ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра программной инженерии

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Операционные системы»

Для студентов направления подготовки 09.03.04 "ПРОГРАММНАЯ ИНЖЕНЕРИЯ"

Донецк-ДонНТУ-2016

ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра программной инженерии

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Операционные системы»

Для студентов направления подготовки 09.03.04 "ПРОГРАММНАЯ ИНЖЕНЕРИЯ"

Рассмотрено на заседании кафедры
Программной инженерии
Протокол № 1 от 30.08.2016
Утверждено на заседании
учебно-издательского Совета ДонНТУ
протокол № от

Донецк –2016

УДК

Методические указания и задания к лабораторным работам по курсу «Операционные системы» для студентов специальности «Программная инженерия»/ Сост.: Чернышова А.В. - Донецк, ДонНТУ, 2016 – 96 стр.

Приведены методические указания и задания к выполнению лабораторных работ по курсу «Операционные системы». Излагаются вопросы, связанные организация работы пользователей в Unix-платформенных операционных системах, в частности в ОС Linux, рассмотрены также вопросы, касающиеся специальных возможностей операционных систем, предназначенных для наиболее эффективной организации работы пользователей, рассмотрены вопросы организации процессов в ОС Linux и использование команд фильтров, программирования средствами Shell, а также организации синхронизации процессов и межпроцессного взаимодействия, рассмотрены вопросы организации работы пользователей в Unix-платформенных ОС с использованием графического интерфейса.

Методические указания предназначены для усвоения теоретических основ и формирования практических навыков по организации работы пользователей в Unix-платформенных операционных системах.

Составители: ст. преп. кафедры ПИ Чернышова А.В.

Лабораторная работа №1

Тема: OS Linux: Общая организация работы. Редактирование текстовых файлов с помощью редактора vi.

Цель работы: Освоить основные принципы работы с ОС Linux. Изучение основных команд для работы с элементами файловой системы. Научиться работать с редактором vi.

Методические указания к выполнению работы

Каталоги и файлы

Корневой каталог имеет имя "/". Он обычно содержит каталоги:

bin - для наиболее используемых команд. Здесь находится много важных системных программ. Здесь можно обнаружить команды, вроде cp, ls, mv. Это и есть программы соответствующих команд. Когда, например, используем команду cp, выполняется программа /bin/cp.

В этом каталоге большинство файлов выполняемых (* рядом с файлом говорит об этом).

usr - каталоги и обычные файлы, содержащие информацию, привлекаемую при решении задач пользователя;

dev - файлы в этом каталоге известны, как драйверы устройств, они используются для доступа к устройствам и ресурсам системы, таким, как диски, модем, память и т.д.

etc - для хранения команд администратора системы; содержит множество всевозможных системных файлов конфигурации. Они включают /etc/passwd (файл паролей), /etc/rc (командный файл инициализации) и т.д.

sbin – используется для хранения важных системных двоичных файлов, используемым системным администратором.

lib – важнейшие библиотеки. Эти файлы содержат код, который могут использовать многие программы. Вместо того, чтобы каждая программа

имела свою собственную копию этих выполняемых файлов, они хранятся в одном общедоступном месте – в /lib. Это позволяет сделать выполняемые файлы меньше и сэкономит место в системе.

proc - это “виртуальная файловая система”, в которой файлы хранятся в памяти, а не на диске. Они связаны с с различными процессами, происходящими в системе и позволяют получить информацию о том, что делают программы и процессы в указанное время.

mnt - для подключения (монтирования) новых файловых систем;

sys - средства для изменения конфигурации системы;

tmp - для хранения временных файлов;

usr - каталоги и обычные файлы, содержащие информацию, привлекаемую при решении задач пользователя.

А также обычные (выполняемые) файлы:

unix - ядро;

boot - загрузчик.

Полные имена файлов будут: /bin, /usr, ..., /unix, /boot.

В свою очередь эти каталоги могут содержать каталоги следующего уровня. Например, каталог "usr", кроме прочего, содержит каталоги:

bin - хранит дополнительные команды;

include - хранит фрагменты системных программ;

lib - хранит дополнительные библиотеки.

полные имена этих файлов будут:

/usr/bin /usr/games /usr/include /usr/lib

Если в каталоге "/usr/include" содержится каталог "sys", который в свою очередь, содержит каталог "conf", то полное имя файла "conf" будет /usr/include/sys/conf

Формальным признаком полного имени является то, что оно начинается со слэша ("/").

Относительное имя начинается не с "/", и определят имя относительно своего местоположения. Если пользователь в данный момент находится в директории /usr файловой системы, то он может обратиться к этому же файлу по относительному имени

include/sys/conf

Есть два специальных имени:

. - это "имя" текущего директория и

.. - это "имя" родительского директория (т.е. директория, находящегося на ступеньку выше данного на пути к корню).

В качестве имени файла как правило может использоваться любая последовательность из букв, цифр и подчеркиваний. Могут использоваться и другие символы, однако ряд этих символов при использовании в имени требует специального экранирования. (Лучше не пользоваться специальными символами в именах - иногда это может привести к сложностям в обращении к таким именам, поскольку спецсимволы могут иметь в shell некоторый специальный смысл).

В ряде систем длина имени ограничивается 14-ю символами (этого ограничения желательно придерживаться для переносимости файлов), однако в других системах допускаются более длинные имена - например, до 256 символов.

В общем случае не являются обязательными и какие-то расширения в именах. Хотя ряд команд требуют наличия некоторых фиксированных расширений в именах, например расширение ".c" для исходных файлов для Си-компилятора.

В ОС Linux большие и маленькие буквы воспринимаются как различные, поэтому "FILE", "file" и "File" - это три различных имени.

Отдельные части файловой системы могут находиться на различных физических устройствах, например, на нескольких жестких и гибких дисках (или в различных частях одного диска). Соответствующие фрагменты (поддеревья файловой системы) монтируются (присоединяются) в единую

файловую систему командой mount (обычно это функция администратора системы), после чего пользователь может обращаться к любым доступным файлам, при этом в имени никак не отражается устройство, на котором файл находится или создается.

Вход в систему

Для подключения к OS Linux, которая установлена на сервере кафедры, необходимо вызвать программу telnet, предназначенную для удаленного подключения к узлу сети, указать IP-адрес узла, к которому хотите подключиться. IP-адрес сервера – 194.44.183.180.

При входе Вы увидите приглашение для ввода логина и пароля:

login:student (Логин student для всех студентов кафедры)

Password: Студент (русское слово «Студент», набранное латинскими буквами)

При вводе пароль не будет отображаться на экране. Если Вы неправильно набрали пароль, то увидите на экране сообщение

Login incorrect

Если Вы правильно ввели имя пользователя и пароль, Вы подключены к ОС Linux. Пользователю, подключившемуся к ОС Linux, предлагается работать с помощью командной строки, которая выглядит следующим образом:

[student@pmi student]$

Это значит, что Вы подключились к серверу pmi под логином student и находитесь в домашнем каталоге «student». После авторизации в системе каждый пользователь попадает в свой домашний каталог. Как правило, домашний каталог пользователя соответствует его имени. После этого можно приступать к работе в OS в режиме командной строки.

Виртуальные консоли

Системная консоль - это монитор и клавиатура, связанные непосредственно с системой. (Поскольку Linux - многопользовательская система, Вы можете иметь дополнительные терминалы, связанные через последовательные порты с Вашей системой, но они не будут консолями). Linux, как и некоторые другие версии UNIX, обеспечивает доступ к Виртуальным консолям (или VC), которые позволяют войти в систему под несколькими именами в одно время.

Для демонстрации этого войдите в систему (как было показано ранее). Теперь нажмите alt-F2. Вы должны снова увидеть подсказку login: , то есть перед вами вторая виртуальная консоль, а вы вошли через первую. Чтобы переключиться обратно на первую VC, нажмите alt-F1. Вы снова на первой консоли. Как правило, переключение между виртуальными консолями возможно только в том случае, если вы работаете не удаленно.

Если вы подключены удаленно к системе, то чтобы войти в систему еще с одной консоли под тем же логином или иным, необходимо запустить еще один telnet, при этом создастся еще одно подключение. Работа по средством telnet представляет собой работу через псевдотерминал. Консоль в этом случае будет обозначаться через pts (псевдотерминал).

В то время, как использование виртуальных консолей ограничено (кроме прочего, в каждый момент времени Вы можете видеть только одну виртуальную консоль) оно дает вам представление о многопользовательских возможностях Linux. Пока вы работаете на VC #1, Вы можете переключиться на VC #2 и начать работу над чем-то другим.

Команды для работы с файловой системой

Просмотр содержимого текущего каталога:

ls, dir

Расширенный просмотр текущего каталога:

ls –l, vdir

ls -l ../, ls ../, dir ../,vdir ../, - если Вы находитесь в своем домашнем каталоге, Вы увидите содержимое предыдущего каталога.

ls -l /, ls /, dir /,vdir / - просмотр корневого каталога

ls –F – просмотр каталога (можно увидеть. Что есть файлом, что каталогом)

где ls - имя команды; -l - флаг, говорящий о том, что информация должна быть представлена в длинном формате (ниже приведен пример работы команды ls -l);

```
drwxrwxr-x    2  root  2048  nov 3  12:11  bin
-rwxr--r--    1 root   861  may 11 20:11  boot
drwxrwxr-x    2  root  1024  jan 9  11:55  dev
drwxrwxr-x    1  root  4096  may 11 20:11  dos
drw-r--r--    3  root  4096  nov 17 12:01  include
drwxr-xr-x    7  root  480   nov 17 12:30  lib
```

Первая строка означает, что это каталог (d-directory), где первая триада "rwx" разрешает владельцу каталога: r - читать, w - писать и x - выполнять (для файлов типа каталог w означает разрешение создавать файлы в каталоге и удалять их из него, а x разрешает доступ к файлам каталога); членам группы, в которую входит владелец, также разрешены все три операции. Последняя триада отражает права доступа прочих пользователей, которым разрешено только читать и выполнять (запрещено писать в этот файл, т.е. изменять содержимое каталога).

Далее, 2 - это число связей файла (т.е. где-то в системе есть еще одно имя, связанное с этим файлом); root - имя владельца, 2048 - число символов в файле, nov 3 12:11 - дата и время создания или последней модификации файла (3 ноября в 12-11); bin - имя файла (каталог команд).

Во второй строке указан обычный текстовый файл (boot), который прочие пользователи могут только читать.

dir – выводит список файлов и каталогов указанного каталога.

Параметры:

-A – запрет включения в список текущего (.) и родительского (..) каталогов

-a – вывод списка всех файлов и подкаталогов в каталоге, включая скрытые файлы

-B – запрет включения в список резервных копий (файлов, имена которых заканчиваются на ~)

-C – вывод списка по столбцам (значение по умолчанию)

-c – сортировка по дате последнего изменения.

-F – пометить исполняемые файлы звездочкой (*), каталоги – наклонной чертой (/) и символические ссылки – символом @

-S – сортировка по размеру файла

-X – алфавитная сортировка по расширению.

-w n – Форматирование вывода из расчета ширины экрана, равной n символам.

Команда изменения прав доступа:

chmod – предназначена для изменения режима доступа к файлу или каталогу. Режим доступа устанавливается отдельно для владельца файла, группы пользователей, которой принадлежит файл, и для всех остальных пользователей. Режим доступа может быть задан в одной из двух форм: символьной и числовой. Символьная форма используется для изменения режима доступа относительно текущего состояния, а числовая – для абсолютного задания режима доступа. Числовая форма задания режима использует восьмиричное число, а символьная – одну или несколько групп символов, разделенных запятыми.

Для изменения режима доступа к файлу вы должны быть его владельцем либо привилегированным пользователем.

Примеры использования символьной формы:

chmod g+x file1.txt

Эта команда предоставляет право на исполнение (x) файла file1.txt группе (g), которой принадлежит данный файл.

chmod go-w file1.txt

Эта команда лишает права на модификацию (w) файла file1.txt группу (g), которой принадлежит этот файл, а также прочих (o) пользователей (кроме владельца файла).

chmod g+x, go-w file1.txt

Эта команда предоставляет группе, которой принадлежит файл pat.memo, право на его исполнение и одновременно лишает группу и прочих пользователей права на изменение этого файла.

Обозначения:

u – пользователь (владелец файла)

g - группа

o - прочие

a – все пользователи

+ - добавить разрешения

- - удалить разрешения

= - установить разрешения

r- разрешение на чтение

w – разрешение на изменение

x – разрешение на исполнение

l – блокировка файла для других пользователей

Примеры использования числовой формы рассматривать не будем.

pwd - команда (без флагов и аргументов) сообщает местоположение пользователя в файловой системе. С ее помощью выводится полное имя текущего каталога.

При входе в систему пользователь оказывается в определенной заранее вершине дерева. Это будет каталог "/home/student".

Команду pwd можно использовать при копировании файлов из нижележащих директорий в вышележащие, когда необходимо указать полный путь нахождения файла.

Команда смены текущего каталога:

cd - изменить местонахождение можно командой.

В корневой каталог можно попасть из любого места командой

cd ../

Так можно перейти в каталог /home/student , набрав команду

cd /home/student, если Вы до этого находились в коневом каталоге, здесь указано полное имя. Если в каталоге student создана директория ps, то для того, чтобы перейти из директории student в директорию ps, можно использовать относительное имя, команда перехода будет выглядеть так: cd ps, либо с указанием полного имени: cd /home/student/ps.

Отличительный признак относительного имени - отсутствие символа "/" в начале.

Команда "cd .." осуществит переход вверх на предыдущий уровень, а команда "cd" (т.е. без параметров) осуществит переход в домашний каталог пользователя (т.е. каталог, в котором пользователь оказывается при входе в систему).

Создание файлов и каталогов:

mkdir - создание каталога;

cp – копирование файлов;

mv – перемещает файлы, или можно сказать иначе – дает им новое имя и новое место в дереве каталогов.

Синтаксис: mv параметры исходный_файл конечный_файл

Параметры:

-b – создание резервных копий удаляемых файлов

Команды mv и cp уничтожают содержимое файла, в который они пишут (если он существовал), не спрашивая вашего разрешения.

Удаление файлов и каталогов:

rm имя файла – удаление файла;

rmdir имя директория – удаление каталога, но пустого.

rm -r - удаление каталога и всего его содержимого

Просмотр и поиск файлов:

more, cat – используются для просмотра содержимого файлов

cat – одна из самых полезных команд в Linux, поскольку она позволяет выполнять массу базовых операций с файлами. В простейшем варианте cat читает указанный файл и выводит его содержимое на экран. Команда cat может использоваться в сочетании с символом перенаправления > для объединения нескольких файлов в один, а также с символом перенаправления >> для дописывания файлов в конец существующего. Наконец, команда cat может использоваться для создания нового тестового файла.

Например:

cat report – выводит на экран содержимое файла report

cat report report2 – выводит на экран сначала содержимое файла report и затем без паузы – содержимое файла report2.

cat report report2 > report3 – объединяет файлы report и report2 в файле report3

cat > report – создает новый файл с именем report и записывает в него все, что вы введете с клавиатуры. Выход Ctrl-D.

cat report >> report2 – эта команда дописывает содержимое файла report в конец файла report2.

cat >> report – дописывает все, что вы введете с клавиатуры, в конец файла report.

more – выдает файл на экран поэкранно, а cat – весь сразу.

При использовании more :

« пробел» нажать для перехода к следующей странице;

b – для возврата к предыдущей;

q – выход из more;

find – осуществляет поиск файлов. Она может производить как простой поиск, так и сложный поиск файлов, удовлетворяющих сложным условиям.

find $HOME

Эта команда выводит список всех файлов, находящихся в вашем домашнем каталоге и его подкаталогах.

Копирование файлов:

cp – выполняет одно из следующих действий: копирование содержимого файла в файл с другим именем либо в другой каталог с сохранением существующего имени файла, всех файлов одного каталога в другой каталог. Исходный файл (каталог) не изменяется.

Параметры:

-a – сохранение атрибутов файлов

-b – создание копии вместо перезаписи существующего файла

-d – поддержка символических ссылок

-f – безусловное копирование

-i – интерактивный режим. Перед записью существующего файла будет произведен запрос подтверждения

-l – создание прямых ссылок вместо копирования (применяется при копировании файлов в каталог)

-p – сохранение существующего режима доступа к файлам, принадлежности файлов к метке времени.

-r – копирование каталога вместе с подкаталогом

Примеры:

cp file1 под именем file2

cp /home/student/file1 /home/student/file2

Если Вы находитесь в своем домашнем каталоге student, то можно выполнить вышеприведенную команду без указания полного имени файла-источника и файла-приемника.

cp file1 file2

Удаление файлов и каталогов:

rm – удаляет файлы

Синтаксис rm <file1>…<file N>,

где<file1>…<file N> - имена удаляемых файлов

rm – i – перед удалением спрашивает подтверждения.

rmdir - удаляет пустые каталоги. При использовании этой команды ваш текущий рабочий каталог должен находиться вне удаляемого каталога.

rm –r – удаляет директорию, если она не пустая.

