и огромное количество другого программного обеспечения, утилит и прикладных программ. Одним из следствий свободного распространения ПО для Linux явилось то, что большое число разных фирм и компаний, а также просто независимых групп разработчиков стали выпускать так называемые дистрибутивы Linux.

Дистрибутив — это набор программного обеспечения, включающий все 4 основные составные части ОС, т. е. ядро, файловую систему, оболочку и совокупность утилит, а также некоторую совокупность прикладных программ. Обычно все программы, включаемые в дистрибутив Linux, распространяются на условиях GPL, так что может сложиться впечатление, что дистрибутив может выпустить кто угодно, точнее любой, кто не поленится собрать коллекцию свободного ПО. И какая-то степень правдоподобия в таком утверждении есть. Однако разработчик дистрибутива должен по крайней мере создать программу инсталляции, которая будет устанавливать ОС на компьютер, на котором никакой ОС еще нет. Кроме того, необходимо обеспечить разрешение взаимозависимостей и противоречий между разными пакетами (и версиями пакетов), что, как мы увидим позже, тоже является нетривиальной задачей.

Тем не менее, в мире существует уже более сотни различных дистрибутивов Linux, и все время появляются новые. Более-менее полный список их можно найти на сервере http://www.linuxhq.com, где даны краткие характеристики каждому дистрибутиву (упоминаются и некоторые локализованные версии). Кроме того, там же есть ссылки на другие списки дистрибутивов, так что при желании можно найти все, что вообще существует в мире.

4.2.13 Файловая система

Файловая система — это структура, с помощью которой ядро операционной системы предоставляет пользователям (и процессам) ресурсы долговременной памяти системы, т.е. памяти на различного вида долговременных носителях информации — жестких дисках, магнитных лентах. CD-ROM и т. п.

Информация в любой ОС хранится на носителях в виде файлов. Файлы группируются в каталоги, которые, в свою очередь, могут быть включены в другие каталоги. В результате получается иерархическая структура каталогов, начинающаяся с корневого каталога. Каждый (под)каталог может содержать как отдельные файлы, так и подкаталоги.

Иерархическую структуру каталогов обычно иллюстрируют рисунком "дерева каталогов", в котором каждый каталог изображается узлом "дерева", а файлы — "листьями". В MS Windows или DOS каталоговая структура строится отдельно для каждого физического носителя (т. е., имеем не отдельное "дерево", а целый "лес") и корневой каталог каждой каталоговой структуры обозначается какой-нибудь буквой латинского алфавита (отсюда уже возникает некоторое ограничение). В Linux (и UNIX вообще) строится единая каталоговая структура для всех носителей, и единственный корневой каталог этой структуры обозначается символом "/". В эту единую каталоговую структуру можно подключить любое число каталогов, физически расположенных на разных носителях (как говорят, "смонтировать файловую систему" или "смонтировать носитель").

Имена каталогов строятся по тем же правилам, что и имена файлов. И, вообще, каталоги в принципе ничем, кроме своей внутренней структуры (до которой ОС уже есть дело) не отличаются от "обычных" файлов, например, текстовых.

Полным именем файла (или путем к файлу) называется список имен вложенных друг в друга подкаталогов, начинающийся с корневого каталога и оканчивающийся собственно именем файла. При этом имена подкаталогов в этом списке разделяются тем же символом "/", который служит для обозначения корневого каталога.

В каждый момент времени пользователь работает с одним экземпляром оболочки shell и эта оболочка хранит значение так называемого "текущего" каталога, т. е. того каталога, в котором пользователь сейчас работает. Имеется специальная команда, которая сообщает вам значение текущего каталога — pwd.

В Linux типовая структура каталогов выдерживается, пожалуй, даже более строго, чем в Windows. Более того, существует даже стандарт на структуру каталогов для UNIX-подобных ОС, так называемый Filesystem Hierarchy Standart (FHS).

Стандарт FHS предлагает создать в корневом каталоге следующие подкаталоги:

/bin - Этот каталог содержит в основном готовые к исполнению программы, большинство из которых необходимы во время старта системы (или в однопользовательском системном режиме, используемом для отладки). Здесь хранится значительное количество общеупотребительных команд Linux .

/boot - неизменяемые файлы, необходимые для загрузки системы;

/dev - файлы устройств;

/etc -этот каталог и его подкаталоги содержат большинство данных, необходимых для начальной загрузки системы и основные конфигурационные файлы. В /etc находятся, например, файл inittab, определяющий загружаемую конфигурацию, и файл паролей пользователей passwd. Часть конфигурационных файлов может находится и в /usr/etc. Каталог /etc не должен содержать двоичных файлов (их следует перенести в /bin или /sbin)

/home - домашние каталоги пользователей;

/lib - основные разделяемые библиотеки и модули ядра; Этот каталог содержит разделяемые библиотеки функций, необходимых компилятору языка C и модули (драйверы устройств). Даже если в системе не установлен компилятор языка C, разделяемые библиотеки необходимы, поскольку они используются многими прикладными программами. Они загружаются в память по мере необходимости выполнения каких-то функций, что позволяет уменьшить объем кода программ — в противном случае один и тот же код многократно повторялся бы в различных программах;

/mnt - это точка монтирования для временно монтируемых файловых систем. Если на компьютере запускается поочередно Linux и MS DOS, то этот каталог обычно используется, чтобы монтировать файловую систему MS DOS. Если вы имеете привычку монтировать несколько дополнительных носителей, например, дискеты, CD-ROM, дополнительный жесткий диск и т. д., то можно создать в нем соответственно дополнительные подкаталоги для каждого носителя;

/root - домашний каталог пользователя суперпользователя root;

/opt - дополнительные пакеты программного обеспечения;

/sbin - основные системные исполняемые файлы;

/tmp - временные файлы;

/usr - Этот каталог огромен и его структура в основном повторяет структуру корневого каталога. В его подкаталогах находятся все основные приложения. В соответствии со стандартом FHS рекомендуется выделять для этого каталога отдельный раздел диска или вообще располагать его на сетевом диске, общем для всех компьютеров в сети. Такой раздел или диск монтируют только для чтения и располагают в нем общие конфигурационные и исполняемые файлы, документацию, системные утилиты и библиотеки, а также включаемые файлы (файлы типа include);

/var - переменные данные.

В соответствии с требованиями стандарта приложения не должны создавать файлы и каталоги или требовать наличия каких-то специальных файлов и каталогов (помимо перечисленных) в корневом каталоге. Во-первых, размер корневой файловой системы желательно сохранять по возможности малым, а во-вторых, стандарт FHS обеспечивает достаточную гибкость и удобство размещения файлов, не попавших в корневую систему, в других файловых системах и подкаталогах. Некоторые подкаталоги корневого каталога факультативны. Но уж если они существуют, то должны размещаться в корневом каталоге, но не обязательно в корневой файловой системе.

4.2.14 Права доступа к файлам и каталогам

Поскольку Linux — система многопользовательская, вопрос об организации разграничения доступа к файлам и каталогам является одним из существенных вопросов, которые должна решать операционная система. Механизмы разграничения доступа, разработанные для системы UNIX в 70-х годах (возможно, впрочем, они предлагались кем-то и раньше), очень просты, но они оказались настолько эффективными, что просуществовали уже более 30 лет и по сей день успешно выполняют стоящие перед ними задачи.

В основе механизмов разграничения доступа лежат имена пользователей и имена групп пользователей. В Linux каждый пользователь имеет уникальное имя, под которым он входит в систему (логируется). Кроме того, в системе создается некоторое число групп пользователей, причем каждый пользователь может быть включен в одну или несколько групп. Создает и удаляет группы суперпользователь, он же может изменять состав участников той или иной группы. Члены разных групп могут иметь разные права по доступу к файлам, например, группа администраторов может иметь больше прав, чем группа программистов.

В индексном дескрипторе каждого файла записаны имя так называемого владельца файла и группы, которая имеет права на этот файл. Первоначально, при создании файла его владельцем объявляется тот пользователь, который этот файл создал. Точнее — тот пользователь, от чьего имени запущен процесс, создающий файл. Группа тоже назначается при создании файла — по идентификатору группы процесса, создающего файл. Владельца и группу файла можно поменять в ходе дальнейшей работы с помощью команд chown и chgrp .

Вообще говоря, права доступа и информация о типе файла в UNIX-системах хранятся в индексных дескрипторах в отдельной структуре, состоящей из двух байтов, т. е. из 16 бит (это естественно, ведь компьютер оперирует битами, а не символами r, w, x). Четыре бита из этих 16-ти отведены для кодированной записи о типе файла. Следующие три бита задают особые свойства исполняемых файлов, о которых мы скажем чуть позже. И, наконец, оставшиеся

9 бит определяют права доступа к файлу. Эти 9 бит разделяются на 3 группы по три бита. Первые три бита задают права пользователя, следующие три бита — права группы, последние 3 бита определяют права всех остальных пользователей (т. е. всех пользователей, за исключением владельца файла и группы файла).

При этом, если соответствующий бит имеет значение 1, то право предоставляется, а если он равен 0, то право не предоставляется. В символьной форме записи прав единица заменяется соответствующим символом (r, w или x), а 0 представляется прочерком.

Право на чтение (r) файла означает, что пользователь может просматривать содержимое файла с помощью различных команд просмотра, например, командой more или с помощью любого текстового редактора. Но, подредактировав содержимое файла в текстовом редакторе, вы не сможете сохранить изменения в файле на диске, если не имеете права на запись (w) в этот файл. Право на выполнение (x) означает, что вы можете загрузить файл в память и попытаться запустить его на выполнение как исполняемую программу. Конечно, если в действительности файл не является программой (или скриптом shell), то запустить этот файл на выполнение не удастся, но, с другой стороны, даже если файл действительно является программой, но право на выполнение для него не установлено, то он тоже не запустится.

4.2.15 Команды Linux

man (от англ. manual — руководство) — команда Unix, предназначенная для форматирования и вывода справочных страниц.

top — консольная команда UNIX-совместимых операционных систем, список работающих в данный момент процессов и информацию о них. Команда top показывает список работающих в данный момент процессов и информацию о них, включая использование ими памяти и процессора. Список интерактивно формируется в реальном времени. Чтобы выйти из программы top, нажмите клавишу [q].

free - Показывает количество свободной и используемой памяти в системе.

ps (от англ. process status) — консольная команда UNIX-совместимых операционных систем, выдающая отчёт о работающих процессах.

ls - выдача информации о файлах или каталогах

Синтаксис команды:

ls [флаги] [имя ...]

Команда ls для каждого имени каталога распечатывает список входящих в этот каталог файлов; для файлов - повторяется имя файла и выводится дополнительная информация в соответствии с указанными флагами. По умолчанию имена файлов выводятся в алфавитном порядке. Если имена не заданы, выдается содержимое текущего каталога. Если заданы несколько аргументов, то они

сортируются по алфавиту, однако сначала всегда идут файлы, а потом каталоги с их содержимым.

cat <имя_файла> - вывод содержимого файла на стандартный вывод (по умолчанию - на экран).

