Занятие 3. Пользовательские средства ОС. Часть вторая.

3.1. Работа с файлами. Архивирование.

Во многих случаях для экономии дискового пространства и ускорения передачи по сети в Линукс используется архивирование файлов. Как правило, в любой дистрибутив Линукс входит несколько различных архиваторов, однако наибольшей популярностью пользуются следующие:tar, gzip, bzip2.
Программа tar (GNU tape archiver) позволяет создавать архивы из файлов и каталогов. Команда имеет формат tar options source_files. Исходными данными для архива могут являться как файлы, так и каталоги. В последнем случае архиватор сохраняет структуру дерева каталогов и восстанавливает ее при разворачивании архива.
Наиболее часто употребляются следующие ключи запуска:
-c – создать архив
-r – добавить файлы в архив
-A – добавить содержимое tar-файлов в архив
--delete – удалить файлы из архива
-t – вывести список файлов в архиве
-x – извлечь файлы из архива
-f filename.tar – указывает файл архива, из которого надо извлечь или в который надо записать информацию
-v – расширенный вывод информации о выполняемых действиях
Например, следующая команда создает архив из каталога:

```
[user@localhost ~]$ tar -cvf swift-1.4.5.tar swift-1.4.5/
```

Для сжатия файлов можно использовать утилиты gzip и bzip2. Команды имеют следующий формат:

```
$ gzip [options] filename
$ bzip2 [options] filename
```

Ключи этих двух команд во многом сходны. Наиболее часто используются:
-c – выводит результат работы программы в стандартный вывод stdout
-d – разворачивает сжатый ранее файл. В качестве альтернативы этой опции можно использовать программы gunzip, bunzip2
-t – проверяет целостность указанного сжатого файла
-v – выводит расширенную информацию о выполняемых действиях, в том числе указывает степень сжатия файлов
Следует отметить, что gzip может оперировать только с обычными файлами. Символические ссылки игнорируются.
Пример:

```
[user@localhost ~]$ bzip2 file
[user@localhost ~]$ ls
file.bz2
```

Создавать и распаковывать zip-архивы позволяют команды zip и unzip соответственно. Они могут быть не установлены в системе, тогда нужно поставить одноимённые пакеты.

```
sudo yum install zip unzip

sudo apt-get install zip unzip
```

Команда sudo — позволяет выполнять команду с правами другого пользователя, по умолчанию root, для её использования нужно знать пароль root пользователя.

Задание:
Создать в домашней директории директорию mirantis/test/task1. В ней создать текстовый файл hello с содержимым «Hello, world!» и пустой файл empty. Заархивировать папку mirantis в mirantis.tar.gz. Сравнить размер исходной директории и архива. Вывести список файлов в архиве. Затем удалите исходную директорию. Распакуйте архив. Убедитесь, что появилась папка аналогичная исходной. Снова удалите папку mirantis. Разверните архив до mirantis.tar, затем распакуйте mirantis.tar. Снова убедитесь, что всё успешно.
Примечание к заданию: познакомьтесь с командой du.

3.2. Процессы и уровни инициализации

3.2.1. Процессы в Linux. Демоны.

Процесс в Линукс – это совокупность программного кода и данных, загруженных в память компьютера. Процесс нельзя ассоциировать с запущенным приложением или командой, поскольку одно приложение может создавать несколько процессов одновременно. Код процесса не обязательно выполняется в данный момент времени, поскольку процесс может находиться в спящем состоянии (sleep). Существует три состояния, в которых может находиться процесс:

