

Лабораторная работа №8

Процессы. Работа с процессами

Введение

В этой лабораторной вы познакомитесь с особенностями жизненного цикла процессов в Linux, их рождением, жизнью и смертью. Вы узнаете, что такое сигналы, для чего они нужны и как их использовать. Будет рассмотрена работа планировщика задач CFS и способы приоритизации процессов в Linux. Мы также научимся продвинутым приемам извлечения информации о процессах и управления ими.

Процессы и потоки в Linux

Основным назначением операционной системы, как мы знаем, является создание условий для работы прикладных приложений. Однако установленные в системе приложения — это всего лишь файлы, которые просто лежат на диске. Для того чтобы приложение «заработало», создается так называемый процесс.

При запуске программы операционная система выделяет экземпляру приложения независимое адресное пространство памяти, назначает уникальный идентификатор (Process Identifier, PID) и вносит запись в таблицу процессов. С этого момента планировщик задач (scheduler) начинает выделять процессу кванты процессорного времени, и программа оживает.

Процессы Linux и Windows во многом очень похожи, но есть и ряд отличий, понимание которых упростит работу с системой. Давайте разбираться.

Процессы в Linux и Windows

Дерево процессов в Linux

Приложение начинает работать, как только информация об экземпляре приложения заносится в таблицу процессов и планировщик начинает выделять ему кванты процессорного времени.

Для просмотра таблицы процессов в CMD мы использовали команду `tasklist`, в PowerShell командлет `Get-Process`, а в Linux для этого предназначена утилита `ps`. И начнем мы с наиболее часто используемых ключей **aux**:

```
sa@astra:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   3.4   0.5 103020 11500 ?        Ss   08:34   0:01 /sbin/init
```

```
root      2  0.0  0.0      0   0 ?      S   08:34  0:00 [kthreadd]
root      3  0.0  0.0      0   0 ?      I<  08:34  0:00 [rcu_gp]
...
```

Назначение использованных нами ключей следующее:

- **a** — отобразить процессы всех пользователей;
- **u** — показать, кто является владельцем процесса;
- **x** — показать процессы, у которых нет управляющего терминала, например, службы.

Внимание

Для обеспечения обратной совместимости Bash-скриптов утилита `ps` поддерживает ключи, используемые разными операционными системами: BSD (без дефисов), POSIX (с дефисами) и GNU (с двумя дефисами и длинными ключами). Поэтому короткие ключи, совпадая по написанию, могут иметь совершенно разный смысл, например, команды `ps -a` и `ps a` покажут совсем не одно и то же.

Еще интереснее пример с командой `ps -aux`, когда перед группой ключей ставится символ дефиса. По правилам POSIX эта команда должна вывести все процессы для пользователя с именем «x», но т.к. такого пользователя не существует в системе, утилита понимает, что мы имели в виду «aux» без дефиса, и показывает нам этот результат.

Давайте с помощью ключей `-ef` выведем ту же информацию, но с идентификатором родителя и поговорим об иерархии процессов в Linux:

```
sa@astra:~$ ps -ef
UID      PID  PPID  C  STIME TTY          TIME CMD
root         1      0  0  08:34 ?          00:00:01 /sbin/init
root         2      0  0  08:34 ?          00:00:00 [kthreadd]
root         3      2  0  08:34 ?          00:00:00 [rcu_gp]
root         4      2  0  08:34 ?          00:00:00 [rcu_par_gp]
root         6      2  0  08:34 ?          00:00:00 [kworker/0:0H-events_highpri]
root         7      2  0  08:34 ?          00:00:00 [kworker/0:1-events]
root         8      2  0  08:34 ?          00:00:00 [kworker/u2:0-events_unbound]
...
```

Назначение использованных нами ключей следующее:

- ключ `-e` — отобразить процессы всех пользователей (то же самое, что ключ `-a`);
- ключ `-f` — вывести больше доступной информации, например, идентификатор родителя.

```
sa@astra:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.3  0.5 103020 11540 ?        Ss   08:34   0:01 /sbin/init
root         2  0.0  0.0      0      0 ?        S   08:34   0:00 [kthreadd]
root         3  0.0  0.0      0      0 ?        I<  08:34   0:00 [rcu_gp]
...
```

Существенным отличием Windows и Linux является строгая иерархия процессов. В системе Windows дочерние процессы, конечно же, сохраняют идентификатор родителя, но система не следит за целостностью этой информации, и через некоторое время это значение может соответствовать несуществующему или неправильному процессу. Так происходит,

если родительский процесс был закрыт, и этот идентификатор был повторно выдан другому процессу.

В системе Linux, напротив, установлена жесткая иерархия, поэтому описанная выше ситуация невозможна. Иерархический подход позволяет при закрытии терминала отправить всем дочерним процессам сигнал `SIGHUP`, чтобы закрыть их автоматически. Если же процесс был запущен через команду `nohup` и нужно, чтобы он продолжил свою работу, то он станет «осиротевшим» и будет автоматически усыновлен процессом `init` (`systemd`) с `PID=1`.

Проведем эксперимент, который наглядно покажет разницу в поведении систем.

- Если запустить приложение `calc.exe` из терминала `cmd` или `PowerShell`, то после закрытия терминала калькулятор продолжит работу как ни в чем ни бывало, что является вполне ожидаемым поведением.

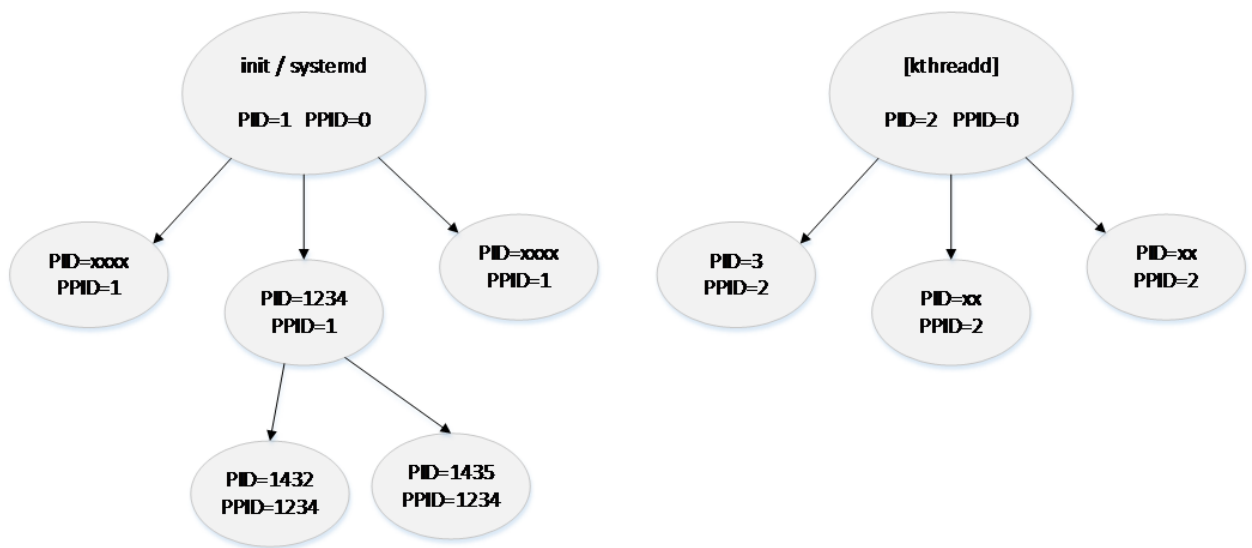
- Если же мы из терминала `Gnome Terminator` запустим калькулятор в фоновом режиме командой `kcalc &` и закроем окно терминала с помощью `ALT + F4`, а не через команду `exit`, то терминал закроется вместе с калькулятором. А это уже ни разу не ожидаемое поведение.

- А теперь повторим то же самое в `fly-term` от `Astra Linux` и увидим, что терминал можно закрывать хоть крестиком, хоть через `exit`, а калькулятор остается на месте, как будто его запустили через `nohup`, что намного привычнее для `Windows`-администраторов.

Используя значение `PPID`, мы можем легко найти все процессы, запущенные из текущей оболочки. При этом мы, конечно, можем поупражняться в составлении трехэтажных регулярных выражений, но проще будет воспользоваться специальной утилитой `pgrep` и системной переменной `$$`, в которой содержится идентификатор текущего процесса. Команда в этом случае будет выглядеть так:

```
sa@astra:~$ ps -f -p$(pgrep --parent=$$)
UID      PID  PPID  C  STIME TTY      TIME CMD
sa        1395  1389  2  10:44 pts/0    00:00:00 kcalc
```

Возвращаясь к дереву процессов, обратим ваше внимание на то, что дерево, на самом деле, не одно (см. рисунок).



Деревья процессов в Linux

Чтобы убедиться в этом на практике, воспользуемся утилитой `pstree` и выведем список всех потомков процесса с PID=0, которые были порождены ядром системы:

```

sa@astra:~$ pstree -p 0
()--kthreadd(2)---acpi_thermal_pm(88)
                  |---ata_sff(76)
                  |---audit_prune_tre(359)
                  |---blkcg_punt_bio(74)
                  ...
...--systemd(1)---NetworkManager(436)---dhclient(690)---{dhclient}(707)
                  |---{NetworkManager}(486)
                  |---{NetworkManager}(492)
                  |---VBoxClient(1184)---VBoxClient(1185)---{VBoxClient}(1186)
                  |---{VBoxClient}(1187)
                  |---{VBoxClient}(1188)
                  |---VBoxClient(1199)---VBoxClient(1200)---{VBoxClient}(1201)
                  ...
  
```

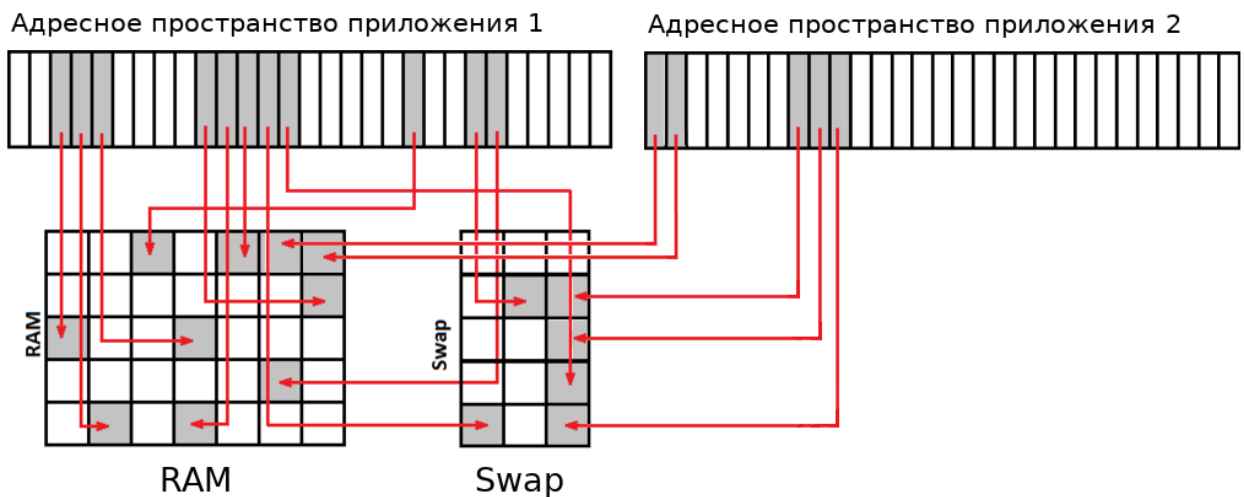
Процесс `[kthreadd]` является диспетчером потоков, а квадратные скобки указывают нам на то, что это процесс самого ядра ОС. Второе дерево потребовалось, потому что для управления потоками нельзя было использовать процесс `systemd`, который работает в пользовательском пространстве.

Процесс `systemd` или `/sbin/init` является процессом подсистемы инициализации и управления службами `systemd`, которая пришла на замену устаревшей подсистеме `init`. Новая реализация обеспечила распараллеливание запуска служб, что позволило существенно ускорить загрузку ОС. В настоящий момент `systemd` полностью вытеснила `init` SysV, но файл `/sbin/init` используется до сих пор и является символической ссылкой на `/lib/systemd/systemd`.

Адресное пространство

Каждому процессу выделяется независимое адресное пространство памяти. Максимальный размер пространства в 64-битных системах в теории может достигать до 2^{64} бит, а на практике для x86 процессоров не превышает 2^{48} бит, т.е. до 256 ТБ (см. вывод команды `cat /proc/cpuinfo` «address sizes ... 48 bits virtual»).

Важно понимать, что размер адресного пространства и объем памяти, потребляемый процессом, — это две большие разницы. Все адресное пространство разбивается на страницы объемом в 4Кб (см. `getconf PAGE_SIZE`). Однако большая часть этих страниц остается пустой, то есть не сопоставлена с реальной памятью компьютера (см. рисунок), поэтому процесс занимает ровно столько памяти, сколько он запросил, или как еще говорят «аллоцировал» (от англ. *allocation* – резервировать).



