Операционные системы (6 семестр)

Лекция 2.1. Процессы и сигналы

Ассистент, к.т.н. Митричев Иван Игоревич

Москва 2018

План лекции

- Многозадачность
- Процессы и задания
- Системные вызовы
- Структура процесса
- Идентификаторы процесса
- Категории процессов
- Фоновый режим выполнения заданий
- Жизненный цикл процесса
- Мониторинг процессов
- Псевдофайловая система /proc
- Сигналы. Перехват и обработка сигналов в командной оболочке bash
- Управление приоритетом процессов.

Многозадачность

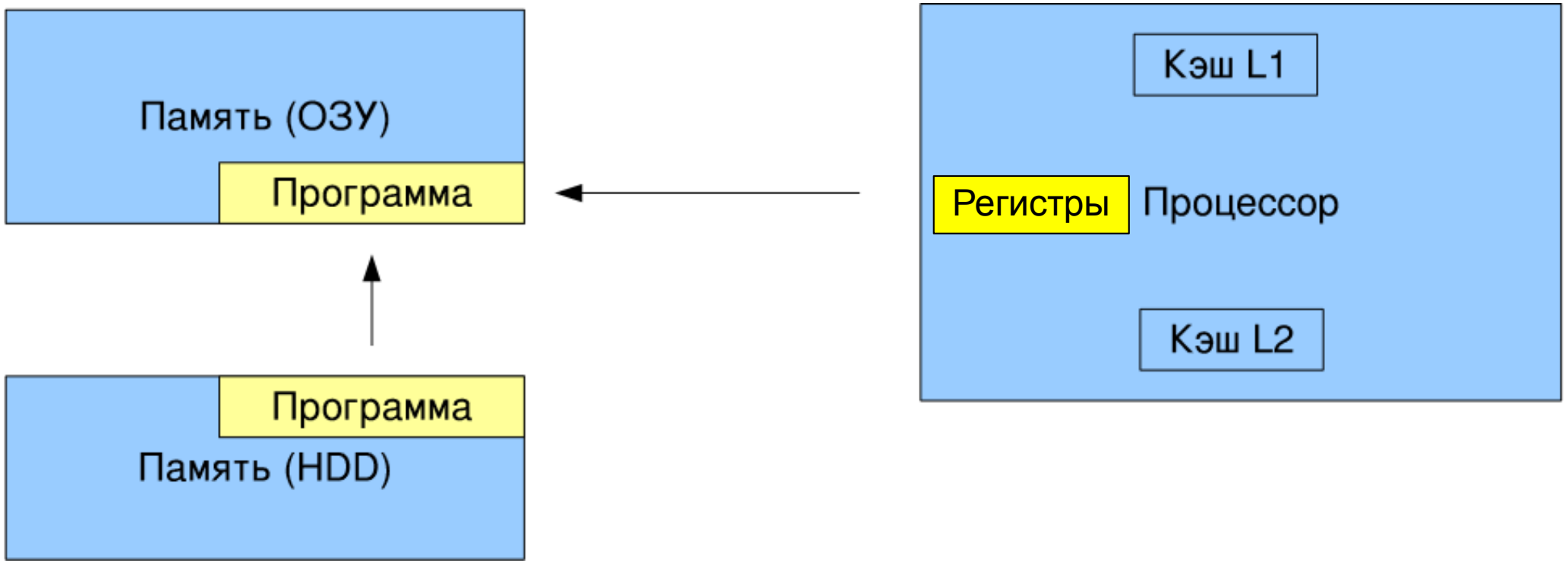
Многозадачность – это одновременное выполнение ряда задач. Каждую из задач принято называть процессом операционной системы. Многозадачность на одноядерных вычислительных устройствах осуществляется за счет механизма переключения контекста – каждый из процессов получает небольшое количество времени процессора, после чего его данные выгружаются из регистров, а данные другого процесса загружаются.

