ЛАБОРАТОРНАЯ РАБОТА №3

Управление системой Linux при помощи командных файлов

3.1. Теоретические сведения

3.1.1. Файловая система

Файловой системой называется некоторая организация данных и метаданных на устройстве хранения. Все файлы в Linux физически состоят из двух частей, реально локализованных в различных блоках дискового накопителя, но обязательно находящихся в одном дисковом разделе, первичном или логическом. Первая часть файла - его так называемые метаданные, которые содержат файловый дескриптор (это просто некое уникальное число), сведения о его атрибутах (принадлежности, правах доступа, времени модификации и т.д.), а также информацию о том, в каких блоках дискового раздела (которые так и называются - блоки данных) физически размещено содержимое файла – его вторая часть.

Все файлы, доступные в Linux системах, составляют иерархическую файловую структуру, которая подобна растущему дереву имеет ветки (каталоги) и листья (файлы в каталогах). Корень этого большого дерева обозначается как /. Физически файлы могут располагаться на различных устройствах.

Команды mount и umount

Команда **mount** служит для подключения файловых систем разных устройств к этому большому дереву. Также существует противоположная ей команда под названием **umount**, которая выполняет демонтирование (отключение) файловых систем. Наиболее часто встречающаяся форма команды **mount** выглядит следующим образом:

mount -t vfstype device dir

Такая команда предлагает ядру смонтировать (подключить) файловую систему указанного типа *vfstype*, расположенную на устройстве *device*, к заданному каталогу *dir*, который часто называют точкой монтирования. Предыдущее содержимое, владелец и режим доступа к каталогу *dir* становятся недоступными (исчезают), а вновь появившиеся продолжают действовать, пока файловая система *device* смонтирована (подключена) к *dir*. Программы **mount** и **umount** поддерживают список текущих смонтированных файловых систем в файле */etc/mtab*. Запущенная без аргументов, **mount** выводит на экран этот список. Подробности работы команд **mount** и **umount**, а также список их параметров приведены в справочной системе Linux.

Команда df

df —показывает список всех файловых систем по именам устройств, сообщает их размер, занятое и свободное пространство и точки монтирования. Подробно работа команды **df** описана в справочной системе Linux.

Команда du

du выдает отчет об использовании дискового пространства заданными файлами, а также каждым каталогом иерархии подкаталогов каждого указанного каталога. Здесь под использованным дисковым пространством понимается пространство, используемое для всей иерархии подкаталогов указанного каталога. Запущенная без аргументов, команда **du** выдает отчет о дисковом пространстве для текущего каталога. Подробности о работе команды **du** смотрите в справочной системе Linux.

i-node

При создании файловой системы создаются также и структуры данных, содержащие информацию о файлах. Каждый файл имеет свой инод, идентифицируемый по номеру

инода (часто называемый 'i-номером' или 'инодом'), в файловой системе, в которой располагается сам файл. Иноды хранят информацию о файлах, такую как принадлежность владельцу (пользователю и группе), режим доступа (чтение, запись, запуск на выполнение) и тип файла. Существует определенное число инодов, которое указывает максимальное количество файлов, допускаемое определенной файловой системой. Обычно, при создании файловой системы примерно 1% ее выделяется под иноды. Номер инода файла можно посмотреть, используя команду **ls -i**, а команда **ls -l** покажет информацию, хранящуюся в иноде.

Жесткие и символьные ссылки

Жёсткой ссылкой в UNIX-подобных операционных системах называется имя файла, привязанное к уникальному индексному дескриптору файла. Таким образом, понятия «жёсткая ссылка на файл» и «имя файла» являются синонимами. Создать жёсткую ссылку в UNIX-подобных ОС можно при помощи команды **ln**, которая по умолчанию создаёт именно жёсткие ссылки. В метаинформации файла всегда хранится число жёстких ссылок на него (то есть количество его имён). Файл считается удалённым после удаления его последнего имени, однако место освобождается лишь когда его индексный дескриптор перестаёт использоваться.