Отображение страниц адресного пространства приложений на ОЗУ и файл подкачки

Обычно процессы могут аллоцировать весь доступный объем памяти, и ничего настраивать дополнительно не требуется. Выполните команду `ulimit -a`, и вы увидите, что для «virtual memory» установлено значение «unlimited»:

```
sa@astra:~$ 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) 7524
max locked memory       (kbytes, -l) 65536
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
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) 7524
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
```

Однако бывают ситуации, когда мы сталкиваемся, например, с утечкой памяти, и, чтобы дожить до патча, устраняющего проблему, приходится подкручивать настройки. Например, с помощью команды `ulimit -v 512000` мы можем ограничить потребление памяти процессами «на лету», а с помощью строки «* - as 512000» в конфигурационном файле `/etc/security/limits.conf` выставить эти ограничения на постоянной основе.

Добавим еще, что из соображений безопасности адресные пространства процессов независимы друг от друга, поэтому один процесс не может напрямую получить доступ к переменным и структурам данных другого процесса. В тех случаях, когда это действительно требуется, процессы должны обмениваться информацией через механизмы межпроцессного взаимодействия (*от англ. Inter-Process Communication, IPC*).

В Linux основными механизмами IPC являются файлы, каналы (`pipe`, `fifo`), сокет (Unix и TCP/IP), сигналы, разделяемая память и др. В Windows есть аналогичные технологии каналов, TCP/IP сокетов, разделяемой памяти, но при организации межпроцессного взаимодействия в большей степени используются программные интерфейсы, такие как объектная модель компонентов (*от англ. Component Object Model, COM*) или удаленный вызов процедур (*от англ. Remote Procedure Call, RPC*). В том числе по этой причине Linux работает пошустрее.

Порождение процесса

В системе Linux новый процесс создается вызовом ядра `fork()`, во время выполнения которого происходит полное копирование адресного пространства родительского процесса, назначается собственный PID/PPID и обнуляется статистика выполнения в таблицах ОС. В ряде случаев этого бывает достаточно, но если требуется запустить другой программный код (например, выполнить `rs` из командной строки `bash`), то сразу после `fork()` следует еще один системный вызов `execve()` (см. рисунок).

Существует целое семейство вызовов `exec*` (см. справку `man 3 exec`), которые отличаются набором параметров и логикой работы. Например, вызов `execl()` имеет дело со списком, а `execv()` с вектором. Но все они решают одну и ту же задачу, просто немного по-разному.

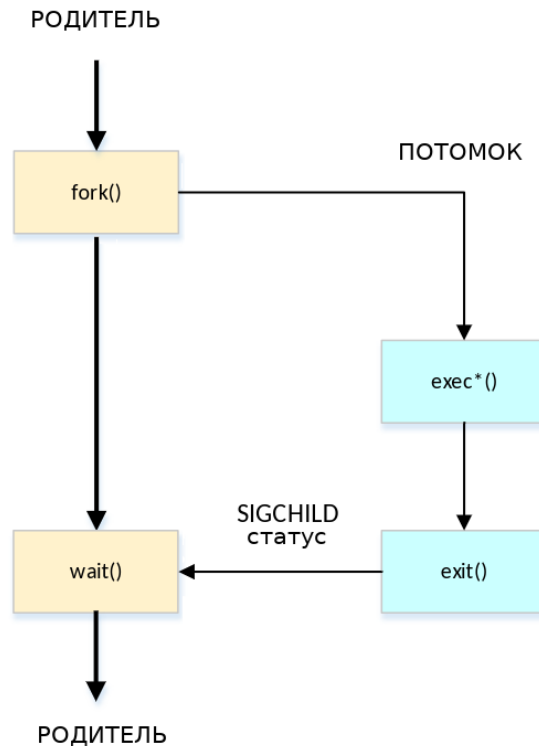


Схема ветвления процессов

Важно понимать, что копирование процесса происходит чрезвычайно быстро, так как копируются не сами данные, а только список страниц памяти, где эти данные хранятся. На все эти страницы устанавливается отметка Copy-on-Write (COW), поэтому, как только одному из процессов требуется внести изменения, для него создается копия соответствующей страницы. В итоге `fork()` оказывается значительно быстрее, чем функция `CreateProcess()` из Win32 API, хотя на современных процессорах это уже не столь важно.

Понятное дело, что системные администраторы не всегда обладают навыками программирования, и подробности о вызовах `fork()` и `execve()` могут показаться излишними. Но мы очень часто сталкиваемся с упоминаниями этих функций в журналах отладки и трассировки, поэтому общие представления о работе этого механизма крайне полезны.

Завершение процесса

Процесс может завершить свою работу штатно, когда программный код полностью выполнил поставленную перед ним задачу, из-за непредвиденной ошибки или аварийно по требованию администратора, например, командой `kill -9 $(pidof kcalc)`.

Вне зависимости от того, как завершился процесс, он освобождает все свои ресурсы и становится пустой записью в таблице процессов, в которой остается только статус завершения, необходимый родительскому процессу для корректной обработки результатов. Для того чтобы родитель поскорее вычитал статус, ему направляется сигнал `SIGCHLD`, но

до тех пор, пока это не произойдет, запись будет оставаться в таблице, поэтому такие процессы называются «зомби» — процесс уже «умер», но продолжает занимать идентификатор.

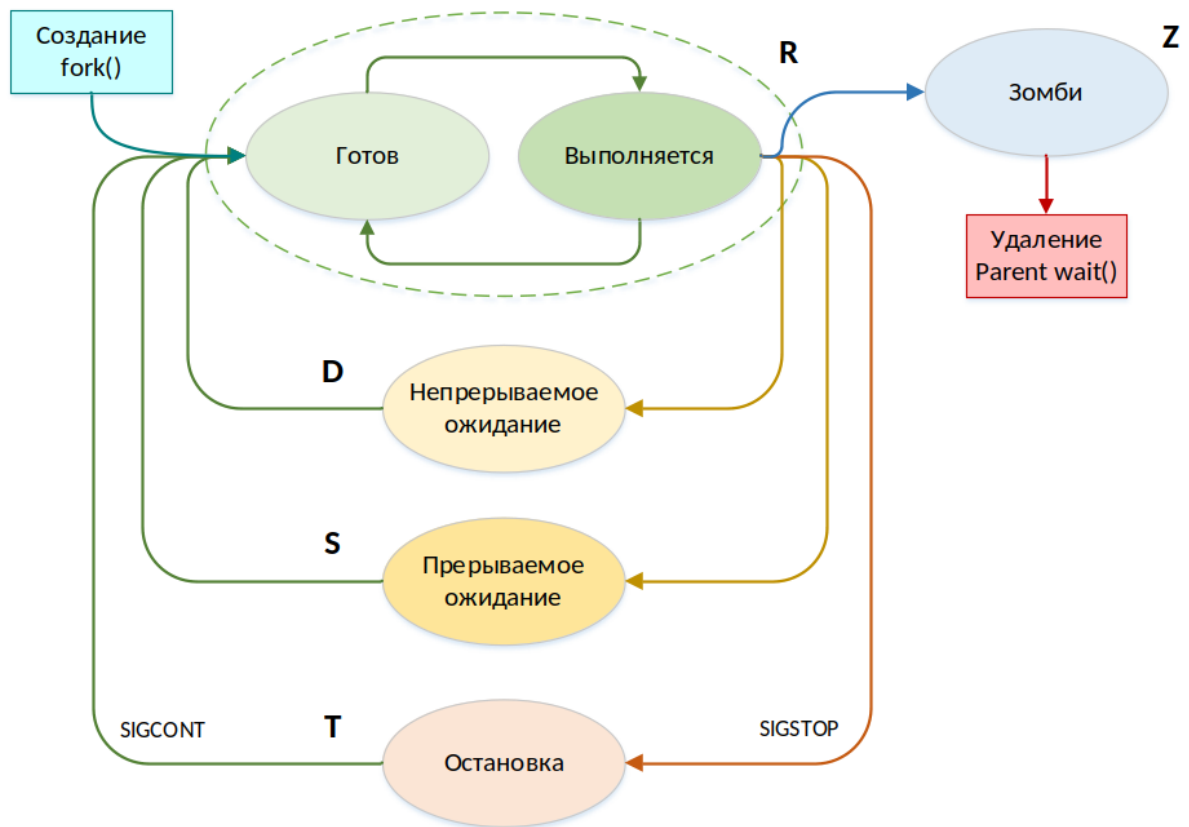
Проблема с «неуспокоенными» процессами заключается в том, что количество идентификаторов PID в дистрибутивах Linux всегда ограничено. К примеру, в Astra Linux их по умолчанию может быть не больше 4 194 304 (см. вывод команды `cat /proc/sys/kernel/pid_max`). Поэтому, когда такие зомби встают регулярно и в большом количестве, наступает зомби-апокалипсис, и система перестает запускать новые процессы.

Зависшие зомби-процессы могут появляться как по причине зависания родительского процесса, так и в связи с ошибками в программном коде, но отправлять SIGKILL зомби-процессам совершенно бесполезно, ибо то, что мертво, умереть не может. Чтобы очистить систему от таких процессов, можно еще раз вручную направить сигнал SIGCHLD родительскому процессу с помощью команды `kill -s SIGCHLD <PPID>`. Если же родительский процесс завис и по-прежнему отказывается «хоронить» зомби-процесс, следующим шагом будет удаление родительского процесса. И не забываем про «семь бед – один reset» — после перезагрузки ни один зомби точно не выживет.

Состояния процесса во время его существования

Жизненный цикл процесса предусматривает 5 основных состояний (см. рисунок). Посмотреть текущее состояние можно с помощью команды `ps aux` в колонке STAT:

- **R** - готов (ready) или выполняется (running).
- **D** - непрерываемое ожидание (uninterruptible sleep) - ожидание определенного события, чаще всего ожидания операции ввода-вывода.
- **S** - прерываемое ожидание (interruptible sleep) - ожидание неопределенного события или сигнала.
- **T** - остановка - процесс приостановлен, например, отладчиком.
- **Z** - зомби (zombie) - завершен, но сигнал SIGCHLD не получен родителем.



Жизненный цикл процесса

Статусы **готов** и **выполняется** обозначаются одной буквой R, т.к. процесс получает кванты процессорного времени по сто раз в секунду, и эта информация не несет для пользователя никакой полезной информации.

Если процессу требуется доступ к устройству ввода-вывода (например, прочитать данные из файла), то без получения этих данных дальнейшее выполнение процесса будет невозможным, и процесс перейдет в состояние **непрерываемого ожидания (D)**. Это ожидание называется непрерываемым, так как процесс может полностью проигнорировать даже команду `kill -9 pid`.

Существует также **прерываемое ожидание (S)**. Это состояние наступает, когда процесс завершил выполнение задач и ожидает следующих запросов. Например, вы открыли калькулятор, и окно приложения ожидает ваших указаний. Прерываемым это ожидание называется потому, что процесс в таком состоянии может быть завершён (убит) администратором. Это не всегда безопасно, особенно применительно к базам данных, хотя и возможно.

Отправкой сигнала SIGSTOP процесс может быть переведен в состояние **остановлен (T)**, в котором он замирает до получения сигнала разморозки SIGCONT. Вы можете

использовать это состояние для приостановки процесса, который начал потреблять слишком много ресурсов, чтобы спокойно проанализировать причины, не завершая работу приложения сигналом SIGKILL.

Пятое состояние **зомби (Z)** мы уже рассмотрели подробно выше. Оно возникает, когда процесс завершил работу, но родитель еще не получил его статус. Если такой процесс зависнет в системе вы можете повторно отправить SIGCHLD родителю, завершить работу родителя или перезагрузить систему

Потоки в Linux

Кроме вызова `fork()` существует также системный вызов `clone()`, который позволяет создавать процессы с разделяемым адресным пространством или так называемые потоки (англ. thread).

Потоки позволяют организовать параллельную обработку данных на нескольких ядрах вычислительного узла, что сильно повышает итоговую скорость выполнения задачи. Например, если у вас 4 ядра плюс гипертрейдинг, то вы можете архивировать данные в 8 потоков, что будет в 8 раз быстрее.

В зависимости от используемых флагов потоки могут обладать разными возможностями:

- **CLONE_VM** — общее адресное пространство (любой из потоков может поменять страницы в памяти, и эти изменения будут видны всем потокам);
- **CLONE_FS** — общие сведения о файловой системе (корневой и текущий каталоги, `umask`, права на создаваемые файлы и каталоги по умолчанию);
- **CLONE_FILES** — общая таблица открытых файлов;
- **CLONE_SIGHAND** — общая таблица обработчиков сигналов;
- **CLONE_PARENT** — позволяет создавать сестринские процессы с тем же PPID, как у себя.

С точки зрения Linux управление процессами и потоками — это очень похожие задачи, поэтому и для работы с ними используются одни и те же инструменты. Например, команда `ps -T -p pid` покажет вам все потоки указанного процесса.

Сигналы для процессов в Linux

Мы уже знаем о том, что сигналы предназначены для межпроцессного взаимодействия. Этот механизм напоминает системные прерывания, поэтому сигналы иногда называют еще мягкими или программными прерываниями. Однако, если системные прерывания обрабатываются ядром, то обработчик сигналов находится внутри приложения.