Еще несколько команд:

man – выдает страницу руководства по данной команде или ресурсу (ресурс – это любая системная утилита, которая не является командой, например, библиотечная функция)

man <command>.

echo – просто повторяет аргументы.

echo<arg1>…<argN>, где

<arg1>…<argN> - “повторяемые” аргументы.

bc – калькулятор

bc –l – подключение к математической библиотеке

kill – команда, оправляет процессу с указанным идентификатором (PID) указанный сигнал. Эта команда часто используется для завершения работы процессов. Только владелец процесса или привилегированный пользователь могут использовать эту команду.

ps - команда, выводящая различную информацию о процессах. Она имеет большой набор сложных параметров. За подробной информацией:

ps –help

renice – команда, позволяющая изменить приоритет одного или нескольких запущенных процессов.

script – позволяет сохранить все выводимые на экран символы в указанном файле.

strings – выполняет поиск текстовых сторок в файле. По умолчанию выволятся только те строки, длина которых составляет не менее 4 символов.

uptime - сообщает вам, сколько времени прошло с момента последней перезагрузки системы, сколько пользователей в настоящий момент подключено к системе и какова средняя загрузка системы за последние 1, 5 и 15 минут.

users – выводит информацию о пользователях, подключенных к системе в данный момент.

Родственные команды – who (см. help), w.

Наряду с перенаправлением ввода-вывода целесообразно иногда использовать так называемый конвейерный запуск команд. Конвейерный запуск команд, это когда результаты одной команды передаются на вход другой команды. Синтаксис конвейерного запуска выглядит следующим образом: «команда1»| «команда2». Например, есть команда, которая выводит содержимое текущего каталога в расширенном виде – ls –l, если кол-во файлов в каталоге невелико, то информация помещается на один экран, иначе, необходимо выполнить постраничный просмотр каталога. Это можно реализовать, выполнив последовательно 2 команды: ls –l|more. При этом выполнится сначала команда ls –l и выходные результаты этой команды подаются на вход команде more, которая осуществляет постраничный просмотр.

Использование редактора vi

В Linux множество разных текстовых редакторов, единственный редактор, который с гарантией можно найти в любой UNIX- платформенной

ОС, это vi (``visual editor"***).*** vi - это не самый простой в использовании редактор.

При работе с vi можно находиться в одном из трех режимов работы. Эти режимы известны как *командный режим, режим вставки и режим последней строки.*

Командный режим: этот режим позволяет использовать определенные команды для редактирования файлов или перехода в другие режимы.

Команды для работы в командном режиме:

- “x” - удаление символа, находящегося перед курсором;
- Стрелки передвигают курсор по редактируемому файлу;
- “o” - вставка текста в строку ниже текущей;
- “i” – переход в режим вставки;
- “a” - вставляет текст, начиная после текущего положения курсора, вместо текущей позиции курсора;
- “dd” – удаление текущей строки;
- “dw” удаление слова, на котором находится курсор;
- “ r” - замена одного символа, отмеченного курсором.
- “˜” изменяет размер буквы, отмеченной курсором: большую делает маленькой и наоборот.
- “ h”, “j”,” k”, ”и”,” l” - для перемещения курсора влево, вниз, вверх и вправо соответственно**$**
- “R” - заменяет прежний текст вместо вставки в него;
- “w” - перемещает курсор на начало следующего слова;
- “b” - перемещает на начало предыдущего слова;
- “0” - передвигает курсор на начало текущей строки;
- “ $” - перемещает на конец строки**;**
- ctrl-F - курсор перемещается на экран вперед;
- ctrl-B - на экран назад.
- “G” - перемещение курсора в конец файла;

- “10G” - перемещение на любую строку, напечатав команду, 10G вы переместите курсор на десятую строку файла.
- “1G” – установить курсор на первую строку);

Большинство команд, используемых в командном режиме, состоит из одного или двух символов.

Режим вставки: Вставка или редактирование текста осуществляется в режиме вставки. При использовании vi большую часть времени находимся именно в этом режиме. Переход в режим вставки с помощью команды “i”" из командного режима**.** В режиме вставки вставляете текст в документ на место, указываемое курсором. Для завершения режима вставки и возврата в командный режим следует нажать esc.

Режим последней строки - это специальный режим, используемый для расширения возможностей командного режима. При вводе таких команд они появляются в последней строке экрана. Напечатав “:”" в командном режиме, осуществляется переход в режим последней строки и возможность использовать такие команды:

- “wq”" - записать (write) файл и выйти (quit) из vi;
- “q!”" - выйти из vi без сохранения изменений;
- “r” – чтение файла, необходимо указать имя загружаемого файла;
- “shell” – временный выход из vi, возврат назад в vi по команде «exit»;
- “w”" - записать (write) файл;

Режим последней строки в общем случае используется для команд vi, которые длиннее одного символа.

Задание к выполнению лабораторной работы

1. Создать личную директорию (согласно фамилии студента).
2. Установить права доступа к этой директории.
3. Создать в своей директории файл.
4. Отредактировать созданный файл с помощью редактора vi.

5. Переименовать файл.
6. Дописать в конец созданного файла результаты команды, которая отображает содержимое корневого каталога.
7. Изменить права доступа к файлу, установить их такими, чтобы только пользователь-владелец файла имел к нему доступ на чтение, запись, исполнение.
8. Попытаться скопировать содержимое своего каталога в корневой каталог.
9. Посмотреть, какие процессы запущены с вашего терминала.
10. Посмотреть, кто из пользователей сейчас работает в системе.
11. Скопировать созданный вами файл на уровень выше.
12. Удалить свою директорию со всем ее содержимым.
13. Выполнить просмотр корневого каталога и результаты работы команды корневого каталога записать в файл.

Содержание отчета

1. Титульный лист
2. Задание
3. Описание команд, которые использовались для выполнения задания.

Контрольные вопросы

1. Что представляет собой файловая система ОС Linux?
2. Каким образом реализована многопользовательская защита в ОС Linux?
3. Какими командами меняются права доступа к файлу (директории), кто имеет права сменить права доступа?
4. Каким образом просмотреть содержимое текстового файла?
5. Каким образом созданный в Windows файл переместить в Вашу директорию?
6. Какую кодировку поддерживает ОС Linux?
7. Как удалить каталог, содержащий файлы?

8. Каким образом осуществляется поиск файлов?
9. Каким образом посмотреть, кто находится в настоящий момент в системе?
10. Что такое виртуальные консоли, каким образом они используются?

Лабораторная работа №2

Тема: Процессы. Управление процессами.

Цель работы: Изучение основных команд для контроля состояния запущенных процессов в ОС Linux.

Методические указания к выполнению работы

Программой называется исполняемый файл, а процессом называется последовательность операций программы или часть программы при ее выполнении. В Unix-подобных системах одновременно может выполняться множество процессов (эту особенность часто называют мультипрограммированием или многозадачным режимом). Процесс – это фундаментальное понятие в Unix. С помощью процессов происходит управление памятью и ресурсами ввода-вывода, используемыми для выполнения программы. На первый взгляд кажется, что в Unix происходит все одновременно, однако на самом деле в конкретный момент времени выполняется только один процесс. Иллюзию параллельного выполнения создает и поддерживает метод, называемый «квантованием времени», с помощью которого Unix через определенные промежутки времени (как правило, каждые 20 миллисекунд) меняет выполняемый процесс. Процессы сменяют друг друга настолько быстро, что кажется, что они выполняются одновременно, тогда как в реальности, выполнение каждого процесса занимает лишь малую долю суммарного времени.

Процесс состоит из адресного пространства и набора структур данных, содержащихся внутри ядра. Адресное пространство представляет собой совокупность страниц памяти, которые ядро выделило для выполнения процесса. Адресное пространство содержит сегменты для кода программы, которую выполняет процесс, используемые процессом переменные, стек процесса и различную вспомогательную информацию, необходимую ядру во время работы процесса.

В структурах данных ядра хранится различная информация о каждом процессе. К наиболее важным сведениям относятся:

- Таблица распределения памяти процесса;
- Текущий статус процесса;
- Приоритет выполнения процесса;
- Информация о ресурсах, которые использует процесс;
- Маска обработки процесса;
- Владелец процесса.

Каждому новому процессу, созданному ядром, присваивается уникальный идентификационный номер (PID).

Исходный процесс в терминологии ОС Unix называют родительским, а его клон – порожденным. Помимо собственного идентификатора, каждый процесс имеет атрибут PPID, т.е. идентификатор своего родительского процесса.

UID – это идентификационный номер пользователя, создавшего данный процесс. Вносить изменения в процесс могут только его создатель и привилегированный пользователь.

От приоритета процесса зависит, какую часть времени центрального процессора он получит. Выбирая процесс для выполнения, ядро находит процесс с самым высоким «внутренним приоритетом».

Непосредственно установить внутренний приоритет невозможно, но можно установить значение nice, которое существенно влияет на внутренний приоритет.

Процессы не появляются в системе просто так и не создаются спонтанно ядром. Новые процессы порождаются другими процессами.

По состоянию процессов процессы могут быть:

Выполнимый – процесс можно выполнять.

Ожидающий – процесс ждет выделения какого-либо ресурса.

Зомби – процесс пытается «умереть».

Остановленный – процесс приостановлен (нет разрешения на выполнение)

Текущий контроль процессов: команда ps.

ps - выдача информации о состоянии процессов,

синтаксис команды:

ps [-e] [-d] [-a] [-f] [-l] [-n файл_с_системой] [-t список_терминалов]
[-p список_идентификаторов_процессов]
[-u список_идентификаторов_пользователей]
[-g список_идентификаторов_лидеров_групп]

Описание команды:

Команда ps выдает информацию об активных процессах. По умолчанию информация дается только о процессах, ассоциированных с данным терминалом. Выводятся идентификатор процесса, идентификатор терминала, истраченное к данному моменту время ЦП и имя команды. Если нужна иная информация, следует пользоваться опциями.

Некоторые опции имеют один аргумент или список аргументов. Аргументы в списке могут быть либо отделены друг от друга запятыми, либо все вместе заключены в двойные кавычки и отделены пробелами или запятыми. Аргументы в списке_процессов и в списке_групп должны быть числами.

Командой ps обрабатываются следующие опции:

-e Вывести информацию обо всех процессах.

-d Вывести информацию обо всех процессах, кроме лидеров групп.

-a Вывести информацию обо всех наиболее часто запрашиваемых процессах, то есть обо всех процессах, кроме лидеров групп и процессов, не ассоциированных с терминалом.

-f Генерировать полный листинг (см. ниже разъяснение смысла колонок).

-l Генерировать листинг в длинном формате (см. ниже).

-n файл_с_системой

Считать, что операционная система загружена из файла_с_системой, а не из файла /unix.

-t список_терминалов

Выдавать информацию только о процессах, ассоциированных с терминалами из заданного списка_терминалов. Терминал - это либо имя файла-устройства, например ttyномер или console, либо просто номер, если имя файла начинается с tty.

-p список_идентификаторов_процессов

Выдавать информацию только об указанных процессах.

-u список_идентификаторов_пользователей

Выдавать информацию только о процессах с заданными идентификаторами или входными именами пользователей. Идентификатор пользователя выводится в числовом виде, а при наличии опции -f - в символьном.

-g список_идентификаторов_лидеров_групп

Выводить информацию только о процессах, для которых указаны идентификаторы лидеров групп. Лидер группы - это процесс, номер которого идентичен его идентификатору группы. Shell, запускаемый при входе в систему, является стандартным примером лидера группы.

Расширенный просмотр запущенных процессов – команда ps –aux.

Пояснения к выходной информации команды ps –aux:

USER – имя владельца процесса.

PID – идентификатор процесса.

%CPU – доля времени центрального процессора (в %), выделенная данному процессу.

%MEM – часть реальной памяти, используемая данным процессом.

VSZ – виртуальный размер процесса в килобайтах.

RSS – Размер резидентного набора (количество 1К-страниц в памяти).

TTY – идентификатор управляющего терминала.

STAT – текущий статус процесса:

R – выполнимый;

D – кратковременное ожидание;

S – ожидающий > 20 с;

I – ожидающий <20 с;

Z – зомби;

T – остановленный;

L – блокирован;

S – процесс лидер сеанса.

START – время запуска процесса;

TIME – Время центрального процессора, потребленное процессом;

COMMAND – имя и аргументы команды.

За более подробной информацией по данной команде обращаться к страницам руководства.

Улучшенный текущий контроль процессов: команда top.

Выдает регулярно обновляемую сводку активных процессов и используемых ими ресурсов.

Изменение приоритета выполнения: команды nice и renice.

nice – выполнение запуска процесса с изменением приоритета. В зависимости от параметров запуска процесс может быть запущен или с повышением или с понижением приоритета. Как правило, запуск процесса с повышением приоритета не всегда разрешен администратором системы.

Синтаксис:

nice [-коэффициент_понижения] команда [аргумент ...]

Описание:

Команда nice выполняет команду с пониженным приоритетом. Коэффициент_понижения задается в диапазоне 1-19 (по умолчанию равен 10).

Суперпользователь может выполнять команды с повышенным приоритетом, для этого нужно указать отрицательный коэффициент_понижения, например --10.

Команда nice возвращает код завершения подчиненной команды.

nice - изменение приоритета процесса

СИНТАКСИС

int nice (incr)

int incr;

ОПИСАНИЕ

Системный вызов nice увеличивает поправку к приоритету вызывающего процесса на величину incr. Поправка к приоритету - неотрицательное число; чем оно больше, тем ниже приоритет процесса в смысле использования процессора.

Максимальное (39) и минимальное (0) значения поправки к приоритету ограничиваются системой. (Подразумеваемое значение поправки равно 20). Эти значения используются, когда делается попытка выйти за соответствующую границу.

[EPERM] Системный вызов nice завершается неудачей и не изменяет поправку к приоритету, если аргумент incr отрицателен или больше 39, а действующий идентификатор пользователя вызывающего процесса не является идентификатором суперпользователя.

В случае успешного завершения системный вызов nice возвращает новое значение поправки к приоритету минус 20. В случае ошибки возвращается -1, а переменной errno присваивается код ошибки.

Прекращение выполнения процессов: команда kill

Команду kill чаще всего используют для прекращения выполнения процесса. Эта команда может послать в процесс любой сигнал, но по умолчанию посылается сигнал TERM, сигнал программного завершения. Команду kill могут использовать как обычные пользователи (для своих собственных процессов), так и суперпользователь(для любого процесса). Она имеет следующий синтаксис:

kill [-сигнал] pid,

где сигнал – номер или символическое имя посылаемого сигнала, а pid – идентификационный номер процесса-адресата. Команда kill без номера сигнала «не гарантирует», что процесс умрет, потому что сигнал TERM можно перехватить, блокировать и игнорировать. Команда kill –9 pid «гарантирует», что процесс умрет, потому что сигнал 9, KILL, другими процессами не перехватывается.

За более подробной информацией по данной команде обращаться к страницам руководства.

Запуск процессов в фоновом режиме

Оболочка позволяет запустить процесс и, не дожидаясь его завершения, запустить другой процесс. Чтобы это сделать, первый процесс должен быть запущен в фоновом режиме. Для запуска процесса в фоновом режиме используется &, который добавляется в конец командной строки.

После запуска процесса в фоновом режиме появится идентификатор процесса.

Когда примитивный интерпретатор команд завершает работу, он посылает во все порожденные им процессы сигнал «отбой». Если процесс выполняется в фоновом режиме, этот сигнал часто уничтожает его, что в некоторых случаях нежелательно. Если нужно запустить программу, которая будет работать и после вашего выхода из системы, ее нужно запускать командой nohup. Эта команда имеет следующий формат:

nohup *команда* &

Подобный запуск заставляет указанную аргументом *команда* команду игнорировать сигнал отбоя.

За более подробной информацией по данной команде обращаться к страницам руководства.

Планирование

Запущенная с помощью nohup команда сразу же начнет выполняться. Если необходимо начать процесс позднее или выполнять его периодически, нужно прибегнуть к услугам демона cron.

Демон cron - это процесс в фоновом режиме, запущенный системой. Можно с помощью cron выполнять программу в определенное время.

За более подробной информацией по данной команде обращаться к страницам руководства.

Команда *at* принимает дату или время в качестве параметра и любое число командных строк из стандартного ввода. Когда встретится признак конца файла, команда at создает сценарий для выполнения команд в указанное время.

За более подробной информацией по данной команде обращаться к страницам руководства.

Задание к лабораторной работе

1. Определите, кто из пользователей работает в системе, и какие процессы запущены каждым из пользователей. Для каждого процесса определите его состояние и приоритет.

2. Попытайтесь уничтожить процессы, запущенные другими пользователями.

3. Запустите какой-либо процесс в фоновом режиме. Найдите процесс в списке процессов. Определите его состояние. Уничтожьте.

4. Выполните те же действия, понижая приоритет запускаемого процесса с помощью команды nice. Измените приоритет запущенного процесса с помощью renice.