Символьная ссылка (*Symbolic link*) — специальный файл в UNIX-подобных операционных системах, для которого в файловой системе не хранится никакой информации, кроме одной текстовой строки. Эта строка трактуется как путь к файлу, который должен быть открыт при попытке обратиться к данной ссылке. Символьная ссылка занимает ровно столько места на файловой системе, сколько требуется для записи её содержимого (нормальный файл занимает как минимум один блок раздела). Целью ссылки может быть любой объект — например, другая ссылка, или даже несуществующий файл (в последнем случае при попытке открыть его должно выдаваться сообщение об отсутствии файла). Ссылка, указывающая на несуществующий файл, называется *висячей*. Практически символьные ссылки используются для более удобной организации структуры файлов на компьютере, так как позволяют одному файлу или каталогу иметь несколько имён и свободны от некоторых ограничений, присущих жёстким ссылкам (последние действуют только в пределах одного раздела и не могут ссылаться на каталоги).

Команда ln -s файл1 файл2 создает символьную ссылку *файл2* на *файл1*. При этом *файл1* может и не существовать. Символическая ссылка с именем *файл2* всё равно будет создана.

Команда chmod

chmod — изменение прав доступа к файлам и папкам. Права записываются сразу для трёх типов пользователей: владельца файла, группы, в которую он входит и для прочих пользователей. **chmod** может быть записан в двух форматах: в числовом и в символьном. Пример символьной записи: *'rwxr-xr-x'*. **chmod +w {имяфайла}** добавит права записи для владельца файла**, chmod -w {имяфайла} –** выполняет обратное действие.

Примером числовой записи может служить *'755*', которая эквивалентна записанной выше строковой записи: каждое право имеет числовой код и может быть задано вручную:
400 — владелец имеет право на чтение;
200 — владелец имеет право на запись;
100 — владелец имеет право на выполнение;
40 — группа имеет право на чтение;
20 — группа имеет право на запись;
10 — группа имеет право на выполнение;
4 — остальные имеют право на чтение;
2 — остальные имеют право на запись;
1 — остальные имеют право на выполнение.

Суммировав эти коды можно получить символьную запись. Например, **chmod 444 {имяфайла}**: 400+40+4=444 — все имеют право только на чтение.

umask

umask — функция, изменяющая права доступа, которые присваиваются новым файлам и директориям по умолчанию. Права доступа файлов, созданных при конкретном значении **umask**, вычисляются при помощи следующих побитовых операций (**umask** обычно устанавливается в восьмеричной системе счисления): *побитовое И* между унарным дополнением аргумента (используя *побитовое НЕ*) и режимом полного доступа.
Режим полного доступа для файлов — *666*, для директорий — *777*. Фактически, **umask** указывает, какие биты следует сбросить в выставляемых правах на файл - каждый установленный бит **umask** запрещает выставление соответствующего бита прав. Исключением из этого запрета является бит исполняемости, который для каталогов выставляется особо. **umask 0** означает, что следует (можно) выставить все биты прав (*rwxrwxrwx*), **umask 777** запрещает выставление любых прав.

Примеры

Допустим, что значение **umask** равняется *174*, тогда каждый новый файл
будет иметь права доступа *602*, а каждая новая директория *603*:

666_8 *И НЕ*(174_8) = 602_8
и
777_8 *И НЕ*(174_8) = 603_8
$777_8 = (111\ 111\ 111)_2$
$174_8 = (001\ 111\ 100)_2$
НЕ$(001\ 111\ 100)_2 = (110\ 000\ 011)_2$
$(111\ 111\ 111)_2$ *И* $(110\ 000\ 011)_2 = (110\ 000\ 011)_2$
777_8 *НЕ* $(174)_8$ $(603)_8$

В bash:

```
$ umask 0174
$ mkdir директория
$ touch файл
$ ls -l
drw-----wx 2 dave dave 512 Sep 1 20:59 директория
-rw-----w- 1 dave dave 0 Sep 1 20:59 файл
```

3.1.2. Пользователи

Файл /etc/passwd

/etc/passwd — файл, содержащий в текстовом формате список пользовательских учётных записей. Является первым и основным источником информации о правах пользователя операционной системы. Существует в большинстве версий и вариантов UNIX-систем. Каждая строка файла описывает одного пользователя и содержит семь полей, разделённых двоеточиями:

1) регистрационное имя, или логин;
2) хеш пароля;
3) идентификатор пользователя;
4) идентификатор группы по умолчанию;
5) информационное поле GECOS;
6) начальный (он же домашний) каталог;
7) регистрационная оболочка, или shell.

