Федеральное государственное бюджетное образовательное учреждение

высшего образования

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(МГТУ им. Н. Э. Баумана)*** |

Факультет ***Информатика и системы управления***

Кафедра ***Компьютерные системы и сети (ИУ6)***

**Отчет**

**по лабораторной работе №4**

**Дисциплина: Операционные системы**

**Название лабораторной работы: исследование способов организации оперативной памяти и взаимодействия процессов**

Студент гр. ИУ6 - 54 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**А.А. Шумаков**\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**В.Ю.Мельников\_**\_**

(Подпись, дата) (И.О. Фамилия)

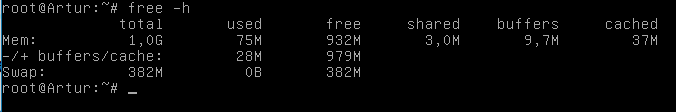
Москва, 2018

**Цель работы**: получение теоретических и практических сведений об управлении процессами, потоками и оперативной памятью в UNIX-подобных системах и в Linux в частности.

В памяти хранится ядро операционной системы, выполняемые программы и их данные. Значительную часть дефицитной оперативной памяти в памяти занимает кэш дисковых операций. Разберёмся, из чего он состоит и для чего он нужен.

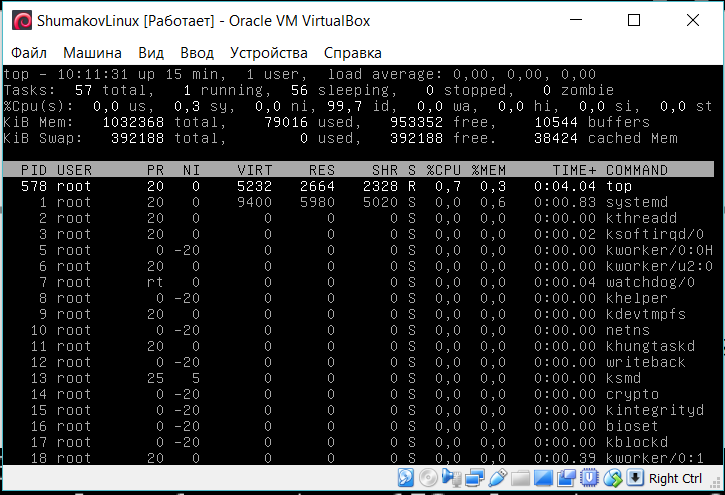
**Страничный кеш** — сюда попадает всё, что мы читали и записывали на диск. Когда программе снова потребуется содержимое файла, который только что записала другая программа, она его моментально получит. Прежде, чем читать файл с диска, ОС пытается найти его в кэше. Если какой-то файл недавно читали, второй раз он будет быстро прочитан из памяти. Этот вид кэша занимает больше всего места в памяти.

Посмотреть размер страничного кэша можно командой *free -h*

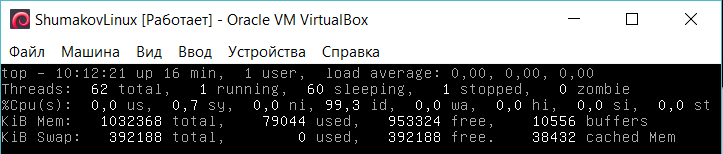


**Команды управления процессами в linux.**

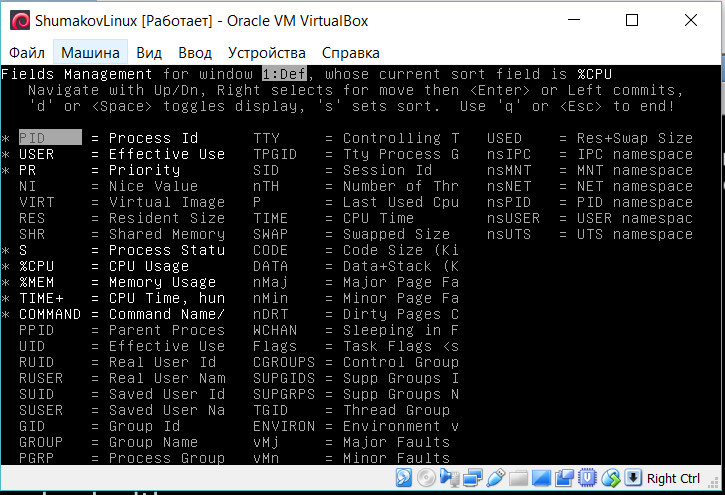
Начнём с утилиты *top*. Она способна отображать списки процессов и потоков, использование процессора и памяти, и причём в динамике, обновляя информацию раз в секунду, что удобно для выявления наиболее активных процессов. Дадим в терминале команду *top.*



Как видно, по умолчанию *top* отображает процессы (*tasks)*. Нажатием кнопки «H» можно перейти в режим отображения потоков (*Threads*).

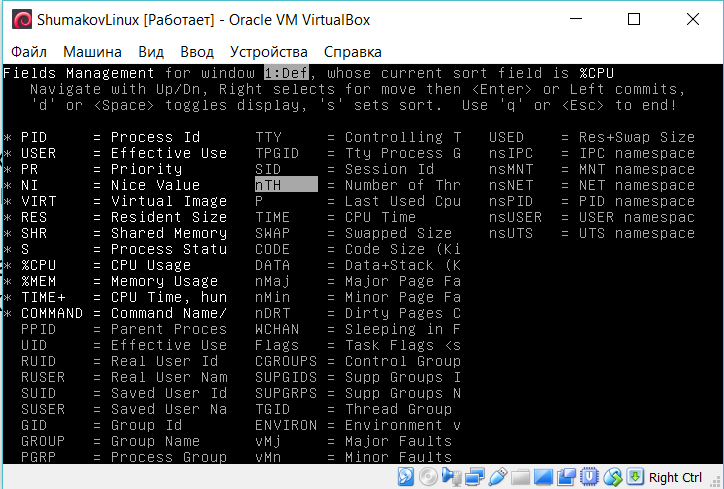


Можно изменить состав и порядок колонок. Для этого нужно нажать клавишу «F» во время работы утилиты.



Для того, чтобы отображать количество потоков и группу и ID группы потоков необходимо сделать следующее:

1. Клавишей со стрелкой «вниз» доведем курсор до записи *nTH*.



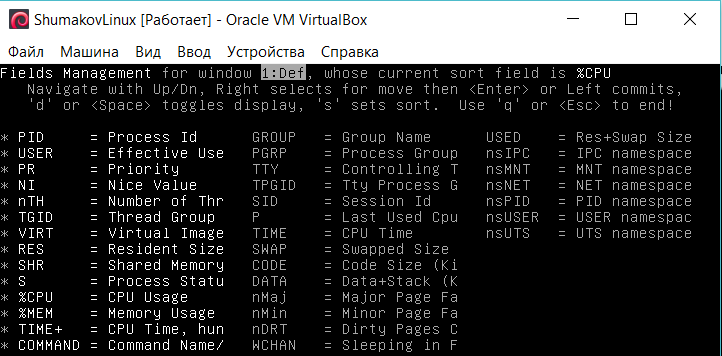
2. Нажмем клавишу со стрелкой «вправо».

3. Нажимая клавишу со стрелкой «вверх», перетащим запись вверх, расположив её после *NI*.

4. Нажмем Enter.

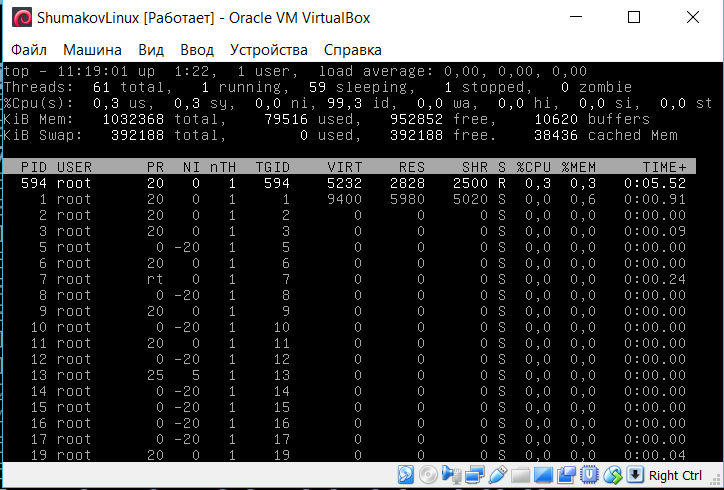
5. Нажмем пробел или клавишу «d». После выполнения этого шага *nTH* должно быть выделено жирным белым шрифтом, а сбоку должна появиться звёздочка.

6. Проделаем аналогичные операции для записи *TGID* (находится в самом конце списка). После выполнения этих шагов файл настроек должен выглядеть так, как показано на рисунке:



7. Закроем настройки (клавишей «q» или Esc).

8. Убедимся, что *top* работает в режиме отображения потоков (во второй строке должно быть написано *«Threads»*, если там написано *«Tasks»*, нажать клавишу «H»). На рисунке показан результат правильного выполнения данной инструкции.