5. Запустите несколько копий процесса в фоновом режиме. Найдите запущенные процессы в списке процессов. Выйдите из системы. Войдите в систему. Просмотрите список процессов.

6. Выполните те же действия, запуская процессы в непрерываемом режиме (nohup). Сравните состояния после повторного входа в систему. Уничтожьте запущенные процессы.

Содержание отчета

1. Титульный лист.
2. Задание.
3. Описание команд, которые использовались для выполнения задания.

Контрольные вопросы

1. Понятие процесса. Типы процессов.
2. В каких состояниях может находиться процесс.
3. Команды для отслеживания процессов, запущенных с разных терминалов и указанными пользователями.
4. Понятие фонового режима.
5. Уничтожение процесса.
6. Понятие приоритета процесса. Типы приоритетов. Изменение приоритета процесса, запущенного пользователем.

Лабораторная работа №3

Тема: Фильтры: sort, grep, wc, awk.

Цель работы: ознакомиться с существующими командами-фильтрами, уметь применять их для получения определенной информации.

Методические указания к выполнению работы

Существует большое число программ Unix-платформенных ОС, которые читают входной поток, выполняют простые операции над ним и записывают результат в выходной поток. Такие программы называются фильтрами. Рассмотрим некоторые из них.

grep - поиск по шаблону, заданному ограниченным регулярным выражением.

Синтаксис: grep [-b] [-c] [-i] [-l] [-n] [-s] [-v] ограниченное_регулярное_выражение [файл ...]

Команда grep сопоставляет строки исходных файлов с шаблоном, заданным ограниченным_регулярным_выражением. Если файлы не указаны, используется стандартный ввод. Обычно каждая успешно сопоставленная строка копируется на стандартный вывод; если исходных файлов несколько, перед найденной строкой выдается имя файла. В grep используется компактный недетерминированный алгоритм. В качестве шаблонов воспринимаются ограниченные регулярные выражения (выражения, имеющие своими значениями цепочки символов, и использующие ограниченный набор алфавитно-цифровых и специальных символов).

В командной строке могут задаваться следующие опции:

-b Перед каждой строкой ставить номер блока, в котором она находится. Используется для поиска блока по контексту (блоки нумеруются с нуля).

-c Выдавать только количество успешно сопоставленных строк.

-i При сопоставлении не различать большие и малые буквы.

-l Выдавать только имена файлов, в которых есть успешно сопоставленные строки, разделяя имена переводами строк.

-n Перед каждой строкой ставить ее номер в файле (строки нумеруются с 1).

-s Подавить выдачу диагностических сообщений о несуществующих и недоступных для чтения файлах.

-v Выдавать только строки, не удовлетворяющие шаблону.

Таблица 1 - Регулярные выражения grep (в порядке убывания приоритета)

c	Любой неспециальный символ с соответствует самому себе
\ c	Указание убрать любое специальное значение символа с
^	Начало строки
$	Конец строки
.	Любой одиночный символ
[…]	Любой символ из …; допустимы диапазоны типа a-z
[^…]	Любой символ не из …
\ n	Строка, соответсвующая n-му выражению \(…\)
r *	Нуль или более вхождений r
r1 r2	За r1 следует r2

Примеры использования grep:

Вывести все файлы и директории, в которых встречается буква s: ls|grep s.

Вывести все файлы и директории, которые начинаются с буквы s: ls|grep ^s.

Вывести все файлы и директории, которые заканчиваются на букву s: ls|grep s$.

Различие между большими и малыми буквами в вышеприведенных примерах существует.

Чтобы снять различие между большими и малыми буквами используйте grep – i.

Например: Вывести все файлы и директории, которые начинаются с буквы S, или s:

ls|grep –i ^s.

Вывести список имен вложенных каталогов:

ls –l|grep ‘^d’

Вывести список файлов, доступных всем для чтения записи, выполнения:

ls –l|grep ‘^-……rwx’

Выяснить работает, ли указанный пользователь в системе:

who|grep student

Использование двойных кавычек: двойные кавычки требуются для размещения в шаблоне пробелов.

Найти все слова их 4-х букв, начинающихся с d:

grep “d…”

Найти все строки, начинающиеся с с букв a,b,c,x,y,z:

grep “^[abcxyz]”.

Найти все слова из 4-х букв, не начинающиеся с D или d, в которых последние три буквы малые от a до z:

grep “[^Dd][a-z][a-z][a-z]”

Фигурные скобки задают количество повторений предыдущего знака:

Найти все слова из 4-х букв, не начинающиеся с D или d, в которых последние три буквы малые от a до z:

grep “[^Dd][a-z]{3}”

Найти все слова, содержащие от 3, до 5-ти малых букв:

grep “[a-z]”{3,5}

Выполнение сортировки.

sort - сортировка и/или слияние файлов

Синтаксис:

sort [-c] [-m] [-u] [-o выходной_файл] [-укилобайт] [-zдлина]
[-d] [-f] [-i] [-M] [-n] [-r] [-b] [-tразделитель]
[+позиция_1 [-позиция_2]] [файл ...]

Команда sort сортирует строки, входящие во все исходные файлы, и выдает результат на стандартный вывод. Если имена файлов не указаны, или в качестве файла указан -, исходная информация поступает со стандартного ввода.

При упорядочении используется один или несколько ключей сортировки, выделяемых из каждой вводимой строки. По умолчанию ключ сортировки один - вся строка, а порядок является лексикографическим, соответствующим принятой кодировке символов.

Следующие опции изменяют стандартный порядок работы:

-c Проверить, является ли (единственный) исходный файл уже отсортированным. На стандартный вывод ничего не выдается. В стандартный протокол выводится соответствующее сообщение только в случае нарушения упорядоченности строк.

-m Только слияние исходных файлов, которые предполагаются отсортированными.

-u Опция уникальности: из всех совпадающих строк выводить только одну.

-o выходной_файл

Результат направляется не на стандартный вывод, а в выходной_файл, который может совпадать с одним из исходных.

Следующие опции позволяют выбрать нужный способ сравнения:

-d "Словарный" порядок: при сравнении являются значимыми только буквы, цифры, пробелы и знаки табуляции.

-f Преобразовывать малые буквы в большие.

-i При нечисловых сравнениях игнорировать символы с (восьмеричными) кодами, не лежащими в пределах 040-0176.

-M Сравнивать как месяца. Первые три символа, отличные от пробела, сравниваются таким образом, что "JAN" < "FEB" < ... < "DEC" (малые буквы преобразуются в большие). Остальные трехсимвольные сочетания считаются меньшими, чем "JAN". Эта опция включает опцию -b (см. ниже).

-n Числовое сравнение. Начальные пробелы отбрасываются, затем цифровые цепочки символов, содержащие, быть может, знак минус и десятичную точку, сравниваются как числа. Эта опция включает опцию -b. Отметим, что опция -b действует только на ключи сортировки с наложенными ограничениями.

-r Заменить результат сравнения на противоположный.

Если опции, задающие способ сравнения, указаны до ограничений на ключи сортировки, то они применяются глобально ко всем ключам. Если же соответствующие флаги ассоциированы с определенными ключами сортировки (см. ниже), они воздействуют только на "свои" ключи.

Полем называется минимальная последовательность символов, за которой следует разделитель полей или перевод строки. По умолчанию символом-разделителем считается пробел или символ табуляции. Пробелы и табуляции сразу вслед за разделителем (если они есть) принадлежат следующему полю. Все пробелы в начале строки входят в первое поле. На трактовку разделителей влияют следующие опции:

-b Игнорировать начальные пробелы при определении начала и конца ключей сортировки. Если опция -b указана перед первым аргументом +позиция_1, она действует на все ключи с наложенными ограничениями. Флаг b можно связать и с отдельными ключами сортировки (см. ниже).

-t разделитель - Использовать заданный символ как разделитель полей.

Разделитель не является частью поля (хотя и может входить в ключ сортировки). Каждое вхождение разделителя является значимым, то есть два рядом стоящих разделителя ограничивают пустое поле.

При наложении ограничения на ключ сортировки указывается позиция начала ключа (+позиция_1) и позиция сразу за концом ключа (-позиция_2). Если опция -позиция_2
отсутствует, ключ занимает весь остаток строки.

Позиция_1 и позиция_2 задаются как пара m.n, возможно, с последующими флагами bdfiMnr. Начальная позиция задается как +m.n, что означает (n+1)-ый символ в (m+1)-ом поле (поля и символы нумеруются с единицы). Отсутствие n означает .0, то есть первый символ (m+1)-го поля.

Если указан флаг b, то n отсчитывается от первого непробела в (m+1)-ом поле; +m.0b означает первый непробел в (m+1)-ом поле.

Позиция за концом ключа записывается как -m.n, что означает (n+1)-ый символ (включая разделители) после последнего символа m-го поля. Если .n опущено, то подразумевается .0, то есть разделитель после m-го поля. Если указан флаг b, то n отсчитывается от первого непробела в (m+1)-ом поле.

Если указано несколько ключей сортировки, то более поздние используются только в случае равенства более ранних. Если значения ключей сортировки двух строк совпадают, строки упорядочиваются с учетом всех символов.

Примеры использования sort:

Допустим, есть некий файл file1, содержащий имя, фамилию, и какое-то числовое значение:

Vasiliy Ivanov	100
Ivan Vasil’ev	200
Aleksandr Petrov	300

sort file1 – записи сортируются по первой букве имени.

Можно отсортировать файл по фамилии, т.е. по второму полю:

sort +1 file1

Сортировка по третьему полю с игнорированием лидирующих пробелов выполняется с использованием ключа –b(blank):

sort –b +2 file1

Для сохранения результата сортировки в файле используется ключ –o:

sort –o file2 +1 file1;

sort можно использовать не только для сортировки файлов.

Например:

ls|sort –f – сортировка имен файлов в алфавитном порядке;

ls –l|sort –n – сортировка в порядке возрастания размеров файлов;

ls -l|sort –nr – сортировка в порядке убывания размеров файлов.

wc - подсчет количества символов, слов и строк в файле

синтаксис:

wc [-l] [-w] [-c] [файл ...]

Команда wc подсчитывает строки, слова и символы, читая их из указанных файлов или со стандартного ввода, если файлы не заданы. Подсчитывается также общий итог для всех указанных файлов.

Слово - это максимальная цепочка символов, не содержащая пробелов, табуляций и переводов строк.

Опциям команды wc приписан следующий смысл:

-l Подсчет числа строк.

-w Подсчет числа слов.

-c Подсчет числа символов.

Можно задавать любую комбинацию этих опций. По умолчанию используется набор -lwc.

Если в командной строке указаны имена файлов, то они выводятся после соответствующих сумм.

Например:

Подсчет числа активных пользователей:

who|wc –l;

awk - сопоставление с шаблонами и преобразование текста

Синтаксис: awk [-Fсимвол] [[-f] программа] [аргумент ...] [файл ...]

Команда awk сопоставляет строки исходных файлов с шаблонами, определенными в программе. Шаблоны можно задать либо непосредственно в командной строке, либо поместить в файл с именем программа и воспользоваться опцией -f. Если шаблоны указаны в командной строке, их следует заключить в одинарные кавычки ('), чтобы избежать интерпретации shell'ом.

За более подробной информацией по awk обращаться к страницам руководства.

Простейший пример использования awk:

who|awk ‘{print $1,$2}’

Команда who показывает, кто в настоящий момент есть в системе. Вывести только пользователей и терминал, можно с помощью выше приведенного примера, где $1 – первое поле – пользователь, $2 – второе поле – терминал.

Задание к лабораторной работе

1. Составьте файл по имени family, содержащий фамилию и дату рождения нескольких лиц. Каждая строка файла должна выглядеть следующим образом:

Ivanov 03/01/1950

Разделите имя и дату рождения с помощью табуляции (<tab>).

2. Отсортировать файл в алфавитном порядке по первой букве фамилии.

3. Отсортировать файл в обратном порядке по первой букве фамилии.

4. Отсортировать файл по дате рождения, запишите в другой файл.

5. Отсортируйте файл по году рождения, запишите в другой файл.

Если Вы используете ">", используйте для сортированного файла новое имя файла.

6. Найдите все фамилии, название которых начинается с буквы «А».

7. Найдите все фамилии, у которых в конце стоит буква «в», посчитать их количество.

8. Вывести фамилии тех, кто родился позже 1970 года.

9. Вывести фамилии тех, кто родился в январе.

После выполнения 9-ти пунктов задания, студент должен получить персональное задание у преподавателя. Пример индивидуального задания: Вывести все файлы текущего каталога, у которых права доступа чтение, запись, исполнение для пользователя-владельца каталога, причем отобрать только те файлы, дата создания которых соответствует текущей дате, отсортировать их по имени в обратном порядке.

Содержание отчета

1. Титульный лист
2. Задание к лабораторной работе (общее задание).
3. Команды, реализующие пункты задания.
4. Индивидуальное задание (формулировка)
5. Набор команд с использованием конвейерного запуска команд, реализующих индивидуальное задание. Пояснение каждой команды.

Контрольные вопросы

1. Пример использование команд-фильтров в ОС Linux.
2. Возможности и назначение команды grep.
3. Возможности и назначение команды sort.
4. Возможности и назначение команды wc.

Лабораторная работа N 4

Тема: Создание команды-скрипта, расширяющей функциональные возможности ОС Linux.

Цель работы: Создание команды, расширяющей функциональные возможности системы Linux с использованием изученных команд, перенаправления ввода-вывода и конвейерного запуска программ

Методические указания к выполнению лабораторной работы

Довольно часто в операционной системе при администрировании требуется выполнять ряд действий, которые могут повторяться на протяжении некоторого промежутка времени. Например, контроль пользователей, работающих в системе или кол-во процессов, запущенных в системе указанным пользователем. С этой целью можно использовать дополнительные команды-скрипта, или исполнимые файлы, которые содержат в себе ряд команд, последовательно выполняющихся.

Для создания такого файла или команды скрипта, необходимо выполнить следующие действия:

1) Создать файл.
2) Записать в этот файл последовательность выполняемых команд с использованием перенаправления ввода-вывода и конвейерного запуска команд.
3) Сделать это файл исполняемым
4) Запустить файл с указанием параметров или без указания параметров.

Рассмотрим пример. Например, нам нужно считать количество файлов в указанном каталоге.

Создаем файл cat>file1

Записываем в него последовательность команд:

ls –l $1|grep –c ^-……

Делаем это файл исполнимым: chmod a+x file1

Запускаем файл: sh file1 /home/student

Выполняя указанные действия, мы получим в результате количество файлов в каталоге /home/student. Указанный каталог при запуске является первым параметром. Количество параметров может быть различным, в зависимости от предъявляемых требований к скрипту.

Иногда необходимо выводить на экран не всю информацию, которую представляет отработавший скрипт, а лишь некоторую ее часть. Для этого можно использовать такие команды, как head, tail.

Рассмотрим пример. Вывести первые 5 процессов, запущенных от имени указанного пользователя.

Команда будет выглядеть так:

ps –u $1|head -$2

Записывает вышеприведенную команду в файл с именем pwu.

Делаем файл исполнимым chmod a+x pwu.

Запускаем файл.

При запуске необходимо передать 2 параметра – имя пользователя и количество процессов, например, sh pwu student 5. Внимание, при запуске файла с такими параметрами на самом деле будет выведена информация о первых 4-х процессах, так как команда head выводит указанное количество строк, начиная с первой. Первая строка – это «шапка», которая содержит информацию о заголовках столбцов команды ps. Т.е. если необходимо вывести первые 5 процессов, то во второй параметр при запуске нужно передавать 6. Команда tail выводит указанное количество строк, начиная с конца, поэтому результатом будет указанное количество процессов.

Еще один нюанс. Если необходимо производить отбор строк, которые удовлетворяют одному из условий отбора, то использование команды grep не даст необходимого результата, так как grep не выполняет логического «или», для выполнения задания, которое предполагает логическое «или» необходимо использовать команду egrep. Например, если мы должны

вывести все файлы, которые хотя бы в одной из триад прав доступа имеют «x», т.е. если файл является исполнимым или для владельца, или для группы, или для всех остальных пользователей. Использование grep не приведет к необходимому результату. В данном случае нужно использовать команду egrep. Синтаксис команды можно посмотреть в страницах руководства (man egrep).

Задания к лабораторной работе

В соответствии с вариантом задания, написать команду, расширяющую функциональные возможности ОС Unix.

1) lx – вывести список файлов указанного каталога, у которых права на чтение, запись и выполнение только для создателя файла (т.е. rwx------), отсортировать их по имени в обратном порядке.
2) pu – посчитать количество процессов, запущенных указанным пользователем.
3) pt – посчитать кол-во процессов, запущенных с указанного терминала.
4) nx - количество исполнимых файлов в указанном каталоге.
5) npu – посчитать количество терминированных процессов, запущенных указанным пользователем.
6) mp – кол-во процессов, запущенных определенного числа.
7) tu - посчитать количество терминалов, с которых запущены процессы в текущий момент времени.
8) bp - вывести информацию об указанном количестве процессов, имеющих наибольшее время использования процессора.
9) bf - вывести информацию об указанном количестве файлов, имеющих наибольший размер.
10) Посчитать кол-во директорий в указанном каталоге, у которых права доступа: rwxrwxrwx.
11) ml - вывести информацию об указанном количестве файлов, имеющих наибольшее число связей.