Основным назначением **/etc/passwd** является сопоставление логина и идентификатора пользователя (*UID*). Изначально поле пароля содержало хеш пароля и использовалось для аутентификации. Однако в связи с ростом вычислительных мощностей процессоров появилась серьёзная угроза применения простого перебора для взлома пароля. Поэтому все

пароли были перенесены в специальные файлы, такие как */etc/shadow* в GNU/Linux или */etc/master.passwd* во FreeBSD. Эти файлы недоступны для чтения обычным пользователям. Поле *GECOS* хранит вспомогательную информацию о пользователе (номер телефона, адрес, полное имя и так далее). Оно не имеет чётко определённого синтаксиса.

Команда finger

finger [-bfilpqsw] [login1 [login2 ...]]

По умолчанию команда **finger** выводит в список для каждого пользователя системы UNIX на данный момент имя регистрации в систему, полное имя, имя терминала и статус записи (при отсутствии разрешения на запись перед терминальным именем указывается символ "*"), время простоя, время регистрации, нахождение места работы и телефонный номер (если они известны). (Время простоя отображается в минутах, если оно выведено одним целым числом, в часах и минутах, если в его отображении присутствует двоеточие (:), или в днях и часах, если в выводе присутствует символ *"d"*.)

Кроме того, существует более длинный формат вывода, и он используется командой **finger** в том случае, если задан список имен пользователей. (Допускаются наряду с первым и последним именами пользователей также и учетные имена.) Этот формат состоит из нескольких строк; он включает всю информацию, описанную выше, и, дополнительно, пользовательские входной каталог и интерпретатор shell регистрации, любой план, который пользователь разместил в файле *.plan* в своем входном каталоге, и проект, в соответствии с которым заданные пользователи работают в данный момент, взятый из файла *.project*, который также находится во входном каталоге. Если в домашней директории указанного пользователя находится файл *.nofinger*, то по команде **finger** информация об этом пользователе не возвращается. Подробности о работе команды **finger** смотрите в справочной системе Linux.

Команда who

who [-u] [-T] [-l] [-H] [-q] [-p] [-d] [-b] [-r] [-t] [-a] [-s] [файл]

Команда **who** сообщает имя пользователя, имя терминальной линии, астрономическое время начала сеанса, продолжительность бездействия терминальной линии с момента последнего обмена, идентификатор процесса интерпретатора команд shell для каждого из пользователей, работающих в системе UNIX. Для получения этой информации команда просматривает файл */etc/utmp*. Впрочем, вместо него может просматриваться другой файл, имя которого должно быть тогда указано в командной строке (файл должен иметь формат *utmp*). Обычно в качестве файла указывают */etc/wtmp*, где зафиксированы времена начала всех сеансов с момента его последнего создания.

Команда **whoami** идентифицирует обратившегося с ней пользователя. Выдаваемые сообщения имеют, вообще говоря, следующий формат:

NAME [STATE] LINE TIME [IDLE] [PID] [COMMENT] [EXIT]

Информация *NAME, LINE* и *TIME* выдается при всех опциях, кроме **-q**; *STATE* - только при **-T;** *IDLE* и *PID* - только при **-u** и **-l**; и, наконец, *COMMENT* и *EXIT* - только при **-a**. Какая информация выдается в случае опций **-p, -d** и **-r**, подробно обясняется для каждой из них отдельно. Задавая различные опции, с помощью команды **who** можно получить информацию о времени начала и конца сеансов, перезагрузок, корректировках системных часов, а также о других процессах, порожденных процессом *init.* Подробности о работе команды **who** смотрите в справочной системе Linux.