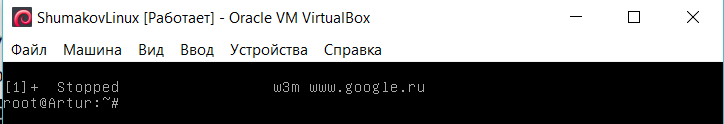


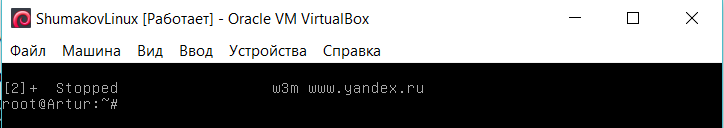
Для выхода из утилиты *top* нажмем клавишу «q».

**Утилиты «ps» и «kill»**

Команда *«top»* не всегда удобна. Для примера запустим пару процессов:

Дадим команду *«w3m www.google.ru»* и нажмем Ctrl+Z, чтобы перевести процесс в фоновый режим. Затем дадим команду «*w3m www.yandex.ru*» и нажмем Ctrl+Z, чтобы перевести процесс в фоновый режим.

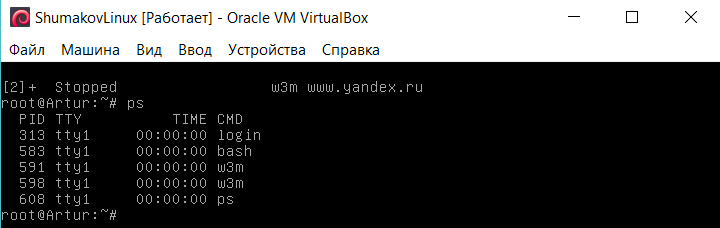




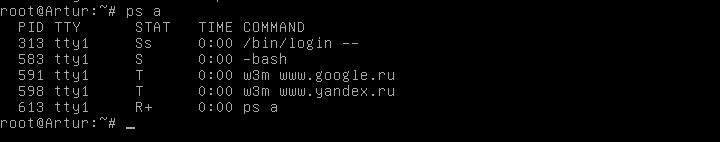
Предположим, мы хотим прервать процесс текстового браузера, в котором открыт «www.yandex.ru». Знакомая нам команда *killall* в этом случае бесполезна, поскольку у нас 2 процесса с одинаковым именем «w3m». В этом случае нам поможет команда «*kill*», но в этой команде надо указать идентификатор процесса. Идентификатор процесса выдаёт команда «*top*», но оба процесса приостановлены, значит окажутся где то в конце списка среди таких же «спящих процессов» и имя процесса у них одинаковое.

Воспользуемся командой «*ps*». Эта команда выводит информацию о процессах и завершается, не переходя в интерактивный режим. Из её вывода легко можно отфильтровать нужные строки. Эту команду часто используют для поиска процесса не только по имени, но и по параметрам команды, которой запущен процесс.

Команда *ps* без параметров выдаёт список процессов, запущенных с текущего терминала.

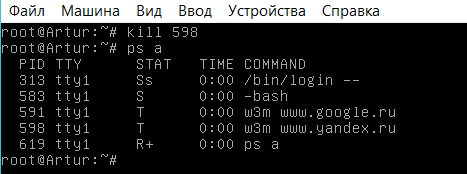


Для нашей задачи информации недостаточно. Выполним команду «*ps a*»



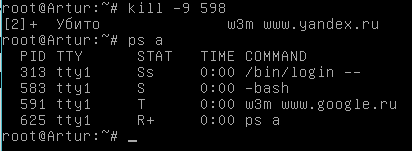
Теперь видно, что страница «www.yandex.ru» открыта в процессе *PID=598*

Попробуем его «убить» командой *«****kill*** *598»*, и снова дадим команду *«ps a»*

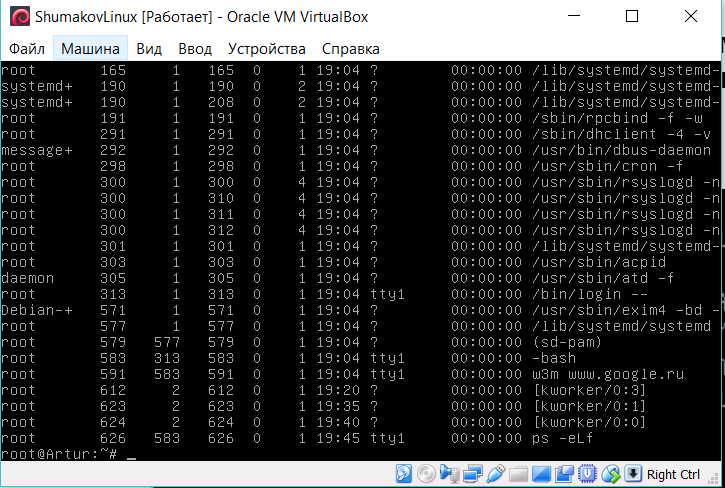


Процесс *«598»* всё ещё жив. Дело в том, что команда *kill* не убивает процесс, она посылает ему сигнал *«SIGTERM»*, но не все процессы на него реагируют. Не прореагирует на этот сигнал и зависший процесс. Если процесс не завершается, как в нашем случае, будем действовать жёстко. Дадим команду *«kill* ***-9*** *598»*. Сигнал *SIGKILL(9)* прерывает любой процесс кроме «Зомби» процессов. Но помните, дочерние процессы останутся в памяти, временные файлы не удалены, сетевые соединения не закрыты.

Вернёмся к выводу команды *«ps a»*



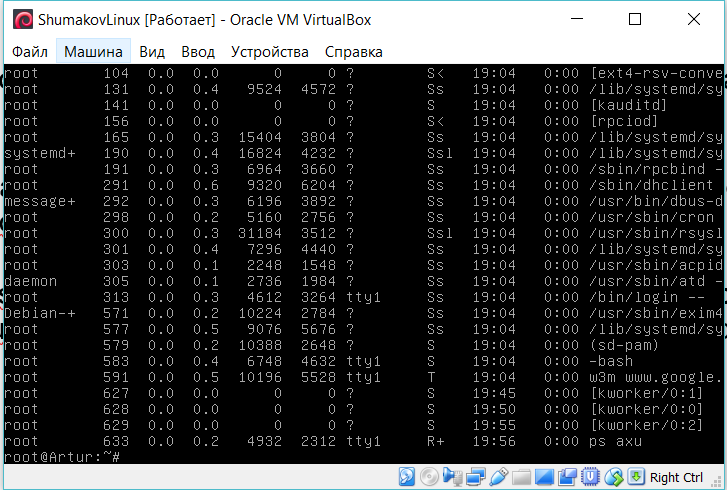
Команда *ps –eLf* также позволяет просмотреть и информацию о запущенных потоках. Как видно из рисунка, появились два новых поля: *LWP* – ID потока (Light Weight Process). *NLWP* – количество поток в процессе (Number of LWPs).



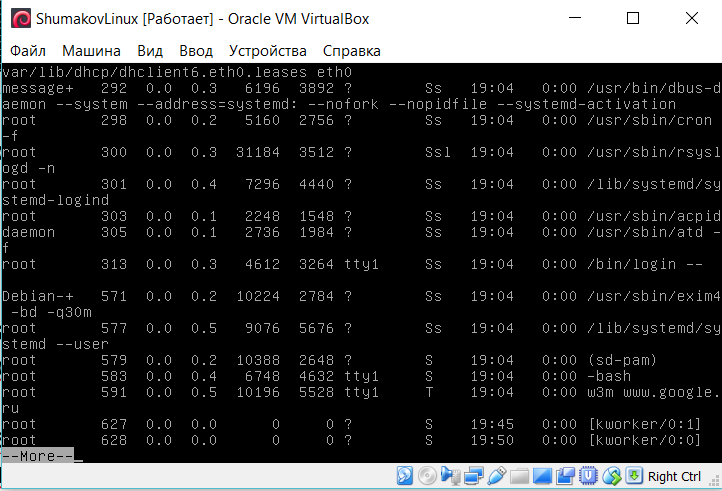
**Объединение команд**

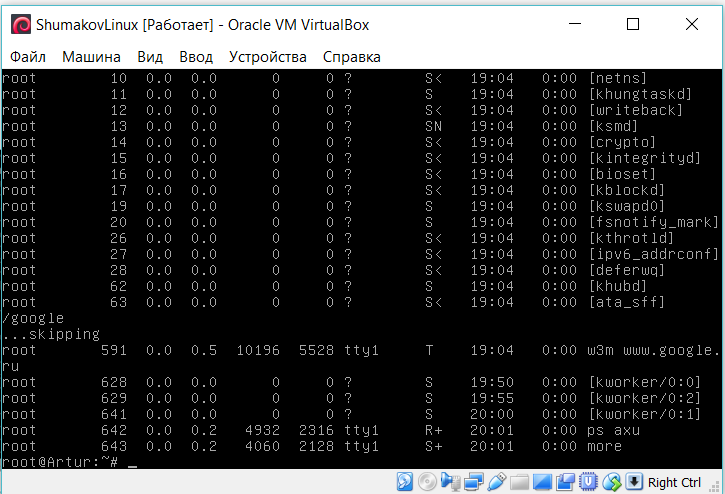
**Конвейеры**

Часто требуется прервать невидимый системный процесс (в linux такие процессы называются демонами, поэтому у многих процессов имя оканчивается на «d»). Команда *«****ps axu****»* выдаёт полный список процессов. Но теперь их слишком много. Все на экране не умещаются.