12) ll - список пользователей - владельцев файлов в указанном каталоге.

13) вывести список всех файлов, с датой создания, равной текущему числу.

14) вывести 5 последних процессов, запущенных root.

15) Вывести 5 процессов, запущенных studentom.

16) Посчитать, какое кол-во пользователей сейчас работает в системе (имя уникально).

Содержание отчета

1) Титульный лист.

2) Задание.

3) Реализация команды, расширяющей функциональные возможности ОС Linux.

4) Описание каждой команды.

5) Пример запуска и результаты работы.

Контрольные вопросы к лабораторной работе

1. Использование перенаправления ввода-вывода и конвейерного запуска команд для создания команд, расширяющих функциональные возможности ОС Linux.

2. Передача параметров.

3. Назначение команд, расширяющих функциональные возможности ОС Linux.

Лабораторная работа № 5

Тема: Написание скриптов на языке командного интерпретатора Shell в ОС Linux.

Цель: Научиться выполнять реализацию скриптов на языке командного интерпретатора Shell в ОС Linux.

Методические указания к выполнению лабораторной работы

Командный интерпретатор – это оболочка, которая принимает команды пользователя и преобразует их в код, исполняемый операционной системой. Можно сказать, что интерпретатор – это удобный интерактивный интерфейс системы и пользователя. Схема взаимодействия пользователя с системой с помощью командного интерпретатора представлено на рисунке 1.

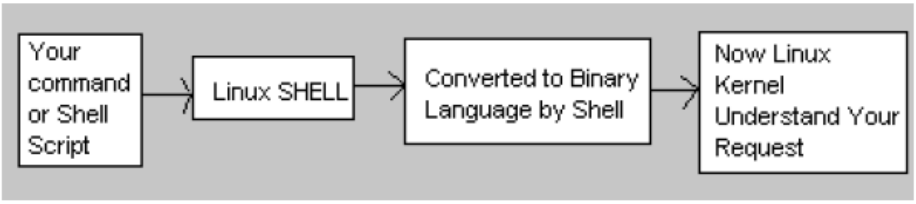

Рисунок 1 – Взаимодействие с системой с помощью командного интерпретатора

Простейший режим работы пользователя и системы – диалоговый. В этом режиме пользователь вводит команды и мгновенно получает ответ в виде реакции системы на его действия. Типовые команды позволяют пользователю работать с файловой системой, процессами, памятью и другими системными ресурсами. Система команд современных ОС, в частности ОС Linux, настолько обширна, что для описания всех команд приходится писать огромные справочные руководства.

Очень часто в практике программиста, системного администратора или простого пользователя появляется необходимость автоматизировать

некоторые действия с ОС. Например, вводом одной команды выполнять синтаксический анализ, компиляцию и сборку программных модулей, обрабатывать большие объемы файлов по заданному сценарию, автоматизировать мониторинг различных событий в системе, реализовать сложную обработку системных журналов и т.д. Конечно, каждая такая задача может быть решена путем написания соответствующего программного продукта. Однако существует значительно более простой способ – использовать команды ОС, запуская их в пакетном режиме. Пакетный режим – это возможность объединить некоторый набор команд в отдельный командный файл («пакет») и запуском этого файла передавать команды на выполнение так же, как если бы мы вводили их в командной строке.

Простейшим примером командных файлов являются командные файлы *.bat в ОС MS DOS и WINDOWS.

Отличительной особенностью командного интерпретатора Linux/Unix систем является богатый набор программных конструкций, которые могут быть реализованы в командном файле (скрипте) - это ветвления, циклы, операции ввода-вывода, вычисление арифметических выражений, логические операции и все это в дополнение к использованию огромной системы команд ОС.

Такие возможности предоставляют разработчику возможность создавать сложные скрипты для реализации практически любых действий в ОС.

Рассмотрим основные программные конструкции языка командного интерпретатора.

Вообще, в состав ОС может входить несколько командных интерпретаторов. Использование одного из них определяется системными настройками. Наиболее известными являются интерпретаторы sh и bash. Исполнимые файлы данных интерпретаторов находятся обычно по путям /bin/sh и /bin/bash, соответственно.

Символ **#** используется для обозначения однострочного комментария, все, что расположено в строке правее данного знака, считается комментарием.

Специальная конструкция, расположенная в первой строке скрипта:

```
#!/bin/bash
```

указывает, какой интерпретатор будет обрабатывать данный скрипт. Данная конструкция не является обязательно, но ее применение позволит избежать множества проблем при запуске скрипта.

Переменные в скриптах не требуют предварительного объявления. Тип переменной определяется автоматически при ее инициализации. Язык командного интерпретатора чувствителен к регистру символов. В именах переменных могут использоваться буквы латинского алфавита, а также цифры и символ подчеркивания, если они идут после одного или нескольких буквенных символов.

Пример присвоения значений переменным:

```
A=10
B4=1000
R_T=cderewfewfwf
```

По умолчанию, переменные в скрипте считаются строковыми и операции над ними производятся как над строками.

Особенным является использование значений переменных. Для того чтобы в программе обратиться к значению переменной, необходимо впереди указать знак **$**.

```
var1=$var2
```

Если написать **var1=var2**, то переменной **var1** будет присвоено строковое значение **var2**.

Для вывода на экран используется команда **echo**, которая может использоваться со следующими параметрами:

\n – переход на новую строку;

\t – вставить знак табуляции;

\\ - отменить специальное значение обратного слэша (или любого другого спец. символа);

-n – не выводить символ перехода на новую строку (по умолчанию выводится всегда);

и т.д.

Например:

```
echo  "The first message"

echo  "Z is equal " $z

echo –n "first line"
```

Для выполнения арифметических операций используется специальная команда expr. Можно сравнить:

```
echo  2 + 3          #на экране будет  2 + 3
echo  expr 2 + 3     #на экране будет 5
```

Еще вариант:

```
Z=`expr 2 +3`
echo  $Z
```

Если результат вычислений или любой другой команды необходимо присвоить некоторой переменной, то строку необходимо взять в обратные кавычки!

При реализации арифметических расчетов доступны операции

сложение +;

вычитание -;

деление /;

умножение *; - обратите внимание на обратный слэш, который отменяет спец. значение символа *;

остаток от деления %.

Все операции – целочисленные!

Специальными переменными в программе являются параметры командной строки скрипта. Для обращения к параметрам используются переменные вида $1, $2, .. . Переменная $0 хранит имя скрипта, а $# - количество параметров командной строки.

Для организации ветвлений в скриптах используется следующая конструкция:

```
if  условие
then
   действие1
else
   действие2
fi
```

или

```
if  условие;    then
    действие1;  else
   действие2
fi
```

Т.е. при наличии нескольких операторов в строке они отделяются точкой с запятой.

Для записи условий используются следующие конструкции:

Условие	Математическая запись	Для числовых значений	Для строковых значений
Равно	=	-eq	= =
Не равно	<>	-ne	!=
Больше	>	-gt	
Меньше	<	-lt	
Больше или равно	>=	-ge	
Меньше или равно	<=	-le	

Объединение простых условий в сложные осуществляется с помощью следующих логических операций:

-a – и;

-o – или;

! – не.

Также часто используются следующие ключи условий:

Ключ	Проверка
-z строка	Является ли строка пустой
-f файл	Существует ли файл
-d каталог	Существует ли каталог
-x файл	Является ли файл выполнимым
-w файл	Является ли файл записываемым
-r файл	Является ли файл читаемым

Возможны два вида записи условий:

```
test  $a  -eq  $b
```

или

```
[  $a –eq  $b ]
```
#после первой и перед последней скобкой обязательны пробелы!

Цикл с заранее известным количеством повторений реализуется следующим образом:

```
for  переменная in список
do
     действия
done
```

Здесь список – явно указанный перечень значений 1 2 3 4 5 6 или множество значений в виде /var/* - имена всех объектов из каталога /var или множество, полученное любым другим путем.

Например, вывод имен всех объектов в каталоге /bin

```
for a in /bin/*
 do
   echo $a
done
```

Цикл с условием имеет следующий формат:

```
while условие
do
    действие
done
```

если вместо условия поставить знак : , то это будет бесконечный цикл.

Еще одна часто используемая конструкция – оператор выбора:

```
case   $переменная in

значение1)  действие1;;
значение2)  действие2;;
.
.
.
*) действие по умолчанию;;
esac
```

Как уже говорилось ранее, в скриптах можно использовать все известные команды операционной системы, например:

```
cp  file1 file2
rm file1
cat file2 | wc –l > file3
```

Данная последовательность команд скопирует первый файл во второй, удалит первый, а затем посчитает количество строк во втором файле и запишет результат в третий файл.

В тех случаях, когда нам необходимо результат выполнения команд системы занести в некоторую переменную, используем обратные кавычки:

```
count=`cat file1 | grep "^aaa" | wc -l`
```

В переменную count будет занесено количество строк файла, которые начинаются на сочетание „aaa”.

Удобным средством в языке командного интерпретатора является возможность условного выполнения команд:

```
команда1 &&  команда2

комнада1 || команда2
```

В первом случае команда2 выполнится, только если работа команды1 завершилась успешно. Во втором случае команда2 выполнится, только если команда1 завершила свою работу с ошибкой.

Как известно, любая команда или программа должна возвращать так называемый код возврата (0 – при успешном завершении или не 0 – при ошибке).

Соответственно из скрипта всегда можно выйти принудительно командой exit код возврата, например:

```
exit 0
```

Результат выполнения последней команды всегда хранится в переменной $?.

В языке командного интерпретатора допускается использование массивов. Массивы фактически представляют собой хэш, т.е. индексом может являться и число и строка, инициализация элемента массива происходит следующим образом:

```
A[aaa]=0
A[5]=2
```

Для обращения к значению элемента массива будут использоваться конструкции вида:

```
${A[aaa]}
${A[5]}
```

В системе всегда существует некоторый набор системных переменных, которые хранят важную информацию, доступную всем прикладным программам, в том числе и нашим скриптам.

Например:

BASH	Имя командного интерпретатора
BASH_VERSION	Версия интерпретатора
HOME	Домашний каталог пользователя
LOGNAME	Логин текущего пользователя
OSTYPE	Тип ОС
PWD	Путь к текущему каталогу

Обращение к системным переменным происходит так же, как и к обычным:

$PWD, $HOME и т.д.

Существует возможность передавать значения переменных из одного скрипта в другой. Это осуществляется командой export.

Например:

```
export var1
```

приведет к тому, что в другом скрипте можно обратится к значению переменной как **$var1**.

Язык командного интерпретатора позволяет использовать функции. Формат описания функции следующий:

```
function1() {

действия

return
}
```

Теперь в любом месте скрипта можно вызвать данную функцию:

```
function1
```

Важным моментом в программировании является диалог с пользователем, а мы пока что видели только вывод информации на экран. Ввод данных осуществляется командой read имя переменной.

Например, организация простого меню:

```
echo  "Введите Ваш выбор"
echo  "[1] Выход"
echo  "[2] Копирование"
echo  "[3] Удаление"
read vibor
case $vibor in
1) exit 0;;
2) cp file1 file2;;
3) rm file1
   rm file2;;
     *)   echo "Неверный выбор"
esac
```

При написании сложных проектов можно переменные и функции собирать в отдельные файлы – библиотеки. Подключение библиотеки к скрипту производится командой:

. путь к файлу

например:

```
. /etc/library/scripts
```

При написании скриптов полезными могут оказаться следующие команды ОС

Команда	Комментарий
date	Вывод текущей даты в заданном формате
who	Список текущих пользователей в системе
pwd	Путь к текущему каталогу
ls	Список объектов текущего каталога
cat	Вывод файла на экран
mv	Перемещение объекта
rm	Удаление объекта
chmod	Смена прав доступа
wc	Подсчет строк, слов и символов в файле
greep	Фильтрация (также см. egrep)
sort	Сортировка
head	Вывод первых строк файла
tail	Вывод последних строк файла
diff	Поиск различий между файлами
ps	Вывод списка процессов
cp	Копирование объекта
sleep	Задержка выполнения на указанное время
и т.д.	

Рассмотрим пример написания следующего скрипта: скрипт организовывает диалог с пользователем и позволяет выбрать одно из четырех действий:

- удаление всех файлов текущего каталога, удовлетворяющих заданному шаблону;

- вывод текущей даты и времени на экран;

- вывод списка процессов, запущенных с указанного терминала;

- выход из программы с задержкой на 10 с.

После выполнения действия (1-3) возвращаться в меню программы.

```
#!/bin/sh

while :
#бесконечный цикл
do
#команда очистки экрана
  clear
#вывод меню
  echo "Сделайте Ваш выбор"
  echo "[1] Удаление заданных файлов"
  echo "[2] Время и дата"
  echo "[3] Список процессов"
  echo "[4] Выход"
#чтение выбранной опции
  read  item
#очистка экрана
  clear
#выбор вариантов
  case $item in
#Удаление файлов
    1) echo "Введите шаблон"
       read pattern
       if [ -s $pattern ]
       then
         echo “Шаблон пустой”
       else
         rm $pattern
         echo “Файлы удалены”
       fi
       ;;
#Вывод текущей даты и времени
    2) date
       ;;
#Вывод списка процессов
    3) echo "Введите терминал"
       read tty
       if [ -s $tty ]
       then
         echo “Терминал не указан”
       else
         ps -t $tty
       fi
```

```
        ;;
#Выход из программы
    4) echo "Выход через 10 секунд"
        sleep 10
        exit 0
        ;;
#Действие по умолчанию
    *) echo "Неверная опция"
        ;;
  esac
  echo "Нажмите клавишу"
  read key
done
```

Следует заметить, что в большинстве случаев скрипты представляют собой бесконечный цикл анализа каких-либо событий, поэтому выход из него возможен или по нажатию Ctrl+C или после выполнения команды kill.

При этом возникает необходимость выполнять некоторые действия по завершению программы при получении соответствующих сигналов. Это можно реализовать с помощью команды trap.

Данная команда имеет формат: trap команда(ы) сигнал(ы).

Например:

```
trap 'rm file1; echo "Exit"; exit 0' 2, 9
```

устанавливает ловушку для сигналов 2 и 9 – удаление файла, вывод сообщения на экран и выход из программы.

Наиболее распространенные сигналы следующие: 0 – выход из командного интерпретатора, 2 – прерывание по Ctrl+C, 3 – выход из программы, 9 – сигнал от команды kill.

Отдельно следует указать, как запускать скрипт на выполнение. Для этого необходимо установить права, разрешающие выполнение данного файла, например командой chmod a+x script. Далее можно запускать из командной строки ./script или нажав <eneter> на имени файла в оболочке (midnight commander).

Общие требования к выполнению лабораторной работы

Разработать программу-скрипт средствами Shell. Программа должна запускаться в следующем формате: *script.sh Time NumItem ItemFile*, где

Time – длительность промежутка времени;

NumItem – количество пунктов меню;

ItemFile – текстовый файл с наименованием пунктов меню.

Скрипт должен выполнять следующие действия:

1. При запуске контролировать наличие необходимых параметров и при необходимости выдавать сообщение об ошибке.
2. Выводить сообщение-подсказку о выполняемом задании.
3. Формировать меню с требуемым количеством пунктов. Информация о названии пунктов берется из текстового конфигурационного файла.
4. Реализовывать выполнение пунктов меню в соответствии с индивидуальным заданием.
5. Для периодических действий период повтора брать из параметра Time.
6. Результаты выполнения должны выводится на экран в формате:

----------Дата---Время----------

Результаты

---

7. Предусмотреть выход из скрипта по заданному условию и возврат в главное меню программы.
8. При необходимости параметры работы скрипта могут вводится в диалоговом режиме.
9. Все действия, производимые скриптом фиксировать в файле журнала.

Задания к лабораторной работе

Вариант 1

Пусть в указанном каталоге появляются файлы-письма, которые имеют такой формат:

From aaaaa@mail.com

To: bbbb@mail.com

Subject:ccccccccc

Message text

Формировать на экране отчет в виде:

Дата, время, от кого, кому, размер

Содержимое писем разбрасывать в каталоги, названия которых совпадают с именем получателя.

Выход при появлении письма с темой «stop».

Обеспечить возможность просмотра отчета, сформированного в предыдущем задании и удаления информации об указанных письмах.

Вариант 2

Вывод на экран информации о процессах, запущенных заданным пользователем и имеющих заданный статус. Для каждого процесса выводить PID, владельца, терминал, статус и приоритет.

Анализ входа в систему новых пользователей. Для каждого пользователя указывать логин, терминал и время пребывания в системе. Выход при превышении суммарного количества пользователей заданной величины.

Удаление процессов заданного владельца.

Вариант 3

Вывод в окне терминала – счетчика таймера с обратным отсчетом от указанной величины. При обнулении счетчика показать на экране первые 5 процессов, имеющих наивысший приоритет в системе.