Для отправки сигналов из командной строки предназначена утилита `kill` (от англ. *убить*). Название утилиты объясняется тем, что без дополнительных уточнений она отправляет сигнал SIGTERM, который как раз и означает завершение работы. Для того чтобы отправить какой-то другой сигнал, нужно указать цифровой код сигнала `-<signal>` или его имя с помощью ключа `-s` (`--signal`):

Запускаем калькулятор в фоновом режиме:

```
sa@astra:~$ kcalc &
[1] 1564
```

Завершаем процесс с выбранным PID 1564 из прошлой команды:

```
sa@astra:~$ kill -SIGTERM 1564
sa@astra:~$
```

Альтернативные варианты написания команды:

- `kill -TERM 1564`
- `kill -15 1564`
- `kill 1564`

Процессы Linux поддерживают 64 сигнала, список которых можно посмотреть с помощью ключа `-l` (`-l`, `--list`) команды `kill`:

```
sa@astra:~$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL     5) SIGTRAP
2) SIGABRT     7) SIGBUS     8) SIGFPE     9) SIGKILL    10) SIGUSR1
1) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
2) SIGSTKFLT  17) SIGCHLD   18) SIGCONT    19) SIGSTOP    20) SIGTSTP
3) SIGTTIN    22) SIGTTOU   23) SIGURG     24) SIGXCPU    25) SIGXFSZ
4) SIGVTALRM  27) SIGPROF   28) SIGWINCH   29) SIGIO       30) SIGPWR
5) SIGSYS     34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
6) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
7) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
8) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
9) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
10) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
11) SIGRTMAX-1 64) SIGRTMAX
```

Чаще всего администраторы используют в работе сигналы:

- **Сигнал (2)SIGINT** предназначен для прерывания работы приложений по требованию пользователя (от англ. *interrupt* — *прервать*) и отправляется автоматически активному процессу терминала по нажатию клавиш `Ctrl + C`.

Чаще всего администраторы отправляют этот сигнал приложениям через горячие клавиши, но можно сделать это и через утилиту `kill`. Продвинутое приложение может реализовывать более сложную логику обработки этого сигнала, чем просто прерывание работы.

Например, Postgres запускает быстрое выключение: в этом случае сервер запрещает новые подключения и посылает всем процессам сигнал SIGTERM, в результате чего их транзакции прерываются, и сами процессы завершаются. Управляющий процесс ждёт, пока будут завершены все эти процессы и затем завершается сам.

- **Сигнал (9)SIGKILL** требует процесс немедленно завершить свою работу. Процесс не имеет права проигнорировать этот сигнал и не может переопределить его обработчик, поэтому, если процесс не завершился по сигналу -9, то это указывает на какие-то проблемы в работе операционной системы. По своей сути сигнал SIGKILL похож на команду `End task` диспетчера задач Windows.

- **Сигнал (15)SIGTERM** отправляется процессу по умолчанию, если команде `kill` передать только идентификатор процесса. Сигнал просит процесс приступить к штатному завершению работы (от англ. *software termination* — завершение работы программного обеспечения).

Если обработчик сигнала реализован в приложении правильно, то при получении SIGTERM процесс выполнит «умное выключение», т.е. обработает все несохраненные изменения, закроет файлы, сохранит состояние и вернет код 0, соответствующий успешному завершению.

При выключении операционной системы процессы сначала получают сигнал SIGTERM и только по таймауту завершаются принудительно.

Продвинутые администраторы расширяют набор своих инструментов сигналами:

- **Сигнал (1)SIGHUP** посылается всем дочерним процессам терминала при его закрытии, чтобы они тоже завершили свою работу (от англ. *hang up* — повесить трубку). Название сигнала берет свое начало со времен аппаратных терминалов и телефонных линий связи.

Для того чтобы приложение игнорировало сигнал -1, его можно запустить с помощью команды `nohup`. Если закрыть терминал, такие процессы «осиротеют» и будут удочерены процессом `init (systemd)`.

```
sa@astra:~$ kcalc &  
[1] 1575
```

Приложение завершает свою работу:

```
sa@astra:~$ kill -SIGHUP 1575  
sa@astra:~$ nohup kcalc &
```

Запускаем команду через `nohup`:

```
sa@astra:~$ nohup kcalc &  
[2] 1583  
[1] Завершено kcalc  
sa@astra:~$ nohup: ввод игнорируется, вывод добавляется в 'nohup.out'
```

Завершить процесс с помощью сигнала HUP больше не удастся, он игнорируется приложением:

```
sa@astra:~$ kill -SIGHUP 1583  
sa@astra:~$
```

- **Сигнал (17)SIGCHLD** направляется родителю при завершении дочернего процесса. При появлении зомби-процессов сигнал может быть направлен вручную для обработки очереди.

- **Сигнал (19)SIGSTOP** переводит процесс в состояние «остановлен», что удобно для отладки проблем с высоким потреблением ресурсов, когда требуется вернуть систему к жизни, не завершая проблемный процесс полностью. Процесс не имеет права проигнорировать этот сигнал и не может переопределить его обработчик. Для возобновления работы процесса предназначен сигнал SIGCONT.

- **Сигнал (20)SIGTSTP** предназначен для остановки работы приложений из терминала (от англ. *TTY stop*) и отправляется автоматически активному процессу терминала по нажатию клавиш `Ctrl + Z`.

- **Сигнал (18)SIGCONT** возобновляет работу процесса, который был остановлен с помощью сигналов SIGSTOP и SIGTSTP.

Каждый сигнал является указателем на область памяти, где находится обработчик, и многие из этих обработчиков могут быть переопределены внутри приложения через функцию `trap()`.

Планировщик задач в Linux и управление приоритетами процессов

Планировщик задач

Планировщик задач – это часть ядра ОС, необходимая для распределения ресурсов процессора между процессами. Планировщик преследует несколько целей:

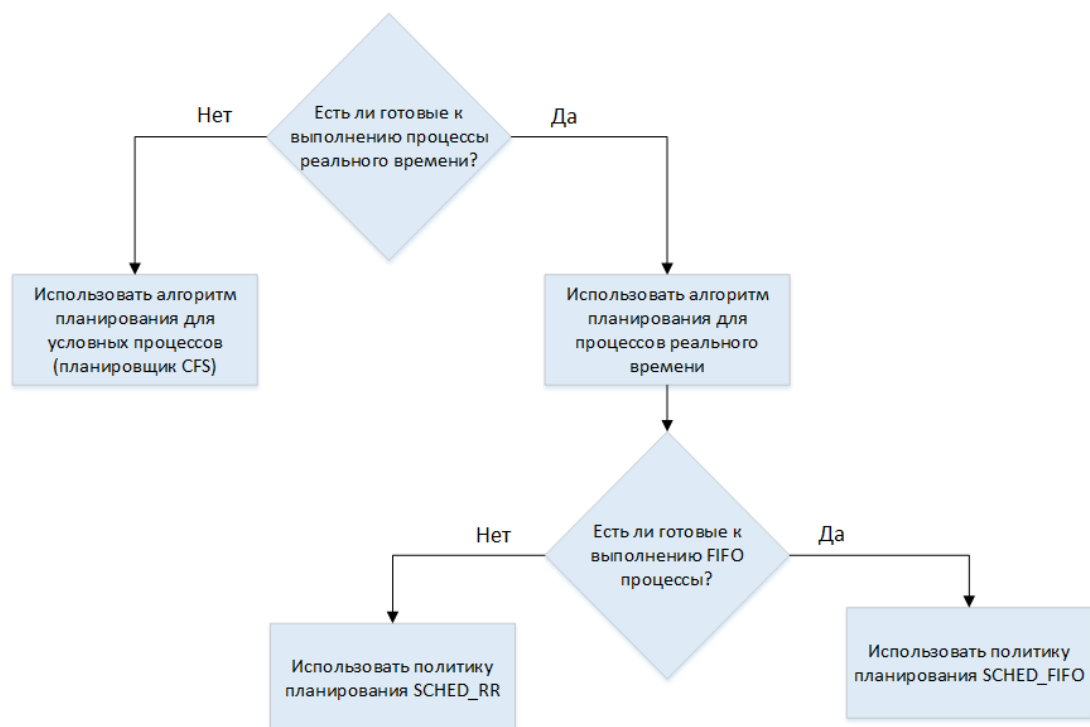
- Максимизировать пропускную способность, то есть количество задач, выполняемых за единицу времени.
- Минимизировать время ожидания, то есть время, прошедшее с момента готовности процесса до начала его выполнения.
- Минимизировать время ответа, то есть время, прошедшее с момента готовности процесса до завершения его выполнения.
- Максимизировать равнодоступность, то есть справедливое распределение ресурсов между задачами.

Для достижения этих целей и обеспечения стабильности работы системы и приоритетного выполнения важных системных процессов используются системы приоритизации и распределения ресурсов по политикам планирования (группам процессов).

Группы процессов

В Linux существуют три основные группы процессов: First In, First Out (FIFO), Round Robin (RR) и Other. Первые две представляют собой процессы реального времени, которые должны выполняться немедленно, без задержек, а последняя, Other - условные процессы, при выполнении которых задержки не являются критичными для работы системы.

Таким образом, в системе существуют три очереди процессов, и логика выделения процессорного времени (логика выбора приоритета процесса) в каждой из них своя (см. рисунок).



Логика выбора алгоритма и политики планирования процессов

FIFO процессы (*first in, first out, первый вошел, первый вышел*) имеют самый высокий приоритет из всех. Такой процесс фактически работает в однопрограммном режиме и не реагирует ни на какие сигналы. Использование таких процессов несет серьезные риски, так как ошибка в его выполнении может критически повлиять на всю систему в целом.

В Linux используется всего несколько FIFO потоков ядра, для выполнения которых критически важно, чтобы их операции не прерывались и завершались без сбоев, например, изменение статуса процесса диспетчером процессов. В противном случае мы могли бы получить некорректную таблицу состояний процессов.

Процессы **Round Robin** (*по круговой схеме*) имеют более низкий приоритет по сравнению с FIFO процессами, поэтому при активизации FIFO процесса будут им вытеснены. Процессы RR равны между собой по приоритету в пределах одной очереди. Пока существует несколько процессов Round Robin. Каждому из них диспетчер выделяет определенный квант времени, и они будут выполняться по кругу, пока все такие процессы не будут завершены или не перейдут в состояния ожидания или остановки.

Процессы **Other** (все остальные) имеют самый низкий приоритет, то есть могут выполняться, только если нет активных FIFO или Round Robin процессов. В современных дистрибутивах Linux этими процессами занимается планировщик CFS (*Completely Fair Scheduler, абсолютно справедливый планировщик*), который старается соблюдать принцип справедливости, чтобы каждый процесс получил одинаковое количество процессорного времени. Кванты времени вычисляются так, чтобы исключить слишком частые переключения между процессами.

Отличительной особенностью планировщика CFS является высокая скорость сортировки процессов по потреблению ими процессорного времени за время $\log(N)$. Благодаря этому удалось добиться действительно справедливого распределения процессорного времени между процессами при высокой скорости самого планирования.

Политики планирования

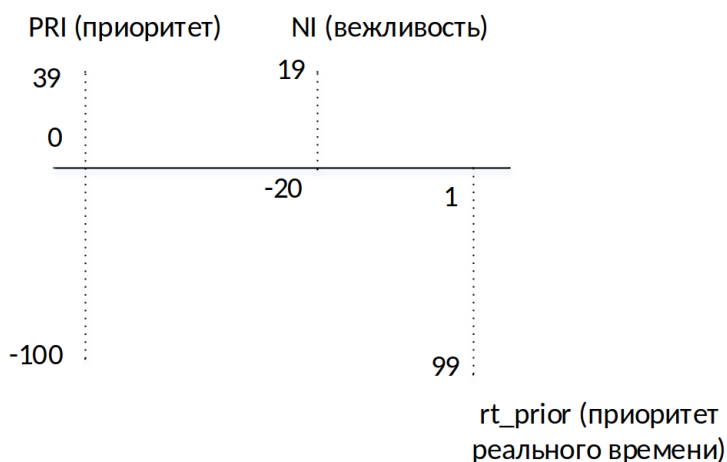
Группы процессов FIFO, RR и Other соответствуют политикам планирования SCHED_FIFO, SCHED_RR и SCHED_OTHER (всего таких политик 6). Посмотреть список политик планирования можно командой `chrt -m`.

```
sa@astra:~$ chrt -m
SCHED_OTHER min/max priority : 0/0
SCHED_FIFO min/max priority  : 1/99
SCHED_RR min/max priority    : 1/99
SCHED_BATCH min/max priority  : 0/0
SCHED_IDLE min/max priority   : 0/0
SCHED_DEADLINE min/max priority : 0/0
```

Как видим, выбор приоритета (а именно приоритета реального времени) доступен только в политиках `SCHED_FIFO` и `SCHED_RR` - от 1 до 99, для остальных политик он всегда 0.

Приоритет процессов

Каждому процессу присваивается приоритет. Это целое число от 39 до -100, где 39 - наименьший приоритет, а -100 - наивысший. Процессы с приоритетами от 39 до 0 - это обычные процессы из группы Other. Их приоритет не может быть меньше 0. По умолчанию все процессы из политики Other имеют приоритет 20, однако это значение может быть скорректировано как при запуске процесса (команда `nice`), так и во время его работы (команда `renice`). Эти команды меняют параметр NICE процессов - его «вежливость» (от англ. *niceness*), которая по умолчанию равна 0.