Можно записать вводимый на экран текст с помощью следующей последовательности действий:

cat > <имя_файла>

more <имя_файла> - просмотр содержимого длинного текстового файла по страницам;

less <имя_файла> - просмотр содержимого текстового файла с возможностью вернуться к предыдущим страницам. Нажмите q, когда захотите выйти из программы. "less" - аналог команды DOS "more", хотя очень часто "less" бывает более удобной чем "more".

pico <имя_файла> - редактировать текстовый файл с помощью текстового редактора pico.

whoami - вывести имя под которым Вы зарегистрированы.

date - вывести дату и время.

time <имя программы> - выполнить программу и получить информацию о времени, нужном для ее выполнения. Не путайте эту команду с date. Например: Я могу определить выполнить команду ls и узнать, как много времени требуется для вывода списка файлов в каталоге, набрав последовательность: time ls

who - определить кто из пользователей работает на машине.

finger <имя_пользователя> - системная информация о зарегистрированном пользователе.

uptime - количество времени, прошедшего с последней перезагрузки операционной системы.

uname - a - вывести информацию о версии операционной системы.

free - вывести информацию по использованию памяти.

df - вывести информацию о свободном и используемом месте на дисках.

login запрос от пользователя имени и пароля (запрос от системы к пользователю) для входа в систему (по умолчанию, при наборе пароля, он не отображается).

logout выход из текущего сеанса оболочки.

startx команда для запуска графического интерфейса X Window.

shutdown останавливает систему и предотвращает повреждение файловой системы при этом, но,используется только при работе в консольном режиме. При работе в режиме X Window, не используйте.

halt быстрое и корректное выключение системы.

poweroff корректное выключение системы.

reboot корректное выключение с последущей загрузкой. Перезагрузка.

vmstat выдаёт сведения о процессах, памяти и загруженности центрально-

го процессора.

cal форматированный календарь на текущий месяц (добавить у и будет календарь на весь текущий год).

oclock простые часы, которые висят на рабочем столе (много дополнительных параметров).

hostname команда отображает идентификатор данного узла сети (его имя). root может изменить имя узла на новое.

hwclock встроенные часы Вашего компьютера. Для изменения даты и времени и синхронизации с системными часами, необходимы привелегии root.

users отображает краткий список пользователей работающих в системе в данный момент.

w подробная информация о всех пользователях, работающих в данный момент и также простой, вход в систему и др.Если нужен один пользователь, то указать имя в параметре.

whatis поиск по базе данных страниц руководства и отображение краткого описания.

which показывает полный путь к исполняемому файлу команды.

write отправляет сообщение другому пользователю, находящемуся в системе, путём копирования строк с терминала отправителя на терминал получателя.

wall отправляет сообщение на терминал каждого пользователя находящегося в системе в данный момент.

history показывает пронумерованный список команд, которые Вы выполняли в этом и предыдущем сеансе. Если в списке историии их довольно много, то увидите последние.

jobs выводит список всех выполняемых и приостановленных задач.

kill завершить процесс (необходимо указать какой).

killall позволят управлять процессами используя их имена или имена файлов, а не идентификаторы как в kill. Завершаются все указанные процессы.

nice позволяет отобразить или настроить приоритет задачи.

pstree показывает иерархию процессов системы, что хорошо показывает их взаимозависимость.

renice задаёт приоритет для указанной задачи.

script позволяет записывать весь вывод с терминала в файл. Что бы остановить запись нажмите Ctrl+d.Если имя файла не указано то записывается в typescript.

times показывает полное время выполнения процессов для всей системы и данного пользователя.

file показывает тип содержимого указанного файла (текст, выполняемый, данные).

last показывает список пользователей, которые заходили в систему с мо-

мента создания файла /var/log/wtmp.

lastlog проверяет историю входа в систему зарегистрированных пользователей.Форматирует и выводит на печать файл /var/log/lastlog.

logger посылает запрос демону syslogd с просьбой поместить сообщение в системный журнал.

lpr отправляет документ на печать демону печати.

chfn изменяет сведения о пользователе в файле /etc/passwd из которого берёт информацию команда finger .

chgrp команда для администратора, для изменения группы владельцев файла.

clear очищает экран терминала (если это возможно).

crontab обеспечивает возможность выполнения определённых задач по расписанию. Чаще используется администратором, хотя свои задачи могут быть и у пользователей.

csplit разбивает файл на несколько частей.Надо задать метод разбивки (строки и т.д.).

dd копирование файла с одновременным выполнением различных, дополнительных преобразований.

dc калькулятор.

debugfs применяется для восстановления файловой системы (ext2, ext3) если недостаточно команды fsck.

du показывает количество блоков диска, занятых каждым из файлов каталога.

mc запускает программу Midnight Commander диспетчер файлов в текстовой консоли. Напоминает MSDOS менеджеры и довольно проста и удобна в использовании.Очень много нужных и удобных функций.

mcat копирует необработанные данные на дискету.

mcopy использует отформатированную дискету MSDOS для копирования файлов в Линукс и из Линукс без предварительного подключения дискеты к файловой системе.

mdel удаляет файл на отформатированной дискете MSDOS.

mdir отображает содержимое каталога на дискете MSDOS.

mdu показывает дисковое пространство занятое каталогом MSDOS.

mesg контролирует доступ к Вашему терминалу, что бы коллеги не могли засыпать Вас сообщениями с помощью команды write

mformat создаёт на дискете файловую систему MSDOS.

mkbootdisk применяется в некоторых дистрибутивах, для создания загрузочной дискеты, содержащей всё необходимое для аварийной загрузки.

mktemp создаёт уникальное имя файла для временной работы.

mlabel создаёт метку тома на MSDOS на отформатированной дискете.

mmd создаёт подкаталог MSDOS на отформатированной дискете.

mmount подключает к файловой системе отформатированное устройство MSDOS.

mmove перемещает или переименовывает файл на дискете MSDOS.

stat отображение всей доступной информации об указанном файле.

touch изменяет время последнего доступа или изменения файла на текущее время.

wc показывает число строк, слов и символов в файле.

bunzip2 распаковывает указанный файл на 30%быстрей чем gzip.

bzip2 сжимает указанный файл по ускоренному алгоритму.

bzip2recover делает попытку восстановить данные из повреждённого файла сжатого bzip2.

compress сжимает указанный файл по другому алгоритму.

uncompress распаковывает файл сжатый предыдущей командой.

gzip сжимает указанный файл.

gunzip распаковывает указанный файл (расширения .Z,.gz,.tgz,.zip).

gzexe позволяет сжать исполняемый файл с указанным именем так, что бы он автоматически распаковывался и выполнялся, когда пользователь даёт команду на выполнение сжатого файла.

gpasswd устанавливает пароль группы.

mcrypt Шифрует указанный файл.Создаётся новый файл в рабочем каталоге с расширением .enc. Вам будет предложено ввести пароль.Не забудьте его.

mdecrypt расшифровывает это же файл.Если этих утилит нет, скачайте http://mcrypt/hellug.grl

tar помещает два и более файлов в новый или существующий архив или извлекает их из архива.При задании каталога, заархивирует все файлы в каталоге и подкаталоге.

echo выводит строку текста на стандартное устройство вывода.

fdformat форматирование гибкого диска. Дополнительно вводится имя устройства и необходимый вид форматирования.

fg переводит процесс выполняемый в фоновом режиме в приоритетный режим.

fgconsole показывает количество активных виртуальных консолей.

fsck проверяет и восстанавливает файловую систему.

mount монтирование файловой системы.

umount отмонтирование файловой системы (в обоих командах необходимо указать, что именно).

rdev при вызове без параметров выводит информацию о текущей файловой системе.

rcp применяется для копирования файлов с одного компьютера на другой.

rdate получает значение даты и времени от другого узла сети.Используется для синхронизации системного времени узлов.

rename переименовывает файлы. Очень удобно, когда много файлов.

sleep приостанавливает начало выполнения процесса на заданное количество секунд.

usleep приостанавливает на микросекунды.

sync очищает буферы файловой системы.

cmp производит быстрое сравнение двух указанных файлов. Если они идентичны, то никакие сообщения не выводятся.

column форматирует входной текст из указанного файла в список из пяти колонок.

diff сравнивает два указанных текстовых файла. Каждое отличие выводится в контексте. Позволяет сравнивать каталоги.

id отображает действующие значения идентификаторов пользователя и группы для текущего пользователя.

ifconfig отображает состояние текущей конфигурации сети или настраивает сетевой интерфейс.

nl команда нумерует строки в указанном файле.

sort команда позволяет отсортировать строки файла в алфавитном порядке.

groupadd создание группы пользователей с указанным именем.

groupdel удаляет группу с указанным именем.

groupmod изменяет параметры группы с указанным именем.

mkpasswd создаёт высококачественный пароль, состоящий по умолчанию из девяти символов и содержащий по крайней мере буквы в разном регистре и цифры.

useradd создание нового пользователя с указанным именем.

userdel удаляет пользователя с указанным именем.

usermod изменяет параметры пользователя с указанным именем.

netstat вывод информации о сетевой подсистеме.Очень много настроек и параметров.

ping отправка на указанный адрес пакетов для проверки возможности соединения с этим узлом.

telnet открывает окно терминала на удалённом узле и запускает интерактивный сеанс.

passwd - смена входного пароля

Синтаксис команды:

passwd [входное_имя]

Команда passwd меняет (или устанавливает) пароль, связанный с входным_именем пользователя.

Обычный пользователь может менять только пароль, связанный с его собственным входным_именем.

Команда запрашивает у обычных пользователей старый пароль (если он

был), а затем дважды запрашивает новый. После первого запроса проверяется, достаточен ли "возраст" старого пароля. Возраст - это промежуток времени (обычно несколько дней), который должен пройти между сменами пароля. Если возраст недостаточен, новый пароль отвергается и passwd завершается.

Если возраст достаточен, делается проверка на соответствие нового пароля техническим требованиям. Когда новый пароль вводится во второй раз, две копии нового пароля сравниваются. Если они не совпали, цикл запроса нового пароля повторяется, но не более двух раз.

Технические требования к паролям:

1. Каждый пароль должен содержать не менее 6 символов. Значащими являются только первые 8.

2. Каждый пароль должен содержать как минимум две буквы (большие или малые) и хотя бы одну цифру или знак.

3. Каждый пароль должен отличаться от входного_имени, прочитанного слева направо или задом наперед, и от его циклических сдвигов. При сравнении не делается различий между большими и малыми буквами.

4. Новый пароль должен отличаться от старого хотя бы тремя символами. При сравнении не делается различий между большими и малыми буквами.

Суперпользователь (root) имеет право изменять любые пароли, поэтому у него старый пароль не запрашивается. Суперпользователь не связан ограничениями на возраст пароля и соответствие техническим требованиям. Суперпользователь может создать пустой пароль, нажимая возврат каретки в ответ на запрос нового пароля.

su (сокр. от англ. Substitute User) — команда Unix-подобных операционных систем, позволяющая пользователю войти в систему под другим именем, не завершая текущий сеанс. Обычно используется для временного входа суперпользователем для выполнения административных работ.

Синтаксис команды:

su [-] [имя_пользователя [аргумент ...]]

Команда su позволяет пользователю выполнять команды от имени другого пользователя, не завершая текущий сеанс, или получить роль. По умолчанию предполагается работа от имени пользователя root (суперпользователя).

Для использования su необходимо ввести соответствующий пароль (если только команду не вызывает пользователь root). Если введен правильный пароль, su создает новый процесс командного интерпретатора, с такими же реальными и эффективными идентификаторами пользователя и группы, а также списком дополнительных групп, что и у указанного пользователя.

sudo - выполнить команду от имени другого пользователя

Синтаксис команды:

sudo -V | -h | -l | -L | -v | -k | -K | -s | [-H] [-P] [-S] [-b] | [-p запрос] [-c класс|-] [-a тип_аутентификации] [-u имя_пользователя/#uid] команда

sudo позволяет разрешенному пользователю выполнять команду как суперпользователь или другой пользователь, как определено в файле sudoers. Реальный и эффективный uid и gid при этом устанавливаются так, чтобы соответствовать таковым целевого пользователя, как определено в файле passwd (также инициализируется вектор группы, если целевой пользователь - не root). По умолчанию sudo требует, что бы пользователи аутентифицировали себя при помощи пароля (ЗАПОМНИТЕ: это пароль пользователя, не пароль root). Как только пользователь аутентифицировал себя происходит обновление временной метки и пользователь может использовать sudo некоторый период времени без пароля (по умолчанию пять минут, если в sudoers не указано другое).

4.2.16 Работа с файлами и каталогами

pwd - выдача имени текущего каталога. Бывает, что при ее изучении, вы попадаете в какой-то каталог, про который уже не помните, как он называется и как вы в него попали. Узнать его полное имя позволяет команда pwd.

cd - смена текущего каталога.