Переключение контекста происходит с фиксированными временными интервалами (упреждающая многозадачность), или по сигналу от программы, разрешающему ее прерывания выполнения (кооперативная многозадачность).

В GNU/Linux многозадачность основана на следующих принципах:

- Используется механизм переключения контекста и упреждающая многозадачность;
- Используется механизм изменения соотношения времени работы процессов за счет назначения приоритетов.
- Процессы с большим приоритетом вытесняют процессы с меньшим приоритетом (вытесняющая многозадачность)

Виды памяти компьютера и процессы

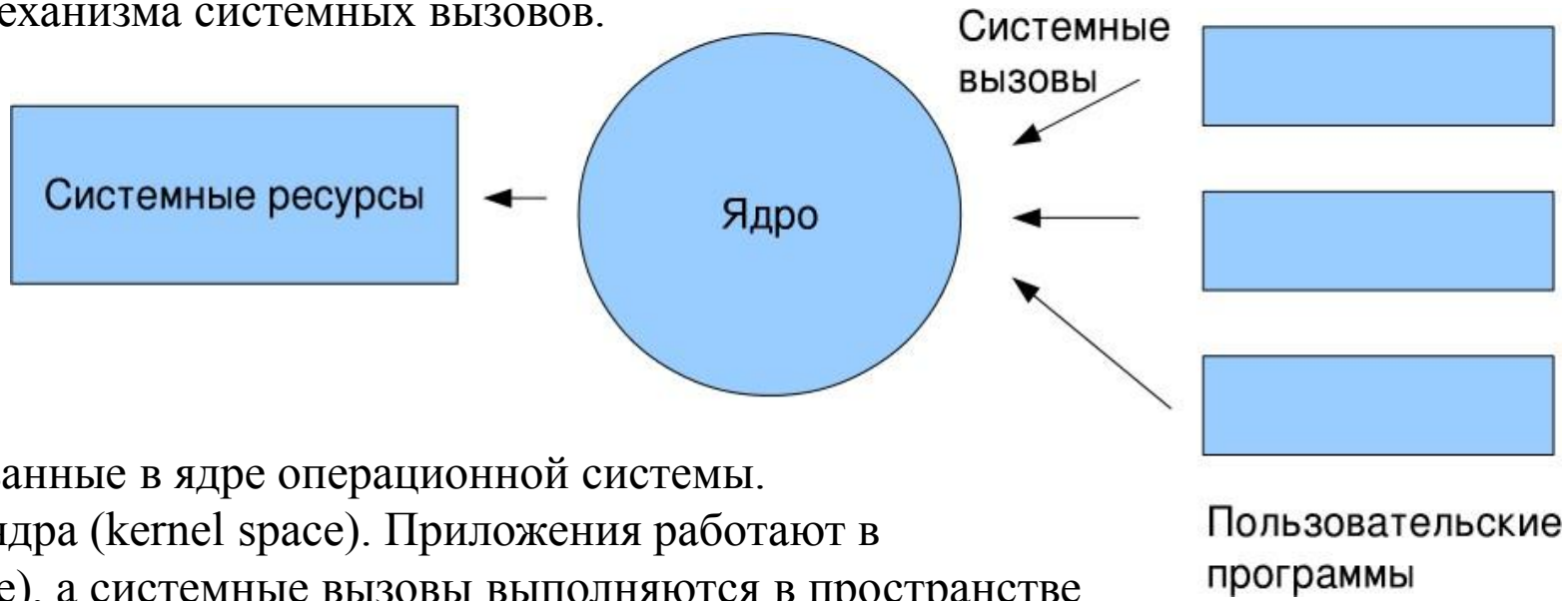


Программа есть файл, предназначенный для исполнения. Процесс – экземпляр программы, исполняемый под управлением операционной системы. Задача (job) – команда, запущенная пользователем (в командной оболочке). Одна задача может приводить к запуска ряда процессов. Задачи могут выполняться удаленно, а также ставиться в очередь задач (job queue).