Соотношение параметров приоритета процесса, его «вежливости» и приоритета реального времени

Параметр NICE может принимать значение от +19 до -20. При этом значение приоритета по умолчанию (20) суммируется со значением NICE, и, как упоминалось ранее, итоговый результат может принимать значение от 39 до 0.

Процессы с приоритетом от -1 до -100 - это процессы реального времени. У таких процессов кроме обычного приоритета существует приоритет реального времени, который используется в политиках планирования FIFO и RR. Для преобразования приоритета реального времени (`rt_prior`) в обычный приоритет процесса необходимо из -1 вычесть `rt_prior`. Например, при `rt_prior` равному 50, обычный приоритет будет равен -51 ($-1 - 50 = -51$).

Узнать текущее значение приоритета и параметра NICE можно в выводе команды `htop` - колонки PRI и NI. Чтобы установить htop в Astra Linux, нужно выполнить команду: `sudo apt install htop`.

Задания, интерактивный и фоновый режимы работы процесса

Оболочка Bash позволяет останавливать работу процессов, переводить их в интерактивный и фоновый режим по ходу работы, для чего предназначено несколько команд.

Остановка процессов. Когда мы запускаем утилиты как обычно, например, `kcalc`, они работают в интерактивном режиме, поэтому блокируют работу с терминалом. Если нам нужно вернуться к работе в терминале, мы можем закрыть приложение полностью клавишами `Ctrl + C` или временно приостановить его работу сочетанием `Ctrl + Z`. При этом калькулятор останется на экране, но перестанет реагировать на наши действия.

Перевод процесса в интерактивный режим. Когда мы захотим продолжить работу с калькулятором, нам нужно будет вернуть его обратно в интерактивный режим командой `fg`. В этом случае калькулятор снова начнет реагировать на наши действия, но терминал опять окажется заблокирован.

Команда `fg <задание>` переводит указанное задание в интерактивный режим (от англ. foreground — передний план). Задание может быть указано следующим образом:

- `<N>` — порядковый номер задания, см. команду `jobs` далее.
- `%шаблон` — поиск задания по его имени, например, `%ping` или `%nmap`.
- `%+` и `%-` — следующая и предыдущая задача.
- `%%` — последняя задача. То же самое, что не указывать номер задания.

Перевод процесса в фоновый режим. Если же мы хотим продолжить работу с калькулятором, но не хотим терять доступ к терминалу, мы можем перевести приложение не в интерактивный, а в фоновый режим командой `bg`. В этом случае нам будут доступны и калькулятор, и терминал одновременно, как будто калькулятор изначально был запущен командой `kcalc &` с амперсандом на конце.

Команда `bg <задание>` переводит указанное задание в фоновый режим (от англ. background — задний план). Задание задается таким же образом, как в команде `fg`, поэтому не будем повторяться.

Просмотр фоновых заданий выполняется командой `jobs`. У этой команды есть следующие ключи:

- Ключ `-l` – выводит список процессов заданий с PID процесса и информацией по умолчанию.
- Ключ `-n` – выводит только те процессы, статус которых был изменен с последнего оповещения.
- Ключ `-p` – выводит список только PID фоновых процессов.
- Ключ `-r` – выводит только запущенные фоновые процессы.
- Ключ `-s` – выводит только остановленные фоновые процессы.

Давайте запустим три фоновых процесса и посмотрим список заданий:

```
sa@astra:~$ kcalc &
[1] 2757
sa@astra:~$ kcalc &
[2] 2763
sa@astra:~$ kcalc &
[3] 2768
sa@astra:~$ jobs
[1]   Запущен          kcalc &
[2]-  Запущен          kcalc &
[3]+  Запущен          kcalc &
```

Извлечение информации о процессах

Информация о процессах доступна через псевдофайловую систему `/proc`. Это универсальный интерфейс для доступа к данным о процессах и работе ядра ОС. Кроме того, используя определенные файлы, можно не только получать информацию о процессах, но и управлять некоторыми настройками ядра ОС. Утилиты для работы с процессами, такие как `ps`, `top` или `htop`, в качестве источника информации используют как раз эту псевдофайловую систему.

Для пользователя взаимодействие с файлами `/proc` выглядит как работа с обычными текстовыми файлами, но это совсем не так. Например, при чтении какого-либо файла командой `cat` происходит системный вызов, и система формирует текстовый ответ, который передается команде.

Часть файлов предоставляет человекочитаемый формат данных, содержащих, например, пары ключ-значение, а часть – машиночитаемые, используемые при работе утилит, которые удобно использовать, например, в скриптах автоматизации.

Вначале мы разберем, что хранится в каталогах `/proc/PID/`, где PID - числовой идентификатор процесса, например, `/proc/1/`, а затем посмотрим содержимое самого каталога `/proc/`.

Содержимое /proc/PID/

```
sa@astra:~$ sudo ls /proc/1
arch_status      cwd              mem              patch_state      stat
attr             environ         mountinfo        personality      statm
autogroup        exe             mounts           projid_map       status
auxv             fd              mountstats       root             syscall
cgroup           fdinfo          net              sched            task
clear_refs       gid_map         ns               schedstat        timens_offsets
cmdline          io              numa_maps        sessionid         timers
comm             limits          oom_adj          setgroups        timerslack_ns
coredump_filter  loginuid        oom_score        smaps            uid_map
cpu_resctrl_groups map_files       oom_score_adj    smaps_rollup     wchan
cpuset           maps            pagemap          stack
```

Файл `/proc/cmdline` – в нем содержится строка запуска процесса.

```
sa@astra:~$ cat /proc/1/cmdline && echo
/sbin/init
```

Файл `/proc/exe` – символическая ссылка, ведущая к полному пути до исполняемого файла. Полный путь необходим для однозначного понимания, какой именно файл был запущен.

```
sa@astra:~$ sudo ls -l --color=always /proc/1/exe
lrwxrwxrwx 1 root root 0 map 19 11:32 /proc/1/exe -> /usr/lib/systemd/systemd
```

Файл `/proc/cwd` – текущий рабочий каталог процесса PID1.

```
sa@astra:~$ sudo ls -l --color=always /proc/1/cwd
lrwxrwxrwx 1 root root 0 map 19 12:49 /proc/1/cwd -> /
```

Файл `/proc/environ` – окружение процесса, создающее контекст его выполнения.

```
sa@astra:~$ sudo cat /proc/1/environ && echo
SHLVL=1HOME=/init=/sbin/initTERM=linuxBOOT_IMAGE=/boot/vmlinuz-5.15.0-33-
genericdrop_caps=PATH=/sbin:/usr/sbin:/bin:/usr/binPWD=/rootmnt=/root
```

Каталог `/proc/fd/` – дескрипторы открытых файлов (*от англ. file descriptors*). Каждый файл в `fd/` представляет собой символическую ссылку и имеет числовое имя. Это не номер айноды в файловой системе, это уникальный номер в пределах процесса, который позволяет процессу обращаться к конкретным файлам, с которыми он работает. Дескрипторы 0, 1 и 2 - это стандартные ввод, вывод и вывод ошибок.

```
sa@astra:~$ sudo ls -l /proc/1/fd --color=always
итого 0
lrwx----- 1 root root 64 map 19 12:38 0 -> /dev/null
lrwx----- 1 root root 64 map 19 12:38 1 -> /dev/null
...
lrwx----- 1 root root 64 map 19 13:03 109 -> 'socket:[19369]'
lr-x----- 1 root root 64 map 19 13:03 11 -> anon_inode:inotify
...
lr-x----- 1 root root 64 map 19 13:03 113 -> 'pipe:[17301]'
lrwx----- 1 root root 64 map 19 13:03 114 -> 'socket:[17168]'
lrwx----- 1 root root 64 map 19 13:03 115 -> 'socket:[16912]'
lrwx----- 1 root root 64 map 19 13:03 116 -> 'socket:[19238]'
lrwx----- 1 root root 64 map 19 12:38 12 -> 'socket:[24743]'
```

```
lr-x----- 1 root root 64 map 19 13:03 13 -> /proc/1/mountinfo
lr-x----- 1 root root 64 map 19 13:03 14 -> anon_inode:inotify
lr-x----- 1 root root 64 map 19 13:03 15 -> /proc/swaps
...
lrwx----- 1 root root 64 map 19 13:03 92 -> /dev/rfkill
lrwx----- 1 root root 64 map 19 13:03 93 -> 'socket:[17218]'
lrwx----- 1 root root 64 map 19 13:03 94 -> /run/initctl
...
```

Файл `/proc/io` – содержит сведения об объемах данных, прочитанных и записанных процессом в хранилище информации.

```
sa@astra:~$ sudo cat /proc/1/io
rchar: 11296824165
wchar: 159986595
syscr: 6481108
syscw: 467703
read_bytes: 801298432
write_bytes: 258985984
cancelled_write_bytes: 43184128
```

Файл `/proc/limits` – отображает различные ограничения процесса, установленные конфигурационным файлом `/etc/security/limits.conf` и командой `ulimit`.

```
sa@astra:~$ sudo cat /proc/1/limits
```

| Limit | Soft Limit | Hard Limit | Units |
|-----------------------|------------|------------|-----------|
| Max cpu time | unlimited | unlimited | seconds |
| Max file size | unlimited | unlimited | bytes |
| Max data size | unlimited | unlimited | bytes |
| Max stack size | 8388608 | unlimited | bytes |
| Max core file size | 0 | unlimited | bytes |
| Max resident set | unlimited | unlimited | bytes |
| Max processes | 7524 | 7524 | processes |
| Max open files | 1048576 | 1048576 | files |
| Max locked memory | 67108864 | 67108864 | bytes |
| Max address space | unlimited | unlimited | bytes |
| Max file locks | unlimited | unlimited | locks |
| Max pending signals | 7524 | 7524 | signals |
| Max msgqueue size | 819200 | 819200 | bytes |
| Max nice priority | 0 | 0 | |
| Max realtime priority | 0 | 0 | |
| Max realtime timeout | unlimited | unlimited | us |

Файл `/proc/maps` – физические адреса страниц памяти, используемые в данный момент.

```
sa@astra:~$ sudo cat /proc/1/maps | head
61e0b3d17000-61e0b3d45000 r--p 00000000 08:01 396408 /usr/lib/systemd/systemd
61e0b3d45000-61e0b3e60000 r-xp 0002e000 08:01 396408 /usr/lib/systemd/systemd
61e0b3e60000-61e0b3eb5000 r--p 00149000 08:01 396408 /usr/lib/systemd/systemd
61e0b3eb5000-61e0b3eee000 r--p 0019d000 08:01 396408 /usr/lib/systemd/systemd
...
```

Файл `/proc/pagemap` для каждой виртуальной страницы определяет её фактическое положение: в физической памяти или в файле подкачки. Информация хранится в двоичном коде.

Файл `/proc/sched` – отображение текущих значений переменных планировщика процессов, необходимых для корректной работы планировщика SFC.

```
sa@astra:~$ sudo cat /proc/1/sched
```

```
systemd (1, #threads: 1)
```

```
-----
se.exec_start          :          1936927.703002
se.vruntime            :           1252.196899
se.sum_exec_runtime    :           1674.625884
se.nr_migrations       :              0
nr_switches            :             4252
nr_voluntary_switches  :              980
nr_involuntary_switches :             3272
se.load.weight         :           1048576
se.avg.load_sum        :              207
se.avg.runnable_sum    :           211968
se.avg.util_sum        :           201728
se.avg.load_avg        :              0
se.avg.runnable_avg    :              0
se.avg.util_avg        :              0
se.avg.last_update_time :          1936927702016
se.avg.util_est.ewma   :              8
se.avg.util_est.queued :              0
uclamp.min             :              0
uclamp.max             :             1024
effective uclamp.min    :              0
effective uclamp.max    :             1024
policy                 :              0
prio                   :             120
clock-delta            :              33
mm->numa_scan_seq      :              0
numa_pages_migrated    :              0
numa_preferred_nid     :             -1
total_numa_faults      :              0
current_node=0, numa_group_id=0
numa_faults node=0 task_private=0 task_shared=0 group_private=0 group_shared=0
```

Файлы `/proc/stat` и `/proc/status` — близкие по своей сути, отражают основные сведения о процессе. Файл **stat** - машиночитаемый формат, а **status** - человекочитаемый. Большинство параметров, выводимых утилитой `ps`, она получает из структур данных, аналогичных тем, которые отображаются в этих файлах.

```
sa@astra:~$ sudo cat /proc/1/stat
```

```
1 (systemd) S 0 1 1 0 -1 4194560 9273 117633 170 8005 62 104 189 273 20 0 1 0 7 105480192 28
28 18446744073709551615 107617717604352 107617718763385 140732246690496 0 0 0 671173123 4096
1260 1 0 0 17 0 0 0 0 0 107617719115472 107617719345472 107617734889472 140732246691659 1
40732246691670 140732246691670 140732246691821 0
```

```
sa@astra: ~$ sudo cat /proc/1/status
```

```
Name:    systemd
Umask:   0000
State:   S (sleeping)
Tgid:    1
Ngid:    0
Pid:     1
PPid:    0
TracerPid:      0
Uid:    0      0      0      0
Gid:    0      0      0      0
FDSize: 128
Groups:
NStgid: 1
NSpid:  1
NSpgid: 1
NSSid:  1
VmPeak: 168544 kB
VmSize: 103008 kB
```

```
...
```

```
Mems_allowed_list:      0
voluntary_ctxt_switches: 986
nonvoluntary_ctxt_switches: 3272
```

