

Московский государственный технический университет
имени Н.Э. Баумана

(национальный исследовательский университет)

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

В.Ю. Мельников, Е.К. Пугачев

Исследование способов организации оперативной памяти и взаимодействия процессов

Электронное учебное издание

Методические указания по выполнению лабораторных работ

по дисциплине "Операционные системы"

2018

Введение

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

Время выполнения - 4 часа.

Организация оперативной памяти.

Оперативная память - это энергозависимая часть компьютерной памяти в которой во время работы компьютера хранятся программы и их рабочие данные.

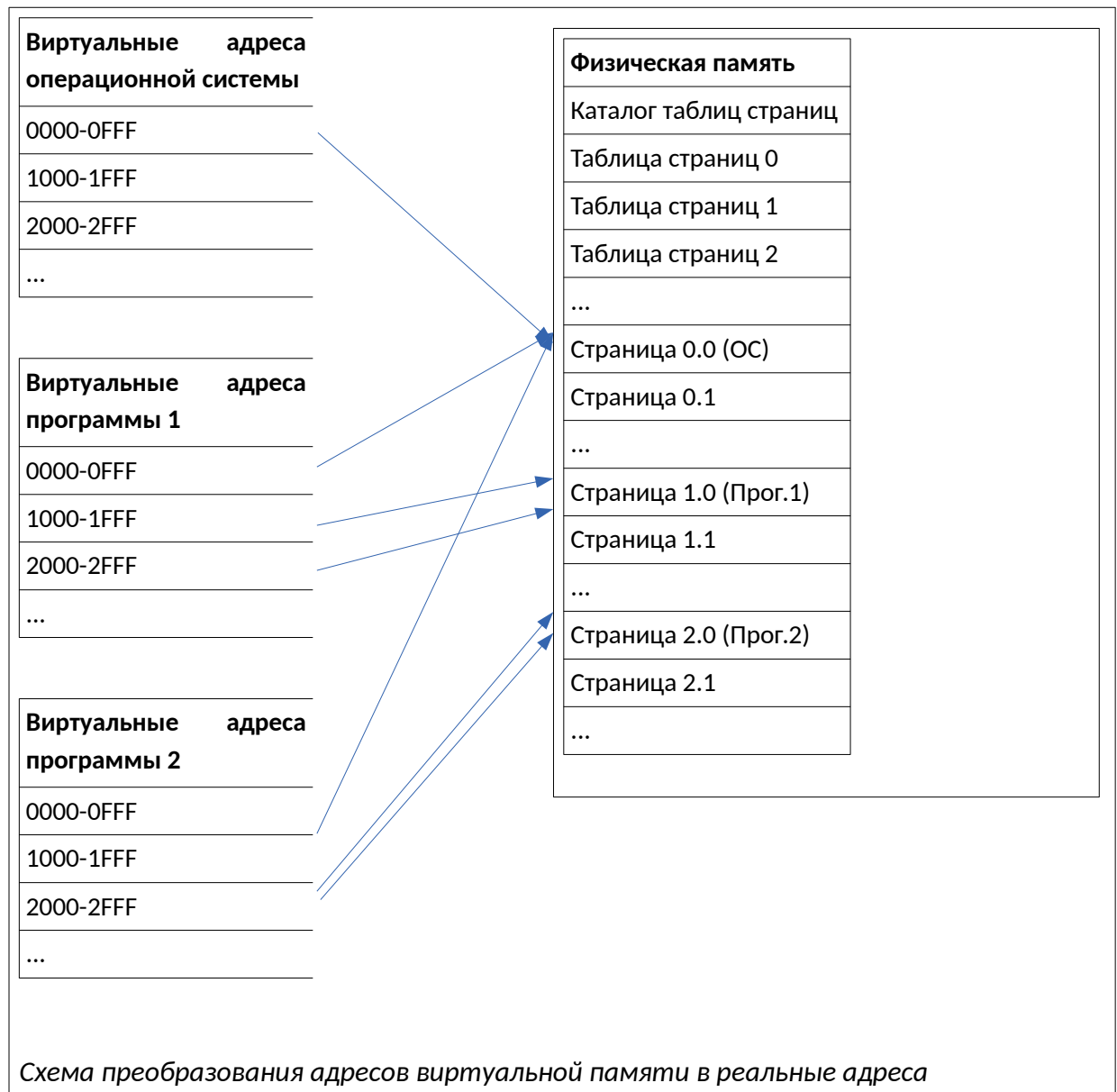
Для выполнения программы, её необходимо загрузить в оперативную память. Программы не могут работать с данными, находящимися непосредственно на жёстком диске, их сначала надо загрузить в оперативную память. А это длительная операция. Поэтому, обычно объём оперативной памяти является критически ресурсом увеличение которого резко ускоряет работу программ, причём гораздо сильнее, чем увеличение быстродействия процессора. Но, оперативная память дорогая, как по деньгам, так и по энергозатратам. На её поддержание постоянно требуется расходовать энергию, что сказывается на времени автономной работы планшетов и ноутбуков. Так что, оперативной много не бывает.

Защита памяти

Ранние версии ОС использовали реальный режим адресации оперативной памяти при котором адреса, используемые программами соответствовали физическим адресам оперативной памяти. В этом режиме, программы имели неограниченный доступ к областям памяти друг друга и могли намеренно или случайно повредить их. Исключительно благоприятная среда для распространения вирусов.

В современных ОС каждый процесс выполняется в виртуальной памяти - виртуальном адресном пространстве, которое аппаратно проецируется на адреса реальной памяти, выделенной программе. При попытке обращения к адресам за её пределами, вызывается прерывание "ошибка страницы" и операционная система прерывает выполнение программы. Для обмена данными между процессами используются области коллективного доступа. Ядро ОС помещается в адресное пространство программы, но помечается как недоступное для пользователя.

Рассмотрим как работает виртуальная память:



Вся физическая память делится на страницы фиксированного размера (обычно 4 КБайт).

Виртуальные адреса (которыми оперирует программа) делятся на 2 части:

- номер виртуальной страницы
- смещение в рамках страницы — биты 11-0 (12 бит).

Для каждого процесса операционная система создаёт таблицу соответствия виртуальных страниц реальным страницам. (Формат таблиц зависит от типа процессора, часто используются многоуровневые таблицы)

При обращении программы к памяти виртуальный адрес аппаратно пересчитывается в реальный согласно этой таблице.

Для программы виртуальная память выглядит, как непрерывное адресное пространство, в пределах выделенного лимита, вне зависимости от наличия у компьютера

такого количества оперативной памяти. При обращении к виртуальной странице, для которой не хватает реальной памяти, вызывается прерывание "ошибка страницы"(page fault). Операционная система сначала выгружает на диск страницу, к которой давно не обращались, а затем заносит в таблицу страниц ссылку на освободившуюся страницу и возвращает управление прерванной программе. Данный процесс называется подкачкой или свопингом.

В ОС Windows содержимое виртуальной памяти сохраняется в файл, в linux используется раздел подкачки.

Но надо понимать, что на выгрузку и подгрузку страниц с диска уходит много времени и при серьёзной нехватке памяти свопинг не спасает. Система начинает работать так медленно, что даже завершение работы лишних программ становится проблемой. Система начинает работать так медленно, что реакция на простейшие действия наступает через несколько секунд. В таком случае, особенно в графическом интерфейсе, надо на некоторое время прекратить нажимать кнопки, дождаться выполнения тех действий, которые вы уже сделали и только после этого продумывать каждое действие и дожидаться на него реакции. При возникновении свопинга на сервере, подключиться к нему в текстовом интерфейсе и прервать выполнение некоторых программ. Затем изменить настройки программ так, чтобы они использовали меньше памяти. Например, уменьшить размер кеша таблиц и индексов для СУБД.

Отображаемые в память файлы

Хотелось бы упомянуть ещё один приём, использования виртуальной памяти - отображаемые в память файлы. Представьте себе, что запускается система управления базой данных (СУБД). При этом надо прочесть большой объём данных (индексы, и описания таблиц), часто превышающий объём физической памяти. При этом часть данных выгружается в файл подкачки, что ещё увеличивает время запуска. Программа может самостоятельно подгружать эти данные по мере необходимости, а может отобразить файл базы данных в виртуальную память и его будет подгружать, по необходимости, менеджер виртуальной памяти. Запуск СУБД в этом случае будет очень быстрым, но первое время СУБД будет работать медленно, пока не загрузятся часто используемые данные.