Ядро и системные вызовы

Каждый процесс работает в своем виртуальном адресном пространстве, получить доступ к адресному пространству другого процесса могут только некоторые системные процессы. Пользовательские процессы могут взаимодействовать с ядром посредством механизма системных вызовов.

Ядро Linux – основная программа, которая обеспечивает взаимодействие пользовательских программ и оборудования. Файл ядра в Ubuntu хранится в папке /boot и называется vmlinuz-номер_версии.



Системные вызовы – функции, реализованные в ядре операционной системы. Процессы ядра работают в пространстве ядра (kernel space). Приложения работают в пользовательском пространстве (user space), а системные вызовы выполняются в пространстве ядра.

time <программа> ←

- определить общее время работы, время работы на стороне пользователя и на стороне ядра.

```
$ time ls
real    0m0.103s
user    0m0.052s
sys     0m0.049s
```

←

- реальное время работы программы updatedb, время работы на стороне пользователя и на стороне ядра.

Структура процесса

Образ процесса в оперативной памяти включает в себя
- Данные пользователя.
- Программа пользователя.
- Системный стек (пространство ядра).
- Блок управления процессом.

Процесс имеет ряд атрибутов, которые можно просмотреть с помощью команды ps:
-Идентификатор процесса (PID): уникальный номер для обозначения процесса.
-Идентификатор родительского процесса (PPID).
-Пользовательский приоритет (число NICE).
-Терминал (TTY): терминал, к которому подключен процесс.
-ID реального и эффективного пользователя (RUID и EUID, обычно совпадают, отличаются при использовании бита доступа SUID), ID реальной и эффективной группы (RGID и EGID): эффективная группа отличается при использовании бита доступа SGID.

Для каждого процесса создается свой каталог в псевдофайловой системе /proc. Здесь также можно просмотреть атрибуты процесса.

Псевдофайловая система /proc

/proc – псевдофайловая система, отражающая дерево процессов в виде дерева директорий, имеющих название, соответствующее PID процесса. Для некоторых процессов ядра можно изменять параметры.

```
bash-4.4$ ls /proc
1           cgroups     driver       ipmi        kpageflags  mtrr          slabinfo                t
imer_stats
17          cmdline     execdomains  irq         loadavg     net           softirqs                t
ty
23          consoles    fb           kallsyms    locks       pagetypeinfo  stat                    u
ptime
25          cpuinfo     filesystems  kcore       mdstat      partitions    swaps                   v
ersion
26          crypto      fs           key-users   meminfo     sched_debug   sys                     v
mallocinfo
acpi        devices     interrupts   keys        misc        schedstat     sysrq-trigger           v
mstat
buddyinfo   diskstats   iomem        kmsg        modules     scsi          sysvipc                 z
oneinfo
bus         dma         ioports      kpagecount  mounts      self          timer_list
```

В каждой директории есть файлы

- cmdline – командная строка процесса;
- status – подробная информация о статусе процесса;
- fd – мониторинг файлов, открытых процессом;
- cwd – ссылка на текущий рабочий каталог процесса;
- environ – окружение.

Создание процесса

Механизм порождения процессов – используется системный вызов fork() для дублирования процесса. В дочернем процессе обычно производят системный вызов exec(), чтобы запустить отдельный, независимый процесс, а в родительском - системный вызов wait(), чтобы ждать сигнала завершения от дочернего процесса.

У каждого процесса есть ровно один родитель, но может быть несколько потомков, или, дочерних процессов. Поэтому, систему процессов удобно изображать в виде дерева.

Если родительский процесс завершается раньше дочернего, то процесс-сирота (orphaned process) наследуется процессом init (PID = 1). Если дочерний процесс завершается раньше, чем родительский процесс успел вызвать wait(), то дочерний помечается как zombie (defunct).