Файл **statm** - статистика по использованию памяти. Это 7 чисел, из которых только 5 имеют значения в современных версиях ядра. Все значения - количество страниц памяти:

- **1** – общий размер памяти, занимаемой программой (то же, что и VmSize в файле status)
- **2** – размер резидентной памяти (то же, что и VmRSS в файле status)
- **3** – разделяемые страницы памяти (то же, что и RssFile+RssShmem в файле status)
- **4** – размер сегмента, отведенного под код
- **5** – не используется (всегда 0, начиная с ядра 2.6)
- **6** – размер сегмента данных и стека
- **7** – не используется (всегда 0, начиная с ядра 2.6)

```
sa@astra:~$ cat /proc/1/statm
25752 2828 2172 283 0 4884 0
```

Содержимое /proc

Ниже представлено содержимое каталога `/proc/`. В выводе команды все каталоги с PID заменены на {PID 1,2,...,N} для лучшего восприятия. Файлы, хранящиеся в корне каталога `/proc/`, характеризуют состояние и характеристики всего вычислительного узла и ядра ОС.

```
sa@astra:~$ ls /proc
{PID 1,2, ... N}
bootconfig      kallsyms        scsi
buddyinfo       kcore           self
bus             keys            slabinfo
cgroups         key-users       softirqs
cmdline         kmsg            stat
consoles        kpagecgroup     swaps
cpuinfo         kpagecount      sys
crypto          kpageflags      sysrq-trigger
devices         loadavg         sysvipc
diskstats       locks           thread-self
dma             mdstat          timer_list
driver          meminfo         tty
dynamic_debug   misc            uptime
execdomains     modules         version
fb              mounts          version_signature
filesystems     mtrr            vmallocinfo
fs              net             vmstat
interrupts      pagetypeinfo    zoneinfo
iomem           partitions
acpi            ioports         pressure
asound          irq             schedstat
```

Файл `/proc/cmdline` – список параметров, которые были переданы ядру при загрузке.

```
sa@astra:~$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-5.15.0-33-generic root=UUID=eba34003-adda-47f3-a50d-8b9513eb81dc ro
parsec.max_ilev=63 quiet net.ifnames=0
```

Файл `/proc/cpuinfo` – сведения о всех установленных процессорах.

```
sa@astra:~$ cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 79
model name    : Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz
stepping      : 9
cpu MHz       : 2394.632
cache size    : 4096 KB
physical id   : 0
siblings      : 1
core id       : 0
cpu cores     : 1
...
```

Файл `/proc/diskstats` – статистика операций со всеми дисками. Для каждого диска своя строка. [Описание полей доступно по ссылке](#).

```
sa@astra:~$ cat /proc/diskstats
 7   0 loop0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
. . .
 8   1 sda1 26397 13868 1078786 19313 4163 14541 185792 7262 0 35336 26576 0 0 0 0 0
 8   2 sda2  2 0 4 2 0 0 0 0 0 8 2 0 0 0 0 0 0
 8   5 sda5  69 0 5000 36 0 0 0 0 0 108 36 0 0 0 0 0
11   0 sr0  9 0 4 1 0 0 0 0 0 24 1 0 0 0 0 0
```

Файл `/proc/meminfo` – отображение информации о состоянии памяти. Предоставляет больше параметров, чем утилита `free`.

```
sa@astra:~$ free
              total        used         free       shared    buff/cache   available
Mem:          2025456      493208        620252         17088         911996       1327116
Swap:          998396           0         998396
```

```
sa@astra:~$ cat /proc/meminfo
MemTotal:      2025456 kB
MemFree:        620252 kB
MemAvailable:  1327160 kB
Buffers:        126248 kB
Cached:         435600 kB
SwapCached:      0 kB
Active:         273028 kB
Inactive:       578948 kB
Active(anon):    2292 kB
Inactive(anon):  304932 kB
...
```

Файл `/proc/devices` – перечень устройств в системе.

```
sa@astra:~$ cat /proc/devices
Character devices:
 1 mem
 4 /dev/vc/0
 4 tty
 4 ttyS
 5 /dev/tty
...
Block devices:
 7 loop
```

```
8 sd
9 md
11 sr
65 sd
66 sd
67 sd
...
```

Файл `/proc/filesystem` – перечень файловых систем, поддерживаемых ядром ОС.

```
sa@astra:~$ cat /proc/filesystems
nodev    sysfs
nodev    tmpfs
nodev    bdev
nodev    proc
...
        ext3
        ext2
        ext4
...
```

Файл `/proc/mounts` – перечень смонтированных файловых систем (формат данных, аналогичен файлу `/etc/fstab`).

```
sa@astra:~$ cat /proc/mounts
sysfs /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
proc /proc proc rw,nosuid,nodev,noexec,relatime 0 0
udev /dev devtmpfs rw,nosuid,relatime,size=963204k,nr_inodes=240801,mode=755,inode64
0 0
devpts /dev/pts devpts rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000 0 0
tmpfs /run tmpfs rw,nosuid,noexec,relatime,size=202548k,mode=755,inode64 0 0
/dev/sda1 / ext4 rw,relatime,errors=remount-ro 0 0
parsecfs /parsecfs parsecfs rw,sync,relatime 0 0
securityfs /sys/kernel/security securityfs rw,nosuid,nodev,noexec,relatime 0 0
tmpfs /dev/shm tmpfs rw,nosuid,nodev,inode64 0 0
tmpfs /run/lock tmpfs rw,nosuid,nodev,noexec,relatime,size=5120k,inode64 0 0
...
```

Файл `/proc/modules` – список подгруженных модулей ядра. Ядро Linux монолитное, но модульное, и мы можем управлять тем, какие модули подгружать, а какие нет, тем самым определяя функционал запускаемой ОС.

```
sa@astra:~$ cat /proc/modules
binfmt_misc 24576 1 - Live 0x0000000000000000 (E)
vboxvideo 36864 0 - Live 0x0000000000000000 (OE)
drm_ttm_helper 16384 1 vboxvideo, Live 0x0000000000000000 (E)
intel_rapl_msr 20480 0 - Live 0x0000000000000000 (E)
intel_rapl_common 32768 1 intel_rapl_msr, Live 0x0000000000000000 (E)
...
```

Файл `/proc/swaps` – список разделов подкачки.

```
sa@astra:~$ cat /proc/swaps
Filename      Type      Size      Used      Priority
/dev/sda5    partition 998396    0         -2
```

Файл `/proc/version` – версия ядра ОС.

```
sa@astra:~$ cat /proc/version
Linux version 5.15.0-33-generic (builder@build5) (gcc (AstraLinuxSE 8.3.0-6) 8.3.0, GNU ld
(GNU Bin utils for AstraLinux) 2.31.1) #astra2+ci96 SMP Fri Oct 28 18:23:37 UTC 2
```


Каталог `/sys/kernel/` содержит набор файлов, которые позволяют нам оперативно без перезагрузки изменять параметры ядра ОС (но таких параметров не очень много). Поэтому часть файлов в этом каталоге доступна для записи, и через эти интерфейсы мы можем управлять ядром. Некоторые из них будут рассмотрены в лабораторной, посвящённой ядру ОС.

```
sa@astra:~$ ls /proc/sys/kernel/
acct                                perf_event_max_stack
acpi_video_flags                   perf_event_mlock_kb
auto_msgmni                         perf_event_paranoid
bootloader_type                    pid_max
bootloader_version                 poweroff_cmd
bpf_stats_enabled                  print-fatal-signals
cad_pid                             printk
cap_last_cap                       printk_delay
core_pattern                       printk_devkmsg
core_pipe_limit                    printk_ratelimit
core_uses_pid                      printk_ratelimit_burst
ctrl-alt-del                       pty
dmesg_restrict                     random
...
```

Управление процессами

Для управления процессами в Linux существует набор утилит. Рассмотрим работу с основными из них: консольными утилитами (`ps`, `top` и `htop`, `kill`), а для графической (системный монитор).

Управление процессами через консольные утилиты

Просмотр процессов

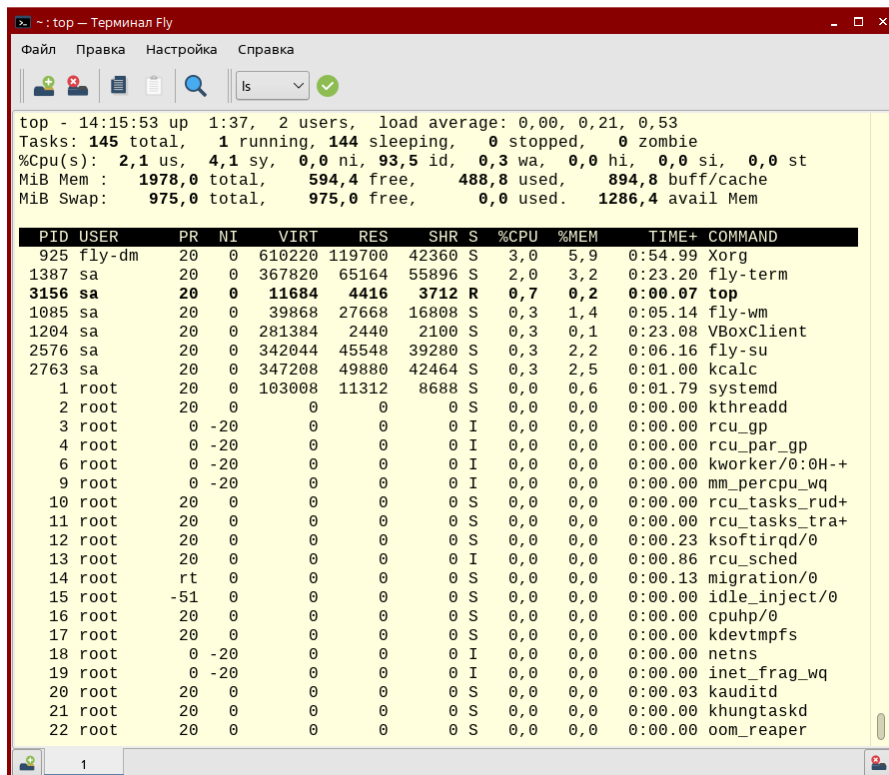
Для просмотра процессов есть три основные консольные утилиты (`ps`, `top` и `htop`) и одна графическая (системный монитор). Утилита `ps` позволяет получить снимок текущего состояния всех или заданных процессов и их параметров.

```
sa@astra:~$ ps
PID TTY          TIME CMD
1390 pts/0        00:00:00 bash
2757 pts/0        00:00:00 kcalc
2763 pts/0        00:00:00 kcalc
2768 pts/0        00:00:00 kcalc
3150 pts/0        00:00:00 ps
```

Просмотр процессов через утилиты `top` и `htop`

Утилиты `top` и `htop` позволяют просматривать загрузку системы и процессы в реальном времени, изменять сортировку, набор и вид отображаемых параметров, менять значение `nice` процессов или завершать процессы.

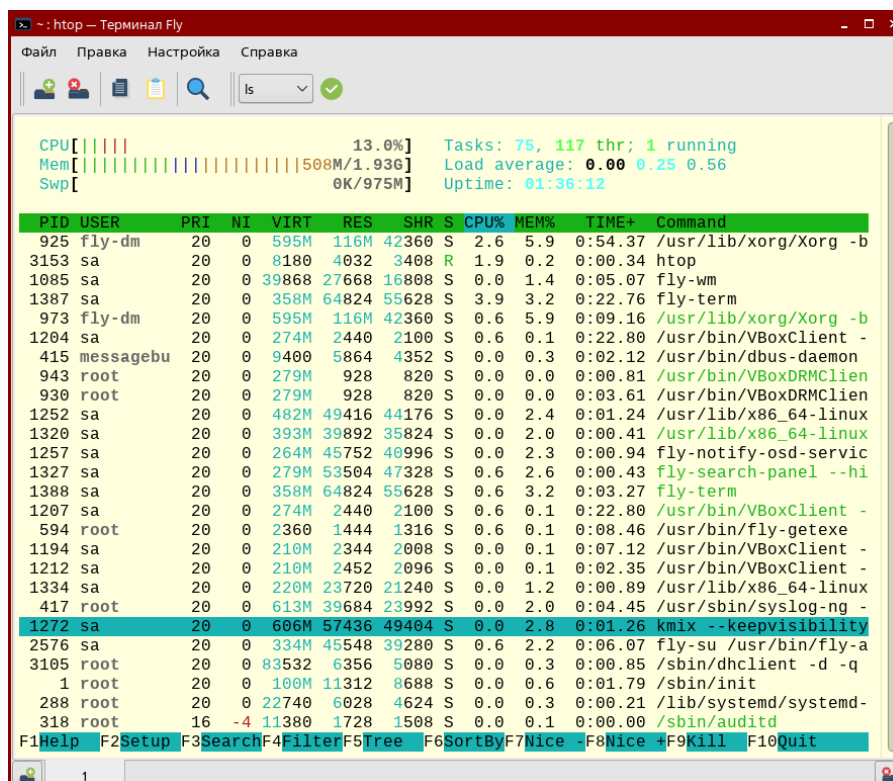
Основное отличие между ними в интерфейсе: **htop** более удобен для новичков и тех, кто работал с Windows Task Manager. Он поддерживает управление мышью и имеет более «дружелюбный» интерфейс управления.