Кеширование операций чтения и записи на диск

Итак, в памяти у нас хранится ядро операционной системы, выполняемые программы и их данные. Но это ещё не всё. 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 вы 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. Вы не можете явно указать сколько мегабайт может система использовать под кэш, но можно настроить скорость удаления просроченных страниц из кэша. В Linux файл `/proc/sys/vm/vfs_cache_pressure` содержит число, которое говорит, насколько агрессивно нужно удалять страницы из кэша. По умолчанию установлен параметр 100. Если его уменьшить ядро будет реже удалять страницы и это приведет к увеличению кэша. При нуле страницы вообще не будут удаляться. Если значение больше 100 страницы будут сразу удаляться быстрее, но чтение замедлится.

Посмотреть размер страничного кэша можно командой «free -h»

	total	used	free	shared	buffers	cached
Mem:	11G	4,7G	6,9G	204M	243M	2,5G
+/+ buffers/cache:		2,0G	9,7G			
Swap:	2,0G	0B	2,0G			

Размер страничного кэша указан в колонке `cached`

Кэш INode — Здесь хранится таблица расположения файлов и папок. Этот кэш невелик, но очень сильно ускоряет работу с файловой системой. Его настройки лучше не менять.

Есть один эффект кэша, который надо учитывать. После того, как программа записала данные в файл, они не сразу записываются на диск. А в параллельном процессе, с небольшой отсрочкой. Это уменьшает нагрузку на диск при записи малыми порциями. Ведь для записи даже одного байта, надо прочитать и целиком перезаписать целый блок не менее 512 байт (обычно несколько Кб).

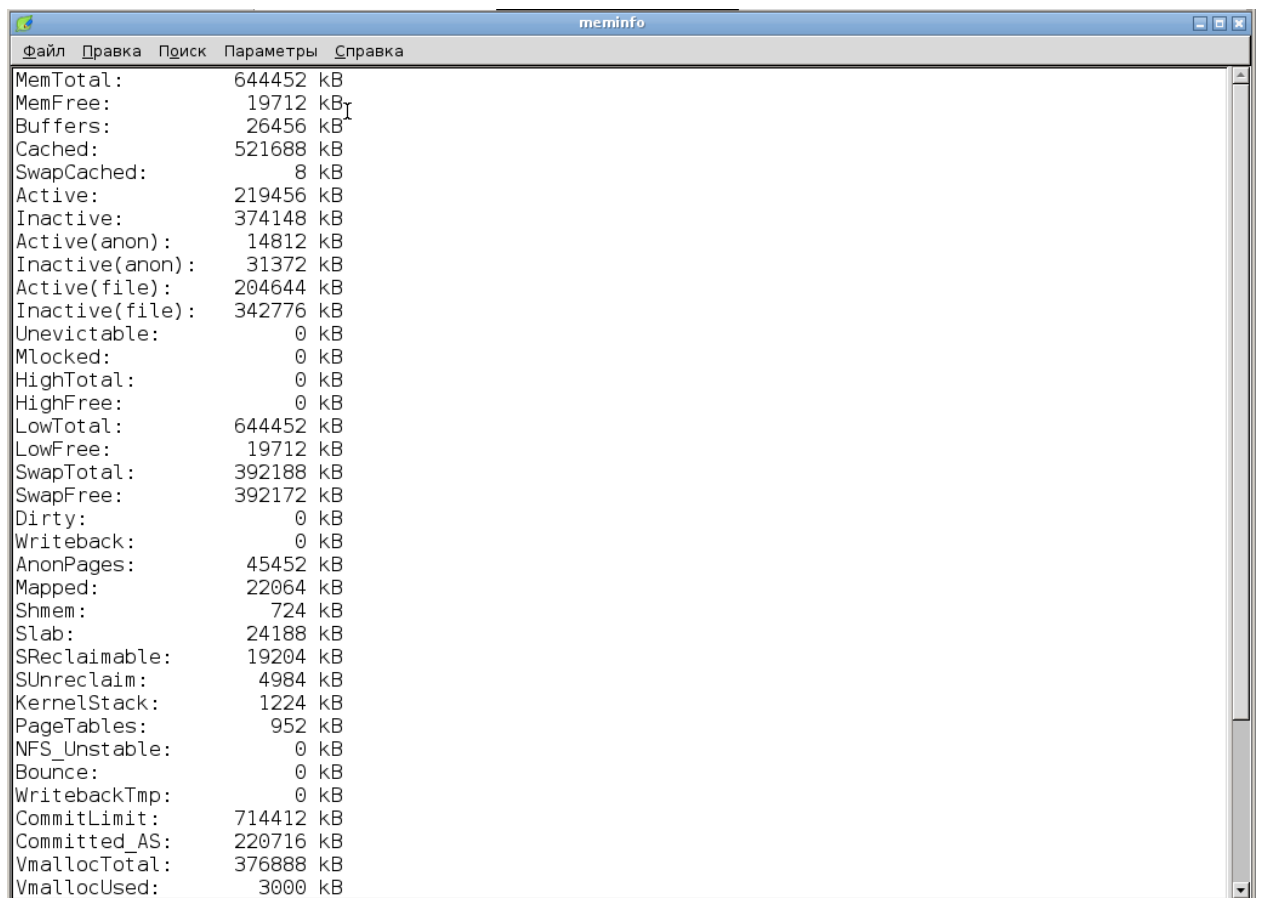
Программы отсрочки не замечают, поскольку получают данные из кэша и единственное тонкое место этой технологии — аварийное выключение компьютера. Последние данные из кэша не успеют записаться, файлы будут повреждены. Наиболее неприятные последствия потери кэша при записи таблицы расположения файлов и каталогов. Для ранних версий файловых систем аварийное выключение могло привести к повреждению операционной системы вплоть до невозможности загрузки. В современных системах предусмотрено дублирование информации и процедура восстановления структуры файловой системы после аварийного завершения, но она может быть продолжительной. Поэтому надо корректно завершать работу ОС.

Впрочем, в современных ОС нажатие кнопки питания просто запускает процедуру завершения работы, которая посылает сигнал завершения всем запущенным процессам, затем, после их завершения сохраняет данные из кэша и только после этого, даёт команду отключения питания. Если спустя пару минут после нажатия кнопки выключения питания компьютер не выключился, придётся отключить его принудительно. Для этого надо

нажать кнопку питания и не отпускать в течении 3-5 секунд. Компьютер аппаратно выключится.

Кэширование применяется и при записи данных на флешку. Поэтому нельзя сразу выдёргивать флешку, как только закончится копирование файлов. Если у флешки есть световой индикатор надо дождаться пока он прекратит мигать. А ещё лучше использовать команду безопасного отключения носителя в операционной системе или файловом менеджере.

Более подробную информацию об использовании памяти можно посмотреть в файле `/proc/meminfo`, именно из этого файла берёт информацию утилита `free`. На следующем рисунке показан пример такого файла:



MemTotal:	644452 kB
MemFree:	19712 kB
Buffers:	26456 kB
Cached:	521688 kB
SwapCached:	8 kB
Active:	219456 kB
Inactive:	374148 kB
Active(anon):	14812 kB
Inactive(anon):	31372 kB
Active(file):	204644 kB
Inactive(file):	342776 kB
Unevictable:	0 kB
Mlocked:	0 kB
HighTotal:	0 kB
HighFree:	0 kB
LowTotal:	644452 kB
LowFree:	19712 kB
SwapTotal:	392188 kB
SwapFree:	392172 kB
Dirty:	0 kB
Writeback:	0 kB
AnonPages:	45452 kB
Mapped:	22064 kB
Shmem:	724 kB
Slab:	24188 kB
SReclaimable:	19204 kB
SUnreclaim:	4984 kB
KernelStack:	1224 kB
PageTables:	952 kB
NFS_Unstable:	0 kB
Bounce:	0 kB
WritebackTmp:	0 kB
CommitLimit:	714412 kB
Committed_AS:	220716 kB
VmallocTotal:	376888 kB
VmallocUsed:	3000 kB

Поищите в этом файле параметры, с которыми мы познакомились в этом разделе.

Как запускается процесс?

Когда мы выполняем команду мы запускаем программу в новом процессе.