Анализировать появление процессов с наличием поправки, внесенной командой nice. Выводить их суммарное количество, поправку, имена и PID-ы новых процессов. Выход при появлении в текущем каталоге файла с именем stop.

Вывод на экран содержимого файла текущего каталога, имеющего максимальный размер.

Вариант 4

Реализация подсчета среднего размера файла в указанном каталоге, а также кол-ва объектов разного вида (файл, каталог, ссылка).

Определение числа файлов заданного каталога, имеющих четный и нечетный размер соответственно.

Отслеживание появления файлов, имя которых удовлетворяет заданному шаблону. Выход, если количество таких файлов превысит указанную величину.

Вариант 5

Поиск в двух указанных каталогах файлов с одинаковым содержимым. Вывод имен этих файлов и размера в строках.

Создание в текущем каталоге папки вида: data-time до тех пор, пока в текущем каталоге не появится файл с именем stop. В каждую папку копировать из текущего каталога файлы, заканчивающиеся на символы a – в первую папку, b – во вторую папку и т. д.

При выходе удаление всех созданных папок и создание файла, в который записывается их количество.

Вариант 6

В заданном каталоге рассортировать файлы по размеру: создать папки имена, которых соответствует размерам, и перенести туда все файлы соответствующих размеров.

В заданном каталоге отслеживать изменение размеров файлов. Для каждого файла выводить на старый размер, новый размер и признак + или -

(увеличился, уменьшился). Выход, если размеры файлов не менялись за три последовательных вызова скрипта.

При выходе из программы вывод на экран отчета о количестве изменений размеров, зафиксированных при анализе, если анализ ни разу не запускался, то выдать соответствующее сообщение.

Вариант 7

Подсчет частоты повторений каждого слова во всех текстовых файлах указанного каталога (*.txt).

Формирование списков файлов имеющих одинаковый первый символ имени. Списки сохранять в файлах вида a.log, b.log и т.д. в указанном каталоге.

Отслеживание изменений прав доступа к объектам в указанном каталоге. Выводить имена изменившихся объектов, старый набор прав и новый. Выход при появлении объекта, у которого права доступа имеют вид ---------.

Вариант 8

Пусть есть файл с перечнем имен файлов, с указанием полного пути. По команде пользователя анализировать наличие в системе файлов из указанного списка. Выводить имена существующих файлов и их суммарное количество. Выход из программы, если в системе присутствуют все файлы из списка.

Определение максимального уровня вложенности файлов в системе. Вывести соответствующий путь.

При выходе из программы создавать каталог, имя которого содержит имя текущего пользователя, время и дату.

Вариант 9

Определить разность между максимальным и минимальным размерами файлов во всех подкаталогах указанного каталога.

Анализировать изменение количества процессов запущенных указанным пользователем в системе. Результат выводить в виде диаграммы вида:

****	4 процесса
**	2 процесса
******	6 процессов
***	3 процесса
*	1 процесс
*	1 процесс
**	2 процесса

и т .д. В начале каждой строки выводить текущее время.

Выход при превышении количества процессов некоторой заданной границы.

При выходе из программы формирование списка процессов, появившихся в системе за время работы скрипта.

Вариант 10

Вывести список файлов заданного каталога, в которых количество строчных и прописных букв одинаково.

Отслеживать появление в указанном каталоге файлов, у которых в тексте присутствуют URL (сочетания вида http://********). Выводить имя файла, список URL, не повторяя одинаковые ссылки. Выход в случае появления файла, содержащего 2 слова stop в тексте.

При выходе из программы вывод списка процессов, запущенных от имени указанного пользователя.

Вариант 11

Вывод списка файлов заданного каталога с возможностью сортировки по размеру, имени, владельцу.

Отслеживание появления и исчезновения файлов, размер которых лежит в заданном диапазоне. Выводить для каждого файла признак

«появился», «исчез», размер и кол-во строк в файле. Выход из цикла анализа при появлении в текущем каталоге пустого файла с именем stop.

Вывод текущего времени на экран.

Вариант 12

Реализовать ввод массива целых чисел. Обеспечить по выбору пользователя:

- поиск максимума;

- поиск минимума;

- расчет среднего значения.

Вывести на экран информацию о процессах, PID-ы которых совпадают с элементами массива.

Отслеживать превышение числа процессов в системе некоторой указанной границы.

Выход из цикла анализа при появлении в системе процесса с указанным именем.

Вариант 13

Выводить на экран права доступа к файлам указанного каталога, которые созданы позже указанной даты. Для указанного пользователем файла изменять права доступа на новые, введенные пользователем.

Отслеживать создание в текущем каталоге подкаталогов, имена которых состоят только из цифр. Выход, если количество таких подкаталогов станет равным количеству файлов в текущем каталоге.

При выходе из программы уничтожать все имеющиеся подкаталоги, имена которых содержат указанную цифру.

Вариант 14

Отслеживать появление в указанном каталоге файлов с указанным расширением. На экран выводить N первых строк файла. Где N – номер текущего терминала. Выход, если количество файлов превысит указанную границу.

При выходе из программы формировать на экране отчет о количестве новых файлов и среднем числе строк, слов и символов в них.

Вариант 15

Выводить информацию о процессах, которые запущены указанным пользователем с указанного терминала.

Отслеживать появление новых процессов в системе. Формировать пятерку первых процессов по проценту использования процессора. Выход при запуске редактора vi любым пользователем системы.

При выходе из программы удаление в текущем каталоге всех файлов с указанным расширением.

Требования к отчету по лабораторной работе:

1. Задание к лабораторной работе.
2. Код скрипта с комментариями основных команд.
3. Входная информация.
4. Результаты работы скрипта.

Контрольные вопросы:

1. Что такое командный интерпретатор?
2. Как запустить скрипт на выполнение?
3. В каких случаях целесообразно использование команды trap?
4. Как реализовать задержку на требуемое время?
5. Какие средства языка shell предусмотрены для арифметических вычислений.

Лабораторная работа № 6

Тема: Межпроцессное взаимодействие в ОС UNIX.

Цель работы: освоить основные принципы работы средств межпроцессного взаимодействия, синхронизации процессов, научиться использовать системные вызовы средств межпроцессного взаимодействия в ОС UNIX.

Методические указания к выполнению работы

Межпроцессное взаимодействие (Interprocess communication, IPC) – это механизм, с помощью которого два и более процессов осуществляют друг с другом взаимодействие, направленное на выполнение определённых задач. Межпроцессное взаимодействие поддерживают все UNIX системы, однако, в каждой из них этот механизм реализован по-разному. В UNIX системах выделяют следующие основные средства межпроцессного взаимодействия:

- сигналы
- сообщения – обмен форматированными данными;
- семафоры – набор общесистемных переменных;
- разделяемая память – совместное использование общей памяти

Кроме перечисленных выше основных средств межпроцессного взаимодействия, в UNIX поддерживаются отображаемая память, каналы, гнёзда и интерфейс транспортного уровня.

1 Сигналы

Сигналы можно рассматривать как запросы на прерывания на уровне процессов. Сигналы в UNIX представляются целыми числами, их определено свыше 30 в файле signal.h

Таблица 1 – Описание сигналов

Сигнал	Описание
SIGFPE	Недопустимая математическая операция
SIGHUP	Разрыв связи с управляющим терминалом
SIGILL	Попытка выполнить недопустимую машинную команду
SIGINT	Прерывание процесса. Обычно генерируется комбинацией клавиш Ctrl+C
SIGKILL	Уничтожение процесса
SIGCHLD	Посылается в родительский процесс при завершении порожденного процесса
SIGTERM	Завершение процесса
SIGSEGV	Ошибка сегментации
SIGSTP	Остановка процесса комбинацией клавиш Ctrl+Z
SIGCONT	Возобновление остановленного процесса

Обычно посылаются 1) от одного процесса к другому как средство IPC, 2) драйвером терминала при возникновении определённого события, например, нажатии комбинации клавиш Ctrl+C, Ctrl+Z, 3) командой kill, 4) ядром в исключительных ситуациях, например, деление на ноль.

Когда поступает сигнал, возможен один из двух вариантов развития события:

1. Если процесс назначил данному сигналу программу обработчик, то она вызывается и ей передаётся информация о контексте, в котором был сгенерирован сигнал. Процедуру вызова обработчика называют перехватом сигнала. Когда выполнение обработчика завершается, процесс возобновляется с той точки, где был получен сигнал.
2. В противном случае ядро выполняет от имени процесса действия, предусмотренные по умолчанию. Эти действия различны для разных сигналов. Чаще всего они завершают процесс, при этом может

создаваться дамп памяти – файл, содержащий образ памяти процесса.

Каждая запись таблицы процесса содержит *массив сигнальных флагов* (по одному флагу для каждого сигнала, определённого в системе). Когда для процесса генерируется сигнал, ядро устанавливает соответствующий флаг в единицу. Если процесс получатель находится в режиме ожидания, ядро «будит» его и ставит в очередь на выполнение. Если процесс работает, то ядро проверяет в *массив спецификаций обработки сигналов*, где каждый элемент массива соответствует сигналу, определённому в системе. С помощью этого массива ядро определяет, как ядро будит реагировать на поступивший сигнал: если элемент равен нулю, то выполняется стандартный обработчик, если равен единице, то сигнал игнорируется, иначе, указанное число считается адресом функции-обработчика. Если обработки ожидают разные сигналы, то порядок передачи и обработки не определён.

Кроме этого каждый процесс содержит сигнальную маску, используемую для блокировки сигналов. При блокировке сигнала процесс ставится в очередь на обработку сигнала, но не делает никаких действий до явного разблокирования сигнала, т.е. до явного изменения сигнальной маски. Обработчик вызывается для разблокированного сигнала только один раз, даже если в течение периода блокировки поступило несколько сигналов.

Для работы с сигналами в UNIX определены следующие функции

```
void (*signal (int sigid, void (*handler)(int)))(int)
```

Функция **`signal`** задаёт обработчик сигнала. Параметр `sigid` указывает идентификатор сигнала (см. таблицу 1). Параметр `handler` является указателем на функцию обработчик сигнала. Функция обработчик должна иметь один целочисленный аргумент и возвращать тип void. Функция **`signal`** возвращает указатель на предыдущий обработчик сигнала. Пример использования этой функции:

```
#include <signal.h>
#include <stdio.h>
//обработчик
```

```
void catch (int sigid)
{
          printf("catch signal %d\n",sigid);
}

int main()
{
          void (*old)(int);
          old = signal(SIGTERM, catch);
          ...
          signal(SIGTERM, old);
}
```

Для игнорирования сигнала необходимо передать функции **signal** в качестве второго аргумента единицу, приведённую к типу указателя на функцию, т.е. void (*)(int)1. Для установки обработчика по умолчанию нужно передать ноль . void (*)(int)0.

```
int kill (pid_t pid, int sigid)
```

Функция **kill** посылает заданный параметром `sigid` сигнал процессу, определённому в параметре `pid`. Функция посылает сигнал, если процесс отправитель и процесс получатель принадлежат одной группе пользователей или отправитель является привилегированным. Если параметр `pid` равен нулю, то сигнал посылается всем процессам, чьи идентификаторы группы GID совпадают с идентификатором группы отправителя. В случае успеха функция возвращает 0, в случае неудачи –1.

2 Сообщения

Сообщения – это средства межпроцессного взаимодействия, позволяющее нескольким процессам, выполняемым на одной UNIX-машине, взаимодействовать между собой путем приёма и передачи форматированных сообщений.

Сообщения позволяют осуществлять доступ к очереди сообщений сразу нескольким процессам. При этом каждый процесс, который помещает сообщение в очередь, должен указать для своего сообщения целочисленный

тип. Процесс-получатель сможет выбрать это сообщение, указав тот же тип. Сообщения не удаляются из очереди, даже если к ней не обращается ни один процесс. Сообщения удаляются из очереди только тогда, когда процессы обращаются к ним явно.

В адресном пространстве ядра имеется таблица очередей сообщений, в которой отслеживается все очереди сообщений, создаваемые в системе. В каждой записи таблицы сообщений можно найти следующие данные, относящиеся к одной из очередей:

- Целочисленный идентификатор – ключ, присвоенный очереди процессом, который её создал. Другие процессы используют этот ключ для доступа к очереди и получения дескриптора очереди.
- Идентификаторы пользователя (UID) и группы (GID). Процесс, идентификатор пользователя (UID) которого совпадает с UID создателя очереди, имеет право удалять очередь и изменять параметры управления ею.
- Права доступа к очереди для чтения-записи по категориям владелец, группа и прочие.
- Указатель на связный список сообщений, находящихся в очереди. В каждой записи списка хранится одно сообщение и присвоенный ему тип.

Когда процесс передает сообщение в очередь, ядро создает для него запись и помещает её в конец связного списка, соответствующего сообщениям указанной очереди. В каждой такой записи указывается тип сообщения, число байт данных и указатель на другую область ядра, где находятся собственно данные сообщения. Ядро копирует данные, находящиеся в сообщении, из адресного пространства процесса-отправителя в область данных ядра, чтобы процесс отправитель мог завершиться, а сообщение при этом осталось доступным для чтения другими процессами.

Для работы с сообщениями в UNIX определены следующие функции

```
int msgget (key_t key, int flag)
```

Функция **msgget** открывает очередь сообщений. Параметр `key` указывает ключ очереди для открытия. Параметр `flag` управляет процессом создания сообщений. Если параметр `flag` имеет нулевое значение, и нет очереди сообщений, ключ которой совпадает с параметром `key`, то функция прерывается. Для создания новой очереди сообщений нужно в качестве второго параметра передать макрос IPC_CREATE и указать права на чтение и запись. Пример использования этой функции:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/messages.h>

//создать очередь
int msgdesc = msgget (10, IPC_CREATE | 0644);

int msgsnd (int desc, const void* msgPtr, int len, int flag)
```

Функция **msgsnd** отправляет сообщение в очередь сообщений, открытую с помощью функции `msgget`. Параметр `desc` задает открытую очередь. Параметр `msgPtr` задает объект, который реально содержит текст и тип сообщения, подлежащий передачи, в этом случае используется следующая структура:

```
struct msgbuf
{
          long mtype;          //тип сообщения
          char mtext[MSGMAX]; //буфер для текста сообщения
}
```

Параметр `len` задает размер в байтах поля нуля `mtext` объекта, на который указывает параметр `msgPtr`. Последний параметр `flag` может иметь значение 0. Это означает, что процесс будет блокирован, пока функция успешно не завершиться. Если параметр `flag` имеет значение IPC_NOWAIT, то при блокировании процесса выполнение функции прерывается.

В случае успешного выполнения функция **msgsnd** возвращает 0, а в случае неудачи –1.

```
int msgrcv (int desc, void* msgPtr, int len, int mtype, int
flag)
```

Функция **msgrcv** принимает сообщение из очереди сообщений, открытой с помощью функции msgget. Параметр desc задает открытую очередь. Полученое сообщение сохраняется в объекте, на который указывает параметр msgPtr. В этом случае используется такая же структура struct msgbuf, как и в функции **msgsnd.**

Параметр len задает максимальный размер в байтах текста сообщения, которое может быть принято объектом msgPtr. Значение параметра mtype определяет тип сообщения, подлежащего приему. Если этот параметр равен нулю, то функция **msgrcv** принимает самое старое сообщение любого типа. Если параметр mtype является положительным целым числом, то принимается самое старое сообщение указанного типа. Отрицательное значение параметра mtype заставляет функцию **msgrcv** принять сообщение, тип которого меньше абсолютного значения mtype или равен ему; если таких сообщений в очереди несколько, то принимается то, которое является самым старым и имеет наименьшее значение типа mtype. Параметр flag может иметь значение 0. Это означает, что процесс будет блокирован, если ни одно сообщение в очереди не удовлетворяет критериям выбора. Если в очереди есть сообщение, которое удовлетворяет критериям выбора, но превышает величину len, то функция возвращает код неудачи. Если параметр flag имеет значение IPC_NOWAIT, то вызов функции будет неблокирующим.

В случае успешного выполнения функция **msgrcv** возвращает количество байтов записанных в буфер mtext, а в случае неудачи –1.