Синтаксис команды:

cd [каталог]

Команда cd применяется для того, чтобы сделать заданный каталог текущим. Если каталог не указан, используется значение переменной окружения $HOME (обычно это каталог, в который Вы попадаете сразу после входа в систему). Если каталог задан полным маршрутным именем, он становится текущим. По отношению к новому каталогу нужно иметь право на выполнение, которое в данном случае трактуется как разрешение на поиск.

chmod - изменение режима доступа к файлам

Синтаксис команды:

chmod режим файл

Права доступа к указанным файлам (среди которых могут быть каталоги) изменяются в соответствии с указанным режимом. Режим может быть задан в абсолютном или символьном виде.

Использование символьного вида основано на однобуквенных обозначениях, которые определяют класс доступа и права доступа для членов данного класса. Права доступа к файлу зависят от идентификатора пользователя и идентификатора группы, в которую он входит. Режим в целом описывается в терминах трех последовательностей, по три буквы в каждой:

Владелец	Группа	Прочие
(u)	(g)	(o)
rwx	rwx	rwx

Здесь владелец, члены группы и все прочие пользователи обладают правами чтения файла, записи в него и его выполнения. В примере показаны обозначения как для класса доступа, так и для прав доступа внутри класса.

Для задания режима доступа в символьном виде используется следующий синтаксис:

[кому] операция права

Часть [кому] есть комбинация букв u, g и o (владелец, члены группы и прочие пользователи соответственно). Если часть кому опущена или указано a, то это эквивалентно ugo.

Операция может быть: + (добавить право), - (лишить права), = (в пределах данного класса присвоить права абсолютно, то есть добавить указанные права и отнять неуказанные).

Права - любая осмысленная комбинация следующих букв:

r Право на чтение.

w Право на запись.

x Право на выполнение (поиск в каталоге).

s При выполнении переустанавливать действующий идентификатор пользователя или группы.

t После выполнения программы сохранять сегмент команд (бит навязчивости).

l Учет блокировки доступа.

Опустить часть права можно только если операция есть = (для лишения всех прав).

Если надо сделать более одного указания об изменении прав, то при использовании символьного вида в правах не должно быть пробелов, а указания должны разделяться запятыми. Например, команда chmod u+w,go+x f1 добавит для владельца право писать в файл f1, а для членов группы и прочих пользователей - право выполнять файл. Права устанавливаются в указанном порядке. Право s можно добавлять только для пользователя и группы, право t - только для пользователя.

Чтобы установить права, позволяющие владельцу читать и писать в файл, а членам группы и прочим пользователям только читать, надо использовать следующую запись:

chmod u=rw,go=r f1

Позволить всем выполнять файл f2

chmod +x f2

chown —изменить владельца файла

Только суперпользователь может изменять владельца файла. Владелец файла может изменять группу файла на любую группу, к которой он принадлежит. Суперпользователь может произвольно изменять группу.

cp - копирование файлов

cp файл1 [файл2 ...] целевой_файл

Команда cp копирует файл1 в целевой_файл. Файл1 не должен совпадать с целевым_файлом (будьте внимательны при использовании метасимволов shell'a). Если целевой_файл является каталогом, то файл1, файл2, ..., копируются в него под своими именами. Только в этом случае можно указывать несколько исходных файлов.

Если целевой_файл существует и не является каталогом, его старое содержимое теряется.

Режим, владелец и группа целевого_файла при этом не меняются.

Если целевой_файл не существует или является каталогом, новые файлы создаются с теми же режимами, что и исходные (кроме бита навязчивости, если Вы не суперпользователь). Время последней модификации целевого_файла (и последнего доступа, если он не существовал), а также время последнего доступа к исходным файлам устанавливается равным времени, когда выполняется копирование. Если целевой_файл был ссылкой на другой файл, все ссылки сохраняются, а содержимое файла изменяется.

mv - перемещение (переименование) файлов

Синтаксис команды:

mv [-f] файл1 [файл2 ...] целевой_файл

Команда mv перемещает (переименовывает) файл1 в целевой_файл. Файл1 не должен совпадать с целевым_файлом (будьте внимательны при использовании метасимволов shell'a). Если целевой_файл является каталогом, то файл1, файл2, ..., перемещаются в него под своими именами. Только в этом случае можно указывать несколько исходных файлов.

Если целевой_файл существует и не является каталогом, его старое содержимое теряется. Если при этом обнаруживается, что в целевой_файл не разрешена запись, то выводится режим этого файла [см. chmod] и запрашивается строка со стандартного ввода. Если эта строка начинается с символа y, то требуемые действия все же выполняются, при условии, что у пользователя достаточно прав для удаления целевого_файла. Если была указана опция -f или стандартный ввод назначен не на терминал, то требуемые действия выполняются без всяких запросов. Вместе с содержимым целевой_файл наследует режим файла1.

Если файл1 является каталогом, то он переименовывается в целевой_файл, только если у этих двух каталогов общий надкаталог; при этом все файлы, находившиеся в файле1, перемещаются под своими именами в целевой_файл. Если файл1 является файлом, а целевой_файл - ссылкой, причем не единственной, на другой файл, то все остальные ссылки сохраняются, а целевой_файл становится новым независимым файлом.

rm - удаление файлов

Синтаксис команды:

rm [-f] [-i] файл ...

rm -r [-f] [-i] каталог ... [файл ...]

Команда rm служит для удаления указанных имен файлов из каталога. Если заданное имя было последней ссылкой на файл, то файл уничтожается. Для удаления пользователь должен обладать правом записи в каталог; иметь право на чтение или запись файла не обязательно. Следует заметить, что при удалении файла в Linux, он удаляется навсегда. Здесь нет возможностей вроде "мусорной корзины" в windows 95/98/NT или команды undelete в DOS. Так что, если файл удален, то он удален!

Если нет права на запись в файл и стандартный ввод назначен на терминал, то выдается (в восьмеричном виде) режим доступа к файлу и запрашивается подтверждение; если оно начинается с буквы y, то файл удаляется, иначе - нет. Если стандартный ввод назначен не на терминал, команда rm ведет себя так же, как при наличии опции -f.

Допускаются следующие три опции:

-f Команда не выдает сообщений, когда удаляемый файл не существует, не запрашивает подтверждения при удалении файлов, на запись в которые нет прав. Если нет права и на запись в каталог, файлы не удаляются. Сообщение об ошибке выдается лишь при попытке удалить каталог, на запись в который нет прав (см. опцию -r).

-r Происходит рекурсивное удаление всех каталогов и подкаталогов, перечисленных в списке аргументов. Сначала каталоги опустошаются, затем удаляются. Подтверждение при удалении файлов, на запись в которые нет прав, не запрашивается, если задана опция -f или стандартный ввод не назначен на терминал и не задана опция -i. При удалении непустых каталогов команда rm -r предпочтительнее команды rmdir, так как последняя способна удалить только пустой каталог. Но команда rm -r может доставить немало острых впечатлений при ошибочном указании каталога!

-i Перед удалением каждого файла запрашивается подтверждение. Опция -i устраняет действие опции -f; она действует даже тогда, когда стандартный ввод не назначен на терминал.

ПРИМЕРЫ Опция -i часто используется совместно с -r. По команде:

rm -ir dirname

запрашивается подтверждение:

directory dirname: ?

При положительном ответе запрашиваются подтверждения на удаление всех содержащихся в каталоге файлов (для подкаталогов выполняются те же действия), а затем подтверждение на удаление самого каталога.

rmdir - удаление каталогов

Синтаксис команды:

rmdir [-p] [-s] каталог ...

Команда rmdir удаляет указанные каталоги, которые должны быть пустыми. Для удаления каталога вместе с содержимым следует воспользоваться командой rm с опцией -r. Текущий каталог [см. pwd] не должен принадлежать поддереву иерархии файлов с корнем - удаляемым каталогом.

Для удаления каталогов нужно иметь те же права доступа, что и в случае удаления обычных файлов [см. rm].

Командой rmdir обрабатываются следующие опции:

-p Позволяет удалить каталог и вышележащие каталоги, оказавшиеся пустыми. На стандартный вывод выдается сообщение об удалении всех указанных в маршруте каталогов или о сохранении части из них по каким-либо причинам.

-s Подавление сообщения, выдаваемого при действии опции -p.

ln - создание ссылки на файл

Синтаксис команды:

ln [-f] файл1 [файл2 ...] целевой_файл

Команда ln делает целевой_файл ссылкой на файл1. Файл1 не должен совпадать с целевым_файлом (будьте внимательны при использовании метасимволов shell'a). Если целевой_файл является каталогом, то в нем создаются ссылки на файл1, файл2, ... с теми же именами. Только в этом случае можно указывать несколько исходных файлов.

Если целевой_файл существует и не является каталогом, его старое содержимое теряется. Если при этом обнаруживается, что в целевой_файл не разрешена запись, то выводится режим доступа к этому файлу [см. chmod] и запрашивается строка со стандартного ввода. Если эта строка начинается с символа y, то требуемые действия все же выполняются, при условии что у пользователя достаточно прав для удаления целевого_файла. Если была указана опция -f или стандартный ввод назначен не на терминал, то требуемые действия выполняются без всяких запросов. Целевой_файл наследует режим доступа к файлу1.

ОГРАНИЧЕНИЯ

Команда ln не создает ссылок между разными файловыми системами, поскольку они (файловые системы) могут добавляться и удаляться.

mkdir – создание каталога

mkdir [-m режим_доступа] [-p] каталог ...

По команде mkdir создается один или несколько каталогов с режимом доступа 0777 [возможно измененном с учетом umask и опции -m]. Стандартные файлы (. - для самого каталога и .. - для вышележащего) создаются автоматически; их нельзя создать по имени. Для создания каталога необходимо располагать правом записи в вышележащий каталог.

Идентификаторы владельца и группы новых каталогов устанавливаются соответственно равными реальным идентификаторам владельца и группы про-

цесса.

Командой mkdir обрабатываются две опции:

-m режим_доступа - (явное задание режима_доступа для создаваемых каталогов [см. chmod]).

-p (при указании этой опции перед созданием нового каталога предварительно создаются все несуществующие вышележащие каталоги).

ПРИМЕРЫ Чтобы создать поддерево каталогов tmpdir/temp/dir, надо выполнить команду

mkdir -p tmpdir/temp/dir

grep - поиск образца в файле

Выполняет поиск образца в текстовых файлах и выдает все строки, содержащие этот образец. Она использует компактный недетерминированный алгоритм сопоставления.

find - поиск файлов

Синтаксис команды:

find список_поиска выражение

Рекурсивно просматривает каждый из каталогов, перечисленных в списке_поиска, отыскивая файлы, удовлетворяющие логическому выражению.

lspci – отображает список PCI устройств компьютера.

lsusb – отображает список USB-устройств компьютера

4.3 Задание на работу и методические указания по выполнению работы

В рамках выполнения данной лабораторной работы студентам необходимо выяснить системные характеристики компьютера. Для этого необходимо воспользоваться консольными командами Linux. Необходимо выяснить следующие характеристики:

- тип процессора, его тактовую частоту, архитектуру;
- размер оперативной памяти компьютера;
- размер swap-области (области подкачки);
- размер свободной и занятой оперативной памяти и области подкачки;
- размер дисковой подсистемы компьютера;
- размер и занятость разделов файловой системы;
- список всех выполняемых процессов на компьютере;
- список выполняемых процессов упорядоченный по использованию процессорного времени (в обратном порядке);
- список выполняемых процессов упорядоченный по использованию памяти;
- список PCI устройств, подключенных к компьютеру;

- список USB-устройств, подключенных к компьютеру.
-

4.4 Содержание отчета

Результаты выполнения работы необходимо представить в отчете. Результаты должны быть оформлены с указанием выполняемых команд и их параметров. Возможно в качестве результатов приводить как листинги (списки) так и скриншоты (снимки экрана).

Лабораторная работа №5 Обработка событий клавиатуры

5.1 Цель работы

Данная лабораторная работа посвящена обработке данных поступающих с клавиатуры. Работа выполняется с использованием системы программирования Free Pascal.