Рассмотрим, что при этом происходит:

1. Родительский процесс (в нашем случае интерпретатор команд) выполняет вызов функции `fork()`. Эта функция помещает аргументы в определённое место и выполняет команду эмулированного прерывания (`syscall`). Управление передаётся соответствующей функции ядра `linux`, одновременно происходит переход из **режима пользователя** в котором запрещены многие команды процессора в **режим ядра**. После выполнения системной функции управление возвращается вызывающему процессу с возвратом в режим пользователя. Такой вызов функции ядра называется «**системным вызовом**». Системные вызовы описаны стандартом POSIX. Системная функция `fork()` выполняет следующие действия:
 - Процесс регистрируется в таблице процессов и получает идентификатор процесса PID (Process IDentificator)
 - Создаётся новая область виртуальной памяти.
 - Копируется содержимое виртуальной памяти родительского процесса. (На самом деле память пока не копируется. Просто создаётся таблица страниц виртуальной памяти, которая указывает на страницы памяти родительского процесса, помеченные, как доступные только для чтения). Когда дочерний процесс пытается писать в эти страницы, ядро ОС выделяет новые страницы памяти, доступные на запись, и копирует туда страницы родительского процесса.
 - Увеличиваются счетчики открытия файлов (порожденный процесс наследует все открытые файлы родительского процесса и может использовать их для обмена данными с дочерним процессом).
 - Получается две копии процесса
 - Возвращает родительскому процессу PID дочернего процесса и 0 дочернему процессу.
2. Загружается новая программа системным вызовом «`exec()`». Новая программа загружается в виртуальную память дочернего процесса на место родительской программы. Освобождается неиспользуемая память родительского процесса. Управление передаётся функции `main()`. В эту функцию передаётся список параметров.
3. Родительский процесс может дожидаться завершения дочернего процесса с помощью системного вызова «`wait()`» (последовательное выполнение команд) или продолжить работу (параллельное выполнение).

Взаимодействие процессов

Каждый процесс выполняется в своей области виртуальной памяти. Способы взаимодействия процессов ограничены следующими:

Обмен через файлы — Один процесс пишет в файл, другой из него читает.

Каналы (pipes) — Один процесс пишет в канал, другой из него читает.

Сигналы — Один процесс может послать сигнал с заданным номером, другому процессу. Обычно это сигнал останова дочернего процесса, но может быть и какой-то другой сигнал, на который запрограммирована реакция.

Общие области памяти (shared memory).

Сокет — Обмен данными между процессами по сети. При необходимости, можно легко перенести часть процессов на другой сервер, чтобы разгрузить первый.

Потоки выполнения (thread)

Назначение потоков (threads) – параллельное выполнение двух и более задач. Каждому процессу соответствует по крайней мере один поток, но процесс это тяжёлая сущность. Требуется довольно много времени для запуска процесса и переключение между процессами. Но главное, каждый процесс выполняется в своей области виртуальной памяти и обмен данными между ними затруднён.

В каждом процессе можно запустить несколько потоков (threads), которые будут работать в одной области памяти и активно обмениваться данными. Переключение между потоками производится гораздо быстрее чем между процессами.

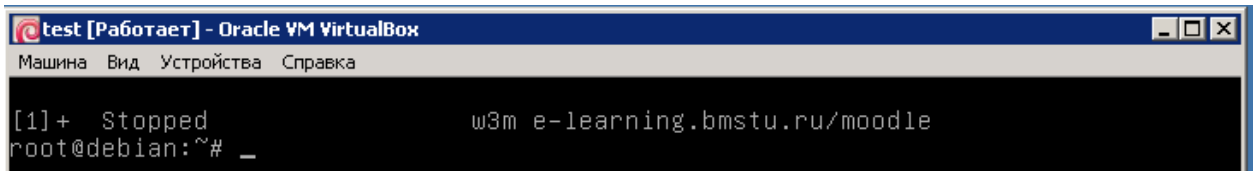
С другой стороны, возникают проблемы синхронизации потоков при обращении к общим ресурсам. Предположим, два потока одновременно переводит деньги со счёта на счёт. Первый поток читает состояние некоего счёта из ячейки памяти, увеличивает его, а затем записывает обратно. Между чтением и записью значения проходит пусть небольшое, но время. Но одновременно, другой поток (случайно) читает значение того же счёта (старое, ещё не изменённое первым процессом), изменяет его и записывает обратно. В ячейке памяти окажется значение, записанное одним из процессов - неправильное. Поэтому, на время обращения одного из процессов к критическому ресурсу, надо блокировать доступ остальных процессам к этому ресурсу. При этом, желательно пользоваться не медленными системными вызовами функций ядра, а функциями процесса.

В современных ОС используется **вытесняющая многозадачность** (preemptive multitasking) – поток с большим приоритетом, может вытеснить поток с меньшим. Например, потоки, обслуживающие прерывания от устройств имеют больший приоритет и будут выполняться в первую очередь. Поэтому, даже когда процессор нагружен на 100%, курсор мыши продолжает двигаться по экрану без задержки.

Команды управления процессами в Linux.

Многозадачность

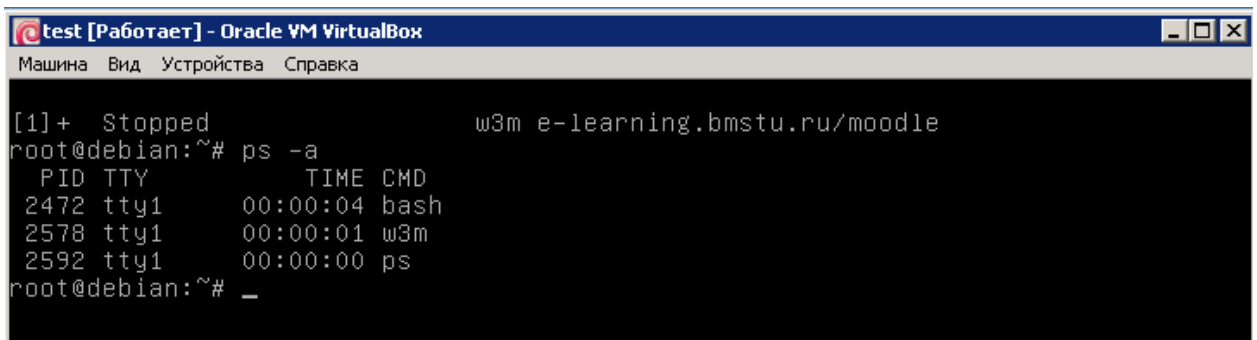
Консольный интерфейс UNIX поддерживает многозадачность. Для того чтобы свернуть приложение, необходимо нажать Ctrl-Z, после чего на экран консоли будет выведено сообщение, показанное на рисунке 9.



Цифра в квадратных скобках обозначает номер задания, он необходим для некоторых команд.

Можно удостовериться, что приложение продолжает выполнение, дав команду

`ps -a`



Развернуть приложение можно с помощью команды `fg` (foreground), имеющей следующий синтаксис:

`fg %id`

где `id` – номер задания. Чтобы развернуть обратно `w3m`, необходимо ввести следующую команду:

`fg %1`

Утилита «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.

Дайте в терминале команду «top»

```
Терминал
Файл Правка Вид Терминал Переход Справка
top 18:44:29 up 1:13, 2 users, load average: 0,10, 0,29, 0,34
Tasks: 73 total, 1 running, 72 sleeping, 0 stopped, 0 zombie
%Cpu(s): 5,7 us, 17,3 sy, 0,0 ni, 76,7 id, 0,0 wa, 0,0 hi, 0,3 si, 0,0 st
KiB Mem: 644452 total, 256376 used, 388076 free, 17888 buffers
KiB Swap: 392188 total, 0 used, 392188 free, 172464 cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM   TIME+  COMMAND
 2746 root        19   -1 57892 23m 5424 S  15,0  3,7   6:23.30 Xorg
 2800 root        20    0 76676 4804 3648 S   5,6  0,7   4:31.49 conky
 3228 root        20    0 36320 10m 8688 S   1,3  1,7   0:05.28 xfce4-terminal
 3237 root        20    0 4508 1436 1084 R   1,0  0,2   0:00.36 top
    4 root        20    0    0    0    0 S   0,7  0,0   0:09.30 kworker/0:0
    3 root        20    0    0    0    0 S   0,3  0,0   0:27.86 ksoftirqd/0
    1 root        20    0 2280 792 688 S   0,0  0,1   0:10.27 init
    2 root        20    0    0    0    0 S   0,0  0,0   0:00.10 kthreadd
    5 root        20    0    0    0    0 S   0,0  0,0   0:00.14 kworker/u:0
    6 root        rt    0    0    0    0 S   0,0  0,0   0:01.09 watchdog/0
    7 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 cpuset
    8 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 khelper
    9 root        20    0    0    0    0 S   0,0  0,0   0:00.06 kdevtmpfs
   10 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 netns
   11 root        20    0    0    0    0 S   0,0  0,0   0:00.50 sync_supers
   12 root        20    0    0    0    0 S   0,0  0,0   0:00.01 bdi-default
   13 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 kintegrityd
   14 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 kblockd
   16 root        20    0    0    0    0 S   0,0  0,0   0:00.02 khungtaskd
   17 root        20    0    0    0    0 S   0,0  0,0   0:00.01 kswapd0
   18 root        25    5    0    0    0 S   0,0  0,0   0:00.00 ksm
   19 root        20    0    0    0    0 S   0,0  0,0   0:00.00 fsnotify_mark
   20 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 crypto
   76 root        20    0    0    0    0 S   0,0  0,0   0:00.16 khubd
   77 root        0 -20    0    0    0 S   0,0  0,0   0:00.00 ata_sff
   92 root        20    0    0    0    0 S   0,0  0,0   0:00.02 scsi_eh_0
   93 root        20    0    0    0    0 S   0,0  0,0   0:00.06 scsi_eh_1
   94 root        20    0    0    0    0 S   0,0  0,0   0:00.09 kworker/u:1
   96 root        20    0    0    0    0 S   0,0  0,0   0:00.00 scsi_eh_2
```