The screenshot shows the 'top' utility running in a terminal window titled 'Терминал Fly'. The interface is text-based and provides a snapshot of system statistics and running processes. At the top, it shows system uptime (14:15:53), load average (0.00, 0.21, 0.53), and task counts (145 total, 1 running, 144 sleeping). Below this, it displays CPU usage (2.1 us, 4.1 sy, 0.0 ni, 93.5 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st) and memory usage (1978.0 total, 594.4 free, 488.8 used, 894.8 buff/cache). The main section is a table of running processes with columns for PID, USER, PR, NI, VIRT, RES, SHR, S, %CPU, %MEM, TIME+, and COMMAND. The processes are sorted by CPU usage, with 'fly-dm' at the top.

| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|------|--------|-----|-----|--------|--------|-------|---|------|------|---------|----------------|
| 925 | fly-dm | 20 | 0 | 610220 | 119700 | 42360 | S | 3.0 | 5.9 | 0:54.99 | Xorg |
| 1387 | sa | 20 | 0 | 367820 | 65164 | 55896 | S | 2.0 | 3.2 | 0:23.20 | fly-term |
| 3156 | sa | 20 | 0 | 11684 | 4416 | 3712 | R | 0.7 | 0.2 | 0:00.07 | top |
| 1085 | sa | 20 | 0 | 39868 | 27668 | 16808 | S | 0.3 | 1.4 | 0:05.14 | fly-wm |
| 1204 | sa | 20 | 0 | 281384 | 2440 | 2100 | S | 0.3 | 0.1 | 0:23.08 | VBoxClient |
| 2576 | sa | 20 | 0 | 342044 | 45548 | 39280 | S | 0.3 | 2.2 | 0:06.16 | fly-su |
| 2763 | sa | 20 | 0 | 347208 | 49880 | 42464 | S | 0.3 | 2.5 | 0:01.00 | kcalc |
| 1 | root | 20 | 0 | 103008 | 11312 | 8688 | S | 0.0 | 0.6 | 0:01.79 | systemd |
| 2 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | kthreadd |
| 3 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | rcu_gp |
| 4 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | rcu_par_gp |
| 6 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | kworker/0:0H+ |
| 9 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | mm_percpu_wq |
| 10 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | rcu_tasks_rud+ |
| 11 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | rcu_tasks_tra+ |
| 12 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.23 | ksoftirqd/0 |
| 13 | root | 20 | 0 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.86 | rcu_sched |
| 14 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.13 | migration/0 |
| 15 | root | -51 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | idle_inject/0 |
| 16 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | cpuhp/0 |
| 17 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | kdevtmpfs |
| 18 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | netns |
| 19 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | inet_frag_wq |
| 20 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.03 | kauditd |
| 21 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | khungtaskd |
| 22 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | oom_reaper |

Интерфейс утилиты top

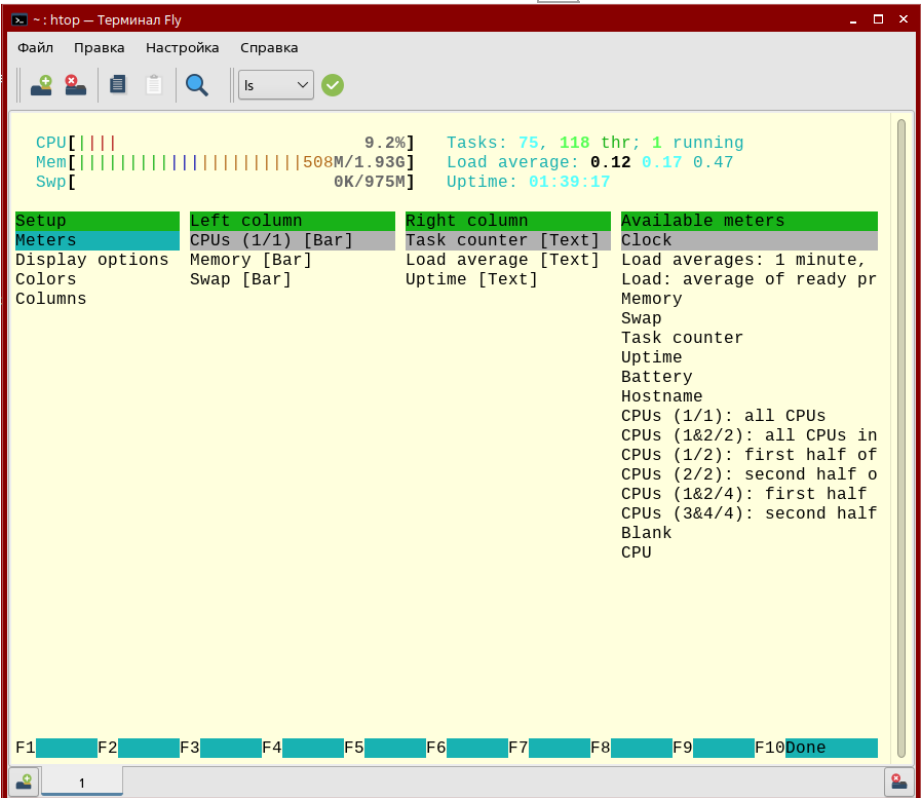


The screenshot shows the 'htop' utility running in a terminal window titled 'Терминал Fly'. The interface is more colorful and user-friendly than 'top'. It features a header with system statistics: CPU usage (13.0%), memory usage (508M/1.93G), swap usage (0K/975M), tasks (75, 117 thr; 1 running), load average (0.00, 0.25, 0.56), and uptime (01:36:12). Below the header is a table of running processes with columns for PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The processes are sorted by CPU usage, with 'fly-dm' at the top. The bottom of the screen shows a status bar with function key shortcuts: F1 Help, F2 Setup, F3 Search, F4 Filter, F5 Tree, F6 SortBy, F7 Nice, F8 Kill, and F10 Quit.

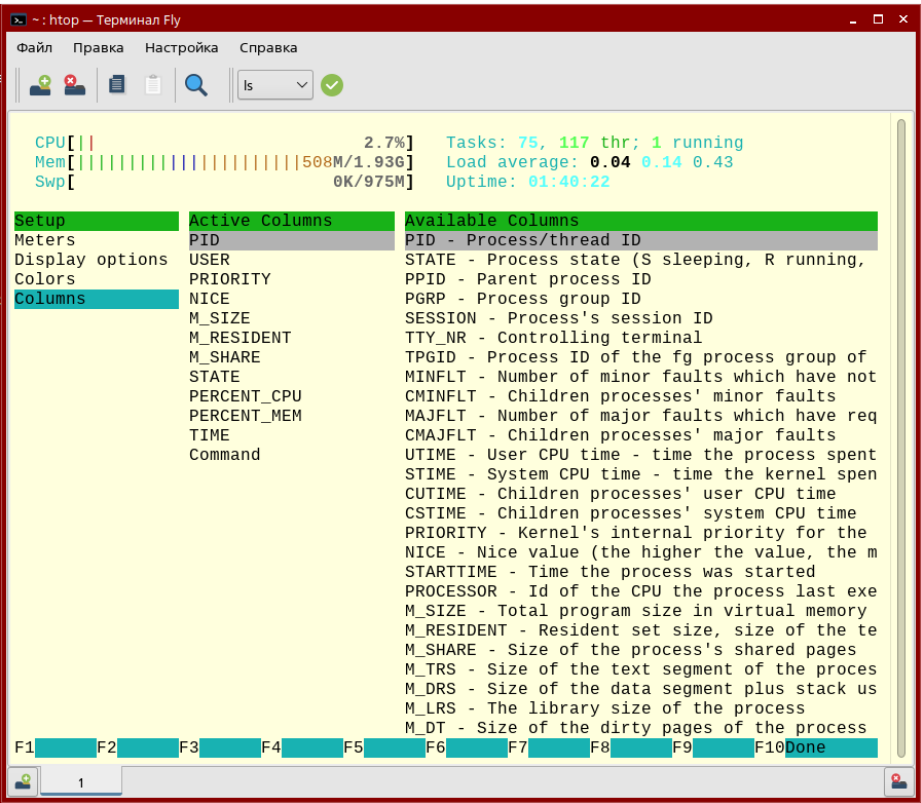
| PID | USER | PRI | NI | VIRT | RES | SHR | S | CPU% | MEM% | TIME+ | Command |
|------|-----------|-----|----|-------|-------|-------|---|------|------|---------|-----------------------|
| 925 | fly-dm | 20 | 0 | 595M | 116M | 42360 | S | 2.6 | 5.9 | 0:54.37 | /usr/lib/xorg/Xorg -b |
| 3153 | sa | 20 | 0 | 8180 | 4032 | 3408 | R | 1.9 | 0.2 | 0:00.34 | htop |
| 1085 | sa | 20 | 0 | 39868 | 27668 | 16808 | S | 0.0 | 1.4 | 0:05.07 | fly-wm |
| 1387 | sa | 20 | 0 | 358M | 64824 | 55628 | S | 3.9 | 3.2 | 0:22.76 | fly-term |
| 973 | fly-dm | 20 | 0 | 595M | 116M | 42360 | S | 0.6 | 5.9 | 0:09.16 | /usr/lib/xorg/Xorg -b |
| 1204 | sa | 20 | 0 | 274M | 2440 | 2100 | S | 0.6 | 0.1 | 0:22.80 | /usr/bin/VBoxClient - |
| 415 | messagebu | 20 | 0 | 9400 | 5864 | 4352 | S | 0.0 | 0.3 | 0:02.12 | /usr/bin/dbus-daemon |
| 943 | root | 20 | 0 | 279M | 928 | 820 | S | 0.0 | 0.0 | 0:00.81 | /usr/bin/VBoxDRMClien |
| 930 | root | 20 | 0 | 279M | 928 | 820 | S | 0.0 | 0.0 | 0:00.61 | /usr/bin/VBoxDRMClien |
| 1252 | sa | 20 | 0 | 482M | 49416 | 44176 | S | 0.0 | 2.4 | 0:01.24 | /usr/lib/x86_64-linux |
| 1320 | sa | 20 | 0 | 393M | 39892 | 35824 | S | 0.0 | 2.0 | 0:00.41 | /usr/lib/x86_64-linux |
| 1257 | sa | 20 | 0 | 264M | 45752 | 40996 | S | 0.0 | 2.3 | 0:00.94 | fly-notify-osd-servic |
| 1327 | sa | 20 | 0 | 279M | 53504 | 47328 | S | 0.6 | 2.6 | 0:00.43 | fly-search-panel --hi |
| 1388 | sa | 20 | 0 | 358M | 64824 | 55628 | S | 0.6 | 3.2 | 0:03.27 | fly-term |
| 1207 | sa | 20 | 0 | 274M | 2440 | 2100 | S | 0.6 | 0.1 | 0:22.80 | /usr/bin/VBoxClient - |
| 594 | root | 20 | 0 | 2360 | 1444 | 1316 | S | 0.6 | 0.1 | 0:08.46 | /usr/bin/fly-getexe |
| 1194 | sa | 20 | 0 | 210M | 2344 | 2008 | S | 0.0 | 0.1 | 0:07.12 | /usr/bin/VBoxClient - |
| 1212 | sa | 20 | 0 | 210M | 2452 | 2096 | S | 0.0 | 0.1 | 0:02.35 | /usr/bin/VBoxClient - |
| 1334 | sa | 20 | 0 | 220M | 23720 | 21240 | S | 0.0 | 1.2 | 0:00.89 | /usr/lib/x86_64-linux |
| 417 | root | 20 | 0 | 613M | 39684 | 23992 | S | 0.0 | 2.0 | 0:04.45 | /usr/sbin/syslog-ng - |
| 1272 | sa | 20 | 0 | 606M | 57436 | 49404 | S | 0.0 | 2.8 | 0:01.26 | kmix --keepvisibility |
| 2576 | sa | 20 | 0 | 334M | 45548 | 39280 | S | 0.6 | 2.2 | 0:06.07 | fly-su /usr/bin/fly-a |
| 3105 | root | 20 | 0 | 83532 | 6356 | 5080 | S | 0.0 | 0.3 | 0:00.85 | /sbin/dhclient -d -q |
| 1 | root | 20 | 0 | 100M | 11312 | 8688 | S | 0.0 | 0.6 | 0:01.79 | /sbin/init |
| 288 | root | 20 | 0 | 22740 | 6028 | 4624 | S | 0.0 | 0.3 | 0:00.21 | /lib/systemd/systemd- |
| 318 | root | 16 | -4 | 11380 | 1728 | 1508 | S | 0.0 | 0.1 | 0:00.00 | /sbin/auditd |

Интерфейс утилиты htop

В нижней части экрана расположена подсказка про доступные действия с помощью функциональных клавиш. Например, при нажатии **F2** мы попадем в меню настроек.