•работающий процесс – в данный момент код этого процесса выполняется

•спящий процесс – код процесса не выполняется в ожидании какого-либо события, например, нажатия клавиши или поступления данных из сети или с диска
•процесс-зомби – сам процесс уже не существует, его код и данные выгружены из оперативной памяти, но запись в таблице процессов по каким-то причинам сохраняется
Каждому процессу в системе назначается числовой идентификатор в диапазоне от 1 до 65535 (PID - Process Identifier), а также идентификатор родительского процесса (PPID - Parent Process Identifier). Идентификатор процесса позволяет адресовать процесс в операционной системе при использовании различных средств управления процессами. PPID определяет родственные отношения между процессами, которые определяют многие их свойства и возможности.
Некоторые процессы в системе привязаны к управляющему терминалу (tty или pty). Процессы, не связанные с терминалом, называются демонами (daemon). Такие процессы, будучи запущены пользователем, не завершают свою работу по окончании сеанса, а продолжают работу, поскольку не могут быть завершены автоматически. Как правило, с помощью демонов реализуются серверные службы, например, sshd или httpd.
Процессы создают древовидную иерархию. Корневой процесс дерева в большинстве дистрибутивов Линукс - это процесс init (PID=1). В настоящее время на смену процедуре загрузки с использованием init приходит более современный событийно-ориентированный загрузчиксystemd. Тем не менее, система управления процессами остается неизменной.
Корневой процесс не является частью ядра, но выполняет важную роль: определяет текущий уровень инициализации системы, отслеживает работу важнейших программ в системе. Некоторые процессы являются частью системы, например, менеджер памяти, планировщик времени процессора, менеджеры внешних устройств и так далее. Прочие процессы являются пользовательскими, запущенными либо из командной строки, либо во время инициализации системы.
Жизненный цикл процесса состоит из следующих фаз:

•Создание процесса – на этом этапе создается полная копия родительского процесса. Например, при запуске программы ls из командного интерпретатора он создает свою полную копию.
•Загрузка кода процесса и подготовка к запуску – копия, созданная на первом этапе, заменяется кодом задачи, которую необходимо выполнить, и создается ее окружение: устанавливаются необходимые переменные и так далее.
•Выполнение процесса – выполнение кода процесса на основе информации из области данных.
•Состояние зомби – на этом этапе выполнение процесса закончилось, его код выгружается из памяти, окружение уничтожается, но запись в таблице (дереве) процессов остается.
•Умирание процесса – после всех завершающих стадий удаляется запись из таблицы процессов и считается, что процесс завершил свою работу.

3.2.2. Отображение информации о процессах

Для отображения информации о процессах в системе используется команда ps. Формат команды:
ps PID options.
Без параметров команда отображает все процессы, которые были запущены в течение текущей сессии, за исключением демонов.
В большинстве Линукс-систем команда ps принимает очень большое число различных опций, которые влияют на формат отображения данных о процессах. Все ключи делятся на 3 основных группы:

•UNIX-тип – указываются с дефисом (ps -ef)
•BSD-тип – указываются без дефиса (ps ax)
•GNU-тип – указываются с двумя дефисами (ps --pid 1

Опции разных типов можно использовать в одном вызове, но при этом между ними могут возникать конфликты.
Наиболее часто используемые ключи:
-A, -e – показать все процессы
-f – полноформатный вывод
-w – показать полные строки описания процессов. Если длина строки превосходит ширину экрана, производится перенос на следующую строку. Ключ -ww снимает все ограничения на длину строки описания, допуская перенос на 2 и более строк.
Наиболее часто используемые комбинации ключей:
aux – показывает все процессы в системе в расширенном формате (BSD-ключи)
-elf – показывает все процессы в системе в расширенном формате (UNIX-ключи)
-ejH, axjf – отображает дерево процессов

Просмотреть информацию о процессах позволяет также программа top. Эта программа позволяет узнать не только PID процесса и его описание, но также показывает параметры производительности процесса и всей системы.

3.2.3. Идентификаторы процессов. Реальный и эффективный идентификаторы. Биты SUID и SGID.

Процессы в Линукс обладают теми же правами, которыми обладает пользователь, от имени которого запущен процесс. Для определения пользователя, запустившего процесс, операционная система использует реальные

идентификаторы пользователя и группы, назначаемые процессу. Однако эти идентификаторы не являются решающими при определении прав доступа. Для этого у каждого процесса существуют эффективныен идентификаторы процесса, которые могут отличаться от реальных. Такая возможность требуется для запуска программ, которые должны использовать права другого пользователя. Например, команда passwd должна использовать эффективный идентификатор суперпользователя, поскольку только он имеет права на запись в файлы паролей.
У каждого файла в дескрипторе, помимо прочих данных, хранятся биты SUID и SGID. Эти биты позволяют при запуске программы присвоить ей эффективные идентификаторы владельца и группы владельцев. Процесс будет выполняться с правами доступа другого пользователя – владельца файла с программой. Так как
файл passwd принадлежит пользователю root и у него установлен бит SUID, то при запуске процесс passwd будет обладать правами пользователя root.
Биты SUID и SGID устанавливаются командой chmod в следующих форматах соответственно:

```
[user@localhost ~]$ chmod u+s filename
[user@localhost ~]$ chmod g+s filename
```