Нагрузку на процессор в целом можно посмотреть в строке «%Cpu».

us — Время (%) работы программ пользователя (%) (**user**)

sy — Время (%) работы ядра (**system**)

id — Время (%) простоя процессора (**idle**)

wa — Время (%) простоя по причине ожидания дисковых операций (**wait**). Бывает так, что загрузка процессора 0, а программа «медленно работает» - посмотрите этот индикатор. Возможно, надо в настройках программы задать больше памяти под кэш.

hi — Время (%), затраченное на обработку аппаратных прерываний (работа с аппаратными устройствами) (**hardware interrupt**)

si - Время (%), затраченное на обработку программных прерываний (**software interrupt**)

st — Время (%), в течении которого реальный процессор не был доступен виртуальной машине. В это время процессор работал с основной ОС или с другой виртуальной машиной

Смотрим предыдущую строку. По умолчанию top отображает процессы (tasks). Нажатием кнопки «Н» можно перейти в режим отображения потоков

```
top - 18:46:53 up 1:16, 2 users, load average: 0,12, 0,22, 0,31
Threads: 153 total, 1 running, 152 sleeping, 0 stopped, 0 zombie
```

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

```
Терминал
Файл Правка Вид Терминал Переход Справка
Fields Management for window 1:Def, whose current sort field is %CPU
Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
'd' or <Space> toggles display, 's' sets sort. Use 'q' or <Esc> to end!

* PID      = Process Id          nDRT    = Dirty Pages Count
* USER     = Effective User Name WCHAN   = Sleeping in Function
* PR       = Priority           Flags    = Task Flags <sched.h>
* NI       = Nice Value        CGROUPS  = Control Groups
* VIRT     = Virtual Image (KiB) SUPGIDS  = Supp Groups IDs
* RES      = Resident Size (KiB) SUPGRPS  = Supp Groups Names
* SHR      = Shared Memory (KiB) TGID     = Thread Group Id
* S        = Process Status
* %CPU     = CPU Usage
* %MEM     = Memory Usage (RES)
* TIME+    = CPU Time, hundredths
* COMMAND  = Command Name/Line
PPID      = Parent Process pid
UID       = Effective User Id
RUID      = Real User Id
RUSER     = Real User Name
SUID      = Saved User Id
SUSER     = Saved User Name
GID       = Group Id
GROUP     = Group Name
PGRP      = Process Group Id
TTY       = Controlling Tty
TPGID     = Tty Process Grp Id
SID       = Session Id
nTH       = Number of Threads
P         = Last Used Cpu (SMP)
TIME      = CPU Time
SWAP      = Swapped Size (KiB)
CODE      = Code Size (KiB)
DATA      = Data+Stack (KiB)
nMaj      = Major Page Faults
nMin      = Minor Page Faults
```

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

1. Клавишей со стрелкой «вниз» довести курсор до записи nTH.
2. Нажать клавишу со стрелкой «вправо».
3. Нажимая клавишу со стрелкой «вверх», перетащить запись вверх, расположив её после NI.
4. Нажать Enter.
5. Нажать пробел или клавишу «d». После выполнения этого шага nTH должно быть выделено жирным белым шрифтом, а сбоку должна появиться звёздочка.
6. Прodelать аналогичные операции для записи TGID (находится в самом конце списка). После выполнения этих шагов файл настроек должен выглядеть так, как показано на рисунке:

```

Fields Management for window 1:Def, whose current sort field is %CPU
Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
'd' or <Space> toggles display, 's' sets sort. Use 'q' or <Esc> to end!

* PID      = Process Id          nMin      = Minor Page Faults
* USER     = Effective User Name nDRT     = Dirty Pages Count
* PR       = Priority            WCHAN    = Sleeping in Function
* NI       = Nice Value         Flags     = Task Flags <sched.h>
* nTH      = Number of Threads  CGROUPS   = Control Groups
* TGID    = Thread Group Id     SUPGIDS   = Supp Groups IDs
* VIRT     = Virtual Image (KiB) SUPGRPS   = Supp Groups Names
* RES      = Resident Size (KiB)
* SHR      = Shared Memory (KiB)
* S        = Process Status
* %CPU     = CPU Usage
* %MEM     = Memory Usage (RES)
* TIME+    = CPU Time, hundredths
* COMMAND = Command Name/Line

```

7. Закрыть настройки (клавишей «q» или Esc).
8. Убедиться, что top работает в режиме отображения потоков (во второй строке должно быть написано «Threads», если там написано «Tasks», нажать клавишу «Н»). На рисунке показан результат правильного выполнения данной инструкции.

```

top - 19:06:47 up 1:35, 2 users, load average: 0,09, 0,27, 0,26
Threads: 154 total, 1 running, 153 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3,9 us, 18,9 sy, 0,0 ni, 77,2 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem: 644452 total, 257128 used, 387324 free, 18100 buffers
KiB Swap: 392188 total, 0 used, 392188 free, 172600 cached

```

PID	USER	PR	NI	nTH	TGID	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2746	root	19	-1	1	2746	58588	23m	5552	S	12,3	3,8	7:53.74	Xorg
2808	root	20	0	8	2800	76676	4812	3648	S	3,6	0,7	2:45.21	conky
3237	root	20	0	1	3237	4624	1512	1096	R	2,6	0,2	0:07.97	top
3228	root	20	0	2	3228	36704	11m	8964	S	1,9	1,8	0:11.45	xfce4-terminal
2800	root	20	0	8	2800	76676	4812	3648	S	1,0	0,7	2:20.18	conky
3241	root	20	0	1	3241	0	0	0	S	0,6	0,0	0:00.50	kworker/0:2
3	root	20	0	1	3	0	0	0	S	0,3	0,0	0:33.84	ksoftirqd/0
2806	root	20	0	8	2800	76676	4812	3648	S	0,3	0,7	0:08.92	conky
2807	root	20	0	8	2800	76676	4812	3648	S	0,3	0,7	0:04.89	conky
1	root	20	0	1	1	2280	792	688	S	0,0	0,1	0:11.12	init
1	root	20	0	1	1	2280	/92	688	S	0,0	0,1	0:11.12	init
2	root	20	0	1	2	0	0	0	S	0,0	0,0	0:00.10	kthreadd
4	root	20	0	1	4	0	0	0	S	0,0	0,0	0:17.38	kworker/0:0
5	root	20	0	1	5	0	0	0	S	0,0	0,0	0:00.14	kworker/u:0
6	root	rt	0	1	6	0	0	0	S	0,0	0,0	0:01.53	watchdog/0
7	root	0	-20	1	7	0	0	0	S	0,0	0,0	0:00.00	cpuset
8	root	0	-20	1	8	0	0	0	S	0,0	0,0	0:00.00	khelper
9	root	20	0	1	9	0	0	0	S	0,0	0,0	0:00.06	kdevtmpfs
10	root	0	-20	1	10	0	0	0	S	0,0	0,0	0:00.00	netns
11	root	20	0	1	11	0	0	0	S	0,0	0,0	0:00.69	sync_supers
12	root	20	0	1	12	0	0	0	S	0,0	0,0	0:00.02	bdi-default
13	root	0	-20	1	13	0	0	0	S	0,0	0,0	0:00.00	kintegrityd
14	root	0	-20	1	14	0	0	0	S	0,0	0,0	0:00.00	kblockd
16	root	20	0	1	16	0	0	0	S	0,0	0,0	0:00.03	khungtaskd
17	root	20	0	1	17	0	0	0	S	0,0	0,0	0:00.01	kswapd0
18	root	25	5	1	18	0	0	0	S	0,0	0,0	0:00.00	ksmd
19	root	20	0	1	19	0	0	0	S	0,0	0,0	0:00.00	fsnotify_mark
20	root	0	-20	1	20	0	0	0	S	0,0	0,0	0:00.00	crypto
76	root	20	0	1	76	0	0	0	S	0,0	0,0	0:00.16	khudb

Для выхода из утилиты top нажмите клавишу «q».

Утилиты «ps» и «kill»

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

Дайте команду «w3m www.yandex.ru» и нажмите Ctrl+Z, чтобы перевести процесс в фоновый режим.