Число процессов в системе ограничено. Узнать максимально поддерживаемое число процессов для запущенной ОС Linux:

cat /proc/sys/kernel/pid_max

По спецификации Linux, 4194304 – предел для архитектуры x86_64, а 32767 – для архитектуры x86. Максимальное значение PID при этом на единицу меньше.

Идентификаторы процесса

- PID (Process ID) – уникальный порядковый номер процесса;
- PPID (Parent Process ID) – PID родительского процесса, позволяющий выстроить иерархию процессов;
- UID (User ID) – ID пользователя, от имени которого выполняется процесс;
- GID (Group ID) – ID группы пользователей, от имени которой выполняется процесс.

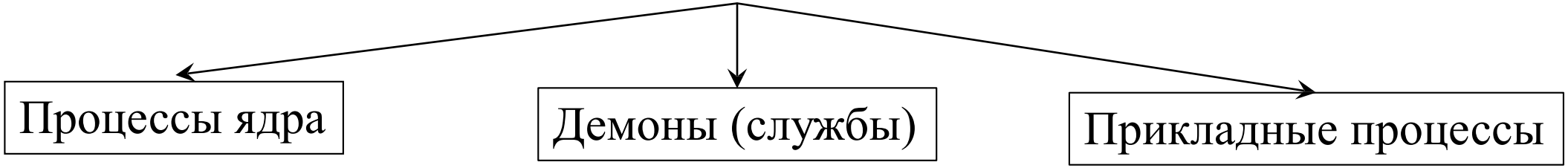
```
bash-4.4$ id
uid=31185 gid=31185 groups=31185
```

```
bash-4.4$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S 31185    27     1  0  80   0 -  7429 wait   pts/2    00:00:00 sh
0 S 31185    29    27  0  80   0 -  7429 wait   pts/2    00:00:00 bash
0 R 31185    33    29  0  80   0 -  8696 -      pts/2    00:00:00 ps
```

По умолчанию, процесс связывается с терминалом, из которого запущен (pts/2 на рисунке)

tty ← • терминал, на котором вы вошли в сеанс.

Категории процессов



Выполняются в пространстве ядра. Располагаются в оперативной памяти и не имеют соответствующих программ в виде исполняемых файлов. Соответствующий код находится в модулях ядра.

Пример – процесс упорядочивания процессов в очереди (scheduling).

Фоновые неинтерактивные процессы. Имеют соответствующие исполняемые файлы. Предназначены для обслуживания соединений по какому-либо сетевому протоколу или регулярное выполнение рутинных операций в системе.

Все остальные. Порождаются в рамках сеанса работы пользователя.

Фоновый режим выполнения заданий

<команда> & ← • запуск задания в фоновом режиме;

jobs ← • статус фоновый заданий.

+ и – для индикации последнего и предпоследнего заданий

```
[1]   Выполняется  sleep 50 &
[2]-  Выполняется  sleep 150 &
[3]+  Выполняется  sleep 70 &
```

disown -h номер_задания ← • запуск с «отвязкой» команды от родительской оболочки;

nohup ./script.sh & ← • «отвязать» от оболочки уже запушенную в фоновом режиме команду;

fg <номер задания> ←

fg <первые буквы команды> ←

• перевести задание в интерактивный режим;

<Ctrl>+<Z>, затем bg <номер задания> ← • перевести задание в фоновый режим;

kill %<номер задания> ← • завершение работы фонового задания.

Программа Screen

screen — запуск нового терминала Screen;

screen -ls — вывести список запущенных экземпляров screen;

<Ctrl>+<a>+<d> — выйти из screen (screen продолжит работу);

screen –r [номер_процесса] — вновь присоединиться к ранее запущенной сессии screen. Если сессия - единственна, номер процесса можно опустить;

screen -R — вновь присоединиться к ранее запущенной сессии screen, или создать новую сессию, если ни одной сессии не запущено.

screen –d [номер_процесса] — отсоединить сессию screen от терминала. Команду следует исполнять извне запущенной сессии при потере управления терминалом для дальнейшего подключения сессии к другому терминалу и использованию.