Для установки в абсолютном режиме следует учитывать, что третьим битом в этой группе является бит прикрепления (sticky bit). Соответствующая цифра добавляется в начало группы. Например, следующая команда выполняет установку битов SUID и SGID, при этом sticky bit остается неустановленным (0):

```
[user@localhost ~]$ chmod 6755 filename
```

3.2.4. Управление процессами. Сигналы.

Посмотреть подробную информацию - man 7 signal.
Во время работы процесса, ядро контролирует его состояние, и в случае возникновения непредвиденной ситуации управляет процессом с помощью посылки ему сигнала. Процесс может воспользоваться действием по умолчанию, или, если у него есть обработчик сигнала, то он может перехватить или игнорировать сигнал. Существуют сигналы SIGKILL и SIGSTOP, которые невозможно ни перехватить, ни игнорировать.
Поддерживаются следующие варианты действий по умолчанию:

- Term – завершить выполнение процесса
- Ign – игнорировать сигнал
- Core – завершить выполнение процесса и создать файл core, содержащий образ памяти процесса, в текущем каталоге
- Stop – приостановить выполнение процесса
- Cont – возобновить выполнение процесса, если он находится в приостановленном состоянии

Впервые сигнальная система была описана стандартом POSIX.1-1990. Номера некоторых сигналов зависят от аппаратной архитектуры, на которой работает операционная система.

```
Signal     Value     Action   Comment
------------------------------------------------------------------
SIGHUP       1       Term    Hangup detected on controlling terminal or death of
controlling process
SIGINT       2       Term    Interrupt from keyboard
SIGQUIT      3       Core    Quit from keyboard
SIGILL       4       Core    Illegal Instruction
SIGABRT      6       Core    Abort signal from abort(3)
SIGFPE       8       Core    Floating point exception
SIGKILL      9       Term    Kill signal
SIGSEGV      11      Core    Invalid memory reference
SIGPIPE      13      Term    Broken pipe: write to pipe with no readers
SIGALRM      14      Term    Timer signal from alarm(2)
SIGTERM      15      Term    Termination signal
SIGUSR1   30,10,16   Term    User-defined signal 1
SIGUSR2   31,12,17   Term    User-defined signal 2
SIGCHLD   20,17,18   Ign     Child stopped or terminated
SIGCONT   19,18,25   Cont    Continue if stopped
SIGSTOP   17,19,23   Stop    Stop process
SIGTSTP   18,20,24   Stop    Stop typed at tty
SIGTTIN   21,21,26   Stop    tty input for background process
SIGTTOU   22,22,27   Stop    tty output for background process
```

В ядре существует механизм распределения времени ЦПУ между процессами. Он называется планировщиком (scheduler). Планировщик распределяет процессорное время на основании параметра процессов, который называется приоритетом. Приоритет процесса можно узнать используя команду ps -l. Чем больше

значение этого параметра, тем меньше процессорного времени получит соответствующий процесс. Высоким приоритетом (0) обладают процессы критических системных задач. Процессам ядра присваивается отрциательный приоритет.
Пользователь также может управлять процессами и их приоритетом. Для этого используются команды nice, nohup, kill.
Команда nice -n N command позволяет установить приоритет (вернее, niceness) N, с которым будет запущен процесс команды command. По умолчанию процессами присваивается приоритет 10. Допустимы значения N в диапазоне от -20 (высший приоритет) до 19 (низший приоритет).
Команда nohup command позволяет запустить процесс, вызываемый командой, таким образом, что сигнал SIGHUP будет игнорироваться этим процессом. Это позволяет запускать длительно работающие процессы, которые смогут "пережить" отключение контрольного терминала.
Команда kill -N <PID> позволяет отправить сигнал с номером N процессу, идентифицируемому PID. Вместо числа N можно использовать символьное обозначение сигнала без приставки SIG-. Например:

```
[root@localhost ~]# kill -9 400
[root@localhost ~]# kill -TERM 400
[root@localhost ~]# kill -TERM `pidof skype`
```

3.3. Командные оболочки и программирование для них

3.3.1. Понятие командной оболочки. Обзор командных оболочек.