Дайте команду «w3m ya.ru» и нажмите Ctrl+Z, чтобы перевести процесс в фоновый режим.

```
[user@host-15 ~]$ w3m www.yandex.ru
[1]+  Stopped                  w3m www.yandex.ru
[user@host-15 ~]$ w3m ya.ru
[2]+  Stopped                  w3m ya.ru
```

Предположим, мы хотим прервать процесс текстового браузера в котором открыт «www.yandex.ru». Команда killall, которая прерывает процесс с заданным именем, 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 можно посмотреть с помощью команды «top», но в данном случае удобнее воспользоваться командой «ps» (Process Status). 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.

```
root@debian:~# ps
  PID TTY          TIME CMD
  760 pts/0        00:00:00 bash
  766 pts/0        00:00:00 w3m
  774 pts/0        00:00:00 w3m
  789 pts/0        00:00:00 ps
```

Для нашей задачи информации недостаточно.

Выполним команду «**ps a**»

```
[user@host-15 ~]$ ps a
  PID TTY          STAT TIME  COMMAND
 1754 tty2      Ss+    0:00 /sbin/mingetty tty2
 1755 tty3      Ss+    0:00 /sbin/mingetty tty3
 1756 tty4      Ss+    0:00 /sbin/mingetty tty4
 1757 tty5      Ss+    0:00 /sbin/mingetty tty5
 1758 tty6      Ss+    0:00 /sbin/mingetty tty6
 1759 tty1      Ssl+   0:25 X -nolisten tcp -auth /var/run/sddm/{4834024c-069f-4d86-921b-995d50b40e9f} -background none
 2139 pts/0      Ss     0:00 /bin/bash
 2172 pts/0      T       0:00 w3m www.yandex.ru
 2177 pts/0      T       0:00 w3m ya.ru
 2198 pts/0      R+     0:00 ps a
```

Теперь видно, что страница «www.yandex.ru» открыта в процессе PID=2172

Попробуем его «убить» командой «kill 2172», и снова дадим команду «ps a»

```
[user@host-15 ~]$ ps a
  PID TTY          STAT TIME  COMMAND
 1754 tty2      Ss+    0:00 /sbin/mingetty tty2
 1755 tty3      Ss+    0:00 /sbin/mingetty tty3
 1756 tty4      Ss+    0:00 /sbin/mingetty tty4
 1757 tty5      Ss+    0:00 /sbin/mingetty tty5
 1758 tty6      Ss+    0:00 /sbin/mingetty tty6
 1759 tty1      Ssl+   0:25 X -nolisten tcp -auth /var/run/sddm/{4834024c-069f-4d86-921b-995d50b40e9f} -background none
 2139 pts/0      Ss     0:00 /bin/bash
 2172 pts/0      T       0:00 w3m www.yandex.ru
 2177 pts/0      T       0:00 w3m ya.ru
 2198 pts/0      R+     0:00 ps a
```

Процесс «2172» всё ещё жив. Дело в том, что команда kill не убивает процесс, она посылает ему сигнал «SIGTERM», но w3m не просто переведён в фоновый режим, а приостановлен (STAT = T) и на сигнал не реагирует. Не прореагирует на этот сигнал и зависший процесс. Тем не менее, сначала следует попробовать завершить процесс «по хорошему». Web сервер, например, получив этот сигнал прекратит принимать новые запросы, но завершит выполнять уже поступившие.

Если процесс не завершается, как в нашем случае, будем действовать жёстко.

Дадим команду «kill -9 774»

Сигнал SIGKILL(9) прерывает любой процесс кроме «Зомби» процессов. Но помните, дочерние процессы останутся в памяти, временные файлы не удалены, сетевые соединения не закрыты. Подождите секунд 30 после обычного kill, только после этого добавляйте.

Чтобы убить «Зомби» процесс надо убить его родителя.

В документации на конкретные программы можно найти полезные сигналы, которые они обрабатывают. Например, сигнал 1(HUP) просит inetd перезагрузить файл конфигурации.

Вернёмся к выводу команды «ps a»

```
root@debian:~# ps a
PID TTY      STAT   TIME COMMAND
 464 tty1      Ss      0:00 /bin/login --
 690 tty1      S        0:00 -bash
 697 tty1      S+       0:00 /bin/sh /usr/bin/startx
 719 tty1      S+       0:00 xinit /etc/X11/xinit/xinitrc -- /etc/X11/xinit/xse
 720 tty1      S<       0:04 /usr/bin/X -nolisten tcp :0 vt1 -auth /tmp/servera
 725 tty1      S        0:00 /usr/bin/openbox --startup /usr/lib/i386-linux-gnu
 745 tty1      S        0:00 /usr/bin/dbus-launch --exit-with-session x-session
 749 tty1      S        0:00 /bin/sh /usr/lib/i386-linux-gnu/openbox-autostart
 754 tty1      S        0:00 sh /etc/xdg/openbox/autostart
 755 tty1      S        0:00 idesk
 756 tty1      Sl       0:01 xfce4-terminal
 759 tty1      S        0:00 gnome-pty-helper
 760 pts/0     Ss      0:00 bash
 766 pts/0     T        0:00 w3m www.google.ru
 774 pts/0     T        0:00 w3m www.yandex.ru
 802 pts/0     R+      0:00 ps a
```

В столбце STAT используются следующие обозначения (и их комбинации):

- R: процесс активен.
- S: процесс ожидает (спит менее 20 секунд).
- I: процесс бездействует (спит более 20 секунд).
- D: процесс ожидает ввода-вывода (или другого недолгого события), непрерываемый.
- Z: зомби.
- T: процесс остановлен.
- W: процесс выгружен на диск (swap).
- < : процесс в приоритетном режиме.
- N: процесс в режиме низкого приоритета

- L: real-time процесс, имеются страницы, заблокированные в памяти.
- s: лидер сессии, если он завершается, то завершаются и все остальные процессы этой сессии. Примером может служить поведение эмулятора терминала в предыдущей ЛР, когда с помощью него была вызвана панель: если завершился родитель-лидер, то завершались и все процессы этой сессии.
- l: многопоточный процесс.
- + : видимый процесс, т.е. находящийся в группе foreground.

Исторически, сложились 3 комплекта параметров команды «ps».

В стиле UNIX	В стиле BSD	«Длинный» стиль	Действие	
-A, -e	ax		Все процессы	
	a		Все процессы, связанные с терминалами	
	x		Все процессы, НЕ связанные с терминалами	
	r		Все процессы в состоянии runn	
-d			Все процессы за исключением лидеров сессий	
-C name			Отобразить процессы с заданным именем	
-p ID		--pid ID	Отобразить процесс с заданным id	
		--ppid ID	Отобразить дочерние процессы по id родителя	
-u	U	--user	Отобразить процессы по имени или id пользователя	
-t			Отобразить процессы по имени терминала	
-N		--deselect	Инверсия выбора	
-f			«Полный» список колонок	
-F			Ещё более «полный» список колонок	
-l	l		«Длинный» список колонок (много узких колонок)	
-o cls	o	--format cls	Задать список колонок (см. таблицу ниже)	
-O cls			Добавить колонки к стандартному списку колонок	
			comm	Имя команды
			args	Команда (с параметрами)
			%cpu	% использования процессора
			%mem	% использования памяти
			prti	Приоритет процесса (планируемый)
			nlwp	Число потоков процесса
			pid	Идентификатор процесса
			ppid	Идентификатор родительского процесса
			pgrp	Идентификатор группы процессов
			sid	Идентификатор сессии = pid процесса

				login)
			tid	Идентификатор потока (Thread ID)
			tty	Имя терминала, из которого запущен процесс
			start	Время запуска (ЧЧ:ММ:СС)
			etime	Время, прошедшее с момента запуска
			time	Суммарное время работы процессора
			majflt	Подгружено страниц виртуальной памяти с диска (первый запуск программы)
			minflt	Подгружено страниц виртуальной памяти из оперативной памяти (из кэша при повторном запуске)
			vsz	Выделено виртуальной памяти всего [Кб]
			rss	Часть памяти, выделенной процессу, находящаяся в ОЗУ [Кб]
			sz	Число страниц процесса в оперативной памяти
			user	Имя пользователя, от имени которого работает процесс
			uid	Идентификатор пользователя
		--sort cls	Задаёт список колонок для сортировки через запятую. Для обратной сортировки надо поставить «-» перед именем колонки	
	f	--forest	Иерархия процессов (отображается псевдографикой)	
-H			Иерархия процессов (отображается сдвигами строк)	

Однобуквенные опции можно объединять, например, ps -ejH или ps axjf

В опциях -pid, --ppid, -t, -u ... можно перечислить несколько идентификаторов или имён через запятую

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» дайте команду «**ps axu | more**»

Для перехода к следующей странице нажмите пробел. Для поиска текста «google» наберите «/google» и нажмите ENTER. Для выхода нажмите q.