```
int msgctl (int desc, int cmd, struct msgid_ds* mbufPtr)
```

Функция **msgctl** позволяет запрашивать управляющие параметры очереди сообщений, обозначенной параметром desc, изменять информацию в управляющих параметрах очереди, а также удалять очередь из системы. С

помощью этой функции можно устанавливать идентификатор UID владельца очереди и идентификатор его группы, а также права доступа. Ниже в таблице указаны возможные значения параметра `cmd` и их смысл.

Таблица 2 – Описание команд для функции **`msgctl`**

Значение параметра `cmd`	Смысл
IPC_STAT	Копировать управляющие параметры очереди сообщений из адресного пространства ядра в объект, указанный аргументом `mbufPtr`
IPC_SET	Заменить управляющие параметры очереди сообщений в адресном пространстве ядра параметрами, содержащимися в объекте, на который указывает `mbufPtr`. Для выполнения этой операции вызывающий процесс должен иметь право либо привилегированного пользователя, либо создателя или назначенного владельца очереди.
IPC_RMIR	Удалить очередь из системы. Для выполнения этой операции вызывающий процесс должен иметь право либо привилегированного пользователя, либо создателя или назначенного владельца очереди.

Тип данных `struct msgid_ds`, используемый параметром `mbufPtr`, определяется в заголовочном файле sys/message.h. В таблице ниже перечислены информационные поля этой структуры

Таблица 3 – Описание полей структуры `struct msgid_ds`

Поле	Описание
msg_perm	Права доступа, хранящиеся в записи типа struct ipc_perm, определяемой в файле ipc.h

msg_first	Указатель на самое старое сообщение в очереди, первое сообщение
msg_last	Указатель на последнее сообщение в очереди, самое новое сообщение
msg_cbyte	Общее число байт во всех сообщениях, находящихся в очереди на текущий момент
msg_qnum	Общее число сообщений, находящихся в очереди на текущий момент
msg_lspid	Идентификатор процесса, который последний передал в очередь сообщение
msg_lrpid	Идентификатор процесса, который последний прочитал из очереди сообщение
msg_stime	Время, когда в очередь было передано самое последнее сообщение
msg_rtime	Время, когда из очереди было прочитано самое последнее сообщение
msg_ctime	Время последнего изменения управляющих параметров очереди сообщений

В случае успешного выполнения функция **`msgctl`** возвращает 0, а в случае неудачи –1.

3 Семафоры

Семафоры – это средства синхронизации множества процессов. Семафоры в UNIX группируются в наборы, каждый из которых содержит один и более семафоров. Семафоры часто используются вместе с разделяемой памятью, реализуя, таким образом, мощный метод межпроцессного взаимодействия.

Значением семафора может быть переменная беззнакового целого типа (unsigned short). Если процесс пытается уменьшить значение семафора так,

что бы оно стало отрицательным, то эта операция и сам процесс будут заблокированы до тех пор, пока другой процесс не увеличит значение данного семафора до величины, достаточной для успешного выполнения операции заблокированного процесса.

Семафоры хранятся в адресном пространстве ядра. В наборе семафоры обозначаются индексом, начиная с нуля. Если осуществляется работа с набором семафоров и возникает блокировка одного семафора из набора, то блокировка выполняется над всеми семафорами из набора.

Для работы с семафорами в UNIX определены следующие функции

```
int semget (key_t key, int num_sem, int flag)
```

Функция **semget** открывает набор семафоров, идентификатор которого задан значением параметра `key`, и возвращает неотрицательный целочисленный дескриптор, который должен быть использован другими функциями для работы с семафорами. Параметр `flag` управляет процессом создания сообщений. Если параметр `flag` имеет нулевое значение, и нет набора семафоров, ключ которого совпадает с параметром `key`, то функция прерывает свою работу. В противном случае возвращается дескриптор набора семафоров. Если необходимо создать новый набор семафоров с идентификатором `key`, и набора с таким идентификатором нет, то значение параметра `flag` должно представлять собой результат побитового сложения константы IPC_CREATE и числовых значений прав доступа к новому набору на чтение и запись.

Значение параметра `num_sem` устанавливается равным нулю, если параметр `flag` не содержит константу IPC_CREATE. Пример использования этой функции:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

//создать набор из двух семафоров
int perm = S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH;
```

```
int semdesc = semget (11, 2, IPC_CREATE | perm);
```

```
int semop (int desc, struct sembuf* opPtr, int len)
```

Функция **semop** позволяет изменить одно или несколько значений семафоров в наборе, открытом с помощью функции semget. Кроме этого функция **semop** позволяет проверять равенство значений семафоров нулю. Параметр desc задает набор семафоров. Параметр opPtr задает массив объектов типа struct sembuf. Каждый такой объект задает одну операцию – запрос или изменение значения семафора. Параметр len задает количество элементов в массиве opPtr . Тип данных struct sembuf определяется в файле sys/sem.h следующим образом

```
struct sembuf
{
          short     sem_num;       // индекс семафора
          short     sem_op;        // операция
     short     sem_flg;       // флаг операции
}
```

Описание возможных значений поля sem_op приводится в следующей таблице

Таблица 4 – Описание значений поля sem_op

Значение поля sem_op	Смысл операции
Положительное число	Увеличить значение семафора с индексом sem_num на величину sem_op
Отрицательное число	Уменьшить значение семафора с индексом sem_num на величину sem_op
Ноль (0)	Проверить равенство значения семафора с индексом sem_num нулю

Если вызов функции **semop** попытается уменьшить значение семафора до отрицательного числа или посчитает, что значение семафора равно нулю, когда на самом деле это не так, то ядро заблокирует вызывающий процесс. Этого не произойдет, если поле sem_flag имеет значение IPC_NOWAIT

В случае успешного выполнения функция **semop** возвращает 0, а в случае неудачи –1.

```
int semctl (int desc, int num, int cmd, union semun arg)
```

Функция **semctl** позволяет запрашивать и изменять управляющие параметры набора семафоров, заданного параметром desc, а также удалять семафор.

Значение параметра num задает индекс семафора, а параметр cmd определяет операцию, которая должна быть выполнена над конкретным семафором данного набора.

Последний аргумент arg может использоваться для задания или выборки управляющих параметров одного или нескольких семафоров набора в соответствии с аргументом cmd. Тип данных union semun определяется в файле sys/sem.h следующим образом

```
union semun
{
          int                  val;      // значение семафора
          struct semid_ds*     buf;      // параметры набора
     ushort              array;    // значения семафоров

}
```

В случае успешного выполнения функция **semctl** возвращает 0, а в случае неудачи –1.

Ниже в таблице указаны возможные значения параметра cmd и их смысл.

Таблица 5 – Описание команд функции **semctl**

Значение параметра `cmd`	Смысл
IPC_STAT	Копировать управляющие параметры набора семафоров из адресного пространства ядра в объект, указанный аргументом `arg.buf`. У вызывающего процесса должно быть право на чтение набора
IPC_SET	Заменить управляющие параметры набора семафоров в адресном пространстве ядра параметрами, содержащимися в объекте, на который указывает `arg.buf`. Для выполнения этой операции вызывающий процесс должен иметь право либо привилегированного пользователя, либо создателя или назначенного владельца очереди.
IPC_RMIR	Удалить семафор из системы. Для выполнения этой операции вызывающий процесс должен иметь право либо привилегированного пользователя, либо создателя или назначенного владельца очереди.
GETALL	Скопировать все значения семафоров в массив, на который указывает `arg.array`
SETALL	Установить все значения семафоров набора равными значениям в массиве, на который указывает `arg.array`
GETVAL	Возвратить значение семафора с индексом `num`. Параметр `arg` не используется.
SETVAL	Установить значение семафора с индексом `num` равным значению, указанному в `arg.val`.
GETNCNT	Возвратить количество процессов, которые в текущий момент заблокированы и ожидают увеличения значения семафора с индексом `num`. Параметр `arg` не используется.

GETZCNT	Возвратить количество процессов, которые в текущий момент заблокированы и ожидают обращения значения семафора с индексом `num` в ноль Параметр `arg` не используется.

В случае успешного выполнения функция **`semctl`** возвращает 0, а в случае неудачи –1.

4 Разделяемая память

Разделяемая память позволяет множеству процессов отображать часть своих виртуальных адресов на общую область памяти. Разделяемая память располагается в адресном пространстве ядра, поэтому области разделяемой памяти не освобождаются, даже если создавшие их процессы завершаются.

Все разделяемые области отслеживаются через таблицу разделяемых областей. В этой таблице, аналогично таблице очередей сообщений или наборов семафоров, хранится информация о параметрах разделяемой области памяти.

Для работы с разделяемой памятью в UNIX определены следующие функции.

```
int shmget (key_t key, int size, int flag)
```

Функция **`shmget`** открывает разделяемую область памяти, идентификатор которой совпадает с параметром `key` и возвращает дескриптор этой области.

Параметр `size` задает размер области, которая может быть позже подсоединена к процессу с помощью функции shmat. Если разделяемая область создается, то параметр `size` задает размер новой области. Параметр `flag` управляет процессом создания области разделяемой памяти. Если параметр `flag` имеет нулевое значение, и нет области разделяемой памяти, ключ которой совпадает с параметром `key`, то функция завершается с

неудачей. Для создания новой области разделяемой памяти нужно в качестве параметра `flag` передать макрос IPC_CREATE и указать права на доступ к области памяти. В случае успешного выполнения функция **shmget** возвращает дескриптор разделяемой области, а в случае неудачи –1.

Пример использования этой функции:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

//создать область разделяемой памяти
int perm = S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH;
int shmdesc = shmget (90, 128, IPC_CREATE | perm);
```

```
void* shmat (int desc, void* addr, int flag)
```

Функция **shmat** подсоединяет область разделяемой памяти, указанную дескриптором `desc`, к виртуальному адресному пространству вызывающего процесса. После выполнения этой функции процесс получает возможность читать и записывать данные в область разделяемой памяти.

Параметр `addr` задает начальный виртуальный адрес адресного пространства вызывающего процесса, в которое необходимо отобразить разделяемую память. Если это значение равно нулю, то ядро самостоятельно определит в вызывающем процессе подходящий виртуальный адрес. Функция **shmat** в случае успеха возвращает виртуальный адрес области отображения разделяемой памяти, в противном случае возвращает –1.

```
int shmdt (void* addr)
```

Функция **shmdt** отсоединяет область разделяемой памяти от заданного параметром `addr` виртуального адреса вызывающего процесса. Параметр `addr` должен быть получен с помощью функции **shmat.** В случае успешного выполнения функция **shmdt** возвращает 0, а в случае неудачи –1.

```
int shmctl (int desc, int cmd, struct shmid_ds* buf)
```

Функция **shmctl** позволяет запрашивать управляющие параметры разделяемой области памяти, заданной параметром desc, изменять информацию в управляющих параметрах области разделяемой памяти, а также удалять область разделяемой памяти. Параметр задает команду функции **shmctl**, ниже в таблице указаны возможные значения параметра cmd и их смысл.

Параметр buf задает адрес объекта типа struct shmid_ds, который используется для задания и выборки управляющих параметров разделяемой памяти.

Таблица 6 – Описание команд функции **shmctl**

Значение параметра cmd	Смысл
IPC_STAT	Копировать управляющие параметры разделяемой области памяти из адресного пространства ядра в объект, указанный аргументом buf
IPC_SET	Заменить управляющие параметры разделяемой области памяти в адресном пространстве ядра параметрами, содержащимися в объекте, на который указывает buf. Для выполнения этой операции вызывающий процесс должен иметь право либо привилегированного пользователя, либо создателя или назначенного владельца очереди.
IPC_RMIR	Удалить разделяемую область памяти из системы. Для выполнения этой операции вызывающий процесс должен иметь право либо привилегированного пользователя, либо создателя или назначенного владельца очереди. Если к разделяемой области памяти, подлежащей удалению, подсоединены процессы, то операция удаления будет отложена до тех

	пор, пока эти процессы не отсоединяться от нее.

Тип данных `struct shmid_ds`, используемый параметром `buf`, определяется в заголовочном файле sys/shm.h. В таблице ниже перечислены информационные поля этой структуры

Таблица 7 – Описание полей структуры `struct shmid_ds`

Поле	Описание
shm_perm	Права доступа, хранящиеся в записи типа struct ipc_perm, определяемой в файле ipc.h
shm_segsz	Размер разделяемой области памяти в байтах
shm_lpid	Идентификатор процесса, который в последний раз присоединялся к области
shm_cpid	Идентификатор процесса-создателя
shm_nattch	Количество процессов, подсоединенных к разделяемой области памяти в текущий момент
shm_atime	Время, когда процесс последний раз присоединялся к области

В случае успешного выполнения функция **`shmctl`** возвращает 0, а в случае неудачи –1.

Для написания программ, использующих средства межпроцессного взаимодействия можно использовать в качестве редактора текстовый редактор vi и стандартный компилятор программ на языке Си: команда «cc» или «gcc».

Например, для компиляции простой программы на языке Си, написанной в файле file.c нужно выполнить команду

```
% gcc –c –W –Wall file.c
```

В результате успешной компиляции получится объектный файл file.o, над которым необходимо выполнить связывание с помощью команды