5.2 Теоретическая часть

5.2.1 Клавиатура

Клавиатура - это одно из основных устройств ввода информации в ЭВМ, позволяющее вводить различные виды информации. Вид вводимой информации определяется программой, интерпретирующей нажатые или отпущенные клавиши. С помощью клавиатуры можно вводить любые символы - от букв и цифр до иероглифов и знаков музыкальной нотации. Клавиатура позволяет управлять курсором на экране дисплея -устанавливать его в нужную точку экрана, перемещать по экрану “прокручивать” экран в режиме скроллинга, отправлять содержимое экрана на принтер, производить выбор при наличии альтернативных вариантов и т.д.

В последнее время наблюдаются тенденции отказа от клавиатуры в пользу альтернативных устройств: мыши, речевого ввода, сканеров. Но полностью эти устройства клавиатуру не заменяют.

Стандартная клавиатура IBM PC имеет несколько групп клавиш:

Алфавитно-цифровые и знаковые клавиши (с латинскими и русскими буквами, цифрами, знаками пунктуации, математическими знаками).

Специальные клавиши: <Esc>, <Tab>, <Enter>, <BackSpace>.

Функциональные клавиши: <F1>...<F10...>.

Служебные клавиши для управления перемещением курсора (стрелки: <Up>, <Down>, <Left>, <Right>, клавиши <Home>, <End>, <PgUp>, <PgDn> и клавиша, обозначенная значком “[ ]” - в центре дополнительной цифровой клавиатуры).

Служебные клавиши для управления редактированием <Ins> <Del>.

Служебные клавиши для смены регистров и модификации кодов других клавиш <Alt>, <Ctrl>, <Shift>.

Служебные клавиши для фиксации регистров <CapsLock>, <Scroll-Lock>, <NumLock>.

Разные вспомогательные клавиши <PrtSc>, <Break>, <Grey +>, <Grey ->.

Если клавиша первой, четвертой, а иногда и пятой группы оказывается нажатой дольше, чем 0,5 с, начинает генерироваться последовательность ее основных кодов с частотой 10 раз в секунду (в IBM PC XT), что имитирует серию

очень быстрых нажатий этой клавиши. Общее число клавиш в основной модификации клавиатуры - 83, в расширенной клавиатуре – до 101. Количество различных сигналов от клавиатуры значительно превышает это число, так как:

1) при нажатии и освобождении клавиши в ЭВМ передаются разные кодовые комбинации: при нажатии - порядковый номер нажатой клавиши на клавиатуре (ее скан-код), а при освобождении - скан-код, увеличенный на 80h;

2) заглавные и строчные буквы первой группы клавиш (алфавитно-цифровых и знаковых) набираются на разных регистрах. Оперативное переключение регистров производится клавишей <Shift>. Если при нажатой (и удерживаемой в нажатом состоянии) клавише <Shift> “клюнуть” (от английского слова “dick”) любую алфавитную клавишу, то в ЭВМ будет отправлен код заглавной буквы, соответствующий нажатой клавише;

3) после однократного нажатия клавиши <CapsLock> (зажигается лампочка на клавиатуре рядом с клавишей) изменяется порядок работы клавиши <Shift>: без нажатия на нее будут набираться заглавные буквы, а при нажатии (совместном) - строчные. После повторного нажатия на <CapsLock> порядок работы клавиши <Shift> восстанавливается, а лампочка гаснет. Такой режим (переключательный) работы клавиши называется триггерным режимом, или flip-flop;

4) аналогично клавише <Shift> действуют <Alt> и <Ctrl> - при одновременном нажатии с ними любой другой клавиши, в ЭВМ передается не scancode, а расширенный код (2 байта). Иногда таким же образом используется клавиша <Esc>;

5) клавиша <NumLock> является триггерным переключателем дополнительной цифровой клавиатуры: при негорящей лампочке она работает как клавиатура для управления курсором; при зажженной - как цифровая;

6) для переключения регистров (или даже групп регистров) иногда используются другие комбинации клавиш: например, программы - русификаторы клавиатуры переключают РУС-ЛАТ с помощью правой клавиши <Shift> или при одновременном нажатии двух клавиш <Shift> (правой и левой) и т.д. Эти комбинации клавиш обладают триггерным эффектом.

Сигналы, поступающие от клавиатуры, проходят трехуровневую обработку: на физическом, на логическом и на функциональном уровнях.

Физический уровень имеет дело с сигналами, поступающими в вычислительн машину при нажатии и отпускании клавиш.

На логическом уровне, реализуемом BIOS через прерывание 9, скан-код транслируется в специальный 2-байтовый код. Младший байт для клавиш группы 1 содержит ASCII-код, соответствующий изображенному на клавише знаку. Этот байт называют главным. Старший байт (вспомогательный) содержит исходный скан-код нажатой клавиши.

На функциональном уровне отдельным клавишам программным путем приписываются определенные функции. Такое “программирование” клавиш осуществляется с помощью драйвера-программы, обслуживающей клавиатуру в

операционной системе.

На IBM PC AT используется клавиатура с большим количеством клавиш. На этих машинах есть возможность управлять некоторыми функциями клавиатуры, например, изменять время ожидания автоповтора, частоту автоповтора, зажигать и гасить светодиоды на панели управления клавиатурой.

Устройство клавиатуры не является простым: в клавиатуре используется свой микропроцессор, работающий по прошитой в ПЗУ программе. Контроллер клавиатуры постоянно опрашивает клавиши, определяет, какие из них нажаты, проводит контроль на “дребезг” и выдает код нажатой или отпущен-_ ной клавиши в системный блок ЭВМ.

Выпускаемые разными производителями клавиатуры различаются также по расстоянию между клавишами, числу специальных клавиш, способу переключения на цифровой регистр для быстрого ввода числовых данных, углу наклона, форме и текстуре поверхности клавиш, усилию нажима и величине хода клавиш, расположению часто используемых клавиш и др.

5.2.2 Система программирования Free Pascal

Free Pascal Compiler (FPC) - это свободно распространяемый компилятор языка Паскаль с открытыми исходными кодами. Он совместим с Borland Pascal 7 и Object Pascal – Delphi, но при этом обладает рядом дополнительных возможностей, например, поддерживает перегрузку операторов. FPC — кросс-платформенный инструмент, поддерживающий огромное количество платформ. Среди них — AmigaOS, DOS, Linux, *BSD, OS/2, MacOS(X) и Win32.

В состав Free Pascal входит большое количество различных библиотек, реализующих в том числе функции работы с периферийными устройствами. Так, в частности, библиотека Crt реализует функции работы с клавиатурой, консолью.

Модуль Crt реализует ряд мощных программ, предоставляющих вам полную возможность управления средствами компьютера PC, такими, как управление режимом экрана, расширенные коды клавиатуры, цвета, окна, и звуковые сигналы.

Одним из основных преимуществ использования модуля Crt является большая скорость и гибкость при выполнении операций работы с экраном. Программы, не работающие с модулем Crt, выводят на экран информацию с помощью средств операционной системы, что связано с дополнительными непроизводительными затратами. При использовании модуля Crt выводимая информация посылается непосредственно в базовую систему ввода-вывода (BIOS), или, для еще более быстрых операций, непосредственно в видеопамять.

Использование модуля CRT

Чтобы использовать модуль Crt, его нужно указать в операторе uses вашей программы:

```
uses Crt;
```

При инициализации модуля Crt для того, чтобы можно было обращаться к CRT, вместо стандартных файлов ввода и вывода DOS назначаются стандартные входные и выходные текстовые файлы. Это соответствует выполнению в начале программы следующих операторов:

AssignCrt(Input); Reset(Input);

AssignCrt(Output); Rewrite(Output);

Это означает, что переопределение входных и выходных файлов далее не допускается до тех пор, пока для данных файлов не будет выполнено обратного переназначения и не произойдет переход к стандартному вводу и выводу с помощью выполнения операторов:

Assing(Input,''); Reset(Input);

Assing(Output,''); RewriteOutput);

Окна CRT

Модуль Crt поддерживает простую, но, тем не менее, мощную форму использования окон. Процедура Window позволяет вам определить в каком-либо месте экрана окно. При записи в это окно оно ведет себя точно также, как целый экран. При этом остальная часть экрана остается нетронутой. Другими словами, доступ к экрану вне окна отсутствует. Внутри окна можно добавлять и удалять строки, при этом курсор возвращается к правому краю и при достижении курсором нижней строки текст продвигается вверх.

Все координаты экрана, кроме тех, которые используются для определения окна, относятся к текущему окну. Координата экрана (1,1) соответствует левому верхнему углу экрана. По умолчанию окном считается весь экран.

Специальные символы

При записи в выходной файл или в файл, который назначен для модуля Crt, специальное значение имеют следующие управляющие символы:

#7 Звонок - Вызывает звуковой сигнал, издаваемый с помощью внутреннего динамика.

#8 Обратный пробел - Возврат на одну позицию. Вызывает перемещение курсора влево на одну позицию. Если курсор уже находится у левого края текущего окна, то никаких действий не производится.

#10 Перевод строки - Перемещает курсор на одну строку вниз. Если курсор уже находится на нижней строке окна, то окно пролистывается вверх на одну строку.

#13 Возврат каретки - Возвращает курсор с левому краю текущего окна.

Ввод строк

При чтении из входного файла (Input) или из текстового файла, который назначен для модуля Crt, текст вводится по одной строке. Строка запоминается во внутреннем буфере текстового файла и когда переменные считываются, то в качестве источника используется этот буфер. Каждый раз когда буфер становится пустым, вводится новая строка. При вводе строк можно использовать следующие клавиши редактирования:

Backspace - Удаляет последний введенный символ.

Esc - Удаляет всю вводимую строку.

Enter - Прекращает ввод строки и записывает метку конца строки (возврат каретки/перевод строки) в буфере.

Ctrl+S - Действует также, как Backspace.

Ctrl+D - Извлекает один символ из последней вводимой строки и выводит его на экран.

Ctrl+F - Восстанавливает на экране последнюю вводимую строку.

Ctrl+Z - Завершает ввод строки и генерирует символ конца файла.

`Для проверки состояния клавиатуры и ввода отдельных символов под управлением программы используйте функции KeyPressed и ReadKey.

Процедуры и функции модуля Crt

AssignCrt - Назначает текстовый файл для устройства CRT.

ClrEol - Очищает все символы, начиная от позиции курсора до конца строки, без перемещения курсора.

ClrScr - Очищает экран и помещает курсор в верхнем левом углу.

Delay - Выполняет задержку на указанное число миллисекунд.

DelLine - Удаляет строку, на которой находится курсор и перемещает все следующие строки на одну строку вверх. Нижняя строка очищается.

GotoXY - Выполняет позиционирование курсора. X – это горизонтальная позиция, Y - вертикальная позиция.

InsLine - Вставляет пустую строку в месте расположения курсора.

KeyPressed - Возвращает значение True, если клавиша на клавиатуре нажата и False - в противном случае.

TextBackground - Выбирает фоновый цвет.

TextColor - Выбирает цвет самого символа.

TextMode - Выбирает конкретный текстовый режим.

Window - Определяет на экране текстовое окно.

Readkey - Считывает символ с клавиатуры.

WhereX - Возвращает координату X для текущей позиции курсора, относящуюся к текущему окну. X представляет собой горизонтальную позицию.

WhereY - Возвращает координату Y для текущей позиции курсора, относящуюся к текущему окну. Y представляет собой вертикальную позицию.

Подробную информацию о модуле Crt и его процедурах и функциях вы можете получить с помощью справочной системы программы Free Pascal.