Чтобы просмотреть список процессов «постранично» дадим команду *«****ps axu | more****».* Для перехода к следующей странице нажмем пробел. Для поиска текста *«google»* наберите «/google» и нажмите ENTER. Для выхода нажмите q.

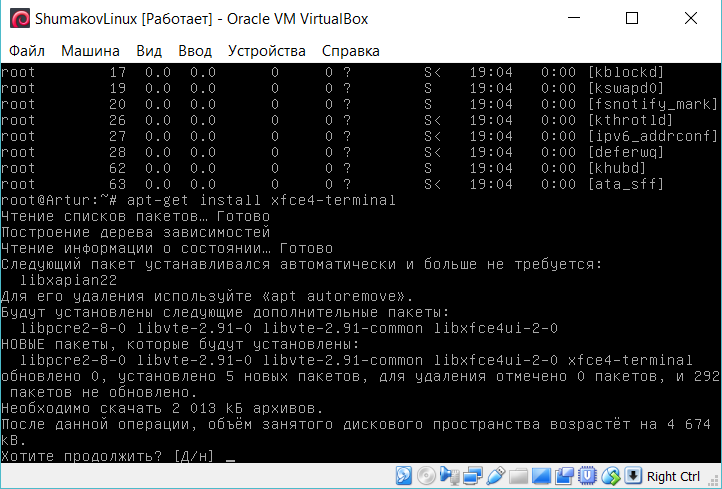




Командой *«ps axu | more»* мы запускаем 2 процесса в **конвейере**.

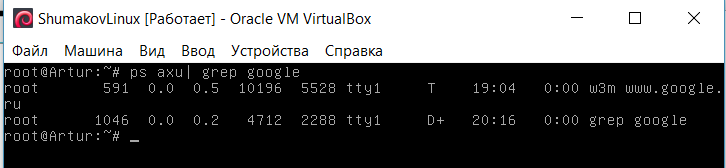
Вывод команды *«ps axu»* направляется на вход команды *more*, которая осуществляет постраничный вывод. Теперь установим другой эмулятор терминала из xfce4, который поддерживает скроллинг.

*apt-get install xfce4-terminal*



Особенно удобно объединять команду *«ps axu»* в конвейер с утилитой *grep*,

Дадим команду *«ps axu |* ***grep*** *google»*

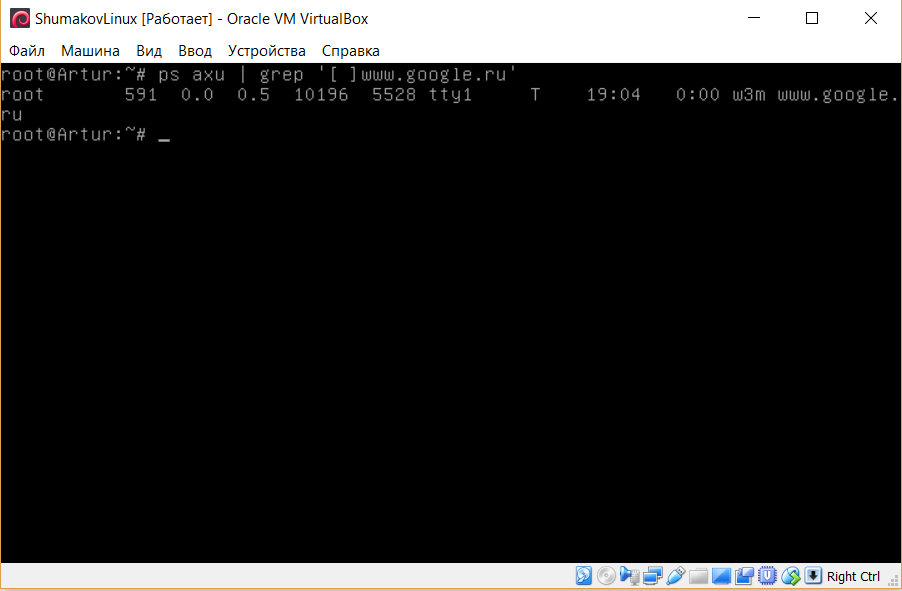


Утилита *«grep»* отбирает из входного потока строки, содержащие текст, заданный первым параметром. Если в искомой строке содержатся пробелы, обязательно заключите такую строку в апострофы *«ps axu | grep* ***'w3m www.google.ru'****»*, иначе *«grep»* будет искать строку «w3m» в файле «www.google.ru». Пробел разделяет в команде параметры.

Утилита *grep* поддерживает регулярные выражения. Символы «.?[]|\» служебные. В частности, в квадратных скобках задаётся диапазон символов которые могут стоять в этой позиции строки.

Выполним команду *«ps axu | grep* ***'[ ]www.google.ru'****»* и попробуем догадаться, как она работает и какая от неё польза.

Польза от этой команды заключается в следующем: данная команда не отберет из входного потока строку, содержащую утилиту *grep*.



**Параллельное выполнение команд**

Команда1 **&** команда 2 — запускает команду 1 в фоновом режиме и сразу начинает выполнение команды 2, не дожидаясь завершения работы команды1

В частности команда «команда **&**» сразу запустит эту команду в фоновом режиме.

Теперь вы знаете, почему в файле «autostart» мы писали «xfce4-panel &»

Запуск команды в фоновом режиме, с указанием «&» отличается от перевода процесса в фоновый режим нажатием Ctrl+Z. По Ctrl+Z процесс получает сигнал SIGTSTR(20) и переходит в состояние «Stoped» (Приостановлен). Чтобы продолжить фоновое выполнение процесса надо дать команду «bg».

**Последовательное выполнение команд**

Команда1**;** команда 2 — последовательно выполняет команды 1 и команды 2

Например, команда «kill 111 **;** sleep 30 **;** kill -9 111» попытается остановить процесс 111, затем подождёт 30 секунд, затем принудительно убьёт этот процесс.

Команды, объединённые «;» выполняются все, все, вне зависимости от результатов работы. Не всегда это хорошо.

Команда1 **&&** команда 2 — выполняет команду 1 и в случае успешного завершения выполняет команду 2.

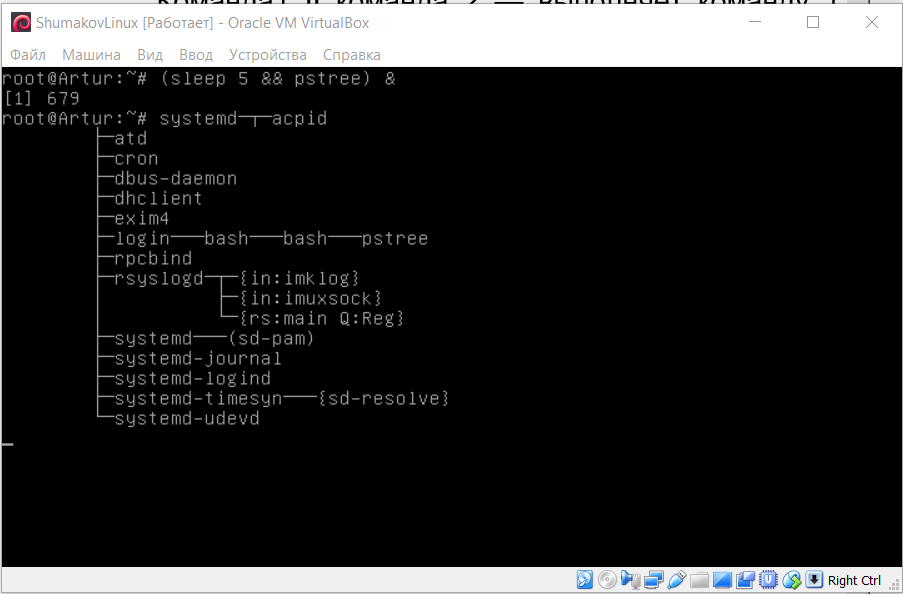
Например, команду отправки архива на другой компьютер следует давать только в случае успешного архивирования нужных файлов.

Команда1 **||** команда 2 — выполняет команду 1 и в случае ошибочного завершения выполняет команду 2.