```
% gcc –o file file.o –lm
```

После этой команды в случае успеха получается исполнимый бинарный файл `file`.

Задание к выполнению лабораторной работы

В соответствии с вариантом написать программу, демонстрирующую возможности средств межпроцессного взаимодействия и синхронизации процессов в операционной системе UNIX.

1. Перехват сигнала нажатия комбинации клавиш CTRL+C
2. Перехват сигнала уничтожения процесса
3. Игнорирование сигнала уничтожение процесса
4. Команда отправки сигналов
5. Отправка и получение сообщений
6. Показать идентификатор процесса, который последний отправил сообщение
7. Показать идентификатор процесса, который последний прочитал сообщение
8. Показать идентификатор пользователя – владельца очереди сообщений
9. Показать количество сообщений в очереди сообщений
10. Удаление очереди сообщений
11. Синхронизация процессов с помощью семафоров
12. Показать значения набора семафоров
13. Показать количество процессов, заблокированных семафором
14. Удаление набора семафоров
15. Обмен сообщениями через разделяемую область памяти
16. Показать размер разделяемой области памяти
17. Показать идентификатор процесса, который в последний раз присоединялся к разделяемой области памяти
18. Показать идентификатор процесса-создателя разделяемой области памяти

19. Показать количество процессов, подсоединенных к разделяемой области памяти в текущий момент
20. Удаление разделяемой области памяти
21. Межпроцессное взаимодействие с помощью разделяемой памяти и семафоров
22. Межпроцессное взаимодействие с помощью разделяемой памяти и семафоров.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Описание функций и команд, которые использовались для выполнения задания.
4. Текст программы.
5. Тестовые примеры.
6. Выводы.

Контрольные вопросы:

1. Что такое средства межпроцессного взаимодействия?
2. Механизм сигналов в UNIX?
3. Что такое перехват сигналов?
4. Функции для работы с сигналами в UNIX?
5. Механизм очередей сообщений в UNIX?
6. Системные вызовы для работы с сообщениями в UNIX?
7. Права доступа и сообщения в UNIX?
8. Семафоры и их наборы в UNIX?
9. Каким образом просмотреть значения семафоров в UNIX?
10. Как осуществляется синхронизация на основе семафоров?
11. Механизм разделяемой памяти в UNIX?
12. Как записать данные в разделяемую память в UNIX?
13. Кто может удалить область разделяемой памяти?

Лабораторная работа №7

Тема: Изучение графического интерфейса ОС Linux.

Цель: Научиться использовать возможности графического интерфейса ОС Linux.

Методические указания к лабораторной работе

ОС Linux, которая изначально обладала интерфейсом командной строки и не позиционировалась как ОС для десктопов, в последнее время стала значительно более дружественной пользователю.

Бурное развитие данной линейки ОС привело к появлению огромного числа разных версий Linux и различных дистрибутивов: ASP Linux, ALT Linux, RedHat, Mandriva и т.д. Все эти дистрибутивы обладают интуитивно понятным графическим интерфейсом для инсталляции, содержат богатый набор графических оболочек для реализации полноценного пользовательского интерфейса («как в Windows»). Современный графический интерфейс ОС Linux – это среда с удобными, эргономично расположенными элементами управления, многообразными средствами настройки системы «под себя», широкий выбор ПО для работы в графической среде. Если раньше сторонники Windows утверждали, что под Linux невозможно найти нормального текстового редактора, графического редактора, игры и т.д., то сейчас все это есть. В графической среде Linux работает масса ПО – редакторы графики, текста, видео, базы данных, мультимедиа ПО, игры и многое другое. В подавляющем большинстве – это все бесплатное ПО с открытым кодом, хотя и под Linux можно встретить коммерческие продукты. Таким образом, ОС Linux в современном мире – это ОС и для серверов и для десктопов, а также ОС, которая активно используется в мобильных устройствах, встраиваемых системах и т.д.

Особую популярность в последнее время приобрели так называемые «живые» (Live CD) дистрибутивы, которые позволяют загружать полноценную ОС без инсталляции на жесткий диск (с любого сменного

носителя – CD, DVD, Flash). Очень часто такие дистрибутивы имеют достаточно малый размер (до 50 Мб), но при этом позволяют за считанные минуты развернуть сервер с полным набором серверного ПО или десктоп с графическими редакторами, текстовыми процессорами и многим другим. Очень удобны такие дистрибутивы на лабораторных практикумах, так как позволяют изучать ОС, не затрагивая текущую конфигурацию компьютеров в аудитории. В данной лабораторной работе графический интерфейс ОС Linux рассматривается именно на примере таких дистрибутивов. Существует множество компактных «живых» дистрибутивов. В работе в качестве примера рассматривается UBUNTU -6.06.1- desktop edition (www.ubuntu.org). Для загрузки нам понадобится дистрибутив на CD или Flash. Дистрибутив можно скачать с сайта разработчика в формате *.iso (размер около 400 Мб).

Для загрузки с компакт-диска или флэш-памяти необходимо установить в биос опцию, которая определяет, с какого устройства будет выполняться загрузка. Варианты такой опции представлены на рисунках 1- 3.

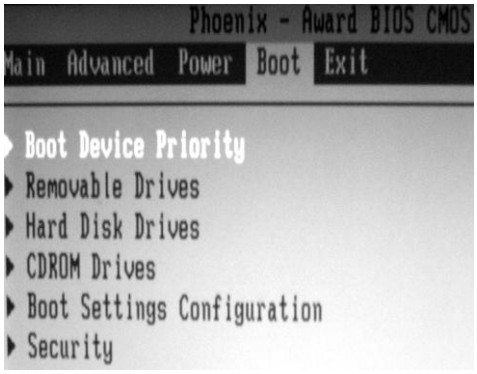

Рисунок 1 – Выбор пункта «Приоритет загрузочных устройств в биос»

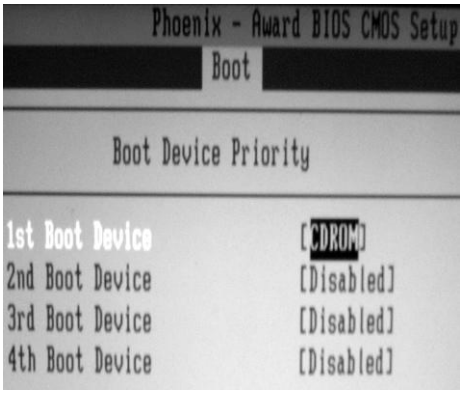

Рисунок 2 – Установка загрузки с носителя CD

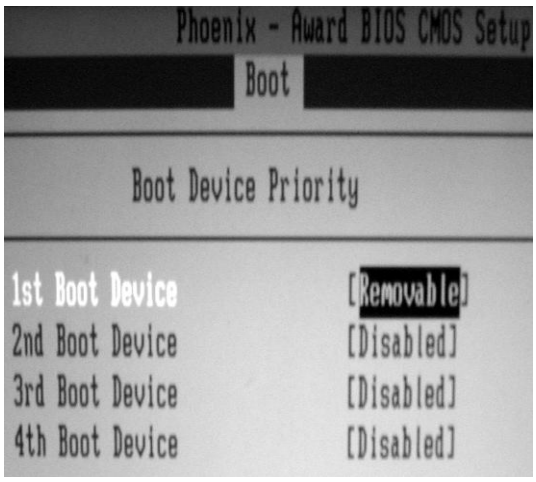

Рисунок 3 – Установка загрузки со сменного носителя (Flash)

После загрузки пользователь получает рабочий стол с элементами управления – рис. 4.

Рисунок 4 – Рабочий стол ubuntu

На рисунке цифрами обозначены:

1 – меню программ (аналог меню «Пуск» и панели быстрого запуска);

2 – папка документов (аналог папки «Мои документы»);

3 – ярлык утилиты установки ПО (аналог «Установка и удаление программ»);

4 – ярлык сменного носителя USB;

5 – кнопка сворачивания всех окон;

6 – меню рабочих поверхностей и корзина;

7 – строка статуса, часы и кнопка выхода из системы.