Модуль keyboard

Весьма эффективным инструментом для низкоуровневой работы с клавиатурой обладает стандартный модуль Free Pascal keyboard. Приведем краткий перечень основных функций и процедур этого модуля:

DoneKeyboard – завершает работу с драйвером клавиатуры

FunctionKeyName – возвращает строку представляющую код функциональной клавиши

GetKeyEvent – возвращает следующее событие клавиатуры

GetKeyEventChar – возвращает символьную часть события клавиатуры

GetKeyEventCode – возвращает функциональную часть события

GetKeyEventFlags – возвращает установленные флаги клавиатурного события

GetKeyEventShiftState – возвращает текущее состояние клавиш Shift

GetKeyEventUniCode – возвращает клавиатурное событие в формате Unicode

InitKeyboard – инициализирует драйвер клавиатуры

IsFunctionKey – возвращает значение Истина если нажатая клавиша – функциональная

KeyEventToString – возвращает строковое описание клавиатурного события

KeyPressed – проверяет наличие клавиатурного события в очереди

Более полную информацию о функциях, процедурах, константах модуля Keyboard смотрите самостоятельно в документации к Free Pascal.

5.3 Задание на работу и методические указания по выполнению работы

В рамках выполнения данной лабораторной работы необходимо написать программу на языке Free Pascal, обеспечивающую обработку событий, поступающих с клавиатуры. Описание событий должно выводиться в нижней части экрана (это может быть последняя строка).

5.4 Содержание отчета

По результатам работы нужно подготовить отчет с исходным текстом программы и таблицей устанавливающей соответствие нажатым клавишам кодов клавиатуры.

Лабораторная работа №6. Исследование различных систем счисления

6.1 Цель работы

Целью данной лабораторной работы является изучение различных систем счисления. Работа выполняется с использованием системы программирования Free Pascal.

6.2 Теоретическая часть

Система счисления — символический метод записи чисел, представление чисел с помощью письменных знаков. Для начала проведём границу между числом и цифрой.

Число — это некоторая абстрактная сущность для описания количества.

Цифры — это знаки, используемые для записи чисел.

Цифры бывают разные: самыми распространёнными являются арабские цифры, представляемые известными нам знаками от нуля (0) до девяти (9); менее распространены римские цифры, мы их можем иногда встретить на циферблате часов или в обозначении века (XIX век).

Итак:

число — это абстрактная мера количества;

цифра — это знак для записи числа.

Поскольку чисел гораздо больше чем цифр, то для записи числа обычно используется набор (комбинация) цифр. Только для небольшого количества чисел — для самых малых по величине — бывает достаточно одной цифры. Существует много способов записи чисел с помощью цифр. Каждый такой способ называется системой счисления. Величина числа может зависеть от порядка цифр в записи, а может и не зависеть. Это свойство определяется системой счисления и служит основанием для простейшей классификации таких систем.

Итак, указанное основание позволяет все системы счисления разделить на три класса (группы):

- позиционные;
- непозиционные;
- смешанные.

Позиционные системы счисления — это системы счисления, в которых значение цифры напрямую зависит от её положения в числе. Например, число 01 обозначает единицу, 10 — десять. Позиционные системы счисления позволяют легко производить арифметические расчёты. Представление чисел с помощью арабских цифр — самая распространённая позиционная система счисления, она называется «десятичной системой счисления». Десятичной системой она называется потому, что использует десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9.

Заметьте: максимальная цифра (9) на единичку меньше количества цифр (10).

Для составления машинных кодов удобно использовать не десятичную, а двоичную систему счисления, содержащую только две цифры, 0 и 1. Таким образом, в двоичной системе максимальная цифра 1.

Программисты для вычислений также пользуются ещё восьмеричной и шестнадцатеричной системами счисления.

Количество цифр используемых в системе счисления называется её «основанием». В десятичной системе основание равно десяти, в двоичной системе — двум, а в восьмеричной и шестнадцатеричной — соответственно, восьми и шестнадцати. То есть в p-ичной системе счисления количество цифр равно p и используются цифры от 0 до $p-1$.

В общем случае в позиционной системе счисления числа представляются следующим образом: $\boxed{a}_n a_{n-1} \ldots a_0 \boxed{f}$, где a_0, a_1,..., a_n — цифры, а f — основание системы счисления. Если используется десятичная система, то f — можно опустить.

Примеры чисел:

11001_2 — число в двоичной системе счисления, $a_0 = 1$, $a_1 = 0$, $a_2 = 0$, $a_3 = 1$, $a_4 = 1$;

31_8 — число в восьмеричной системе счисления, $a_0 = 1$, $a_1 = 3$;

25_{10} — число в десятичной системе счисления, $a_0 = 5$, $a_1 = 2$;

6.3 Задание на работу и методические указания по выполнению работы

На языке FreePascal необходимо написать программу, которая производит преобразование вводимых с клавиатуры чисел в следующие системы счисления:

- десятичную;
- двоичную;
- восьмеричную;
- шестнадцатиричную.

При этом необходимо с клавиатуры вводить число в разных системах счисления. Систему счисления вводимого числа можно задавать либо с помощью экранного меню, либо с помощью спецсимволов.

6.4 Содержание отчета

По результатам работы необходимо подготовить отчет с приведением текста программы и скриншотов, демонстрирующих ее выполнение.

Лабораторная работа №7. Создание программы-демона

7.1 Цель работы

Целью данной лабораторной работы является создание программы-демона, которая может выполнять поступающие к ней задания пользователя.

7.2 Теоретическая часть

Де́мон (англ. daemon) — в системах класса UNIX — служба, работающая в фоновом режиме без прямого общения с пользователем.

Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: серверы сетевых протоколов (HTTP, FTP, электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается процесс, который не имеет управляющего терминала. Чаще всего (но не обязательно) предком демона является init — корневой процесс UNIX.

7.3 Задание на работу и методические указания по выполнению работы

На языке FreePascal или на языке скриптов Bash необходимо написать программу, которая периодически (раз в 10 секунд, например) просматривает определенную папку, куда помещаются файл (или файлы) с заданиями пользователя. Задания пользователя представляют собой командные строки которые просто необходимо выполнить. Демон должен выполнить задания пользователя и удалить файлы с завершенными заданиями.

7.4 Содержание отчета

По результатам работы необходимо подготовить отчет с приведением текста программы и скриншотов, демонстрирующих ее выполнение.

Лабораторная работа №8. Работа с регулярными выражениями

8.1 Цель работы

Целью данной лабораторной работы является изучение регулярных выражений и создание программы, определяющей правильность ввода форматированной информации.

8.2 Теоретическая часть

Регулярные выражения (англ. regular expressions — система обработки текста, основанная на специальной системе записи образцов для поиска. Образец (англ. pattern), задающий правило поиска, по-русски также иногда называют «шаблоном», «маской».

Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Многие языки программирования уже поддерживают регулярные выражения для работы со строками. Например, Perl и Tcl имеют встроенный в их синтаксис механизм обработки регулярных выражений. Набор утилит (включая редактор sed и фильтр grep), поставляемых в дистрибутивах Unix, одним из первых способствовал популяризации понятия регулярных выражений.

8.3 Задание на работу и методические указания по выполнению работы

Изучить синтаксис регулярных выражений на основе документации Linux. Возможно также использование источников сети Интернет.

С помощью языка PHP создать web-приложение со следующим функционалом:

- страница с html-формой со следующими полями:
  - Фамилия Имя;
  - адрес электронной почты;
  - адрес web-ресурса;
- по нажатию на кнопку "Отправить" скриптом производится проверка на правильность заполнения полей формы и пользователь получает отчет.

8.4 Содержание отчета

По результатам работы необходимо подготовить отчет с приведением текстов скриптов и скриншотов, демонстрирующих ее выполнение.

Лабораторная работа №9 Работа с архивами в Linux

9.1 Цель работы

Целью данной лабораторной работы является изучение особенностей упаковки и распаковки данных в операционной системе Linux.

9.2 Теоретическая часть

Утилита tar предназначена для создания архивов файлов и каталогов. С помощью этой программы можно архивировать файлы, обновлять их в архиве и вводить в этот архив новые файлы. Можно архивировать и целые каталоги со всеми их файлами и подкаталогами. При необходимости все эти файлы и подкаталоги можно восстановить из архива. Программа tar предназначалась для создания архивов на лентах, отсюда и название tar (tape archive, т.е. "архив на ленте"). Архив можно создавать на любом устройстве, например на дискете или в архивном файле на диске. Программа tar - идеальное средство для создания резервных копий файлов или объединения нескольких файлов в один с целью передачи его по сети.

В операционной системе Linux программу tar часто используют для создания архивов на устройствах и в файлах. Ей можно дать указание архивировать файлы на определенном устройстве или в определенном файле, для чего служит опция f с именем устройства или файла. Синтаксис команды tar с опцией f очевиден из нижеследующего примера. Имя устройства или файла часто называют именем архива. При создании файла для tar-архива к имени этого файла обычно добавляется расширение .tar Это условное обозначение; оно не обязательно. В команде можно указать сколько угодно имен файлов. Если указано имя каталога, то в архив включаются и все подкаталоги этого каталога.

```
$ tar опцииf имя_архива.tar имена_файлов_и_каталогов
```

Для создания архива служит опция c. В сочетании с опцией f опция c приводит к созданию архива в файле или на устройстве. Эта опция ставится непосредственно перед опцией f. Обратите внимание, что дефиса перед опцией нет. В следующем примере каталог mydir и все его подкаталоги сохраняются в файле myarch.tar.

```
$ tar cf myarch.tar mydir
```

Потом пользователь может извлекать каталоги из архива, применяя команду tar с опцией x. Опция xf позволяет извлекать файлы из архивного файла или устройства. При извлечении формируются и все подкаталоги. В следующем примере посредством опции xf команде tar дается указание извлечь все файлы и подкаталоги из файла myarch.tar.

```
$ tar xf myarch.tar
```

Для добавления файлов в существующий архив служит опция r. В приведенном ниже примере пользователь добавляет файлы из каталога letters в архив myarch.tar.

```
$ tar rf myarch.tar letters
```

Если нужно изменить какой-либо файл в архивированных ранее каталогах, можно с помощью опции u дать команде указание обновить архив, заменив некоторые файлы их новыми версиями. Программа tar сравнивает время последнего изменения каждого архивированного файла и соответствующего файла в каталоге и копирует в архив все файлы с более поздней датой модификации. В архив будут добавлены и все вновь созданные в этих каталогах файлы. В следующем примере пользователь обновляет файл myarch.tar, вводя в него все измененные и вновь созданные в каталоге mydir файлы.

```
$ tar uf myarch.tar mydir
```

Если вы хотите посмотреть, какие файлы хранятся в архиве, дайте команду tar с опцией t. В следующем примере показано, как с помощью этой команды можно вызвать список всех файлов, хранящихся в архиве myarch.tar.

```
$ tar tf myarch.tar
```

Для создания резервных копий файлов на определенном устройстве укажите имя этого устройства в качестве имени архива. В следующем примере пользователь создает архив на дискете в устройстве /dev/fd0 и копирует в него все файлы из каталога mydir.

```
$ tar cf /dev/tdO mydir
```

Для того чтобы извлечь архивированные таким образом файлы, используйте опцию xf.

```
$ tar xf /dtv/fd0
```

Если архивируемые файлы занимают больше места, чем имеется на носителе, например на дискете, создайте tar-архив, состоящий из нескольких томов (дискет или лент).