**Последовательное выполнение группы команд в фоновом режиме**

Выполним команду *(sleep 5 && pstree) &*

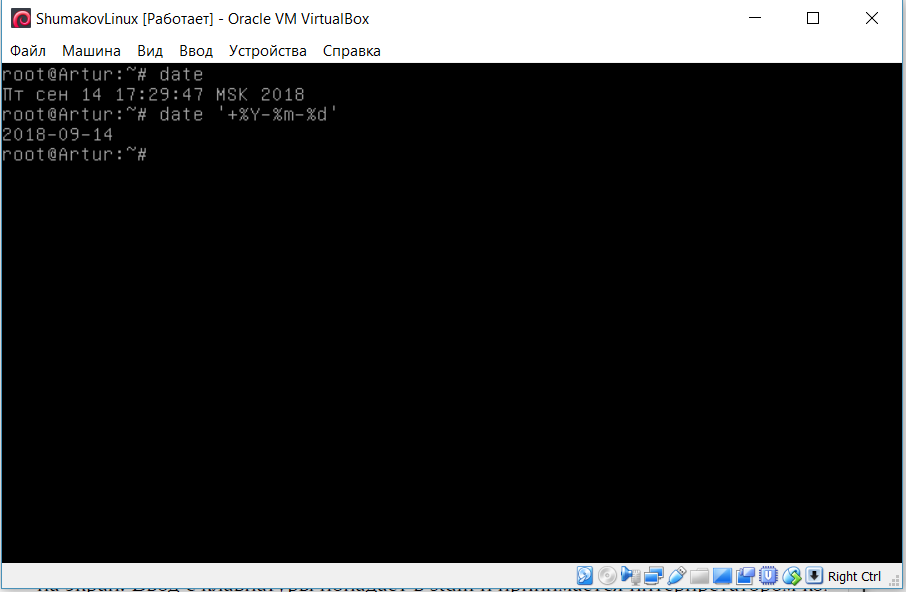
Запускается фоновый процесс, в котором, с задержкой 5 секунд выполнится команда *pstree*. В течении этих 5 секунд мы можем выполнять команды в основном процессе.



Обратим внимание на результаты команды *«pstree»*. Команды, заключённые в круглые скобки, выполняются в новом экземпляре интерпретатора команд *bash*.

Команда «*date*» позволяет не только установить системную дату и время, но и вывести его в заданном формате.

Например, команда «*date '+%Y-%m-%d'*» выводит текущую дату в формате год-месяц-день.



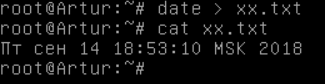
Формат год-месяц-день удобен для сортировки файлов. При сортировке по имени файлы окажутся отсортированными по имени.

**Перенаправление входных и выходных потоков**

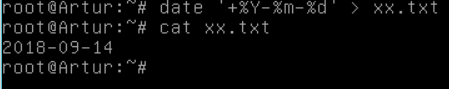
В стандартной библиотеке языка С и других языков программирования есть константы «*stdin, stdout, stderr*», типа поток. Именно в *stdout* осуществляют вывод все команды linux данные из этого потока принимает эмулятор терминала и выводит на экран. Ввод с клавиатуры попадает в *stdin* и принимается интерпретатором команд *bash*. Эти потоки можно перенаправить.

Выходной поток можно записать в файл. Для этого надо задать в команде «***>****file*»

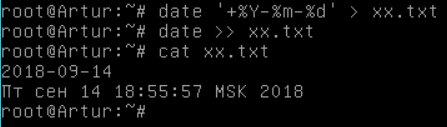
Например, команда «*date* ***>*** *xx.txt*» запишет вывод команды *date* в файл «*xx.txt*». Просмотреть файл можно командой «*cat xx.txt*»



При повторном выполнении команды файл заменяется. Например, дадим команду *date ‘+%Y-%m-%d’ > xx.txt*



Если мы хотим дописать результаты команды в конец существующего файла, надо задать в команде «***>>****file*».



Например, так можно организовать ведение файла протокола.

*date >> log.txt*

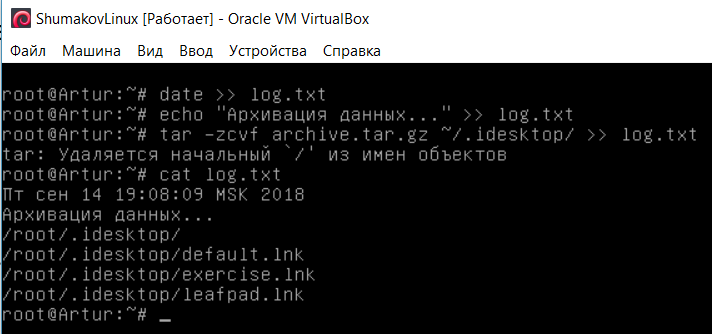
*echo "Архивация данных...">>log.txt*

*tar -zcvf archive.tar.gz ~/.idesktop/ >> log.txt*

Командой *«date»* записываем в файл текущее время (при анализе файла протокола мы должны знать когда произошла ошибка)

Командой «*echo*» записываем в файл строку комментария на производимые действия.

Команда «*tar*» - архивирует, в нашем примере, каталог *.idesktop*



Можно перенаправить *stderr* в другой файл, и тогда у вас будет отдельный протокол ошибок, указав в команде «*2>file*». Например:

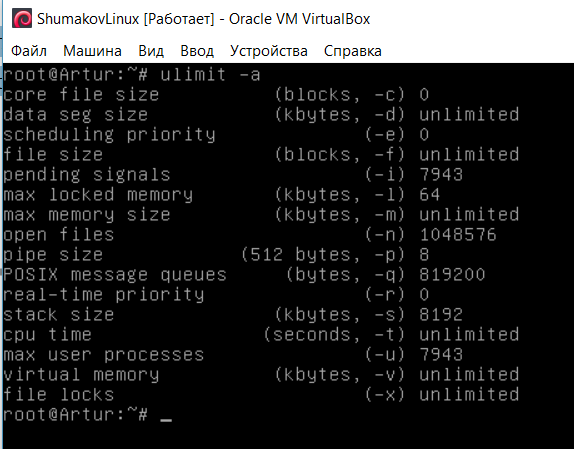
«*tar -zcvf archive.tar.gz ~/.idesktop/ >>log.txt* ***2>>err.txt***»

А можно перенаправить и *stdout* и *stderr* в один и тот же файл, и тогда у вас будет общий протокол, указав в команде *&>file*. Например:

«*tar -zcvf archive.tar.gz ~/.idesktop/* ***&>>log.txt***»

**Ограничения на ресурсы процесса**

В linux имеется возможность ограничить ресурсы, предоставляемые процессу с помощью команды *ulimit*. Для начала просмотрим действующие ограничения с помощью команды «*ulimit -a*»



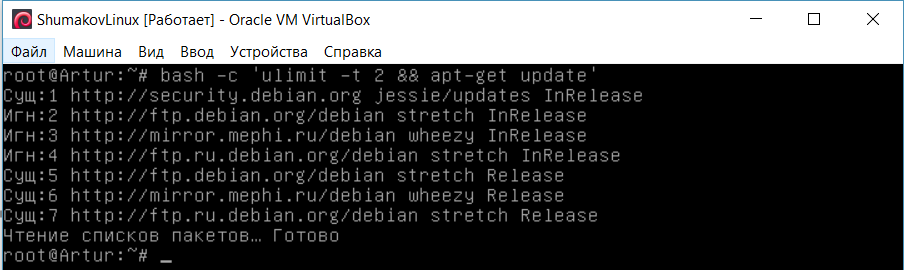
В скобках указаны опции, которые управляют этими параметрами.

Особый интерес представляют ограничения на выделяемую память (-m) и на время выполнения (-t).

Но прежде чем выполнять из командной строки эту команду, запомните, что это ограничение будет действовать на все процессы, запускаемые из интерпретатора команд (*bash*), связанного с текущим терминалом. Поэтому, надо выполнить команды в дочернем *bash*.

Команда «*bash -c 'команда'*» запускает дочерний интерпретатор команд и выполняет в нём заданную команду. Обратим внимание на то, что мы заключили команду в прямые одиночные кавычки. В отличии от двойных кавычек, Linux передает такую строку запускаемой программе как единый параметр, не заглядывая в него, благодаря этому в команде можно использовать символы объединения потоков, не опасаясь, что текущий интерпретатор команд начнёт их интерпретировать, как свои.

Для примера, запустим команду *apt-get update* c ограничением по времени выполнения: «*bash -c 'ulimit -t 2 && apt-get update*'»

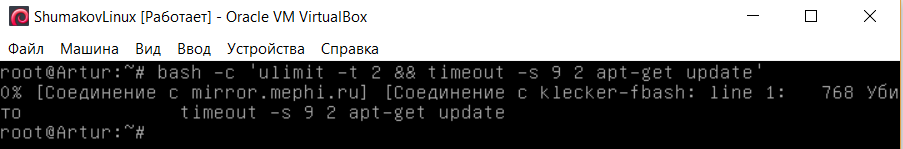


Этой командой мы запускаем в дочернем процессе интерпретатор команд *bash* и просим его выполнить команду «*ulimit -t 2 && apt-get update*».

Дочерний интерпретатор команд сначала устанавливает ограничение времени команду «*ulimit*», а затем выполняет команду «*apt-get update*».