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

Помните в языке «C» потоки «stdin» и «stdout». При выполнении в конвейере «stdout» первого процесса сразу перенаправляется в «stdin» второго процесса. Другими словами, вывод команды «ps axu» направляется на вход команды more, которая осуществляет постраничный вывод. При этом, ничего на диск не пишется. Если второй процесс не успевает обрабатывать поступающие данные, небольшой буфер обмена заполнится и первый процесс приостанавливается. Если второй процесс слишком быстро читает, то когда буфер обмена опустеет, приостанавливается работа второго процесса.

Можно выстраивать в конвейер и более 2 команд. Например, первая команда формирует дампы базы данных, вторая его архивирует, а третья пересылает на архивный сервер. Очень удобно. И быстро, ведь ничего на диск не записывается.

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

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

```
root@debian:~# ps axu | grep google
root      778  0.0  0.8  9340  5504 pts/0    T   21:57   0:00 w3m www.google.ru
root      946  0.0  0.3  4560  2348 pts/0    S+  23:37   0:00 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. Например? [a-z]

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

Кроме конвейеров существует ещё несколько способов объединения команд.

Параллельное выполнение команд

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

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

Команда `1 ; 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.

Команда `1 && 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.

Команда `1 || команда 2` — выполняет команду 1 и в случае ошибочного завершения выполняет команду 2. Например, команда «`ping -c 1 -W 3 192.168.99.99 || mail -s 'сервер недоступен' test@mail.ru`» отправит e-mail с текстом 'сервер недоступен', если не отвечает сервер с IP адресом «192.168.99.99». (если у вас установлена и настроена утилита отправки электронной почты mail)

Последовательное выполнение группы команд в фоновом режиме

Выполните команду `(sleep 5 && pstree)&`

Запускается фоновый процесс, в котором, с задержкой 5 секунд выполнится команда `pstree`. В течении этих 5 секунд вы можете выполнять команды в основном процессе.

```
root@debian:~# (sleep 2;pstree)&
[1] 726
root@debian:~# systemd└─acpid
                        └─atd
                        └─auditd─{auditd}
                        └─cron
                        └─dbus-daemon
                        └─dhclient
                        └─exim4
                        └─login─bash─bash─pstree
```

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

Подстановка результатов одной команды в другую

Пусть у нас ежедневно ночью, автоматически запускается процесс архивирования файлов из заданного каталога, и я хочу, чтобы архив создавался каждый день с новым именем, чтобы можно было восстановить файлы, удалённые по ошибке несколько дней назад. Для этого достаточно подставить текущую дату в имя файла.

Начнём с команды «date». Эта команда 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.

Команда «tar -cvzf ~/archive\$(date '+%Y-%m-%d').tar ~/folder/*» подставит результаты работы команды «date» на место, \$(...) и выполнит получившуюся команду.

Перенаправление входных и выходных потоков

В стандартной библиотеке языка C 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.

При организации рассмотренного выше конвейера, stdout первой команды перенаправляется в stdin второй команды. При этом, ничего на диск не записывается, если вторая команда обрабатывает данные быстрее, чем первая их поставляет, то её процесс переводится в режим ожидания, пока первый процесс не запишет в выходной поток хотя бы один байт. И наоборот, если второй поток не успевает обрабатывать данные, то как только заполнится небольшой буфер обмена, первый поток перейдёт в режим ожидания, пока буфер не освободится.

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»

```
root@debian:~# date>xx.txt
root@debian:~# cat xx.txt
Пт окт 21 13:42:41 MSK 2016
```

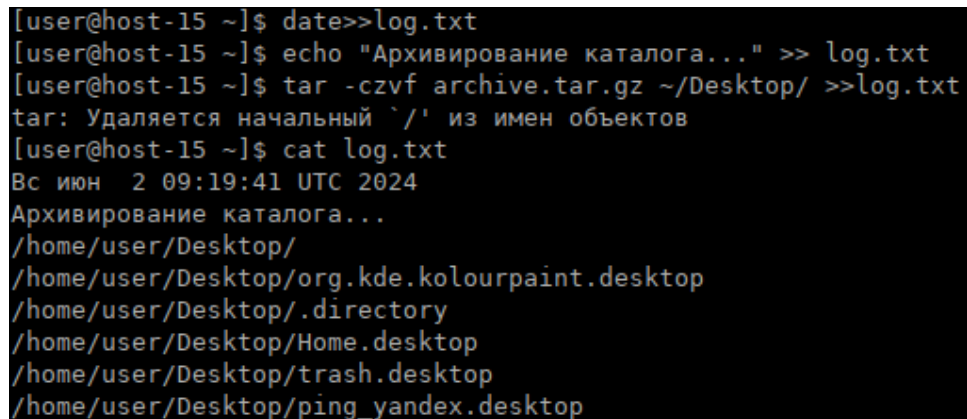
При повторном выполнении команды файл заменяется. Если мы хотим дописать результаты команды в конец существующего файла, надо задать в команде «>>file».

Например, так можно организовать ведение файла протокола:

```
date >> log.txt
```

```
echo "Архивирование каталога...">>log.txt
```

```
tar -czvf archive.tar.gz ~/.Desktop/ >> log.txt
```



```
[user@host-15 ~]$ date>>log.txt
[user@host-15 ~]$ echo "Архивирование каталога..." >> log.txt
[user@host-15 ~]$ tar -czvf archive.tar.gz ~/Desktop/ >>log.txt
tar: Удаляется начальный '/' из имен объектов
[user@host-15 ~]$ cat log.txt
Вс июн  2 09:19:41 UTC 2024
Архивирование каталога...
/home/user/Desktop/
/home/user/Desktop/org.kde.kolourpaint.desktop
/home/user/Desktop/.directory
/home/user/Desktop/Home.desktop
/home/user/Desktop/trash.desktop
/home/user/Desktop/ping_yandex.desktop
```

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 .Desktop (эту команду подробно будем рассматривать в следующей лабораторной работе)

Обратите внимание, что несмотря на то, что вывод команды **tar** был перенаправлен в файл, на экран было выведено предупреждение «**tar: Удаляется ...**»

Дело в том, что этот текст выводится не в stdout, а в stderr, а он не перенаправлен.

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

```
tar -czvf archive.tar.gz ~/.Desktop/ >>log.txt 2>>err.txt
```

А можно перенаправить stderr в stdout, и тогда у вас будет общий протокол. Для этого надо указав в команде «2>&1». Например:

```
tar -czvf archive.tar.gz ~/.Desktop/ >>log.txt 2>&1
```

Если вы пишете программу и вызываете в ней дочерний процесс, обязательно направьте весь вывод этого процесса в файл или обеспечьте чтение из stdout и stderr. Иначе буфер одного из этих потоков заполнится и дочерний процесс остановится. В вы в основном процессе вечно будете ждать его завершения.

А ещё можно подавить вывод сообщений об ошибках, используя «2>&-»

Ограничения на ресурсы процесса

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

```
root@debian:~# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited
pending signals         (-i) 4904
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files              (-n) 65536
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority      (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes      (-u) 4904
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
```

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`.

В разделе «Последовательное выполнение группы команд в фоновом режиме» мы пользовались для этого круглыми скобками. Попробуем альтернативный способ.

Команда «`bash -c 'команда'`» запускает дочерний интерпретатор команд и выполняет в нём заданную команду. Обратите внимание, я заключил команду в прямые одиночные кавычки. В отличие 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` с ограничением по времени выполнения: `«bash -c 'ulimit -t 2 && apt-get update'»`