Посредством опции M команде tar дается указание выводить сообщение о том, что текущий носитель заполнен. При архивировании файлов на дискете с использованием опции M в случае заполнения дискеты программа tar предложит вам вставить новую дискету. Таким образом вы сможете записать свой ар-

хив на нескольких дискетах.

```
$ tar cMf /dev/fd0 mydir
```

Чтобы распаковать архив, записанный на нескольких дискетах, вставьте первую дискету в дисковод и введите команду tar с опциями x и M, как показано ниже. Программа подскажет вам, когда надо вставить следующую дискету.

```
$ tar xMf /dev/fd0
```

При использовании команды tar операция сжатия архивных файлов не выполняется. Если вы хотите сжать файлы, дайте tar указание вызвать утилиту gzip. Если команда tar применяется с опцией z, то сначала программа gzip выполняет сжатие, а затем tar архивирует файлы. Та же опция z обеспечит вызов gzip для распаковки файлов при извлечении их из архива.

```
$ tar czf myarch.tar mydir
```

Помните, что между сжатием отдельных файлов с последующим архивированием и сжатием всего архива есть разница. Во многих случаях архив создается, чтобы переслать по сети несколько файлов в виде одного tar-файла. Для сокращения времени передачи размер этого архива должен быть по возможности небольшим. Чтобы добиться этого, можно с помощью утилиты gzip сжать архивный tar-файл, уменьшив его размер, а затем переслать сжатую версию. Получатель распакует его и восстановит файл. В результате применения утилиты gzip к tar -файлам часто получаются файлы с расши-рением .tar.gz. Расширение .gz добавляется к сжатому gzip-файлу. В следующем примере создается сжатая версия файла myarch.tar под тем же именем, но с расширением .gz.

```
$ gzip myarch.tar
$ la
myarch.tar.gz
```

Если вы хотите создать архив на некотором устройстве, например на ленте или в файле, нужно дать команду tar с опцией f и именем устройства или файла. Такой вариант эффективен при создании резервных копий файлов. Имя устройства по умолчанию хранится в файле /etc/default/tar. Синтаксис команды tar, подразумевающей использование устройства, заданного по умрлчанию (накопителя на магнитной ленте), приведен в показанном ниже примере. Опция f и имя устройства не задаются. Если указано имя каталога, то в архив включаются все его подкаталоги.

```
$ tar опция имена_каталогов_и_файлов
```

В представленном ниже примере каталог mydir со всеми подкаталогами

сохраняется на ленте как на носителе по умолчанию.

```
$ tar c mydir
```

А в этом примере каталог mydir со всеми файлами и подкаталогами извлекается из устройства, принятого по умолчанию, и помещается в рабочий каталог пользователя.

```
$ tar x mydir
```

Сжатие файлов: программа gzip

Уменьшать размер файла приходится по разным причинам. Чаще всего это делается для экономии места и, если вы пересылаете файл по сети, для экономии времени передачи. Сжатие и распаковка файлов осуществляются с помощью утилиты gzip. При сжатии в качестве аргумента вводится имя файла. Этот файл заменяется сжатой версией с расширением .gz.

$ gzip mydata
$ la
mydata.gz

Для распаковки grip-архива введите либо команду gzip с опцией -d, либо команду gunzip . Эти команды приводят к распаковке файла с расширением .gz и замене его распакованной версией с тем же именем, но без расширения .gz . При использовании команды gunzip не нужно даже вводить расширение .gz. Команды gunzip и gzip -d заведомо предполагают его наличие.

$ gunzip mydata.gz
$ ls
mydata

Пусть, например, вы хотите вывести на экран или напечатать содержимое сжатого файла, не распаковывая его. Команда zcat создает распакованную версию файла и посылает ее на стандартный вывод. Затем этот вывод можно переадресовать в утилиту печати или отображения, например в more. Оригинал файла остается записанным в сжатом виде.

$ zcat mydata.gz | more

Можно сжимать и архивированные файлы. Эта операция дает в результате файлы с расширением .tar.gz. Сжатые архивированные файлы часто используются для передачи очень больших файлов по сетям.

```
$ gzip myarch.tax
$ ls
myarch.tar.gz
```

Файлы, входящие в архив, можно сжимать и по отдельности, используя команду tar с опцией z, которая вызывает утилиту gzip. В этом случае файл сначала сжимается, а затем помещается в архив. Следует отметить, однако, что архивы с файлами, сжатыми с применением опции z, обновлению не подлежат, и добавлять в них файлы нельзя. Все файлы необходимо сжимать одновременно и добавлять тоже одновременно.

Для создания сжатых файлов можно также пользоваться командами compress и uncompress. В утилите compress используется другой формат сжатия. В результате ее использования образуются файлы с расширением .Z. Команды compress и uncompress применяются не очень широко, но файлы с расширением .Z иногда встречаются. Для распаковки файла с расширением .Z можно использовать не только команду uncompress, но и команду gunzip. Однако gzip является стандартной утилитой сжатия из набора программного обеспечения GNU, поэтому вместо команды compress по возможности следует использовать именно ее.

9.3 Задание на работу и методические указания по выполнению работы

Изучить синтаксис использвания утилит tar и gzip с помощью системы man.

Создать архив .gz в который поместить один текстовый файл

Добавить новый текстовый файл в созданный архив

Удалить файл из архива

Создать архив на основе папки с вложенными папками и файлами с помощью утилиты tar

Создать архив на основе папки с вложенными папками и файлами в формате .tgz

9.4 Содержание отчета

По результатам работы необходимо подготовить отчет с приведением команд и результатов работы.

Лабораторная работа №10 Работа с файлами в Linux

10.1 Цель работы

Целью данной лабораторной работы является изучение особенностей работы с файлами в операционной системе Linux.

10.2 Теоретическая часть

Файловая система — это структура, с помощью которой ядро операционной системы предоставляет пользователям (и процессам) ресурсы долговременной памяти системы, т. е. памяти на различного вида долговременных носителях информации — жестких дисках, магнитных лентах. CD-ROM и т. п.

С точки зрения ОС файл представляет собой непрерывный поток (или последовательность) байтов определенной длины. Внутренний формат файла операционную систему не интересует. Но ОС должна дать файлу какое-то имя, с помощью которого пользователь, а точнее, программы-приложения, будут обращаться к файлу. Как организовать это обращение — дело файловой системы, пользователя это чаще всего не интересует. Поэтому с точки зрения пользователя файловая система выглядит как логическая структура каталогов и файлов.

Имена файлов в Linux могут иметь длину до 255 символов и состоять из любых символов, кроме символа с кодом 0 и символа / (слэша). Однако имеется еще ряд символов, которые имеют в оболочке shell специальное значение и которые поэтому не рекомендуется включать в имена. Это следующие символы:

! @ # $ & ~ % * ( ) [ ] { } ' " \ : ; > < ` пробел.

Если имя файла содержит один из этих символов (это не рекомендуется, но возможно), то вы должны перед этим символом поставить символ обратного слэша "\" (в том числе и перед самим этим слэшем, т. е. повторить его дважды).

```
[user]$ mkdir \\my\&his
```

Можно также заключить имя файла или каталога с такими символами в двойные кавычки. Например, для создания каталога с именем "My old files" следует использовать команду:

```
[user]$ mkdir "My old files"
```

так как команда

```
[user]$ mkdir My old files
```

создаст каталог с именем "My".

Аналогичным образом можно поступать и с другими символами, перечисленными выше, т. е. их можно включать в имена файлов, если имя файла взять в двойные кавычки или отменить специальное значение символа с помощью обратного слэша. Но все же предпочтительнее не использовать эти символы, включая пробел, в именах файлов и каталогов, потому что могут возникнуть проблемы при обращении к таким файлам из некоторых приложений, а также при переносе таких файлов в другие файловые системы.

Но к точке сказанное не относится, и в Linux часто ставят более одной точки в именах файлов, например, This_is.a.forth-chapter_of_my_book.about.Linux. При этом теряет смысл такое понятие (принятое в DOS), как расширение имени файла, хотя все же часто последние части имени, отделенные точками, используют для обозначения файлов каких-то особых типов (например, .tar.gz используется для обозначения сжатых архивов). Но исполняемые и неисполняемые файлы в Linux распознаются не по расширениям имен файлов. Для этого существуют другие признаки, о которых мы скажем чуть позже. Точка имеет особое значение в именах файлов. Если она является первым символом имени, то данный файл считается скрытым для некоторых команд, например, он не показывается при выполнении команды ls.

В Linux различаются символы верхнего и нижнего регистра в именах файлов. Поэтому FILENAME.tar.gz и filename.tar.gz вполне могут существовать одновременно и являться именами разных файлов.

10.3 Задание на работу и методические указания по выполнению работы

Изучить синтаксис использования утилит работы с файлами в Linux с помощью системы man.

Получить список файлов в папке

Найти все файлы в файловой системе, которые начинаются с символа "a"

Написать на языке Bash скрипт, который выводит на экран содержимое всех файлов в указанной папке

10.4 Содержание отчета

По результатам работы необходимо подготовить отчет с приведением команд и результатов работы.

Лабораторная работа №11 Установка дистрибутива Linux

11.1 Цель работы

Изучить особенности установки дистрибутивов Linux на компьютер пользователя. Познакомиться с процедурой разбивки диска на тома и форматирование томов. Познакомиться со структурой каталогов файловой системы Linux.

11.2 Теоретическая часть

Для выполнения лабораторной работы будем использовать популярный дистрибутив Linux Ubuntu.

Ubuntu — это разрабатываемая сообществом, основанная на ядре Linux операционная система, которая идеально подходит для использования на персональных компьютерах, ноутбуках и серверах. Она содержит все необходимые программы, которые вам нужны: программу просмотра Интернет, офисный пакет для работы с текстами, электронными таблицами и презентациями, программы для общения в Интернет и много других.

Любой дистрибутив GNU/Linux позволяет заменять практически каждую свою часть на другую, и Ubuntu не исключение. Поэтому существует множество вариантов Ubuntu, отличающихся в первую очередь окружением пользователя и предустановленными программами.

Важной и основной особенностью всех этих систем является полная совместимость между собой, поскольку все они используют общую огромную базу программ, созданную для Ubuntu. То есть в процессе работы с любой из перечисленных ниже систем вы сможете легко доустановить нужные вам программы и компоненты, пусть даже присущие другому варианту Ubuntu.

Основным разработчиком Ubuntu является компания Canonical. Компания поддерживает два дистрибутива:

– Ubuntu - Базовая версия операционной системы, сочетающая простоту, удобство и функциональность. Основана на фирменной оболочке Unity и компонентах рабочей среды GNOME;

– Ubuntu Server - Серверный вариант операционной системы, включающий средства быстрого развёртывания облачной инфраструктуры, создания серверов LAMP, LTSP и прочих.

Кроме этого, сообщество разработчиков самостоятельно поддерживает развитие иных вариантов системы:

– Edubuntu - Вариант Ubuntu, нацеленный на использование в образовательных учреждениях. Содержит самые востребованные образовательные приложения;

– Kubuntu - Ubuntu с рабочим окружением KDE и программами, типичными для него;

– Lubuntu - Минималистичный вариант Ubuntu, основанный на рабочем окружении LXDE;.

– Mythbuntu - Мультимедийный вариант Ubuntu, содержащий программное обеспечение MythTV для создания домашних кинотеатров;

– Ubuntu Studio - Вариант Ubuntu, предназначенный для людей, активно зани-

мающихся редактированием и созданием мультимедийного контента;

– Xubuntu - Ubuntu с рабочим окружением Xfce. Отлично подходит для старых, маломощных компьютеров.

Нужный вариант системы выбирает сам пользователь. Наибольшее распространение получил, разумеется, Ubuntu. Тем не менее, имея установленный дистрибутив в системе, можно легко дополнить его иными вариантами и переключаться между ними в процессе работы. Так, например, в обычном Ubuntu в среде Unity можно пользоваться программами, написанными для KDE и наоборот. Возможно переключение между вариантами рабочей среды системы: между Unity и Gnome или XFCE, например.

Многие операционные системы поддерживают несколько файловых систем, например. Однако нужно различать файловые системы, которые могут использоваться в качестве корневой файловой системы, и файловые
системы, которые просто поддерживает ОС, но которые
не используются для установки Linux.