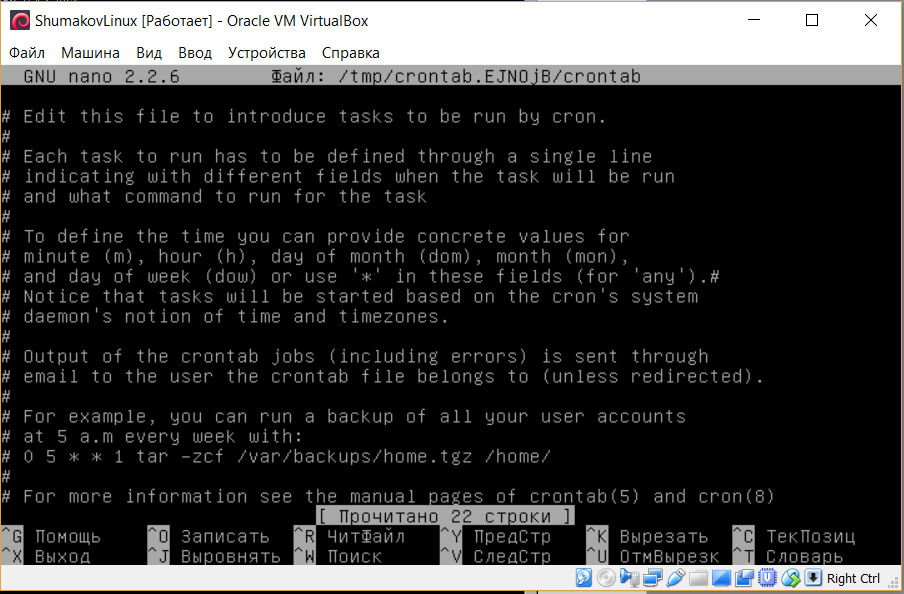
Чтобы ограничить полное время выполнения команды, следует использовать команду «*timeout -s сигнал время команда*»

Например, «*timeout -s 9 2 apt-get update*»



В Linux имеется планировщик задач «*cron*». Каждый пользователь может настроить запуск программ по расписанию, и они будут выполняться от его имени в заданное время.

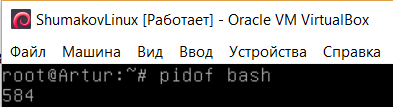
Для того, чтобы настроить расписания следует дать команду «*crontab -e*»



В редакторе «nano» открывается конфигурационный файл текущего пользователя. Текстовый редактор «nano» имеет привычный интерфейс, и подсказки по командам в нижней части экрана. Возможно, запустится другой текстовый редактор в зависимости от установленных пакетов.

**Другие команды управления процессами.**

*pidof* – утилита, выдающая *pid* процесса по его имени.



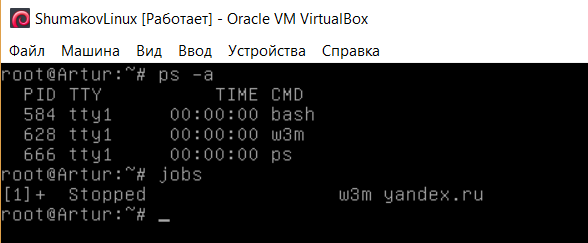
*kill* – завершение процесса с определённым *pid*, возможна отправка сигнала: *kill [-номер сигнала] PID*.

*killall* – завершает все процессы с совпадающим именем, в остальном команда похожа на *kill*.

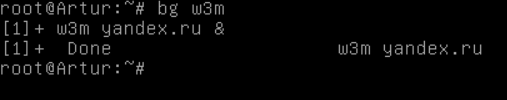
*nice* – изменить приоритет процесса для планировщика. По умолчанию понижает приоритет процесса, по умолчанию он равен 10. Позволяет установить приоритет от -20 (наивысший приоритет) до 19 (низший, в *FreeBSD* низшим является приоритет 20). Для повышения приоритета процесса надо писать смещение приоритета с двойным минусом: --5. Следует отметить, что *nice* не устанавливает приоритет, а указывает смещение приоритета. Поле *nice (NI)* есть, например, на рисунке 10.

*nohup* – игнорирование всех сигналов прерывания, кроме *SIGHUP* и *SIGQUIT*.

*jobs* – показывает список процессов, выполняющихся в фоновом режиме.



*bg* – продолжить исполнение приостановленной программы в фоновом режиме.



*fg* – вывод программы в обычный режим (foreground, «на передний план»).

**POSIX.**

*POSIX (Portable Operating System Interface for Unix)* – набор стандартов, определяющих взаимодействие ОС с прикладными программами. Разрабатывается *IEEE (Institute of Electrical and Electronical Engineers)*, комитетом 1003. Состоит из следующих частей:

1.*POSIX 1003.1* – определяет стандарт на основные компоненты ОС, API для процессов, ФС и т.д.

2.*POSIX 1003.1b* – стандарт на расширения «реального времени» (планировка приоритетов, семафоры, передача сообщений, сигналы реального времени, асинхронный и синхронный ввод/вывод, разделяемая память и т.д.).

3.*POSIX 1003.1c* – стандарт на потоки (создание и завершение, планирование, синхронизация, обработка сигналов).

4.*POSIX 1003.1d* – стандарт на дополнительные расширения «реального времени» (обработчики прерываний и т.д.).

5.*POSIX 1003.2* – стандарт на основные утилиты.

ОС, полностью отвечающие стандартам *POSIX*, называются *POSIX-совместимыми*, или *полностью совместимыми*. Также существуют частично совместимые и по большей части совместимые. Для официального признания совместимости ОС должна пройти сертификацию, потому некоторые POSIX-совместимые ОС могут находится в группе «частично совместимых».

**D-Bus.**

*D-Bus* представляет собой систему межпроцессного взаимодействия (*IPC, Inter-Process Communication*). Эта система предоставляет различные шины для обмена сообщениями. Система была разработана в рамках проекта freedesktop.org для возможности коммуникации процессов вне зависимости от окружения рабочего стола.

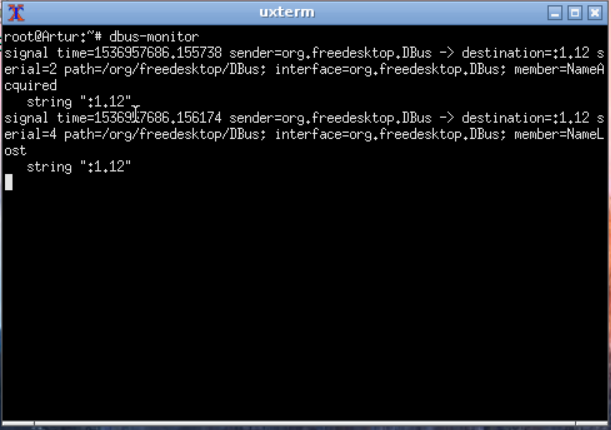
В данной лабораторной работе будут рассмотрены сообщения, передаваемые по различным шинам. Для этого будет использована утилита *dbus-monitor*.

Некоторые опции утилиты *dbus-monitor*:

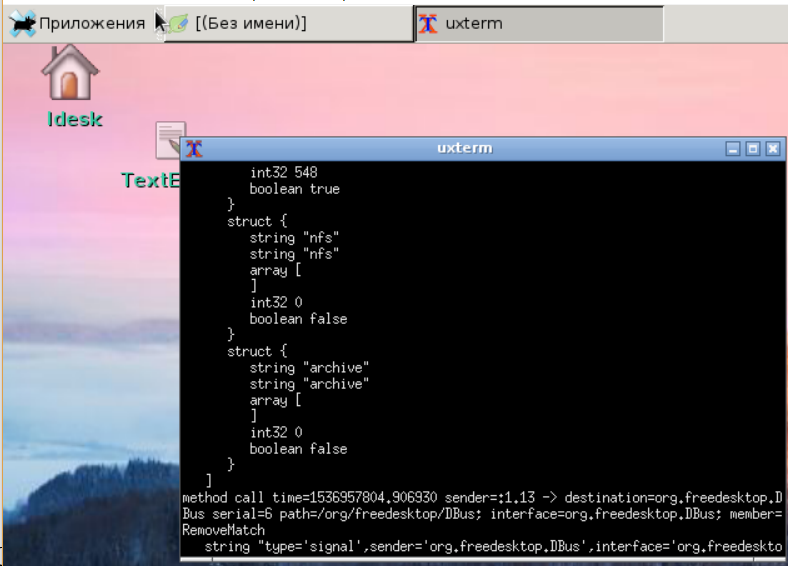
*--system* – мониторинг системной шины сообщений.

*--session* – мониторинг шины сообщений сессии (по умолчанию).