```
root@debian:~# bash -c 'ulimit -t 2 && apt-get update'
Игн http://ftp.debian.org jessie InRelease
В кэше http://security.debian.org jessie/updates InRelease
В кэше http://ftp.debian.org jessie Release.gpg
В кэше http://ftp.debian.org jessie Release
В кэше http://security.debian.org jessie/updates/main Sources
В кэше http://ftp.debian.org jessie/main Sources
В кэше http://security.debian.org jessie/updates/main i386 Packages
В кэше http://ftp.debian.org jessie/main i386 Packages
В кэше http://security.debian.org jessie/updates/main Translation-en
В кэше http://ftp.debian.org jessie/main Translation-ru
В кэше http://ftp.debian.org jessie/main Translation-en
E: Method copy has died unexpectedly!
E: Порождённый процесс copy получил сигнал 9.
```

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»`.

Обратите внимание, что `«apt-get update»` прервалась гораздо позже, чем через 2 секунды. Дело в том, что `«-t»` накладывает ограничение на чистое время работы процессора. В это время не входит время дисковых операций. А если процесс перешёл в состояние ожидания (например, ждёт ввода данных от пользователя или соединения по сети), процессорное время вообще не расходуется и прерывания не произойдёт.

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

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

Запуск процесса по расписанию

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.

Рассмотрим конфигурационный файл.

Каждая строка (кроме комментариев и пустых строк) определяет день, время и выполняемую команду, которые задаются через пробел в следующей последовательности:

Последовательность	Минуты	Часы	Дни месяца	Месяцы	Дни недели	Команда
--------------------	--------	------	------------	--------	------------	---------

Диапазон	0-59,*	0-23,*	1-31,*	1-12,*	1-7,*	
Примеры значений	0 * */10	0 */12	*	*	* 1-5 6,7	
Пример1	*/10	*	*	*	*	date>>log1.txt

Значения могут задаваться в виде:

- Диапазона. Например, 1-5 Означает множество чисел {1,2,3,4,5}
- Перечисления. Например, {6,7} Означает множество чисел {6,7}
- Кратного. Например для часа */2 означает раз в 2 часа
- Любое значение (*).

ВНИМАНИЕ! После последней строки должен быть перевод строки, иначе cron не будет выполнять эту команду

Примеры:

Выполнять по выходным в 01:00:

0 1 * * 6,7 Команда

Выполнять только по будням в 01:30

30 1 * * 1-5 Команда

Выполнять каждый час в 0 минут

0 * * * * Команда

Выполнять каждые 10 минут

*/10 * * * * Команда

Другие команды управления процессами.

pidof – утилита, выдающая pid процесса по его имени.

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»).

shutdown -h now — Рассылает сигналы завершения всем запущенным процессам (сначала SIGTERM, затем SIGKILL), завершает запись кэша на диск, подготавливает внешние устройства к выключению (например, головка жёсткого диска смещается в

парковочную позицию, безопасную для транспортировки) и только после этого отключает питание.

`shutdown -h 21:00` - завершить работу ОС и отключить питания в 21:00

`halt` — немедленно отключить питание, не пытаясь корректно завершить работу.

`reboot` — перезагрузить компьютер

Горячие комбинации клавиш

Ctrl+Alt+F1 — переключиться в текстовый режим (терминал 1)

Ctrl+Alt+F2 — переключиться в текстовый режим (терминал 2)

Ctrl+Alt+F7 — вернуться в графический режим

Alt+PrtScr+ E — послать всем процессам SIGTERM

Alt+PrtScr+ I — послать всем процессам SIGKILL

Alt+PrtScr+ S — завершить запись кэша на диск

Alt+PrtScr+ U — перемонтировать файловые системы в режим readonly

Alt+PrtScr+ B — перезагрузить компьютер

Ctrl+C — прервать текущий процесса

Ctrl+S — Приостановить вывод в терминала. Если Вы не можете ввести команду, возможно вы случайно нажали эту комбинацию клавиш.

Ctrl+Q — Продолжить вывод в терминала. приостановить вывод в терминала

Ctrl+Z — Перевести текущий процесс в фоновый режим.

Службы Linux

Кроме программ, которые запускает пользователь, есть службы. Службы запускаются автоматически, при загрузке ОС и следят за подключением устройств, позволяют процессам взаимодействовать с оборудованием. Как службы регистрируются web-серверы и СУБД.

В ранних версиях linux для запуска использовалась система управления службами «System V init». В ней для запуска и остановки служб использовались скрипты инициализации, содержащие специальные псевдо-комментарии. Эти скрипты при установке записываются в каталог «/etc/init.d». Некоторые службы используют этот метод регистрации по сей день.

Для запуска службы надо дать команду «/etc/init.d СЛУЖБА start»

Для остановки службы надо дать команду «/etc/init.d СЛУЖБА stop»

Для перезапуска службы надо дать команду «/etc/init.d СЛУЖБА restart»

В настоящее время используется система управления службами «Systemd». В ней службы регистрируются с помощью файлов описания служб (имеют расширение .service). Это текстовые файлы, содержащие команды запуска и остановки службы, перечень служб,

которые надо запустить до запуска данного сервиса. Файлы описания служб лежат в каталоге «/etc/systemd/system/» и его подкаталогах.

Дайте команду «systemctl --version». Если команда не найдена — значит в Вашей ОС используется устаревшая система управления службами «System V Init». Здесь и далее в лабораторных работах используйте выше приведённые команды управления службами. Остаток этого раздела **выполнять не надо**. Прочтите для ознакомления.

Рассмотрим работу со службой на примере службы времени «ntp», которая автоматически синхронизирует часы вашего компьютера с сервером службы времени.

Установим службу командой:

```
apt-get install ntp
```

При установке служба автоматически регистрируется и запускается

Посмотрим, успешно ли прошёл запуск:

```
systemctl status ntp
```

```
● ntp.service - LSB: Start NTP daemon
   Loaded: loaded (/etc/init.d/ntp; generated; vendor preset: enabled)
   Active: active (running) since Sun 2019-08-18 15:35:45 MSK; 16min ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/ntp.service
           └─13030 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 111:118

авг 18 15:35:46 debian95 ntpd[13030]: Soliciting pool server 89.221.207.113
авг 18 15:35:47 debian95 ntpd[13030]: Soliciting pool server 185.22.60.71
авг 18 15:35:47 debian95 ntpd[13030]: Soliciting pool server 185.209.85.222
авг 18 15:35:48 debian95 ntpd[13030]: Soliciting pool server 192.36.143.130
авг 18 15:35:48 debian95 ntpd[13030]: Soliciting pool server 85.21.78.91
авг 18 15:35:48 debian95 ntpd[13030]: Soliciting pool server 91.207.136.50
авг 18 15:35:49 debian95 ntpd[13030]: Soliciting pool server 85.21.78.23
авг 18 15:35:49 debian95 ntpd[13030]: Soliciting pool server 88.212.196.95
авг 18 15:35:54 debian95 ntpd[13030]: Soliciting pool server 195.91.239.8
авг 18 15:35:55 debian95 ntpd[13030]: Soliciting pool server 194.190.168.1
```

Зелёная надпись «active (running)» означает, что служба стартовала успешно.

В случае проблем с запуском, надпись была бы красная. В нижней части выводятся последние сообщения протокола. Обычно причина ошибки видна сразу. Но иногда надо первопричину ошибки запуска сервиса надо искать в более ранних сообщениях.

Полностью системный журнал службы «ntp» можно просмотреть командой:

```
journalctl -u ntp
```

```
-- Logs begin at Sun 2019-08-18 14:48:00 MSK, end at Sun 2019-08-18 16:03:54 MSK. --
авг 18 15:35:44 debian95 systemd[1]: Starting LSB: Start NTP daemon...
авг 18 15:35:45 debian95 ntpd[13027]: ntpd 4.2.8p10@1.3728-o Sun Feb 25 21:15:29 UTC 2018 (1): Starting
авг 18 15:35:45 debian95 ntpd[13027]: Command line: /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 111:118
авг 18 15:35:45 debian95 ntpd[13020]: Starting NTP server: ntpd.
авг 18 15:35:45 debian95 systemd[1]: Started LSB: Start NTP daemon.
авг 18 15:35:45 debian95 ntpd[13030]: proto: precision = 1.043 usec (-20)
авг 18 15:35:45 debian95 ntpd[13030]: Listen and drop on 0 v6wildcard [::]:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen and drop on 1 v4wildcard 0.0.0.0:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen normally on 2 lo 127.0.0.1:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen normally on 3 enp0s3 10.0.2.15:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen normally on 4 enp0s8 192.168.56.101:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen normally on 5 lo [::1]:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen normally on 6 enp0s3 [fe80::a00:27ff:fe3c:461d%2]:123
авг 18 15:35:45 debian95 ntpd[13030]: Listen normally on 7 enp0s8 [fe80::a00:27ff:fec8:1463%3]:123
авг 18 15:35:45 debian95 ntpd[13030]: Listening on routing socket on fd #24 for interface updates
авг 18 15:35:46 debian95 ntpd[13030]: Soliciting pool server 89.221.207.113
авг 18 15:35:47 debian95 ntpd[13030]: Soliciting pool server 185.22.60.71
```

Для настройки, откроем файл конфигурации `/etc/ntp.conf`

Если в вашей организации имеется собственный сервер службы времени, надо закомментировать строки:

```
pool 0.debian.pool.ntp.org iburst
pool 1.debian.pool.ntp.org iburst
...
```

И добавить аналогичную строку с адресом вашего сервера.

Настройками в этом же файле можно сделать ваш компьютер сервером службы времени.

После изменения файла конфигурации надо рестартовать сервис командой