Пункт 5 следует рассмотреть особо. Дело в том, что при работе с ОС Linux пользователю предоставляется большое виртуальное рабочее пространство, которое разбивается на 4 части. На мониторе пользователь в один момент времени видит одну из частей. Навигации между частями осуществляется соответствующим меню. Данная возможность позволяет не захламлять экран множеством открытых окон и эффективно использовать пространство рабочего стола.

Рассмотрим базовый набор ПО в графическом интерфейсе системы. Через меню программ можно запускать типовой набор утилит: калькулятор, таблицу символов, словарь, редактор меню, терминал и т.п. (рис. 4).

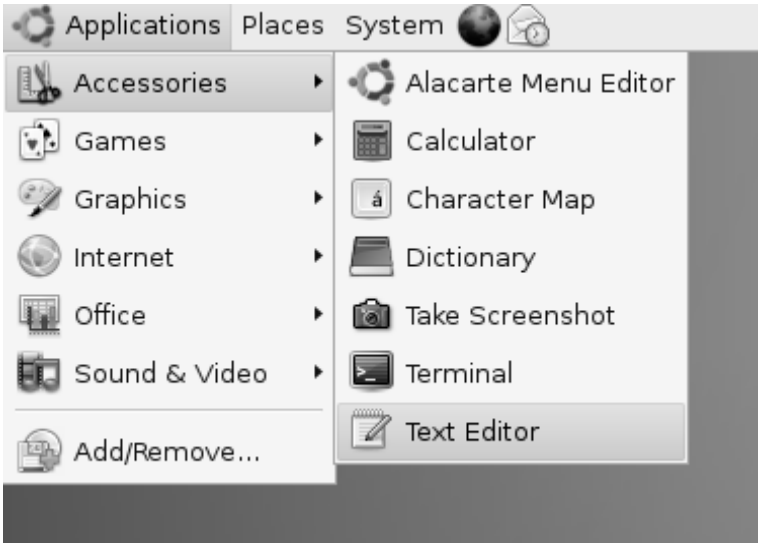

Рисунок 4 – Пункт „Принадлежности” меню программ

Терминал представляет собой возможность работать с системой через интерфейс командной строки: рис. 5.

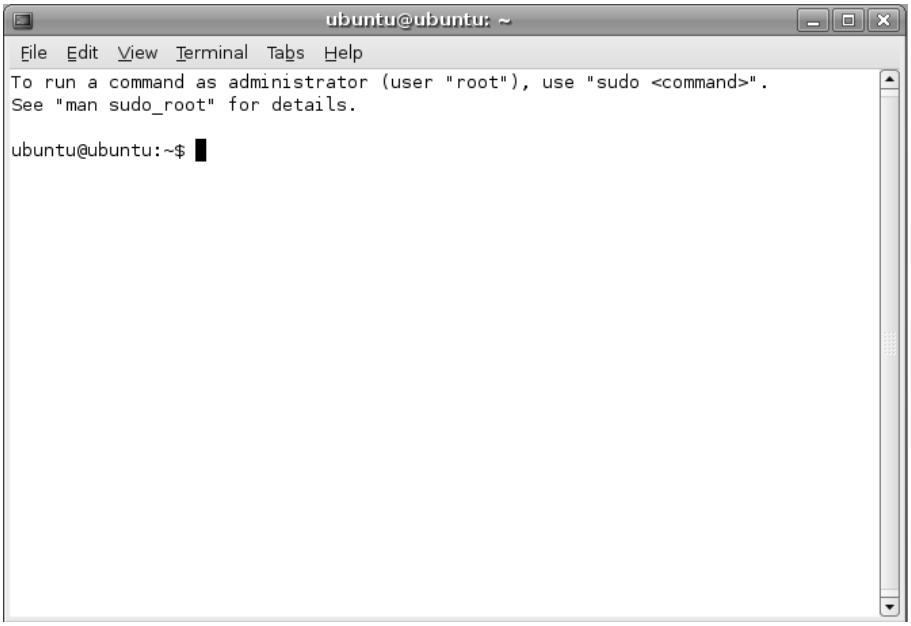

Рисунок 5 – Окно терминала

Пункт «Add/Remove» заменяет раздел «Установка и удаление программ» в ОС Windows. Его вид представлен на рис 6.

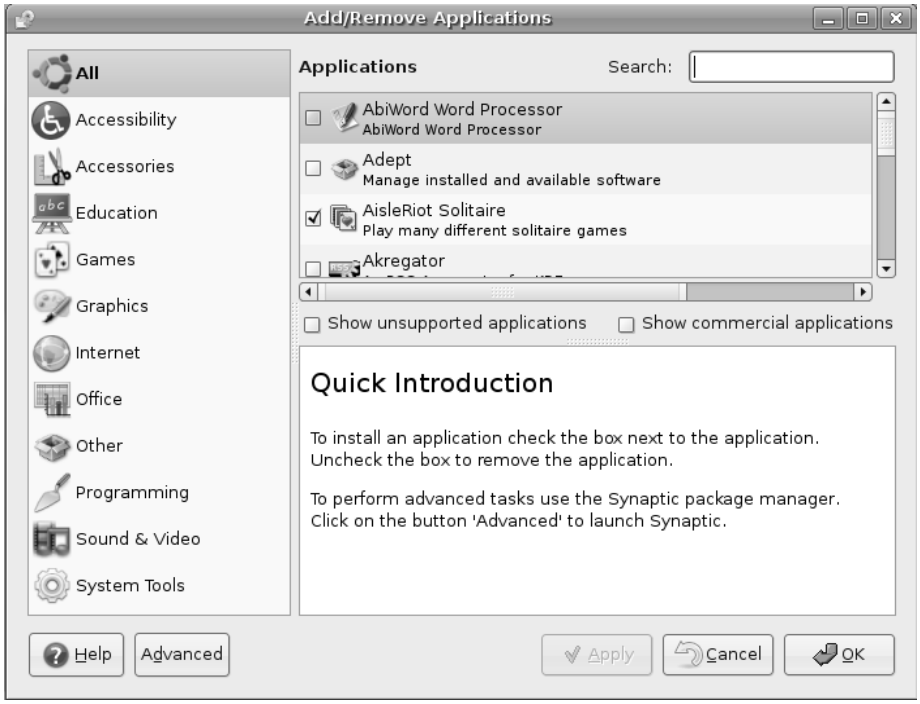

Рисунок 6 – „Установка и удаление программ”

Пункт меню „Internet” позволяет работать с браузером, почтой, менеджером сообщений и т.п. На рисунке 7 представлен вид браузера с открытой стартовой страницей, а на рис. 8 – интерфейс почтового клиента.

Рисунок 7 – Браузер в графической среде Linux

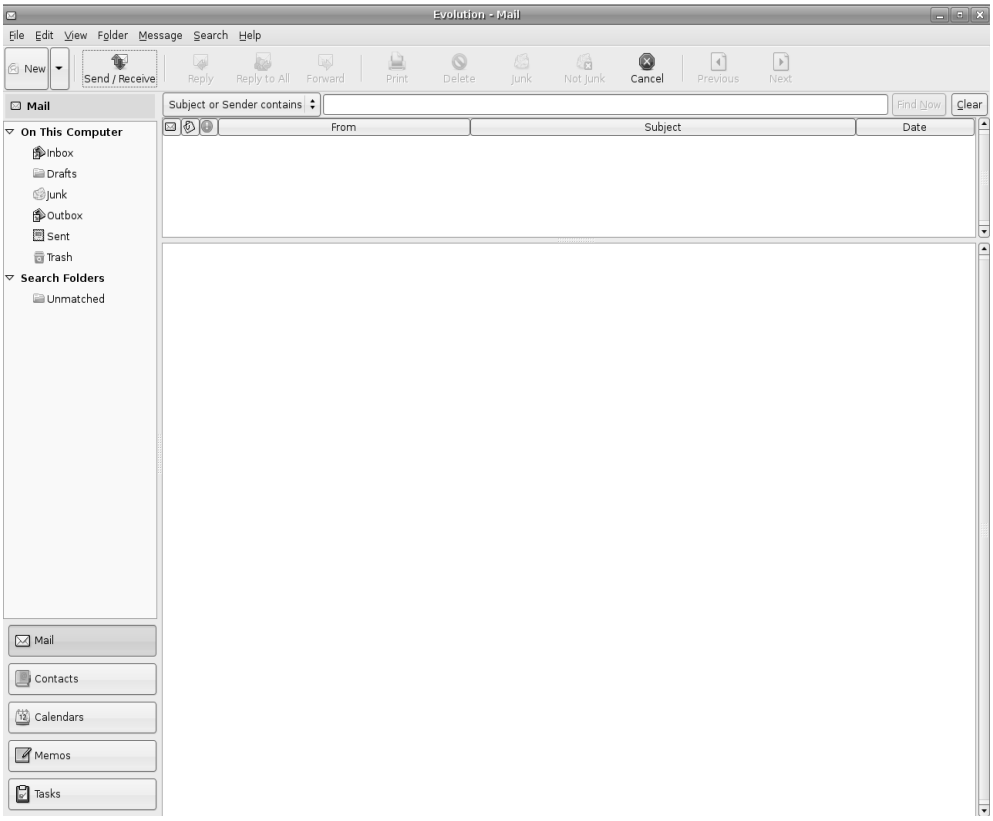

Рисунок 8 – Интерфейс почтового клиента в графической среде Linux

Также в графической среде Linux функционирует полноценный текстовый редактор из пакета OpenOffice.

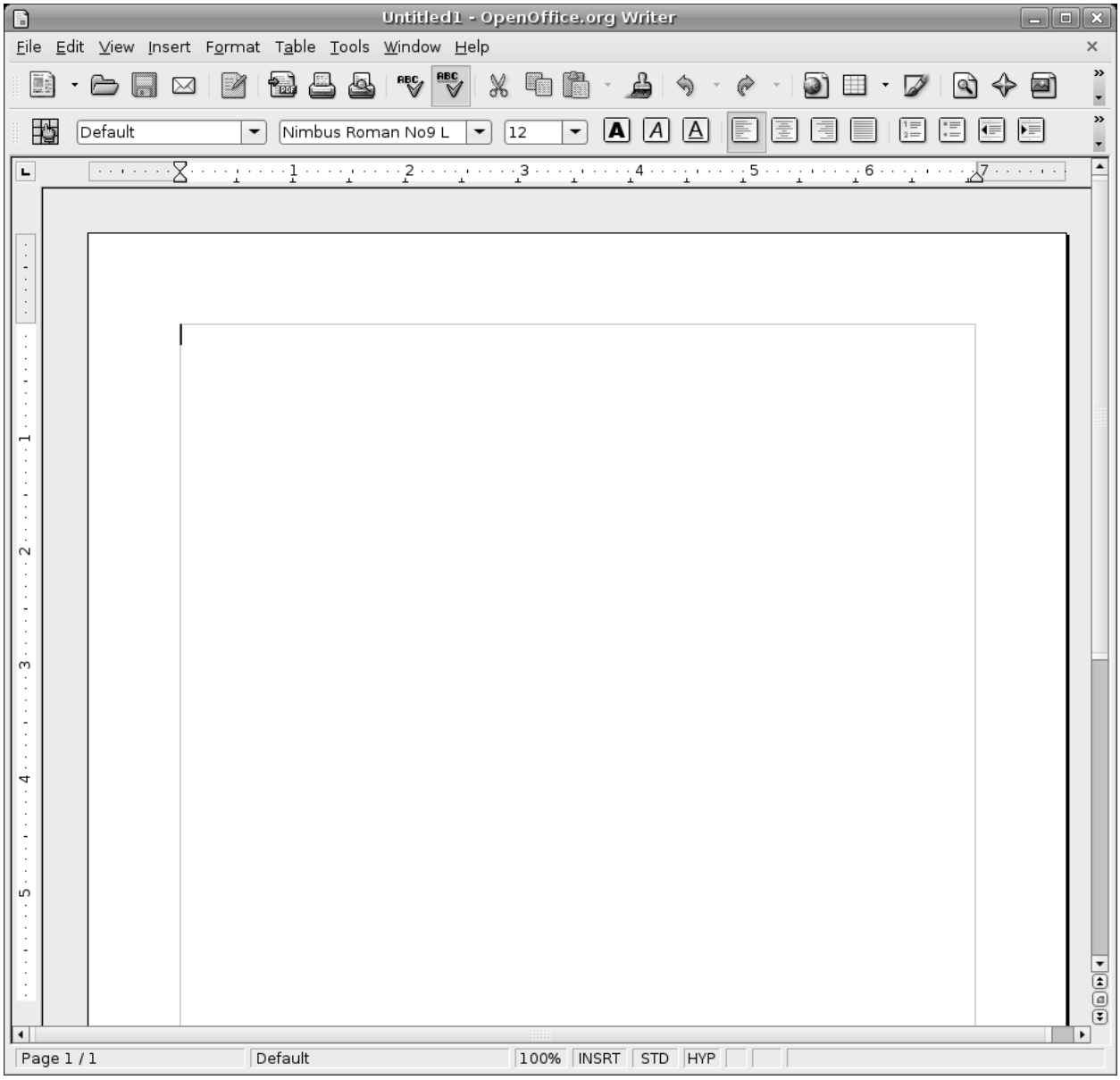

Рисунок 9 – Текстовый редактор OpenOffice

В разделе «Places» меню программ расположены ярлыки, позволяющие переходить в различные места системы: носители, рабочий стол, домашний каталог пользователя, сервер в сети и т.д. (рис. 10).

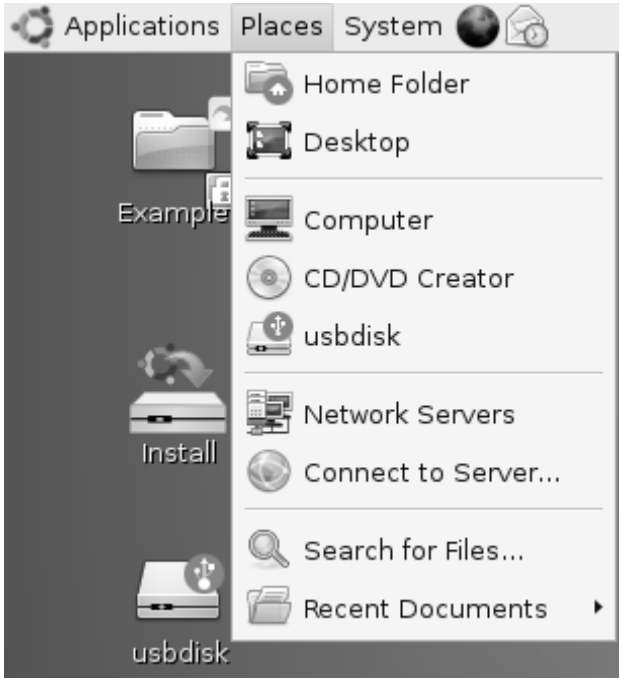

Рисунок 10 – Ярлыки перехода в системе

Раздел «System» содержит развитые средства управления и администрирования, как можно видеть из рис. 11-12.

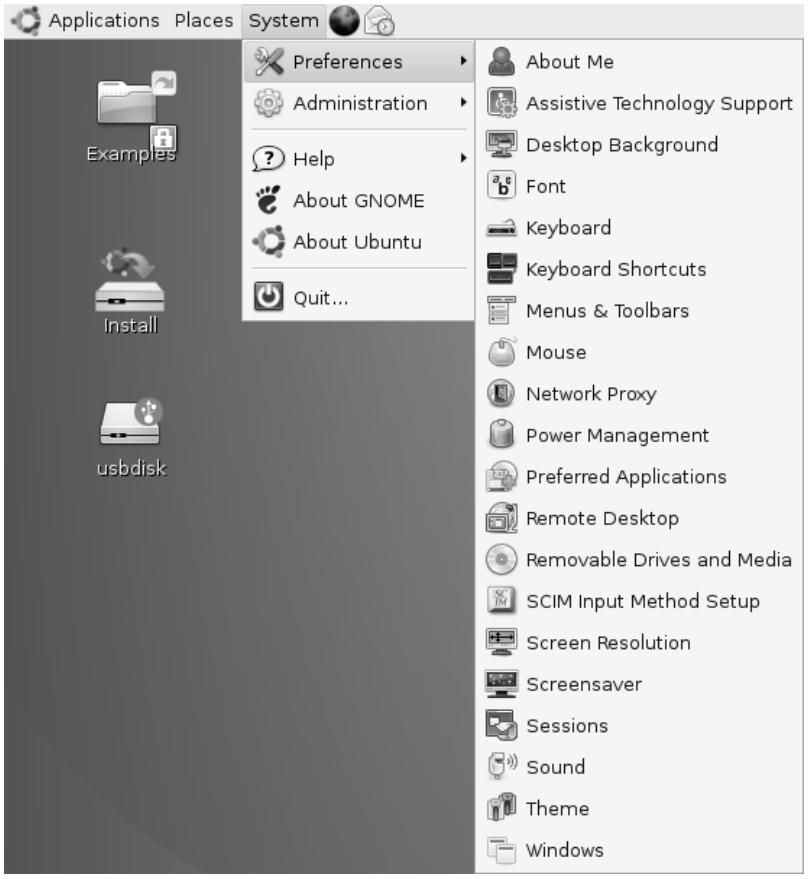

Рисунок 11 – Пункт «Настройки» раздела «System»

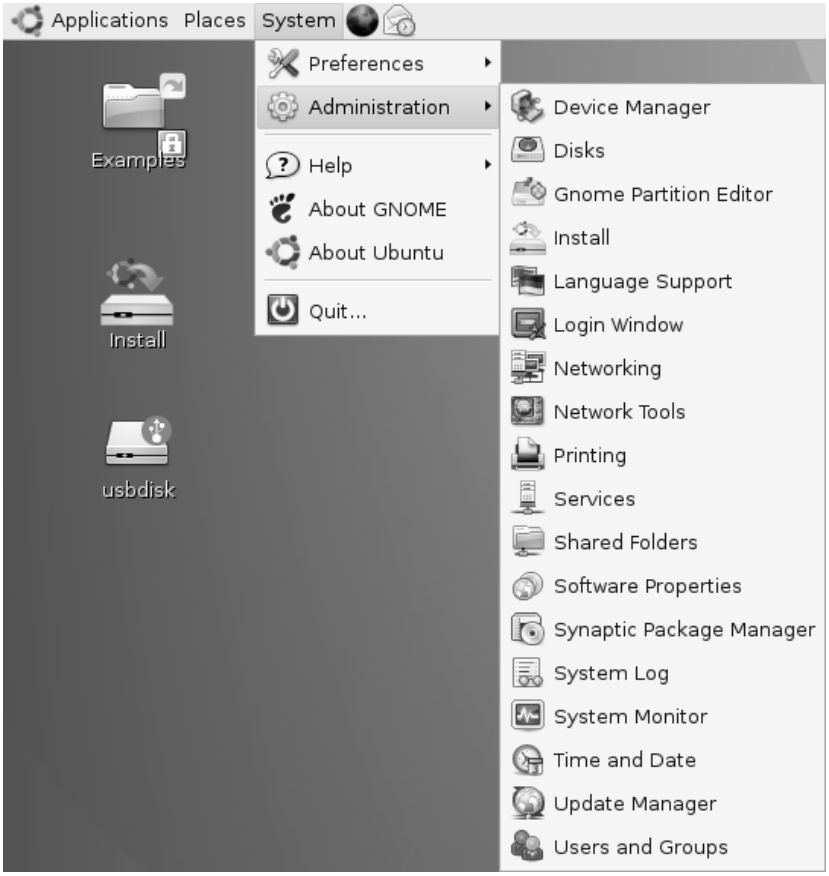

Рисунок 12 – Пункт «Администрирование» раздела «System»

На рисунке 13 показан броузер файлов, аналог проводника Windows. Вход в данный браузер возможен или через меню программ или через ярлык „Examples” на рабочем столе системы.

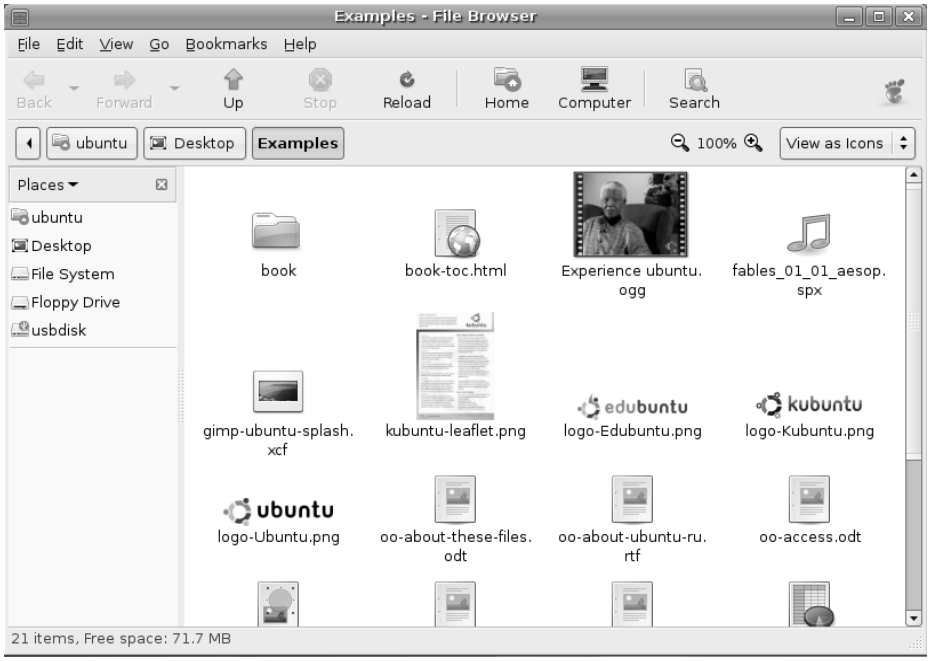

Рисунок 13 – Браузер файлов

Отдельно следует остановиться на работе с накопителями в системе Linux. Любой накопитель (раздел жесткого диска, CD/DVD-привод, дисковод и т.п.) должен быть примонтирован к файловой системе ОС.

Для этого можно использовать менеджер дисков, который вызывается через меню программ – «System/Administration/DisksManager» (рис. 14).

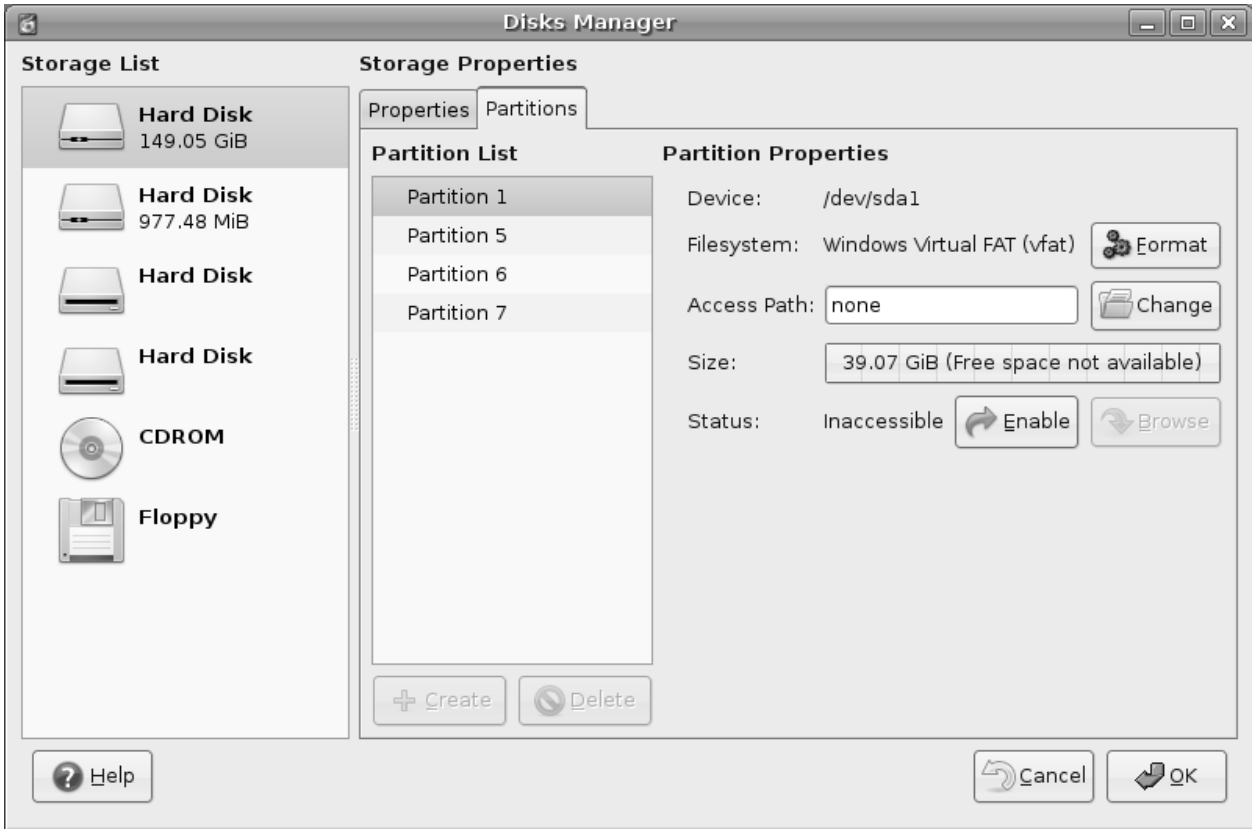

Рисунок 14 – Менеджер дисков

На рис. 14 можно видеть перечень дисков в системе, информацию о разделах и другую служебную информацию. Монтирование заключается в связывании устройства с некоторым каталогом в ФС системы (аналог команды mount интерфейса командной строки).

Для примера укажем в поле Access Path путь /root/disk и нажмем клавишу „Enable”. В случае удачного монтирования мы увидим следующий результат: рис. 15.

Рисунок 15 – Корневой каталог смонтированного диска в браузере файлов

После завершения работы с диском его нужно отмонтировать клавишей «Disable» в менеджере дисков.

Для выхода из системы необходимо щелкнуть мышью на соответствующей кнопке в правом верхнем углу экрана. В результате появится меню (рис. 16).

При этом станут доступны следующие пункты:

- завершение работы с данной учетной записью;
- блокировка экрана;
- переключение пользователей;
- перезапуск;
- выключение компьютера.

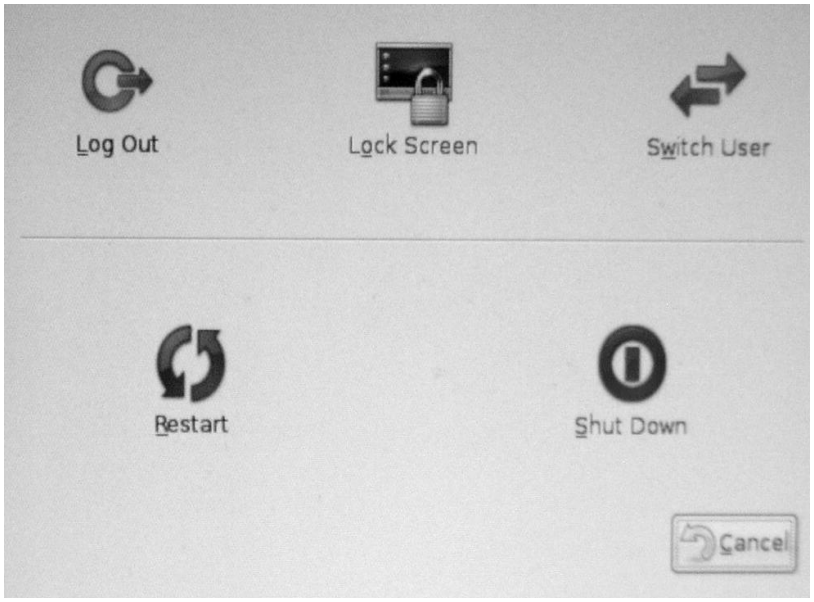

Рисунок 16 – Меню выхода из системы

Следует отметить, что при работе в графической среде остаются доступными текстовые терминалы. Перейти в текстовый терминал можно нажатием комбинаций Ctrl+Alt+F1…Ctrl+Alt+F6. Нажатие Ctrl+Alt+F7 возвращает нас в графическую среду.

Любой другой дистрибутив Linux, запускаемый со сменного носителя будет в чем то похож на рассмотренный ubuntu. Поэтому, навыки, приобретенные в работе с одним дистрибутивом, легко можно использовать и на других варианта системы.

Задание к лабораторной работе

В процессе лабораторной работы студенты должны изучить графический интерфейс операционной системы Linux на примере одного из дистрибутивов.

По требованию преподавателя студенты должны демонстрировать умение:

- загрузить компактный дистрибутив Linux;
- работать с элементами рабочего стола;

- оперировать файлами (копировать, удалять, создавать каталоги) в графической среде;

- выполнять запуск прикладного ПО;

- работать с графическим редактором;

- работать в текстовом редакторе;

- настраивать систему через меню администрирования;

- монтировать и размонтировать устройства хранения информации;

- работать в Интернет, настраивать броузер и почтовый клиент;

- переключаться в режим командной строки и выполнять в нем команды ОС; и т.д.

- настройка сети;

- настройка параметров экрана.

Контрольные вопросы:

1. 1. Как выполнять запуск прикладного ПО?

2. Какие средства есть для создания текстовых файлов? Достоинства и недостатки.

3. Какие возможности предоставляет меню «администрирование»?

4. Каким образом можно выполнить монтирование и размонтирование устройств информации?

ЛИТЕРАТУРА

1. Б.В. Керниган, Р.Пайк «Универсальная среда программирования Unix», М. - Финансы и статистика, 1992 г., 304 с.
2. А.Робачевский «Операционная система Unix», БХВ-Санкт-Петербург, 1999 г, 514 с.
3. Бах М.Д., «Архитектура операционной системы Unix»
4. Кейслер С., «Проектирование операционных систем для малых ЭВМ.» М.: Мир, 1986
5. Дейтел Х. М., «Введение в операционные системы». В 2-х томах – М.: Мир, 1987

6. Таненбаум Э., «Современные операционные системы» , - «Питер», 2002 г.

7. Кью Питер., "Использование UNIX. Специальное издание", - "Вильямс", 1999 г., 624 стр.

8. Пик Дж., О'Райли Т., Лукидис М. UNIX. Инструментальные средства. Киев: BHV, 1999 г., 944 стр.,

Дополнительная:

9. Страницы руководства ОС Linux.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Операционные системы»

Для студентов направления подготовки

09.03.04 "ПРОГРАММНАЯ ИНЖЕНЕРИЯ"

Составители: Алла Викторовна Чернышова