Команда id

id - утилита, выводящая информацию об указанном пользователе или текущем пользователе, который запустил данную команду и не указал явно имя пользователя. По умолчанию выводятся подлинные числовые идентификаторы пользователя (*UID*) и группы

(*GID*), действующие (именные) идентификаторы пользователей и групп, а также идентификаторы других групп, в которых состоит пользователь. Подробно работа команды **id** описана в справочной системе Linux.

3.1.3. Процессы

Каждая выполняемая программа называется процессом. Эти процессы варьируются от вещей типа X Window System до системных программ (демонов), которые запускаются во время загрузки системы. Каждый процесс выполняется от имени какого-либо пользователя. Процессы, запускаемые во время загрузки, обычно выполняются от имени *root* или *nobody*. Процессы, запускаемые пользователями, работают с правами этих пользователей. Пользователь имеет полный контроль, над процессами, которые он запустил. Кроме того, *root* может управлять всеми процессами в системе, включая те, что были запущены другими пользователями. Процессами можно управлять и наблюдать за ними с помощью различных программ, а также с помощью некоторых команд shell.

Приоритетные и фоновые задачи

Программы, запускаемые из командной строки, начинают работать в приоритетном режиме (*foreground*). Это позволяет пользователю видеть данные, выводимые программой, и взаимодействовать с ней. Однако в некоторых случаях хотелось бы, чтобы программа работала, не захватывая полностью терминал. Это называется работой программы в фоновом режиме (*background*), и для осуществления этой задачи существует несколько способов.

Первым способом перевода процесса в фоновый режим является добавление в командную строку амперсанда (**&**) при запуске программы. Например, допустим, воспользуемся консольным mp3-плеером *amp* для воспроизведения mp3-файлов, однако в то же время нужно работать в том же самом терминале. Следующая команда запустит *amp* в фоновом режиме:

*% amp *.mp3 &*

Программа запустится и будет работать как обычно, а пользователь снова вернётся в командную строку. Другим способом перевода процесса в фоновый режим является осуществление этого во время работы процесса. Сначала запустим программу. Во время её работы нажмем **Control+z**. При этом процесс будет приостановлен. Иными словами, приостановленный процесс “ставится на паузу”. Он мгновенно прекращает свою работу, но может быть вновь запущен в любое время. После того, как пользователь приостановили процесс, он возвращаетесь в командную строку. Затем можно перевести процесс в фоновый режим, набрав **bg.** При этом приостановленный процесс вновь продолжит свою работу, но уже в фоновом режиме. Если необходимо взаимодействовать с фоновым процессом, то можно снова перевести его в приоритетный режим. Если выполняется только один фоновый процесс, можно переключиться на него, набрав **fg.** Если программа не завершила своё выполнение, она возьмёт под свой контроль терминал и будет невозможно получить доступ к приглашению командной строки. Иногда программа завершает своё выполнение, работая в фоновом режиме. В этом случае будет получено сообщение типа:

[1]+ Done /bin/ls $LS_OPTIONS

Это означает, что фоновый процесс (в данном случае *ls*) завершил свою работу. Одновременно могут выполняться несколько фоновых процессов. В этом необходимо выяснить, какой именно процесс нужно перевести назад в приоритетный режим. Просто набрав **fg**, будет возвращен назад процесс, который был последним переведён в фоновый режим. Для вывода перечня всех заданий существует команда **jobs**.

% jobs
[1] Stopped vim
[2]- Stopped amp
[3]+ Stopped man ps

По сути, команда выводит список всех процессов, переведённых в фоновый режим. В данный момент, все они остановлены (*stopped*). Это означает, что их работа этих процессов приостановлена. Числа слева – это идентификаторы, согласно которым сортируются все фоновые процессы. Идентификатором с плюсом отмечен процесс, который будет переведён в приоритетный режим, если будет набрано просто **fg**. Если нужно переключиться в *vim*, то необходимо набрать:

% fg 1

и *vim* возьмёт консоль под свой контроль. Перевод процессов в фоновый режиме весьма полезно, если у пользователя есть только один терминал через коммутируемое подключение.