```
systemctl restart ntp
```

Приведу другие команды управления службами:

`systemctl list-units --type service --all` — выводит список служб (постранично, листание пробелом, для выхода нажмите клавишу «q»).

`systemctl disable СЛУЖБА` — удалить службу из автозагрузки. Если вам редко нужна некоторая служба, можно отключить её автозагрузку и ваша ОС будет загружаться быстрее. А когда служба понадобится, надо дать команду:

`systemctl start СЛУЖБА` — запустить службу linux

`systemctl stop СЛУЖБА` — остановить службу linux

`systemctl enable СЛУЖБА` — добавить службу в автозагрузку

`systemctl status СЛУЖБА` — посмотреть состояние службы

`journalctl -u СЛУЖБА` — посмотреть протокол службы

`systemctl restart СЛУЖБА` — перезапустить службу — используется после изменения конфигурационных файлов или «зависания» службы.

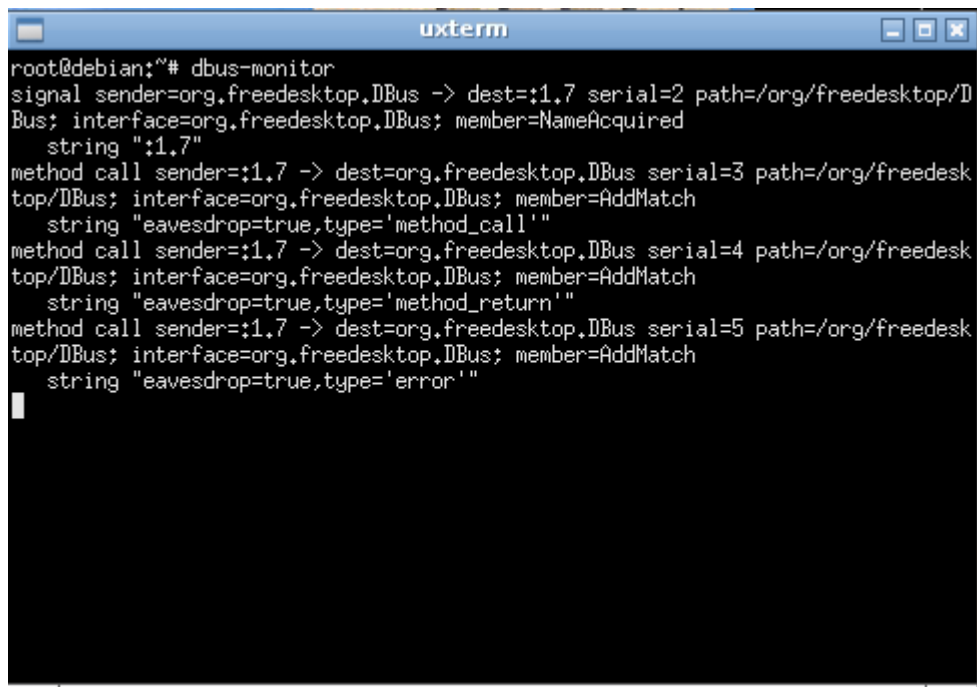
`systemctl reload СЛУЖБА` — перечитать параметры из конфигурационных файлов (не все службы поддерживают эту команду)

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.

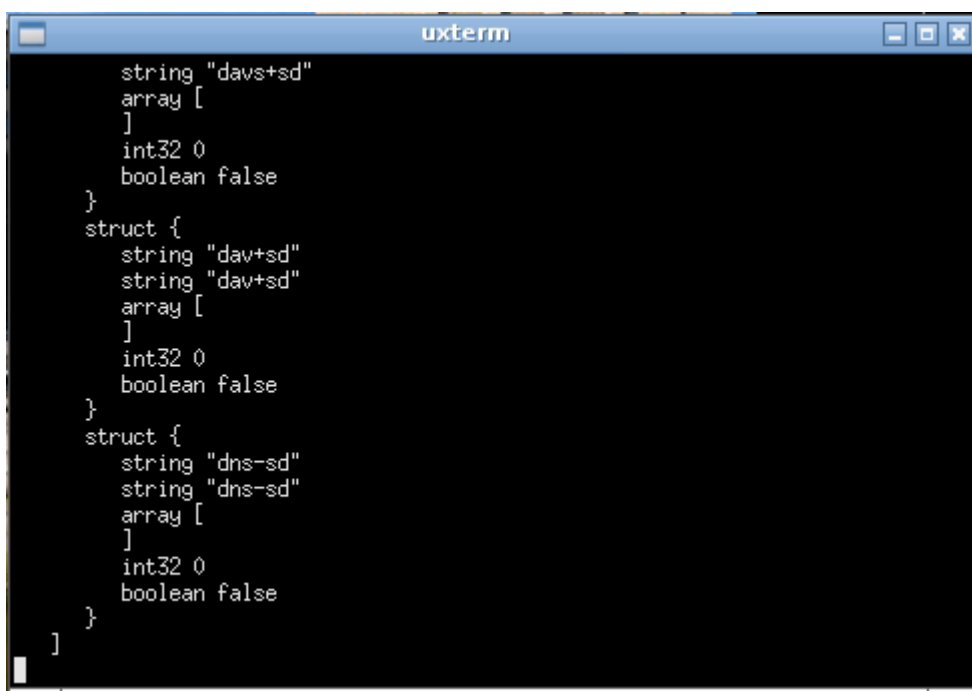
Дайте команду «dbus-monitor».



```
root@debian:~# dbus-monitor
signal sender=org.freedesktop.DBus -> dest=:1.7 serial=2 path=/org/freedesktop/DBus; interface=org.freedesktop.DBus; member=NameAcquired
  string ":1.7"
method call sender=:1.7 -> dest=org.freedesktop.DBus serial=3 path=/org/freedesktop/DBus; interface=org.freedesktop.DBus; member=AddMatch
  string "eavesdrop=true,type='method_call'"
method call sender=:1.7 -> dest=org.freedesktop.DBus serial=4 path=/org/freedesktop/DBus; interface=org.freedesktop.DBus; member=AddMatch
  string "eavesdrop=true,type='method_return'"
method call sender=:1.7 -> dest=org.freedesktop.DBus serial=5 path=/org/freedesktop/DBus; interface=org.freedesktop.DBus; member=AddMatch
  string "eavesdrop=true,type='error'"

```

Не закрывая окно эмулятора терминала с запущенным dbus-monitor, запустите Leafpad через панель xfce. После этого в окне dbus-monitor произойдут изменения, показанные на следующем рисунке.



```

    string "davs+sd"
    array [
    ]
    int32 0
    boolean false
  }
  struct {
    string "dav+sd"
    string "dav+sd"
    array [
    ]
    int32 0
    boolean false
  }
  struct {
    string "dns-sd"
    string "dns-sd"
    array [
    ]
    int32 0
    boolean false
  }
]

```

На следующем рисунке показано начало этого сообщения.

```
signal sender=org.freedesktop.DBus -> dest=(null destination) serial=7 path=/org/
/freedesktop/DBus; interface=org.freedesktop.DBus; member=NameOwnerChanged
  string ":1.12"
  string ""
  string ":1.12"
method call sender=:1.12 -> dest=org.freedesktop.DBus serial=1 path=/org/freedes
ktop/DBus; interface=org.freedesktop.DBus; member=Hello
method call sender=:1.12 -> dest=org.gtk.vfs.Daemon serial=2 path=/org/gtk/vfs/m
ounttracker; interface=org.gtk.vfs.MountTracker; member=listMountableInfo
method return sender=:1.3 -> dest=:1.12 reply_serial=2
```

Чтобы прервать «dbus-monitor» нажмите «Ctrl+C»

Можно отправлять сообщения по D-Bus прямо из консоли, для этого служат команды dbus-send и qdbus. Например, я использовал «qdbus», для копирования содержимого файла в буфер обмена (не открывая его).

Выводы.

Linux имеет гибкие механизмы управления процессами и потоками. Процессы могут посылать друг другу сигналы или сообщения, обмениваться данными через файлы или каналы. Суперпользователь может изменять приоритеты процессов. Также Linux имеет стандартные гибкие средства мониторинга ресурсов.

Порядок выполнения работы.

1. Ознакомиться с теоретическими сведениями.
2. Изучить механизмы управления процессами и потоками.
3. Выполнить следующие задания из файла «zadanie Process» (последней версии). В имена используемых в командах создаваемых файлов должна быть включена фамилия студента (латиницей) во избежание копирования отчётов.

Отчет должен включать:

- Название работы и ее цель;
- Действия, выполняемые по данному руководству и результаты их работы.
- Задания из файла «zadanie Process», команды для их выполнения и результаты их работы.

Литература

[https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_\(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0\)](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) Процесс (Википедия)

<https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F> - Поток выполнения (Википедия)

Управление процессами в Linux — OpenNET
(http://www.opennet.ru/docs/RUS/lrx_process/process2.html)

Процессы и потоки ос Linux – StudFiles (<https://studfiles.net/preview/5357373/page:3/>)
<https://habrahabr.ru/company/infobox/blog/241237/> Шпаргалка по управлению сервисами CentOS 7 с systemd / Хабрахабр

<https://losst.ru/avtozagruzka-linux> — Автозагрузка в Linux