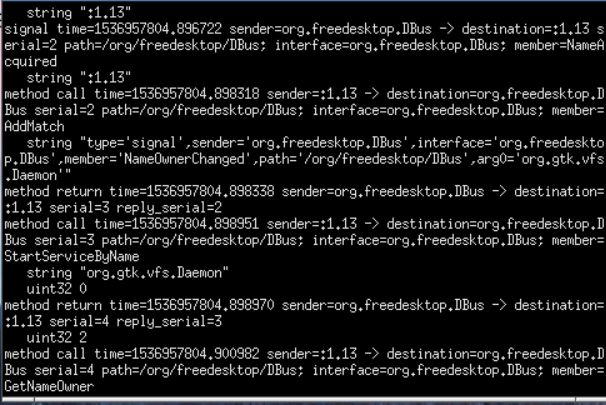
Просмотрим сообщения, переданные по шине сессии (система сразу после запуска).



Сообщения не отличаются разнообразием. Не закрывая окно эмулятора терминала с запущенным *dbus-monitor*, запустим *Leafpad* через панель *xfce*. После этого в окне *dbus-monitor* произойдут изменения, показанные на следующем рисунке



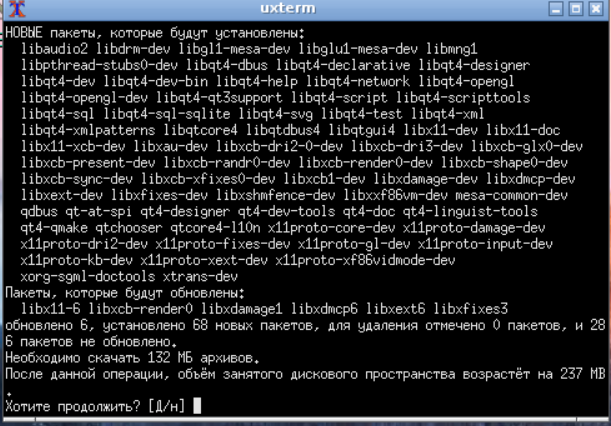
На следующем рисунке показано начало этого сообщения.



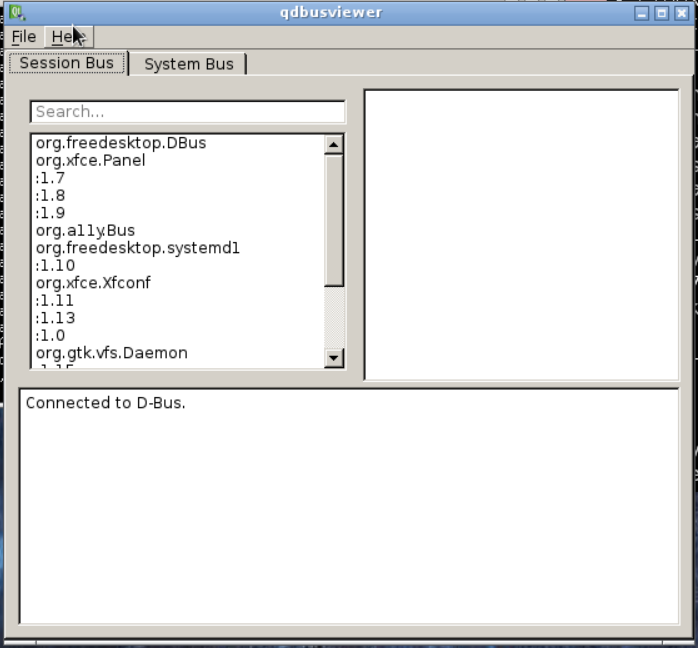
Можно отправлять сообщения по *D-Bus* прямо из консоли, для этого служит команд*а dbus-send*.

Библиотека Qt поддерживает работу с *D-Bus*. Для работы с *D-Bus* используются два приложения: *qdbus* (консольный интерфейс) и *qdbusviewer* (графический интерфейс). Это приложение находится в пакете *qt4-dev-tools*:

*apt-get install qt4-dev-tools*

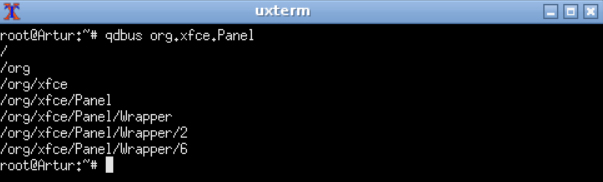


Команда*: qdbusviewer*



Как видно, утилита *qdbusviewer* позволяет просматривать интерфейсы и их методы для приложений, использующих *D-Bus*. Это же можно сделать с помощью консольной утилиты *qdbus*.

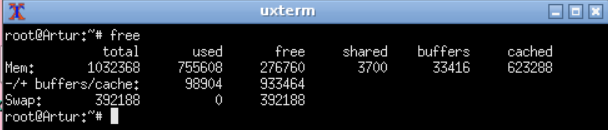
Посмотрим теперь вывод *DBus для xfse4-panel*. Выполним *qdbusorg.xfce.Panel*



**Управление оперативной памятью в Linux.**

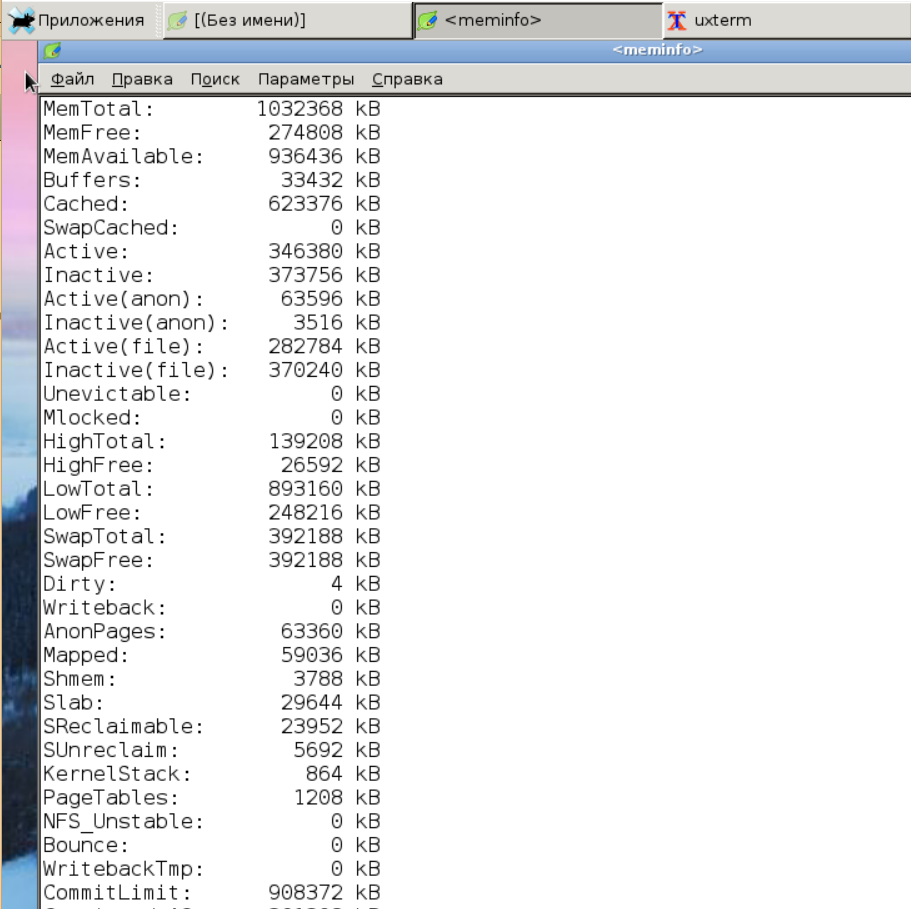
Данная секция описывает некоторые полезные команды для мониторинга состояния ОЗУ. Предполагается, что предварительно был освоен теоретический материал.

Для мониторинга оперативной памяти можно использовать утилиту *free*. Эта утилита показывает статистику по использованию ОЗУ и файла подкачки (*swap*).



Строка «*-/+ buffers/cache*» показывает, сколько памяти использовано и сколько свободно с точки зрения её использования в приложениях.

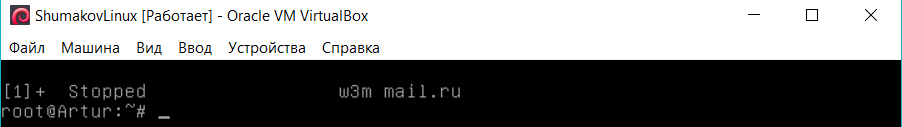
Более подробную информацию об использовании памяти можно посмотреть в файле */proc/meminfo*, именно этот файл использует утилита *free*. На рисунке 17 показан пример такого файла.



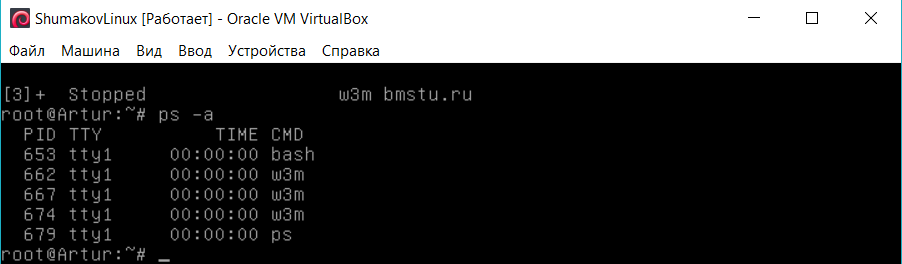
Здесь указывается используемая, неиспользуемая память, статистика использования, свопинга.