Командная оболочка - это программа, позволяющая пользователю вводить команды, которые запускают процессы в системе. В составе Линукс присутствует несколько командных оболочек, в зависимости от дистрибутива.

- Bourne Shell – sh – наиболее распространенная и старая оболочка для Unix-систем. Различные ее версии присутствуют и используются во всех Unix-системах.
- Bourne Again Shell – bash – первоначально расширенная версия sh, впоследствие выделилась в самостоятельный проект. В настоящее время является наиболее популярной оболочкой во всех дистрибутивах Линукс.
- csh – оболочка с системой команд, близкой к языку программирования C.
- tclsh – система команд, близкая к языку Tcl.
- zsh – альтернативное расширение sh, оболочка с наиболее широкими возможностями.

3.3.2. Командная оболочка bash. Особенности работы.

Командная оболочка bash первоначально являлась свободно распространяемым аналогом оболочки sh. Впоследствии стала самостоятельным программным продуктом. Основные особенности таковы:
1. Возможность редактирования введенной команды. sh поддерживает только удаление команды и ввод новой.
2. Перенаправление ввода-вывода, организация каналов между задачами.
3. Поддержка псевдонимов команд, истории команд, автодополнения.
4. Возможность создания фоновых заданий и управления ими.
5. Настраиваемость и индивидуализация.

В командной оболочке существуют так называемые встроенные команды. Эти команды не имеют соответствующих программ в системе, а выполняются самой командной оболочкой.
bash автоматически записывает введенные пользователем команды в файл ~/.bash_history. Для управления этим файлом служит встроенная в bash команда history. При запуске по умолчанию она отображает историю команд.
Флаг history -c очищает текущую историю команд. Флаг history -a добавляет в файл команды, введенные в текущей сессии bash. Ключ history -w перезаписывает файл истории данными текущей сессии.
Перемещение по истории команд возможно клавишами <up>, _<down>+.
Поиск по истории осуществляется сочетанием клавиш <Ctrl+r>.
Кроме истории, оболочка bash поддерживает автодополнение команд. Это означает, что при вводе первых символов команды можно использовать клавишу <Tab> для получения вариантов команды.
Псевдонимы команд - это внутренняя подстановка в командной оболочке вместо введенной команды какой-то другой команды, или той же команды с параметрами. Псевдонимы управляются командой alias. Без флагов и аргументов эта команда выводит перечень установленных в текущей сессии псевдонимов в формате, совпадающем с форматом команды создания псевдонима. Например:

```
[user@localhost ~]$ alias
alias cp='cp -i'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-
tilde'
```

Программная оболочка взаимодействует с системой с использованием переменных среды. Это стандартные переменные, которые позволяют определить параметры, предпочитаемые пользователем. Наиболее часто используются следующие переменные:
PATH – содержит пути, по которым системе следует искать исполняемые файлы, если в командной строке не набирается полное имя файла.
PWD – содержит имя текущей директории.
HOME – содержит полный путь домашнего каталога пользователя.
HOSTNAME – содержит имя компьютера.
LOGNAME – содержит имя пользователя, от имени которого открыт текущий сеанс.
SHELL – содержит имя командной оболочки, запущенной в текущем сеансе.
USER – содержит имя пользователя, от имени которого открыт текущий сеанс.
Каждый интерпретатор также содержит внутренние переменные. Сделать внутреннюю переменную переменной окружения можно командой export. Эта команда без аргументов выводит списке установленных в системе переменных.

Свои пользовательские настройки можно сохранять в файл ~/.bashrc (переменные окружения, алиасы).

3.3.3. Многозадачность в консоли. Задания. Управление заданиями.