Команда ps

Для того, чтобы получить перечень всех программ, выполняющихся в системе нужно воспользоваться командой **ps**. У этой команды есть много опций, поэтому мы рассмотрим только самые важные из них. Чтобы получить список программ, выполняемых в терминале нужно просто набрать **ps** без параметров. В этот список входят процессы, работающие в приоритетном режиме (любой командный процессор, в котором работает пользователь, и сама команда **ps**). Также перечисляются фоновые процессы, которые могут выполняться в данный момент. В большинстве случаев это будет очень короткий список:

% ps
PID TTY TIME CMD
7923 ttyp0 00:00:00 bash
8059 ttyp0 00:00:00 ps

PID - это идентификатор процесса (*process ID*). Все работающие процессы имеют уникальные идентификаторы в диапазоне от *1* до *32767*. Каждому новому процессу присваивается следующий свободный *PID*. Когда процесс завершает свою работу (или убивается, как вы увидите в следующем разделе), он отдаёт свой *PID*. При достижении в системе максимального *PID*, следующим берётся первый свободный *PID* с наименьшим номером, и так по кругу.

Колонка *TTY* обозначает терминал, в котором выполняется процесс. При простом вызове **ps** будет выведен список только тех программ, которые выполняются в текущем терминале. Поэтому все процессы будут иметь одну и ту же информацию в колонке *TTY*. Как видно в примере, оба процесса в списке выполняются в терминале *ttyp0*. Это означает, что они выполняются или удалённо, или в каком-нибудь X-терминале. Колонка *TIME* содержит данные о времени, в течение которого процесс использует ресурсы центрального процессора. Однако это не то время, в течение которого выполняется процесс. В колонке *CMD* представлена сама программа. В ней отображается только имя программы, безо всяких опций командной строки или подобной информации. Для получения боле полной информации нужно использовать одну из множества опций команды **ps**. Можно получить полный список процессов, выполняемых в вашей системе, используя правильный набор опций. Это приведёт к появлению большого списка процессов.

% ps –ax
PID TTY STAT TIME COMMAND
1 ? S 0:03 init [3]
2 ? SW 0:13 [kflushd]
3 ? SW 0:14 [kupdate]
4 ? SW 0:00 [kpiod]
5 ? SW 0:17 [kswapd]
11 ? S 0:00 /sbin/kerneld
30 ? SW 0:01 [cardmgr]
106 tty1 S 0:08 -bash
108 tty3 SW 0:00 [agetty]

109 tty4 SW 0:00 [agetty]
111 tty6 SW 0:00 [agetty]
[вывод сокращён]

Большинство этих процессов запускаются во время загрузки на большинстве систем. Уязвимость в ядре в функции *ptrace* стала причиной исправления, в результате которого для многих выполняемых процессов больше не показываются опции командной строки. В данном примере названия этих процессов заключены в квадратные скобки, например, у процессов с *PID* от *108* до *111*. В этом листинге также появилось несколько новых колонок. Во-первых, большинство процессов в списке работают в *tty "?"*. Они не привязаны ни к одному из терминалов. Это является самым распространённым случаем для демонов, т.е. процессов, которые выполняются без привязки к какому-либо терминалу. Во-вторых, здесь появился новый столбец: *STAT*. В нём отображается состояние процесса. *S* означает, что процесс спит (*sleeping*), т.е. ожидает какого-либо события. *Z* означает процесс-зомби. Такой процесс появляется в том случае, когда умирает его родительский процесс, оставив после себя дочерний процесс. Это нехорошая ситуация. *D* означает процесс, который перешёл в "непробудный" сон. Часто такие процессы невозможно убить даже путём отправки им сигнала *SIGKILL*. *W* означает *paging* (страничную подкачку файлов). Мёртвые процессы обозначаются как *X*. Процессы, отмеченные как *T*, трассируются или остановлены. *R* означает, что процесс можно запустить. Ещё более подробную информацию о выполняемых процессах можно получить, выполнив следующую команду:

% ps –aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME
COMMAND
root 1 0.0 0.0 344 80 ? S Mar02 0:03 init [3]
root 2 0.0 0.0 0 0 ? SW Mar02 0:13 [kflushd]
root 3 0.0 0.0 0 0 ? SW Mar02 0:14 [kupdate]
root 4 0.0 0.0 0 0 ? SW Mar02 0:00 [kpiod]
root 5 0.0 0.0 0 0 ? SW Mar02 0:17 [kswapd]
root 11 0.0 0.0 1044 44 ? S Mar02 0:00 /sbin/kerneld
root 30 0.0 0.0 1160 0 ? SW Mar02 0:01 [cardmgr]
bin 50 0.0 0.0 1076 120 ? S Mar02 0:00 /sbin/rpc.port
[вывод сокращён]

Это практически вся информация о системе. В ней присутствует дополнительная информация о процессе: какой пользователь его запустил, сколько он использует системных ресурсов (колонки *%CPU, %MEM, VSZ* и *RSS*) и когда был запущен. Следует отметить ещё один момент: данные теперь выходят за пределы экрана, и вы не можете увидеть их полностью. Опция **-w** заставит **ps** переносить длинные строки.

Команда kill

Для принудительного завершения программы можно использовать команду **kill.** Также её можно использовать для управления процессами разными способами. Чтобы убить процесс, нужно знать его *PID* или имя. Чтобы узнать идентификатор, можно воспользоваться командой **ps**, описанной в предыдущем разделе. Например, чтобы убить процесс *4747*, нужно выполнить следующее:

% kill 4747

Чтобы убить процесс, нужно быть его владельцем. Это особенность системы безопасности. *root* может убить любой процесс в системе.

Существует ещё одна разновидность утилиты **kill** под название **killall**. Эта программа убивает все работающие процессы с заданным именем. Если нужно убить все процессы *vim*, можно набрать следующую команду:

% killall vim

Эту команду с правами администратора убьет все процессы *vim*, запущенные пользователями системы. Иногда обычный **kill** не справляется с поставленной задачей.

Определённые процессы не будут завершаться. Если *PID 4747* не отвечает на запрос **kill**, можно выполнить следующее:

% kill -9 4747

То же самое вы можете использовать и с **killall**. В данном случае процессу просто отправляется другой сигнал. Обычный вызов **kill** отправляет процессу сигнал *SIGTERM (terminate)*, который сообщает ему, что нужно остановить свою работу, сбросить буферы и выгрузить себя из памяти. **kill -9** отправляет процессу сигнал *SIGKILL (kill),* который по сути просто убивает его. Процессу не разрешается "чисто" завершить свою работу, и иногда это приводит к нежелательным последствиям, таким как повреждение данных.

Одним из полезных вариантов использования **kill** является перезапуск процесса. Отправка большинству процессов сигнала *SIGHUP (-1)* заставит их повторно прочитать свои конфигурационные файлы. Это особенно полезно для сообщения системным процессам о том, что им нужно перечитать свои конфигурационные файлы после их редактирования.

Команда top

Команда **top** позволяет просматривать постоянно обновляющуюся информацию о процессах, выполняющихся в системе. На экран выводится информация о выполняющихся в системе процессах плюс некоторая дополнительная информация о системе: средняя загрузка, количество процессов, состояние процессора, информация о свободной памяти. Также предоставляется подробная информация о процессах, включая *PID*, пользователя, приоритет, использование процессора и памяти, время работы и название программы.