**Выполнение заданий к лабораторной работе №4**

1) Открыть в текстовом браузере некую страницу и перевести его в фоновый режим.



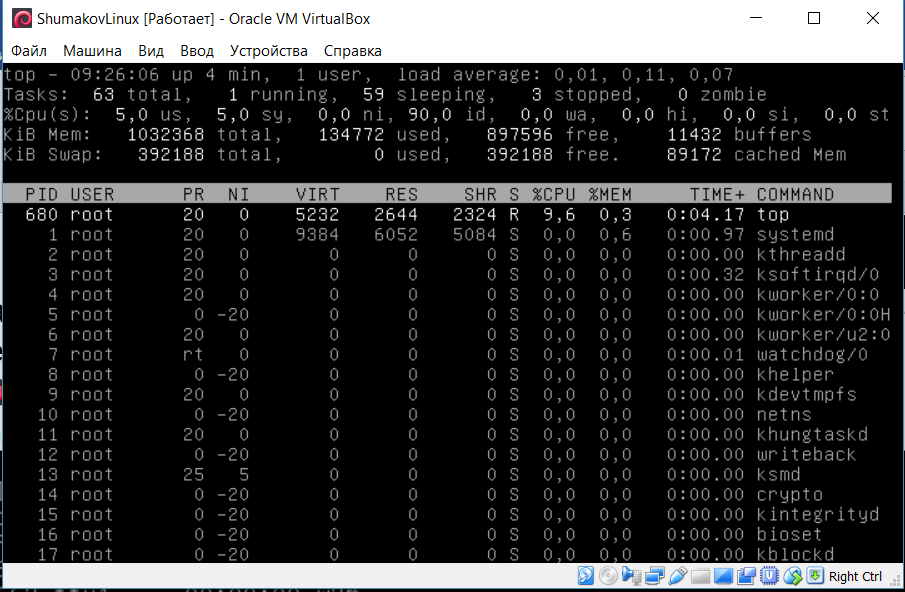
2) Запустить ещё два экземпляра текстового браузера в фоновом режиме



3) Найти процесс, максимально нагружающий процессор

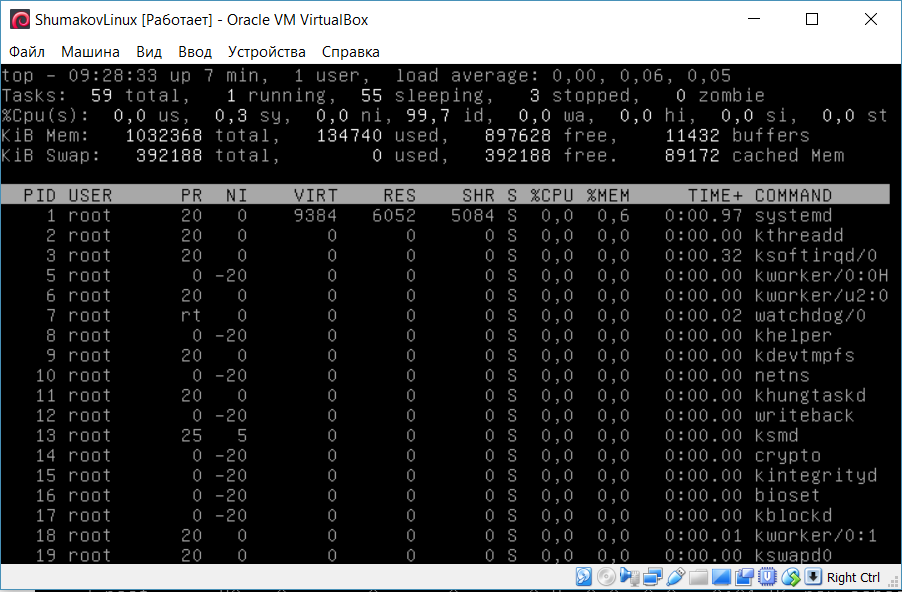
Введем команду в терминале *top* см *CPU*

Смотрим на столбец *Command* и определяем, что *top* занимает 9,6% процессорного времени

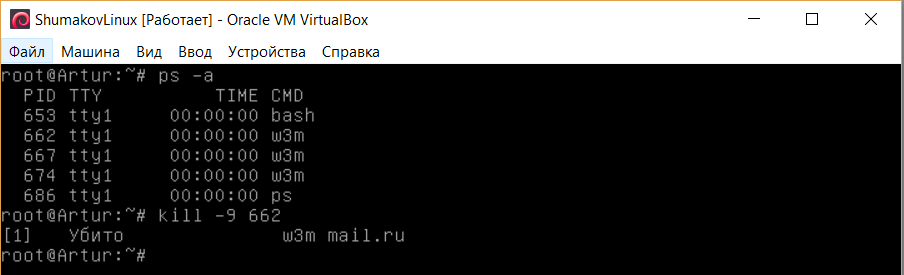


4) Вывести список процессов текущего пользователя

выполним *top -u root*

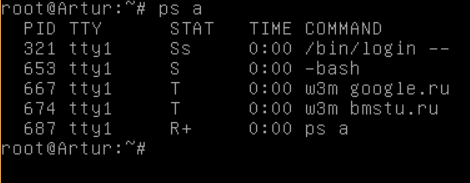


5) «Убить» первый процесс браузера в котором открыта 1 страница



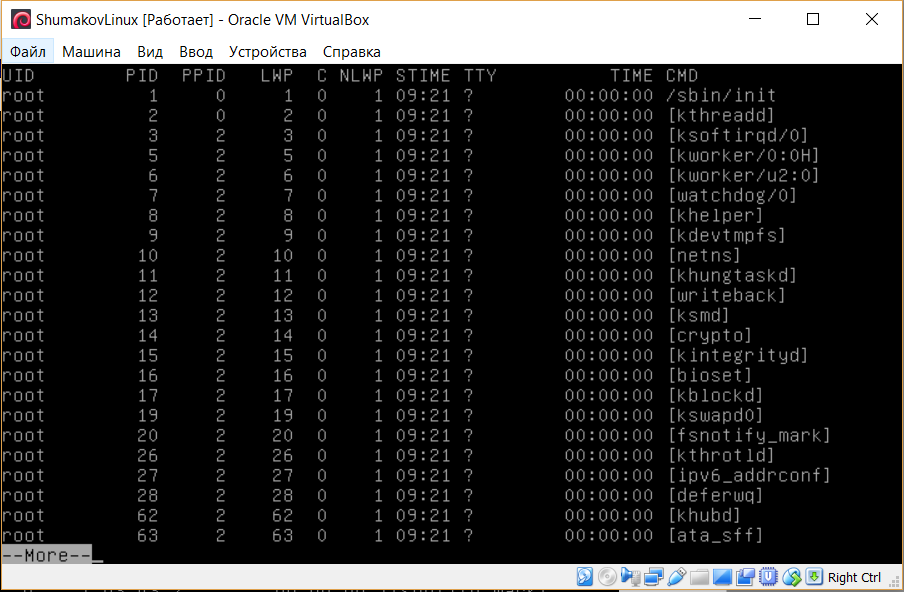
6) Вывести список всех процессов всех пользователей

Дадим команду *ps a*



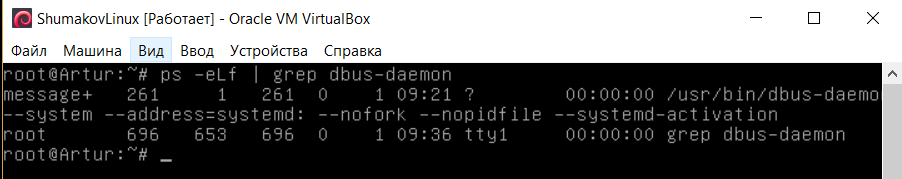
7) Просмотреть список процессов постранично

Введем команду *ps –eLf | more*



8) Отобрать из вывода команды *ps* строку, соответствующую процессу «*dbus-daemon*», определить, где лежит её выполняемый файл и с какими параметрами он запущен.