В качестве корневой файловой системы применяются следующие файловые системы.

ext	первая файловая система Linux, использовалась в ранних версиях Linux.
ext2	стандартная, но уже устаревшая файловая система Linux. Долгое время использовалась практически во всех дистрибутивах по умолчанию, но была заменена файловой системой ext3.
ext3	модифицированная версия файловой системы ext2, но с поддержкой журнала, существенно повышающего надежность файловой системы. Максимальный размер раздела с файловой системой ext3 — 4 Тбайт. Максимальный размер файла — 1 Тбайт.
ext4	новейшая файловая система Linux. Поддержка ext4 как стабильной файловой системы появилась в ядре Linux версии 2.6.28. Если сравнивать эту файловую систему с ext3, то производительность и надежность новой файловой системы существенно увеличена, а максимальный размер раздела теперь равен 1024 Пбайт. Максимальный размер файла больше 2 Тбайт.
ReiserFS	основная особенность этой файловой системы заключается в хранении в одном блоке нескольких маленьких файлов. Например, если у вас размер блока 4 Кбайт, то в него поместится до четырех файлов по одному килобайту каждый. Если у вас много маленьких файлов, то такая файловая система — просто находка, ведь она позволяет экономить дисковое пространство. Однако с большими файлами эта файловая система работает медленно, потом она чувствительна к сбоям и ее нужно регулярно дефрагментировать.
JFS	разработка IBM, обладает высокой производительностью, но оптимизирована под сервер баз данных, поскольку размер блока небольшой — от 512 байт до 4 Кбайт. Если приходится работать с большими файлами, например с видео, то файловая система — это не очень удачный выбор.
XFS	обладает относительно высокой производительностью — она быстрее, чем

	ext3, ReiserFS и JFS, но медленнее, чем ext4. Устанавливает большой размер блока — до 64 Кбайт, что позволяет ее использовать на графических станциях для обработки видео.

При установке Linux на выбранном разделе жесткого диска создается корневая файловая система Linux. Корневая файловая система содержит набор стандартных каталогов и утилит, без которых невозможна работа Linux.

Корневая файловая система обозначается как /. Полный путь к файлу обязательно начинается с корневой файловой системы. Вот полный путь к файлу report.doc, который находится в домашнем каталоге пользователя den: /home/den/report.doc.

/	Корневой каталог
/bin	Содержит стандартные утилиты Linux
/boot	Содержит конфигурационные файлы загрузчика GRUB, образы ядра, файлы Initrd
/dev	Содержит файлы устройств.
/etc	Содержит конфигурационные файлы операционной системы и всех сетевых служб. В Linux настройки хранятся в разных конфигурационных файлах, которые можно редактировать обычным текстовым редактором
/home	Содержит домашние каталоги пользователей. В домашних каталогах пользователей хранятся пользовательские файлы, а также пользовательские настройки различных программ
/lib	Здесь находятся различные библиотеки и модули ядра
/misc	В данном каталоге может быть все, что угодно
/mnt	Обычно в этом каталоге содержатся точки монтирования.
/proc	Это каталог псевдофайловой системы procfs, которая используется для предоставления информации о процессах
/root	Каталог пользователя root (пользователь с максимальными полномочиями)
/sbin	Набор утилит для системного администрирования
/tmp	“Мусорка”, т.е. каталог, в котором хранятся временные файлы.
/usr	Содержит пользовательские программы. По размеру это один из самых больших каталогов файловой системы. В этот каталог устанавливаются практически все программы. Также в этом каталоге находятся вспомогательные файлы, необходимые для работы установленных программ.
/var	Данные системы, которые постоянно изменяются, например, очередь печати, почтовые ящики и т.д.

11.3 Задание на работу и методические указания по выполнению работы

Задание на работу:

– с помощью установочного диска Ubuntu произвести установку операционной системы на компьютер;

– в процессе установки произвести разбивку диска таким образом, чтобы пользовательские данные находились на отдельном томе винчестера;

– создайте пользователя с логином user;

– после установки системы установите дополнительные программные пакеты: mc

11.4 Содержание отчета

По результатам работы необходимо подготовить отчет с приведением команд и результатов работы.

Лабораторная работа №12. Изучение среды рабочего стола KDE

12.1 Цель работы

Знакомство со средой рабочего стола KDE. Использование возможностей KDE и прикладных программ.

12.2 Теоретическая часть

KDE Software Compilation (KDE SC) — свободная среда рабочего стола и набор программ от проекта KDE. До начала 2010 года была известна как KDE (сокращение от K Desktop Environment). Построена на основе кросс-платформенного инструментария разработки пользовательского интерфейса Qt. Работает преимущественно под UNIX-подобными операционными системами, которые используют графические подсистемы X Window System и Wayland. Новое поколение технологии KDE 4 частично работает на Microsoft Windows и Mac OS X.

В состав KDE SC входит набор тесно интегрированных между собой программ для выполнения повседневной работы. Также в рамках проекта KDE разрабатываются интегрированная среда разработки KDevelop, офисный пакет Calligra Suite, музыкальный проигрыватель Amarok и многие другие. Эти программы не являются частью KDE SC.

KDE был основан в 1996 году Маттиасом Эттрихом, который в то время был студентом Тюбингенского университета. Его беспокоили проблемы UNIX-десктопа, одной из которых было отсутствие приложений, которые выглядели бы и вели себя одинаково. Он предложил не просто создание набора программ, а скорее среды для рабочего стола, в которой пользователь мог ожидать однородного поведения программ. Кроме того, он хотел сделать эту среду простой и понятной в эксплуатации.

В качестве инструментария разработки пользовательского интерфейса был выбран Qt. Инициатива получила распространение и стараниями разработчиков к началу 1997 года среда насчитывала уже достаточное количество приложений. На тот момент Qt не использовал свободную лицензию, и участники проекта GNU были обеспокоены тем фактом, что свободная среда и программы, входящие в её состав, создаются с использованием несвободных инструментов. Это послужило причиной создания двух проектов: «Harmony» и GNOME. Имея одинаковые цели (создание свободной среды свободными средствами), два проекта выбрали совершенно разные пути реализации задуманного. Проект Harmony ставил своей задачей переписать библиотеки Qt, выпустив их под свободной лицензией, проект GNOME — отказался полностью от использования Qt.

В ноябре 1998 года инструментарий Qt стал использовать свободную лицензию — open source Q Public License. Организациями Trolltech и специально созданной для этого KDE e.V. была основана KDE Free Qt Foundation, между которой и Trolltech было подписано соглашение, позволяющее KDE Free Qt Foundation в экстренном случае (прекращение разработки Qt Free Edition) выпустить Qt под лицензией типа BSD.

В сентябре 2000 года Trolltech выпускает UNIX-версию Qt под лицензией GNU General Public License, после чего споры, касающиеся лицензирования Qt, сошли на нет. Qt 4.0 доступна под лицензией GNU GPL для платформ *nix, Mac и Windows, что

позволяет приложениям и библиотекам KDE 4 иметь полную официальную поддержку на всех перечисленных платформах.

В основе KDE лежат следующие технологии:

– KDELibs — коллекция базовых библиотек KDE, общих для всех программ среды;

– KHTML — компонент для просмотра HTML документов;

– KIO — фреймворк, обеспечивающий прозрачный доступ к файлам, как по сети, так и локально;

– KWin — оконный менеджер;

– XMLGUI — позволяет генерировать элементы пользовательского интерфейса на основе файлов формата XML.

– Стандартные пакеты

– aRts — звуковой сервер (в KDE4 заменён на phonon).

– kdelibs — основные библиотеки, требуются для сборки других пакетов.

– kdepimlibs — библиотеки для PIM (для KDE4)

– kdebase — рабочий стол и основные приложения.

– kdeaccessibility — дополнительные программы для людей с ограниченными способностями (экранная лупа, синтезатор речи и т. д.).

– kdeaddons — дополнительные модули и скрипты.

– kdeadmin — инструменты графического администрирования.

– kdeartwork — содержит дополнительные темы, экранные заставки, звуки, обои и различные стили оформления окон.

– kdeedu — образовательное программное обеспечение.

– KDE Games — игры.

– kdegraphics — ПО для работы с графикой.

– kde-i18n — интернационализация; пакет для пользователей, которые хотят использовать в меню, справке и в приложениях языки, отличные от английского (в KDE4 заменён на kde-l10n).

– kdemultimedia — ПО для работы с файлами (и устройствами) мультимедиа.

– kdenetwork — инструменты для работы с сетью.

– kdepim — персональный органайзер.

– kdesdk — инструменты разработчика.

– kdetoys — бесполезные «игрушки».

– kdeutils — разнообразные утилиты.

– kdeplasmoids — пакет дополнительных плазмоидов и тем plasma (для kde4.1)

– kdewebdev — пакет программ для веб-разработчиков.

– Основные программы

– Amarok — проигрыватель аудиофайлов;

– Ark — архиватор;

– Calligra Suite — офисный пакет;

– digiKam — программа для управления коллекциями фотографий;

– Dolphin — файловый менеджер;

– Gwenview — просмотрщик изображений;

– K3b — программа для записи CD-, DVD- и BluRay-дисков;
– Kdenlive — видеоредактор;
– KDevelop — интегрированная среда разработки;
– KolourPaint — растровый графический редактор;
– Konqueror — веб-браузер, со множеством дополнительных возможностей;
– Konsole — эмулятор терминала;
– Kontact — персональный информационный менеджер, включающий клиент электронной почты, адресную книгу, планирование задач, календарь и многое другое;
– Kopete — мультипротокольный клиент мгновенных сообщений;
– Krusader — двухпанельный файловый менеджер;
– KStars — программа-планетарий;
– KTorrent — Bittorrent-клиент;
– KWallet — менеджер паролей;
– Okular — универсальный просмотрщик файлов различных типов, в частности, PDF, DjVu, FB2, CHM.

Помимо названных, в программную среду KDE входят множество других программ.

12.3 Задание на работу и методические указания по выполнению работы

Задание на работу:

– установить окружение рабочего стола KDE с помощью команды: sudo apt-get install kubuntu-desktop
– переключиться в среду KDE
– с помощью прикладных программ системы выполнить следующие задания:
– создать папку
– создать в ней текстовый файл с помощью программы kwrite
– скопировать этот файл на рабочий стол
– переименовать файл на рабочем столе

12.4 Содержание отчета

В отчете необходимо указать все команды которые были сипользованы, привести скриншоты рабочего стола и использованных программ.

Лабораторная работа №13. Изучение среды рабочего стола Gnome

13.1 Цель работы

Знакомство со средой рабочего стола Gnome. Использование возможностей Gnome и прикладных программ.

13.2 Теоретическая часть

GNOME — свободная среда рабочего стола для Unix-подобных операционных систем. GNOME является частью проекта GNU.

Разработчики GNOME ориентируются на создание полностью свободной среды, доступной всем пользователям вне зависимости от их уровня технических навыков, физических ограничений и языка, на котором они говорят. В рамках проекта GNOME разрабатываются как приложения для конечных пользователей, так и набор инструментов для создания новых приложений, тесно интегрируемых в рабочую среду.

GNOME — акроним от англ. GNU Network Object Model Environment («сетевая среда объектной модели GNU»). Под GNU в данном случае подразумевается не проект, а операционная система, официальной средой рабочего стола которой он является.

В основе среды GNOME лежит ряд библиотек и технологий. Некоторые из них разрабатываются как часть самого проекта GNOME, иные же являются результатом работы других проектов (например, freedesktop.org) и используются в других рабочих средах (KDE, Xfce).

В основном GNOME написана на языке Си, однако для библиотек GNOME существуют механизмы (так называемые привязки, англ. bindings), позволяющие использовать их из других языков. Поэтому многие приложения для GNOME пишутся на языках C++, Python, C# и других.

Центральную роль в GNOME играет инструментарий GTK+, который предоставляет средства для создания графических интерфейсов. В состав GTK+ также входят вспомогательные библиотеки:

GTK+ написан на Си, однако в последнее время всё больше GNOME-приложений разрабатываются на языках более высокого уровня. Это стало возможным благодаря тому, что в GTK+ изначально предусмотрена возможность относительно простого построения интерфейсов для других языков. Существуют надстройки для таких языков программировая, как C++ (gtkmm), Python (PyGTK), Perl (gtk2-perl), Java (java-gnome), Ruby (ruby-gnome2), C# (Gtk#), Tcl (Gnocl) и многих других. Только в программах, являющихся частью официального релиза GNOME, используются C, C++, C#, Python и Vala.

В качестве графических и мультимедиа-инструментов в GNOME используются некоторые проекты freedesktop.org.

Библиотека Cairo обеспечивает вывод векторной графики. Она используется в GTK+ для отрисовки элементов интерфейса.

Технология GStreamer обеспечивает «прозрачную» работу с аудио и видео раз-

личных форматов — ввод, обработку и вывод. Её используют, в частности, мультимедиапроигрыватель Totem и программа извлечения аудио с компакт-дисков Sound Juicer.