Основная характеристика системы Линукс – многозадачность. Она поддерживается в том числе и в консоли. Во-первых, возможно запустить несколько командных оболочек в разных виртуальных консолях. Переключение между ними производится клавишами <Alt+F[1-6]>.
Многозадачностью можно воспользоваться и в рамках одной консоли. Для поддержки многозадачности в bash можно использовать следующие средства:

- <Ctrl+Z> – комбинация клавиш, отправляющая текущему процессу немаскриуемый сигнал SIGSTOP. Выполнение процесса приостанавливается, управление передается родительскому процессу, в данном случае, командной оболочке.
- command & – символ & позволяет запустить ее в фоновом режиме.

```
make &>file.log &
```

- jobs – выводит список текущих заданий командного интерпретатора.
- bg <#job> – переводит задание <#job> в фоновый режим. При этом задание должно находиться в остановленном состоянии. Номер задания можно не указывать, если оно единственное.
- fg <#job> – передает управление консолью заданию номер <#job>. Задание должно находиться в остановленном состоянии или в фоновом режиме.

3.4. Планирование заданий

3.4.1. Понятие планирования заданий

Механизмы планирования заданий используются в тех случаях, когда необходимо либо запустить программу в определенное время, когда администратор не сможет этого сделать, либо требуется выполнять программу с некоторой периодичностью в заранее известное время.
Методика планирования требует представления о процессах, происходящих с сервером в течение определенных периодов времени.

3.4.2. Команда at, демон cron, команда crontab.

Команды семейства at применяется для однократного запуска программы в указанное время. Для работы этой программы необходимо, чтобы в системе был запущен демон atd. Этот демон поддерживает очередь заданий. Формат указания времени, в которое нужно выполнить задание, является достаточно гибким. По умолчанию указание времени в формате ЧЧ:ММ устанавливает задание на текущий день (или на следующий, если сегодня этот час и минута уже прошли). Можно указать время в формате ЧЧ:ММ am/pm (граджанский формат США). Если задание нужно выполнить в указанное время через несколько минут/часов/дней/недель, можно указать этот преиод в формате now + <num> <units>, где num - число, units - minutes/hours/days/weeks. Если известна дата, когда нужно выполнить программу, можно указать ее в формате MMDDYY, или MM/DD/YY, или DD.MM.YY, или YYYY-MM-DD. Примеры:

- at 4pm + 3 days - выполнить задание в 16:00 через 3 дня.
- at 10:00 Jul 31 - выполнить задание в 10:00 31 июля будущего года.
- at 19:00 2011-12-23 - выполнить задание в 19:00 23 декабря 2011 года.

По умолчанию команда работает в интерактивном режиме. Например:

```
[user@localhost ~]$ at 19:00
at> /home/user/task.sh
at> <EOT>
job 1 at 2011-12-22 19:00
```

Для того, чтобы назначить выполнение команды или серии команд на определенное время, можно воспользоваться ключом -f:

```
[user@localhost ~]$ at -f /home/user/task.sh 19:00
job 1 at 2011-12-22 19:00
```

Просмотреть очередь заданий можно с помощью команды atq:

```
[user@localhost ~]$ atq
1       2011-12-22 19:00 a root
```

Удалить задание из очереди позволяет команда atrm <job>:

```
[user@localhost ~]$ atrm 1
```

Если задание требуется выполнять регулярно в указанное время, используется демон crond. Он поддерживает таблицы заданий для всех пользователей системы. Управление таблицами осуществляется командой crontab, либо путем прямого редактирования файлов, например, файла /etc/crontab, или, чаще, таблиц в каталоге /etc/cron.d/, /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly.
Формат таблицы crontab:
<minute> <hour> <day_of_month> <month> <day_of_week> <command>
Для просмотра таблицы используется команда crontab -l:

```
[user@localhost ~]$ crontab -l
46 2 * * * find /var/spool/bacula/backup.ok -ctime -2 2>/dev/null
0 1 * * Sun /usr/sbin/raid-check
```

Редактировать таблицу заданий по умолчанию позволяет ключ команды crontab -e. Он открывает файл с таблицей в текстовом редакторе, определенном в переменных окружения VISUAL или EDITOR.
Очистить список заданий можно с помощью команды crontab -r.