В меню **Columns** нам доступен выбор отображаемых столбцов и их описание.



Через клавишу **F1** нам доступна кратка справка. Полная справка доступна через man htop.

Утилита ps

Почти все те же действия можно выполнять и с помощью утилиты ps, но без удобного интерактивного режима. Вот некоторые её опции:

- `-e` – вывести на дисплей процессы всех пользователей. Не отображаются только процессы, не связанные с терминалом.
- `-a` – вывести информацию обо всех процессах, за исключением лидеров сессий и процессов, не ассоциированных с терминалом.
- `-t` – показывать только процессы из этого терминала.
- `-p` – показывать информацию только об указанном процессе.
- `-u` – вывод подробной информации в пользовательско-ориентированном формате.
- `-x` – показать процессы без управляющего терминала.
- `-o` – задать формат вывода информации.

Таким образом, команда `ps aux` выводит все активные процессы.

Для сортировки процессов можно воспользоваться параметром `--sort`, например, `ps aux --sort=%mem` или `ps aux --sort=%cpu`.

```
sa@astra:~$ ps aux --sort=%mem | tail -n 3
sa      1264  0.0  3.3 784012 68692 ?        Ssl  12:39   0:00 nm-applet
root    2599  0.0  3.8 671080 77856 ?        SNl  12:42   0:03 fly-admin-gmc --app
lication_name fly-admin-smc --plugin_path /usr/lib/x86_64-linux-gnu/fly-admin-smc --
app_icon fly-admin-smc --window_title Управление политикой безопасности --vertex_ele
ment_id LocalComputer
...
```

Команда `ps -eo euser,ruser,suser,fuser,f,comm,label` выведет информацию об атрибутах EUID, RUID, SUID.

```
sa@astra:~$ ps -eo euser,ruser,suser,fuser,f,comm,label
EUSER  RUSER  SUSER  FUSER  F COMMAND          LABEL
root   root   root   root   4 systemd          0:63:0:0
root   root   root   root   1 kthreadd          0:63:0:0
root   root   root   root   1 rcu_gp            0:63:0:0
...
sa      sa      sa      sa      4 systemd          0:63:0:0
sa      sa      sa      sa      5 (sd-pam)         0:63:0:0
sa      sa      sa      sa      4 fly-wm           0:63:0:0
...
```

С помощью утилиты ps можно посмотреть и потоки. Воспользуйтесь командами `ps -eLf` и `ps axms`.

```
sa@astra:~$ ps -eLf
UID      PID  PPID  LWP  C  NLWP  STIME  TTY          TIME CMD
root      1    0      1  0    1  12:38  ?            00:00:01 /sbin/init
. . .
sa      2768 1390  2770  0    3  12:48 pts/0      00:00:00 kcalc
root    3063   2   3063  0    1  14:04  ?            00:00:00 [kworker/u2:42-events_unb
```

```

root      3069      2  3069  0    1 14:04 ?          00:00:00 [kworker/0:2-events]
root      3077      1  3077  0    1 14:04 ?          00:00:00 /lib/systemd/systemd-jour
root      3105     419  3105  0    2 14:04 ?          00:00:01 /sbin/dhclient -d -q -sf
root      3105     419  3111  0    2 14:04 ?          00:00:00 /sbin/dhclient -d -q -sf
root      3146      2  3146  0    1 14:09 ?          00:00:00 [kworker/u2:0-events_unbo
root      3154      2  3154  0    1 14:15 ?          00:00:00 [kworker/u2:1-events_unbo
root      3175      2  3175  0    1 14:17 ?          00:00:00 [kworker/0:0-events]
sa        3182    1390  3182  0    1 14:23 pts/0      00:00:00 ps -elf

```

```
sa@astra:~$ ps axmu
```

```

USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0   0.5 103008 11312 ?        -    12:38   0:01 /sbin/init
root        -   0.0   -    -    -    -    -    Ss    12:38   0:01 -
. . .
root        -   0.1   -    -    -    -    -    Sl    14:04   0:01 -
root        -   0.0   -    -    -    -    -    Sl    14:04   0:00 -
root      3146   0.0   0.0     0     0 ?        -    14:09   0:00 [kworker/u2:0-event
root        -   0.0   -    -    -    -    -    I    14:09   0:00 -
root      3154   0.0   0.0     0     0 ?        -    14:15   0:00 [kworker/u2:1-event
root        -   0.0   -    -    -    -    -    I    14:15   0:00 -
root      3175   0.0   0.0     0     0 ?        -    14:17   0:00 [kworker/0:0-events
root        -   0.0   -    -    -    -    -    I    14:17   0:00 -
sa        3186   0.0   0.1  11112  3868 pts/0    -    14:28   0:00 ps axmu
sa        -   0.0   -    -    -    -    -    R+   14:28   0:00 -

```

Утилита pstree

С помощью утилиты `pstree` можно вывести дерево процессов. Для её использования необходим пакет `psmisc` `sudo apt install psmisc`.

```
sa@astra:~$ pstree
```

```

systemd--NetworkManager--dhclient--{dhclient}
      |               |      |
      |               |      |--2*[{NetworkManager}]
      |               |      |
      |--3*[VBoxClient--VBoxClient--3*[{VBoxClient}]]
      |--VBoxClient--VBoxClient--4*[{VBoxClient}]
      |--VBoxDRMClient--4*[{VBoxDRMClient}]
      |--VBoxService--8*[{VBoxService}]
      |--agetty
      |--alsactl
      |--anacron
      |--astra-orientati--2*[{astra-orientati}]
      |--at-spi2-registr--2*[{at-spi2-registr}]
      |--auditd--{auditd}
      |--avahi-daemon--avahi-daemon
      |--cron
      |--cupsd
      |--3*[dbus-daemon]
      |--2*[dbus-launch]
      |--fly-dm--Xorg--5*[{Xorg}]
      |      |
      |      |--fly-dm--fly-wm--astra-event-wat--{astra-event-wat}
      |      |      |
      |      |      |--at-spi-bus-laun--dbus-daemon
      |      |      |      |
      |      |      |      |--3*[{at-spi-bus-laun}]
      |      |      |      |
      |      |      |      |--baloo_file--2*[{baloo_file}]
      |      |      |      |
      |      |      |      |--fly-cups-watch--{fly-cups-watch}
      |      |      |      |
      |      |      |      |--fly-notify-osd--2*[{fly-notify-osd}]
      |      |      |      |
      |      |      |      |--fly-reflex-serv--3*[{fly-reflex-serv}]
      |      |      |      |
      |      |      |      |--fly-search-pane--2*[{fly-search-pane}]
      |      |      |      |
      |      |      |      |--fly-term--bash--3*[kcalc--2*[{kcalc}]]
      |      |      |      |      |
      |      |      |      |      |--pstree
      |      |      |      |      |
      |      |      |      |      |--2*[{fly-term}]
      |      |      |      |
      |      |      |      |--kmix--3*[{kmix}]
      |      |      |      |
      |      |      |      |--kscreend--2*[{kscreend}]
      |      |      |      |
      |      |      |      |--nm-applet--7*[{nm-applet}]
      |      |      |      |
      |      |      |      |--org_kde_powerde--5*[{org_kde_powerde}]
      |      |      |      |
      |      |      |      |--polkit-kde-auth--4*[{polkit-kde-auth}]
      |      |      |      |
      |      |      |      |--ssh-agent
      |      |      |      |
      |      |      |      |--su-to-root--fly-su--sudo--su_stub--fly-adm+

```

Ключ `-p` позволяет вывести PID процессов `ps tree -p`.

```
sa@astra:~$ pstree -p
systemd(1)─NetworkManager(419)─dhclient(3105)─{dhclient}(3111)
      │
      │   {NetworkManager}(455)
      │   {NetworkManager}(461)
      └─VBoxClient(1178)─VBoxClient(1179)─{VBoxClient}(1180)
      │
      │   {VBoxClient}(1181)
      │   {VBoxClient}(1182)
      ...
```

Поиск процессов

Утилита htop

Для поиска процессов в htop необходимо воспользоваться пунктом Search. Для этого нажмите клавишу **F3**. Вам станет доступна строка поиска. После выполнения поиска вы можете переходить от одного процесса к другому. Для выхода из режима поиска нажмите клавишу **Esc**.

```
1085 sa      20    0 39868 27316 16456 S  0.0  1.3  0:07.10 fly-wm
1324 sa      20    0 606M 57020 48988 S  0.0  2.8  0:00.65 kmix --keepvisibility
1334 sa      20    0 220M 23460 20980 S  0.0  1.2  0:01.28 /usr/lib/x86_64-linux
F3Next EscCancel Search: fly
```

В целях поиска может использоваться и фильтрация. Она доступна по нажатию клавиши **F4**.

Для очистки фильтра нажмите клавишу **Esc**.

Утилита ps

Для фильтрации вывода команды `ps` используйте встроенные опции: `-123` или `123`, `-p "1 2"` – вывод информации по PID процессу(ам).

```
sa@astra:~$ ps -1
  PID TTY          STAT       TIME COMMAND
    1 ?            Ss          0:01 /sbin/init

sa@astra:~$ ps 1
  PID TTY          STAT       TIME COMMAND
    1 ?            Ss          0:01 /sbin/init

sa@astra:~$ ps -p "1 2"
  PID TTY          STAT       TIME CMD
    1 ?            00:00:01 systemd
    2 ?            00:00:00 kthreadd
```

`-C <имя_команды>` – вывод информации по процессам, имя команды которых соответствует указанному, например, `ps -C bash`.

```
sa@astra:~$ ps -C bash
PID TTY      TIME CMD
1390 pts/0    00:00:00 bash
```

Используя опцию `-o pid`, можно вывести только колонку с PID найденных процессов, а используя `-o pid=`, только PID без названия колонки.

```
sa@astra:~$ ps -C bash -o pid
PID
1390

sa@astra:~$ ps -C bash -o pid=
1390
```

Другие опции фильтрации `ps` в справке `man 1 ps`.

Для дополнительной фильтрации можно использовать утилиту **grep**, например, `ps aux | grep tty`.

```
sa@astra:~$ ps aux | grep tty
root        619  0.0  0.0  5808  1768 tty1      Ss+  12:38   0:00 /sbin/agetty -o -p
-- \u --noclear tty1 linux
fly-dm      925  1.1  5.9 610196 119700 tty7      Ssl+  12:39   1:29 /usr/lib/xorg/Xorg
-br -novtswitch -quiet -keeptty :0 vt7 -logfile /var/log/fly-dm/Xorg.%s.log -seat se
at0 -auth /var/run/xauth/A:0-OOMBoc
sa          3369  0.0  0.0   6096   812 pts/0    S+   14:44   0:00 grep tty
```

Завершение процессов

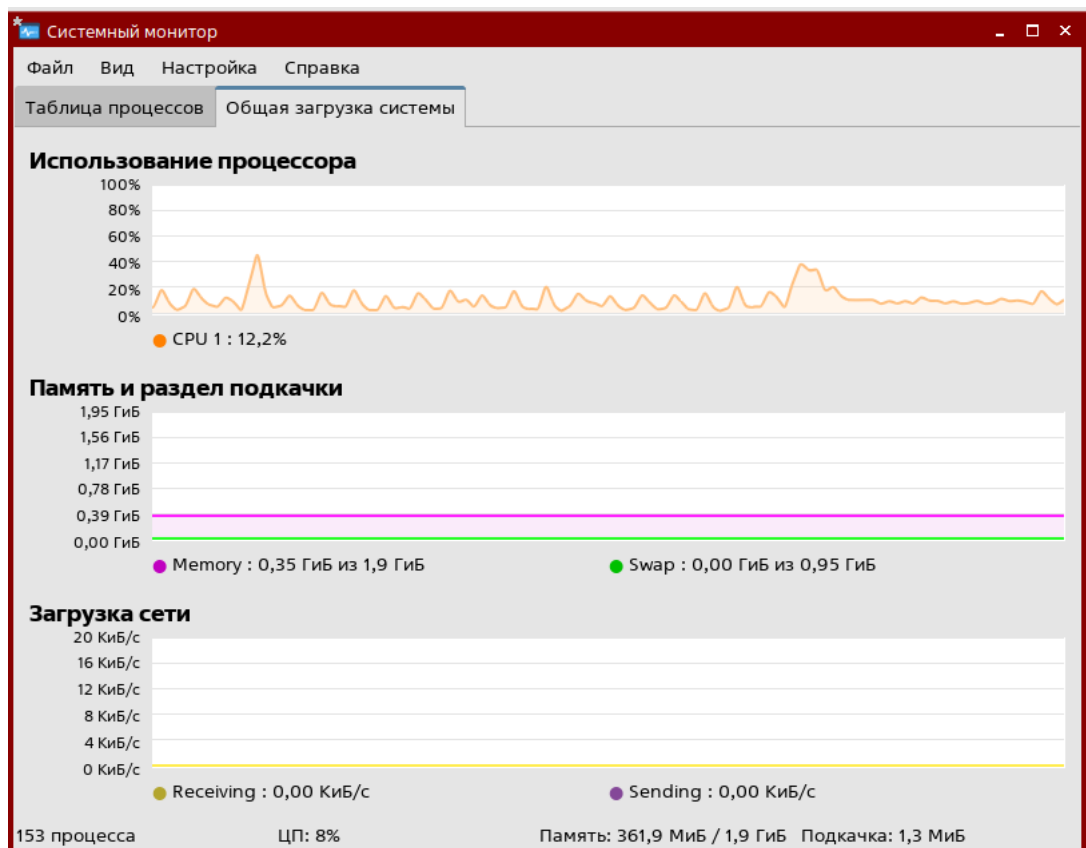
При работе с утилитой `htop` для отправки сигнала процессу необходимо нажать `F9`. По умолчанию будет выбран сигнал `SIGTERM`. Для завершения процесса с этим сигналом необходимо нажать `F9` и `Enter` (или перед нажатием `Enter` вы можете выбрать по необходимости другой сигнал).