6:47pm up 1 day, 18:01, 1 user, load average: 0.02, 0.07, 0.02
61 processes: 59 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: 2.8% user, 3.1% system, 0.0% nice, 93.9% idle
Mem: 257992K av, 249672K used, 8320K free, 51628K shrd, 78248K
buff
Swap: 32764K av, 136K used, 32628K free, 82600K cached
PID USER PRI NI SIZE RSS SHARE STAT LIB %CPU %MEM TIME
COMMAND
112 root 12 0 19376 18M 2468 R 0 3.7 7.5 55:53 X
4947 david 15 0 2136 2136 1748 S 0 2.3 0.8 0:00 screenshot
3398 david 7 0 20544 20M 3000 S 0 1.5 7.9 0:14 gimp
4946 root 12 0 1040 1040 836 R 0 1.5 0.4 0:00 top
121 david 4 0 796 796 644 S 0 1.1 0.3 25:37 wmSMPmon
115 david 3 0 2180 2180 1452 S 0 0.3 0.8 1:35 wmaker
4948 david 16 0 776 776 648 S 0 0.3 0.3 0:00 xwd
1 root 1 0 176 176 148 S 0 0.1 0.0 0:13 init
189 david 1 0 6256 6156 4352 S 0 0.1 2.4 3:16 licq
4734 david 0 0 1164 1164 916 S 0 0.1 0.4 0:00 rxvt
2 root 0 0 0 0 0 SW 0 0.0 0.0 0:08 kflushd
3 root 0 0 0 0 0 SW 0 0.0 0.0 0:06 kupdate
4 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 kpiod

Она называется **top** потому, что программы, наиболее активно использующие процессор, будут находиться вверху. Интересное замечание: сам **top** будет первым в списке в большинстве систем с низкой активностью вследствие своего использования процессора. Если нужно выводить список только своих процессов или процессов какого-то другого пользователя, то нужные процессы могут отсутствовать среди тех, что активно используют процессор. Опция **-u** позволит вам указать имя пользователя или его *UID* и наблюдать только процессами, владельцем которых является этот *UID.*

% top -u alan
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+
COMMAND
3622 alan 13 0 11012 10m 6956 S 1.0 2.1 0:03.66 gnome-terminal

3739 alan 13 0 1012 1012 804 R 0.3 0.2 0:00.06 top
3544 alan 9 0 640 640 568 S 0.0 0.1 0:00.00 xinit
3548 alan 9 0 8324 8320 6044 S 0.0 1.6 0:00.30 gnome-session
3551 alan 9 0 7084 7084 1968 S 0.0 1.4 0:00.50 gconfd-2
3553 alan 9 0 2232 2232 380 S 0.0 0.4 0:00.05 esd
3555 alan 9 0 2552 2552 1948 S 0.0 0.5 0:00.10 bonobo-activati

Задания к лабораторной работе

Задание 1. Создайте скрипт, который предоставляет пользователю следующие функции:
1. Создает отчет о количестве доступных файловых систем, их свободном и занятом пространстве;
2. Выводит информацию об имени и размере файла отчета, а также о правах пользователя на него.

Задание 2. Создайте скрипт, который предоставляет пользователю следующие функции:
1. Создает отчет об именах файлов их инодах, правах, размерах и временах последнего доступа для каталога, указанного пользователем (отчет формируется в порядке возрастания инодов);
2. Открывает полные права на файл отчета для всех пользователей;
3. Создает жесткую ссылку на файл отчета в домашнем каталоге пользователя и символьную ссылку на рабочем столе.

Задание 3. Создайте скрипт, который предоставляет пользователю следующие возможности:
1. Выясняет, зарегистрирован ли в системе пользователь с заданным именем.
2. Выясняет, работает ли в текущий момент пользователь с заданным именем.
3. Выводит список зарегистрированных пользователей с именами их домашних каталогов и числовыми идентификаторами, в порядке убывания идентификаторов (для редактирования вывода используйте команды **grep** или **sed**).

Задание 4. Реализуйте два исполняемых файла, выполняющих следующие задачи:
1. Расчет значения функции sin(x) с шагом 0.01 от 0 до бесконечности.
2. Расчет значения функции cos(x) с шагом 0.001 от 0 до бесконечности.
3. Для реализации исполняемых файлов можно воспользоваться компиляторами **gcc** и **g++**.
4. Запустите первый файл в фоновом режиме.
5. Запустите второй файл в приоритетном режиме.
6. Переведите второй процесс в фоновый режим.
7. Переведите первый процесс в приоритетный режим.
8. Для каждого из процессов выведите:
- долю оперативной памяти, занимаемой процессом,
- время, прошедшее с момента запуска процесса,
- имя управляющего терминала,
- время старта процесса.