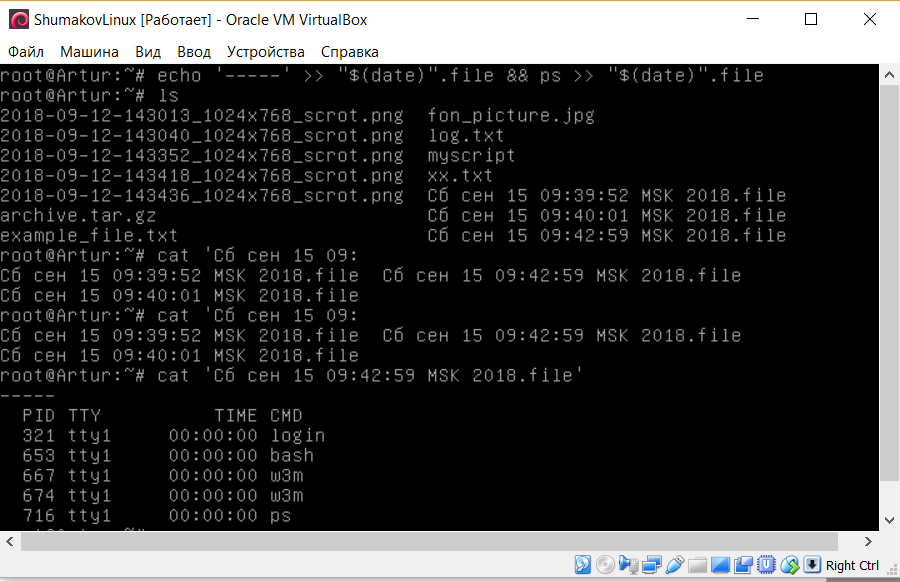
Выполним *ps -eLf | grep dbus-daemon*



9) Записать в файл с именем, содержащим текущее время, строку «-------» и список процессов

*echo ‘-------‘ >> “$(date)”.file && ps >> “$(date)”.file*

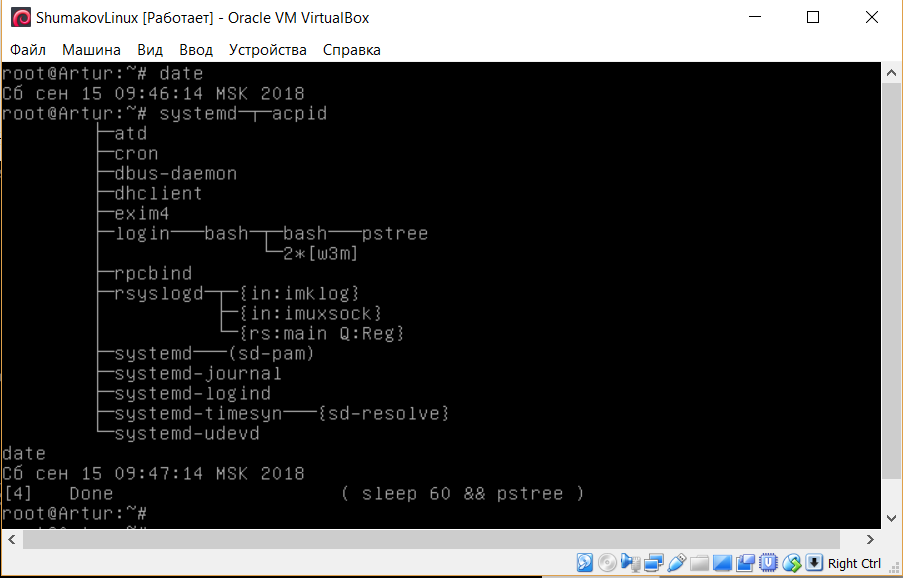
Затем находим только что созданный файл командой *ls* и открываем его при помощи команды *cat*



10) Выполнить команду в фоновом режиме с отсрочкой запуска на 1 минуту. Продемонстрировать, что команда выполнилась именно через минуту.

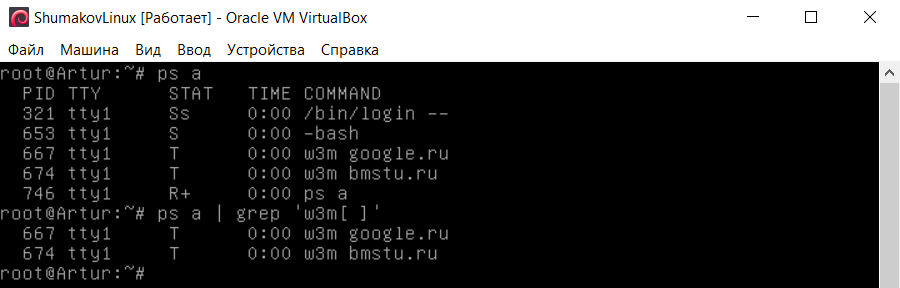
Введем команду *(sleep 60 && pstree) &*

Первая ее часть означает, что выполнение команды *pstree* выполнится с задержкой в 60 секунд. После успешного выполнения *sleep 60* выполнится *pstree* за это отвечает связка &&, использующаяся в команде. Последний символ & в команде означает, что команда будет выполняться в фоновом режиме. Мы используем это для того, чтобы успеть дать команду *date*, тем самым показав, что команда выполнилась ровно через минуту.



11) Отобрать из одного из сформированных файлов строки, относящиеся к одному из процессов.

Отберем строки, относящиеся к процессу браузера *w3m: ps a | grep ‘w3m[]’*

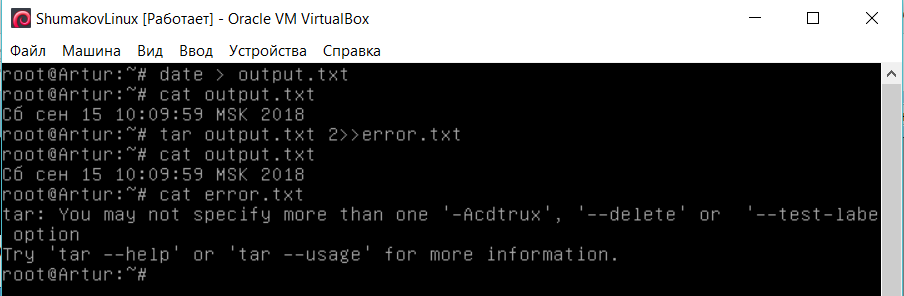


12) Вывести результаты работы произвольной команды в один файл, а сообщения об ошибках в другой. Продемонстрировать правильность работы.

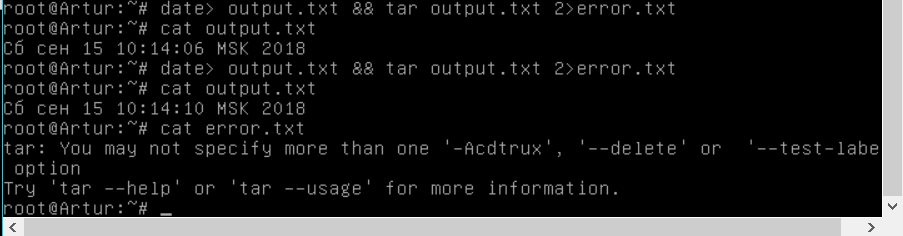
Создадим файл *output.txt*, содержащий текущую дату. При помощи команды

*tar output.txt 2>>error.txt* выводим сообщения об ошибках в отдельный файл *error.txt*

Командой *cat* попытаемся открыть файлы *output.txt* и *error.txt*, чтобы проверить их содержимое.

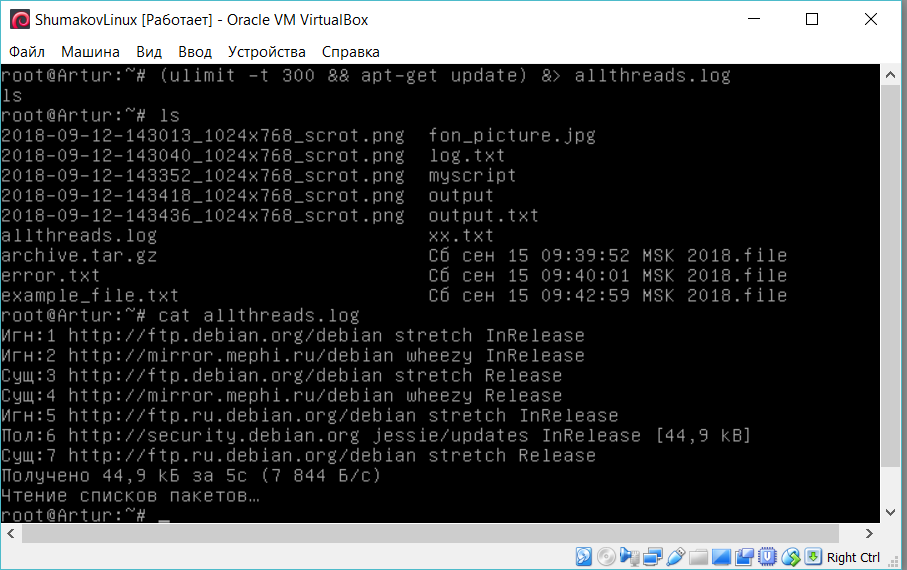


В файл *error.txt* записываются ошибки, но их нет, поэтому файл не был создан. Можно предложить и другое решение данной задачи.



13) Выполнить произвольную команду с ограничением использования процессорного времени 300 секунд и выводом результатов и сообщений об ошибках в один файл.

Дадим команду: *(ulimit –t 300 && apt-get update) &> allthreads.log*



**Вывод:** в ходе работы были изучены управление процессами и памятью в Linux.