Управление через графику в Системном мониторе

Системный монитор напоминает «Менеджер задач» в Windows, что было сделано специально для удобства пользователей и администраторов. Он имеет интуитивно понятный интерфейс и не должен вызывать сложностей при своем использовании.

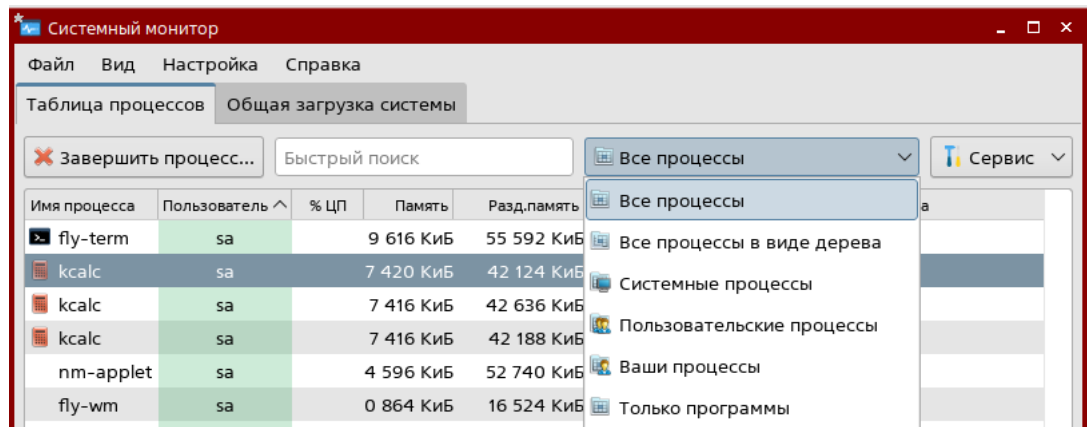
| Системный монитор | | | | | | | |
|--|-----------------|----------|-----------|-------------|-----------------|----------|----------|
| Файл Вид Настройка Справка | | | | | | | |
| Таблица процессов | | | | | | | |
| Общая загрузка системы | | | | | | | |
| <div> <div>✕ Завершить процесс...</div> <div>Быстрый поиск</div> <div>Все процессы</div> <div>Сервис</div> </div> | | | | | | | |
| Имя процесса | Пользователь | % ЦП | Память | Разд.память | Заголовок окна | Загрузка | Передача |
| kworker/... | root | известны | | | | | |
| kworker/... | root | известны | | | | | |
| irq/18-v... | root | | | | | | |
| kworker/... | root | известны | | | | | |
| kworker/... | root | известны | | | | | |
| astra-ori... | astra-orient... | | 2 444 КиБ | 20 932 КиБ | | | |
| avahi-da... | avahi | | 436 КиБ | 3 192 КиБ | | | |
| avahi-da... | avahi | | 436 КиБ | | | | |
| Xorg | fly-dm | 5% | 7 596 КиБ | 42 044 КиБ | | | |
| (sd-pam) | fly-dm | | 2 776 КиБ | | | | |
| systemd | fly-dm | | 1 316 КиБ | 7 400 КиБ | | | |
| dbus-da... | messagebus | | 1 512 КиБ | 4 336 КиБ | | | |
| fly-term | sa | | 9 616 КиБ | 55 592 КиБ | ~ : bash — T... | | |
| kcalc | sa | | 7 420 КиБ | 42 124 КиБ | KCalc | | |
| kcalc | sa | | 7 416 КиБ | 42 636 КиБ | KCalc | | |
| kcalc | sa | | 7 416 КиБ | 42 188 КиБ | KCalc | | |
| nm-applet | sa | | 4 596 КиБ | 52 740 КиБ | | | |
| fly-ur... | sa | | 0 864 КиБ | 16 534 КиБ | | | |
| <div> <div>153 процесса</div> <div>ЦП: 6%</div> <div>Память: 361,9 МиБ / 1,9 ГиБ</div> <div>Подкачка: 1,3 МиБ</div> </div> | | | | | | | |

Системный монитор: таблица процессов

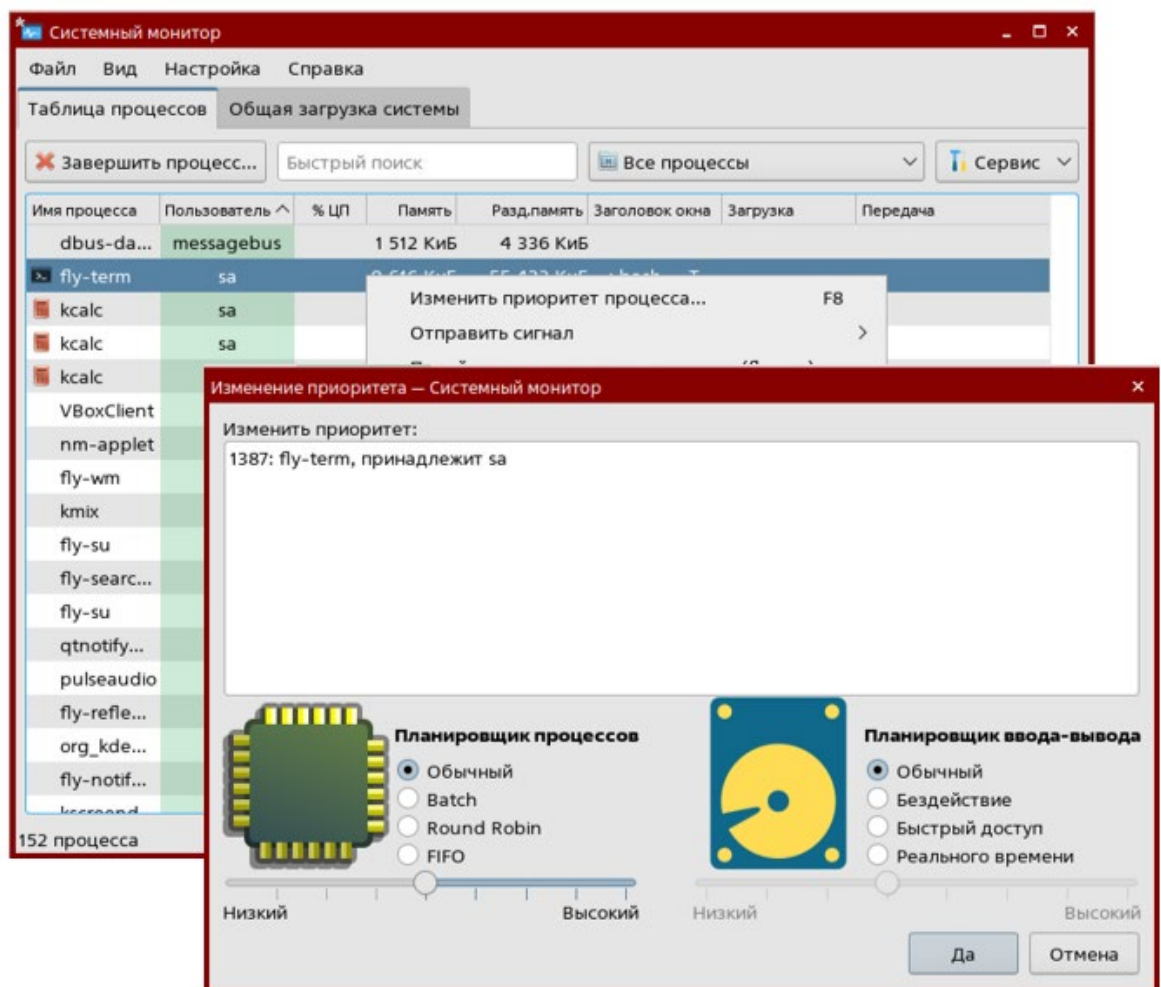


Системный монитор: общая загрузка системы

Управление доступно через контекстное меню и органы управления. Для поиска процессов доступна строка быстрого поиска, для завершения процесса – соответствующая кнопка над списком задач. С помощью выпадающих списков вы можете отфильтровать процессы по категориям.



В контекстном меню процесса можно перейти к окну этого приложения, вывести подробную информацию о процессе, перейти к родительскому процессу, отправить сигнал процессу или изменить его приоритет.



С подробной справкой вы можете ознакомиться в справочной системе **Alt + F1**.

Практические задания

Задание 1

Внимание

Для корректного выполнения первого задания необходимо выделить виртуальной машине не менее 2 vCPU.

- Выведите на экран все потоки процесса syslog-ng, используя команду `ps`.
- Попробуйте выполнить то же задание, используя утилиту `htop`.

Задание 2

Выведите на экран информацию, в том числе PID, %MEM, EUID, RUID, SUID и Command о топ-15 процессов по использованию памяти, используя команду `ps`. Попробуйте выполнить то же задание, используя утилиту `htop`.

Задание 3

Создайте файл в домашнем каталоге `test_nano` и откройте его текстовым редактором nano. В этом же терминале, используя командную строку, найдите этот процесс и повысьте его приоритет до 15. Верните из фона процесс nano и закройте его. Откройте файл `test_nano` текстовым редактором `nano`, задав ему приоритет 25 при создании. Завершите процесс `nano`, отправив ему сигнал SIGKILL.

Задание 4

Тут вам необходимо будет управлять приоритетами процессов. Для наглядности работы по управлению приоритетом процессов используйте скрипт, который создаст нагрузку на CPU.

Создайте файлы `~/scripts/resource-demanding1.sh` и `~/scripts/resource-demanding2.sh` со следующим содержанием (два файла нам нужны, чтобы впоследствии различать процессы не только по PID, но и по команде запуска):

```
#!/bin/bash
while [ 1 == 1 ]; do
  for x in {1..60}; do
    let "res=2**x"
  done
done
```

- Запустите утилиту `htop` в новом терминале и отсортируйте процессы по использованию CPU.

- Сделайте файлы исполнимыми для владельца и запустите их с помощью команды `taskset` в **фоновом режиме на первом ядре** процессора. Сколько процентов первого ядра процессора используют созданные процессы?
- Измените приоритет первого процесса, понизив его до 25, 30, 35 и 39 последовательно. Посмотрите, как менялось при этом использование процессора.
- Повысьте приоритет первого процесса до 15, 10, 5, 0 последовательно. Посмотрите, как менялось при этом использование процессора?
- Задайте приоритет второму процессу так, чтобы он потреблял не менее 75% ресурсов первого ядра CPU, не изменяя при этом приоритет первого процесса (0).
- Завершите процессы, отправив первому сигнал SIGINT, а второму SIGKILL.

Задание 5

Задайте ограничение для текущей оболочки на количество запущенных потоков. Задайте ограничение, превышающее текущее количество запущенных потоков пользователем на 3, запустите несколько раз новый процесс `bash`, убедитесь, что лимит действует, и закройте терминал с ошибкой.

Задание 6

Получите информацию из псевдофайловой системы `/proc`:

О процессе `syslog-ng`:

- строка запуска процесса
- окружение процесса
- сведения об объемах данных, прочитанных и записанных процессом в хранилище информации
- лимиты процесса
- информация о статусе процесса

Об ОС:

- список параметров, которые были переданы ядру при загрузке
- сведения о всех установленных процессорах
- информация о состоянии памяти
- перечень устройств в системе
- перечень файловых систем, поддерживаемых ядром ОС
- перечень смонтированных файловых систем
- список разделов подкачки

Вопросы

1. Дополните фразу «проблема иерархической структуры процессов в современных Linux дистрибутивах заключается в»
2. Сколько состояний есть у процесса?
3. Чем больше значение параметра приоритета процесса, тем...
4. Перечислите все команды, позволяющие управлять процессами в фоновом режиме работы.
5. Соотнесите команду для назначения политики планирования и тип политики:
 - A. SCHED_OTHER
 - B. SCHED_RR
 - C. SCHED_FIFO
1. `sudo chrt -f {priority} {PID}`
2. `sudo chrt -o -p 0 {PID}`
3. `sudo chrt -r -p {priority} {PID}`
6. Какая команда выведет процессы и потоки?
7. Какое назначение у сигналов SIGINT, SIGKILL, SIGCHILD, SIGSTOP, SIGSTP, SIGCONT, SIGTERM?
8. Какая команда выполнит поиск по имени команды, использующейся при запуске процесса?
9. Что содержит в себе файл `/proc/cmdline/`?
10. Что делает команда `ulimit -u <число>`?