Poppler — библиотека отображения PDF-документов, основанная на xpdf. Она используется приложением просмотра документов Evince.

Tango Desktop Project — это попытка создать единый визуальный стиль для свободного программного обеспечения, в первую очередь в области значков. Официальная тема значков GNOME следует рекомендациям проекта Tango по внешнему виду значков, а также соответствует спецификациям наименования значков freedesktop.org.

Файловый менеджер Nautilus обеспечивает отрисовку рабочего стола со значками на нём, а также работу с файлами и директориями. Nautilus может работать в двух режимах: пространственном (англ. spatial) и режиме браузера. В первом режиме (по умолчанию в версиях 2.6 — 2.28)) каждая директория открывается в своём собственном окне, причём положение окон запоминается. Во втором режиме, подобно Проводнику Windows, перемещение по директориям производится в рамках одного окна, оснащённого панелями инструментов, деревом каталогов и другими элементами. Этот режим используется по умолчанию с версии 2.30. Начиная с версии 2.24, Nautilus поддерживает работу с вкладками.

В качестве оболочки по умолчанию, начиная с GNOME 3.0, используется GNOME Shell, основанная на оконном менеджере Mutter. Также доступна «Классическая оболочка», основанная на двух панелях. Есть возможность менять их количество, внешний вид, размер и набор апплетов панелей.

13.3 Задание на работу и методические указания по выполнению работы

Задание на работу:

- установить окружение рабочего стола GNOME с помощью команды: sudo apt-get install ubuntu-desktop;
- переключиться в среду Gnome;
- с помощью прикладных программ системы выполнить следующие задания:
  - создать папку;
  - создать в ней текстовый файл с помощью программы gedit;
  - скопировать этот файл на рабочий стол;
  - переименовать файл на рабочем столе.

13.4 Содержание отчета

В отчете необходимо указать все команды которые были использованы, привести скриншоты рабочего стола и использованных программ.

Лабораторная работа №14. Изучение среды рабочего стола XFCE

14.1 Цель работы

Знакомство со средой рабочего стола XFCE. Использование возможностей XFCE и прикладных программ.

14.2 Теоретическая часть

Xfce — свободная среда рабочего стола для UNIX-подобных операционных систем, таких как Linux, NetBSD, OpenBSD, FreeBSD, Solaris и т. п. Конфигурация данной среды полностью управляется мышью, конфигурационные файлы скрыты от пользователя.

Xfce основана на GTK+ 2 и использует собственный менеджер окон Xfwm. Начало своей истории Xfce берет с 1998 года. Тогда эта оболочка представляла собой дополнение к популярной тогда среде CDE, потому первоначально Xfce очень напоминала коммерческую CDE, но с каждой выпущенной версией всё дальше отходит от данной системы (Xfce была полностью переписана дважды — между версиями 2 и 3 и между версиями 3 и 4).

Xfce воплощает в себе традиционную философию UNIX, а именно концепции модульности и возможности многократного использования. Функциональные компоненты вынесены в отдельные приложения, и пользователь имеет возможность конфигурировать систему оптимальным образом.

Приложения

Mousepad — это простой текстовый редактор, написанный Эриком Харрисоном (Erik Harrison). Он является текстовым редактором по умолчанию среды Xfce. Согласно сайту Xfce, Mousepad основан на более старом текстовом редакторе Leafpad и был разработан с целью поддержки печати. За время существования Mousepad, его развитием занимались разные разработчики. Следующая версия Mousepad, 0.3.x, была переписана Ником Шермером (Nick Schermer).

Начиная с Xfce версии 4.8 Mousepad не поддерживается по причине несовместимости с новой библиотекой libxfce4ui и отсутствием деятельности по его разработке.

Orage. Начиная с версии 4.4, Xfcalendar был переименован в Orage и получил некоторые новые возможности. Orage имеет оповещения и использует формат iCalendar, что делает его совместимым с многими другими календарными приложениями. Он также включает плагин часов для панели и приложение международных часов, способных одновременно показывать часы с различными часовыми поясами.

Parole. Медиаплеер. Для проигрывания аудио и видеофайлов эта программа использует мультимедийный фреймворк GStreamer.

Ristretto. Программа для просмотра изображений.

Thunar — файловый менеджер по умолчанию для Xfce, заменяющий Xffm. Он напоминает Nautilus из GNOME и был разработан с расчётом на высокую скорость работы при низком потреблении памяти, при этом предоставляя возможность расширения функциональности посредством плагинов. Xfce также включает упрощённый

менеджер архивов Xarchiver, но он не является частью ядра Xfce 4.4.0.

Xfwm. Начиная с версии 4.2, Xfwm включает в себя собственный композитный менеджер окон. Многие пользователи называли его самым стабильным из доступных, хотя в то время, в конце 2004 года, другим доступным композитным менеджером был только xcompmgr.

Xfburn. Программа для записи оптических дисков.

14.3 Задание на работу и методические указания по выполнению работы

Задание на работу:

- установить окружение рабочего стола XFCE с помощью команды: sudo apt-get install xubuntu-desktop;
- переключиться в среду XFCE;
- с помощью прикладных программ системы выполнить следующие задания:
  - создать папку
  - создать в ней текстовый файл
  - скопировать этот файл на рабочий стол
  - переименовать файл на рабочем столе

14.4 Содержание отчета

В отчете необходимо указать все команды которые были использованы, привести скриншоты рабочего стола и использованных программ.

Лабораторная работа №15. Работа с кодовыми таблицами русского языка

15.2 Цель работы

Познакомиться с разнообразием кодовых таблиц для русских символов, научиться производить конвертацию текста из одной кодировки в другую.

15.2 Теоретическая часть

Кодировка букв русского алфавита

В настоящее время наиболее широко используются пять (!) различных таблиц кодировки для формального представления русских букв:

- I.ISO 8859-5 — международный стандарт;
- II.Кодовая страница 866 (Microsoft CP866) — используется в MS- DOS;
- III.Кодовая страница 1251 (Microsoft CP1251) для Microsoft Windows;
- IV.На базе ГОСТ КОИ-8, koi8-r — применяется в мире Unix;
- V.Unicode — используется в Microsoft Windows, Unix и клонах Unix.

Основная кодировка ГОСТ (государственный стандарт СССР) от 1987 года создана на основе рекомендаций ISO и в дальнейшем стала основой для представления знаков русских букв в Unicode. В ней и в кодировках II, III и V все буквы кроме ё и Ё расположены в алфавитном порядке. На практике эту кодировку можно встретить только на старых IBM PC совместимых компьютерах ЕС-1840 и в некоторых принтерах. Internet браузеры обычно поддерживают ее наряду с кодировками II–IV.

Кодировка CP866, разработанная на основе альтернативной кодировки ГОСТ, создана специально для ОС MS-DOS, в которой часто используются символы псевдографики. В этой кодировке эти символы имеют те же коды, что и в стандартном IBM PC совместимом компьютере.

Альтернативная кодировка ГОСТ, которая имеет два варианта, совпадает с CP866 по позициям для букв русского алфавита и знакам псевдографики. Основная кодировка ГОСТ совпадает с ISO 8859-5 только по всем знакам русских букв, кроме заглавной буквы Ё. Использование CP1251 обусловлено почти исключительно влиянием на компьютерные технологии разработок фирмы Microsoft. В ней наиболее полно по сравнению с I, II, IV представлены такие символы как (c), №, различные виды кавычек и тире и т. п.

Кодировка koi8-r основана на стандартах по обмену информацией, используемых на компьютерах под управлением ОС Unix, CP/M и некоторых других с середины 1970-х. В 1993 она стандартизирована в Internet документом RFC1489.

Кодировка Unicode опирается на каталог символов UCS (Universal Character Set) стандарта ISO 10646. UCS может содержать до 231 различных знаков. Коды UCS-2 — 2-байтные, UCS-4 — 4-байтные. Используются также коды переменной длины UTF-8 (Unicode Transfer Format) — 1–6-байтные, наиболее совместимые с ASCII, и UTF-16 — 2 или 4-байтные. Unicode в прикладных программах реализуется лишь частично, и в полном объеме пока нигде не поддерживается. В Linux используется UTF-8.

Достаточно широко используется кодирование на основе ASCII: VI. На базе

КОИ-7 — можно использовать при отсутствии кириллических шрифтов, код получается вычитанием 128 от соответствующего кода в koi8-r, что, как правило, дает код латинской буквы, близкой фонетически к русской. В кодировке VI нет видимого символа для для Ъ.

Кроме перечисленных можно встретить еще используемую до введения кодировок ГОСТ болгарскую кодировку, называемую также MIC, Interprog или “старый вариант ВЦ АН СССР”. На компьютерах под управлением Macintosh OS используется также своя собственная таблица кодировки для русских букв, по своему набору знаков почти совпадающая с CP1251.

15.3 Задание на лабораторную работу

Исходный текст - файл с русским текстом. Кодировка файла - CP1251. Произвести кодирование текста в кодировки KOI8-R, ISO 8859-5 , Microsoft CP866 , Unicode .

15.4 Порядок выполнения работы

Выберите исходный файл на русском языке, может содержать цифры, знаки и буквы латинского алфавита. Кодировка файла CP1251. Размер файла - от 200 байт до 1 килобайта. Имя файла: <filename>.cp1251

На языке Pascal написать программу которая производит преобразование файла в другие кодировки. Выбор кодировки производится с помощью меню. Возможен также выбор кодировки с помощью ключей командной строки, например:

transcode -koi8r <filename>

transcode -cp866 <filename>

Результат работы прораммы записывается в файлы с именами: <filename>.cp866, <filename>.koi8r и т.д.

Проверку правильности преобразования можно произвести с помощью программы kwrite.

Кодовые таблицы можно найти либо в учебном пособии, в разделе приложений, либо в интернет.

По окончании работы необходимо подготовить отчет.

Лабораторная работа №16. Основы криптографии

16.1 Цель работы

Изучить базовые методы криптографической защиты документов. Создать программы для шифрования документов несколькими способами.

16.2 Задание на лабораторную работу

Исходный текст - текстовый файл на английском или русском языках (только однобайтовая кодировка, например CP1251). Необходимо на языке Pascal создать следующие программы:

- шифрование/дешифрование простой заменой или подстановкой;
- кодирование/декодирование шифром-перестановкой;
- шифрование/дешифрование с ключевым словом.

Все программы должны получать имя файла с исходным текстом в командной строке. Для работы с командной строкой воспользуйтесь функцией ParamStr() языка Pascal.

16.3 Порядок выполнения работы

Выберите исходный файл на русском или английском языках, может содержать цифры, знаки и буквы латинского алфавита. Кодировка файла CP1251. Размер файла - от 200 байт до 1 килобайта. Имя файла: <filename>.src

На языке Pascal написать программы которая производит шифрование/дешифрование файла. Выбор режима работы программы производится с помощью меню. Возможен также выбор режима работы с помощью ключей командной строки, например:

cyfer -encode <filename>

cyfer -decode <filename>

Результат работы программ записывается в файлы с именами: <filename>.dst.

По окончании работы необходимо подготовить отчет. В отчете привести тексты как исходного сообщения, так и закодированного шифром.

Список рекомендуемой литературы

1 Цилькер Б.Я. Организация ЭВМ и систем : Учебник для вузов - СПб. : Питер, 2007. - 667[5] с

2 Бройдо В.Л., Ильина О.П. Архитектура ЭВМ и систем : Учебник для вузов - СПб. : Питер, 2006. - 717[3]

3 Таненбаум, Эндрю. Архитектура компьютера : Пер. с англ. / Э. С. Таненбаум ; пер. : Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб. : Питер, 2007. - 843[5] с

Учебное пособие

Шандаров Е.С.

Персональные компьютеры и компьютерные сети

Методические указания к лабораторным работам

Усл. печ. л. Препринт
Томский государственный университет
систем управления и радиоэлектроники
634050, г.Томск, пр.Ленина, 40