screen –d –r [номер_процесса] — отсоединить сессию screen от терминала и вновь присоединить к текущему терминалу

Мониторинг процессов

ps ←
- вывести информацию о процессах, связанных с текущим терминалом

PID – ID процесса, TTY – терминал, TIME – суммарное процессорное время;

ps -f ←
- вывести более подробную информацию UID – пользователь, PPID – ID родителя, C – уровень загрузки процессора, STIME – время запуска процесса; CMD – командная строка процесса;

ps -l ←
- вывести еще более подробную информацию;

ps -e, ps -A, ps aux ←
- вывести информацию о всех процессах в системе.

top ←
- вывести обновляющийся список процессов.

Горячие клавиши top: q – выход; k – послать сигнал процессу; i – отключить вывод неактивных процессов; f – показать доступные поля.

```
sh-4.4$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
32388         1  5.3  0.0 1342600 51704 ?        Ssl  22:36   0:01 --CODINGGROUND--
32388        11  0.0  0.0  29716  2272 pts/0    Ss   22:36   0:00 sh
32388        17  0.0  0.0  29716  2044 pts/0    S+   22:36   0:00 /bin/bash
32388        18  0.0  0.0  29716  2264 pts/1    Ss   22:36   0:00 sh
32388        20  0.0  0.0  39256  1660 pts/1    R+   22:36   0:00 ps aux
```

VSZ – объем используемой процессом виртуальной памяти,
RSS – объем используемой физической памяти.
%CPU – процент использования ядра CPU
%MEM – процент использования оперативной памяти

Команда ps -l

Длинный формат вывода

```
bash-4.4$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S 31185    27     1  0  80   0 -  7429 wait   pts/2    00:00:00 sh
0 S 31185    29    27  0  80   0 -  7429 wait   pts/2    00:00:00 bash
0 R 31185    33    29  0  80   0 -  8696 -      pts/2    00:00:00 ps
```

F – флаги процесса;
S – статус:
•D – приостановлен и не может быть прерван;
•R – выполняется или находится в очереди;
•S – приостановлен;
•T – трассируется;
•Z – defunct (zombie)
NI (Nice Number) – пользовательский приоритет процесса;
SZ – общее количество памяти, занимаемое процессом;
WCHAN –идентификатор системного вызова.

Фильтрация списка процессов

`ps -C bash` ←

- вывести процессы, содержащие в команде запуска заданную строку;

`pgrep <имя команды>` ←

- получить PID процесса по заданной строке .

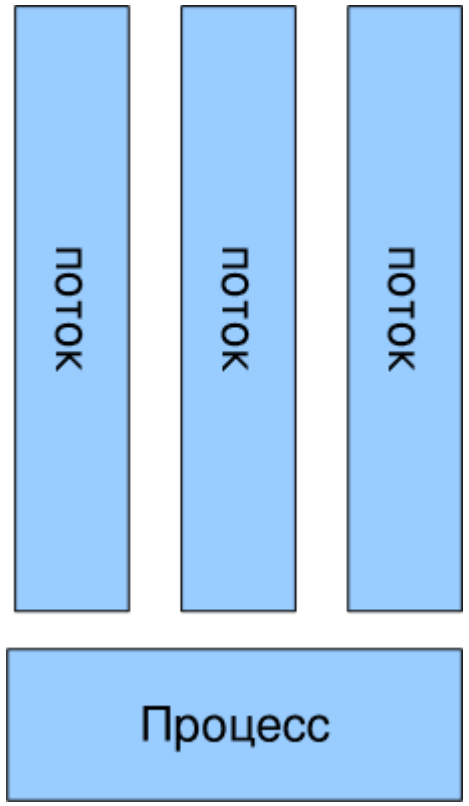
`pgrep -l <имя команды>` ←

- вывести также название процесса.

`pstree` ←

- просмотр списка процессов в виде дерева.

Потоки исполнения (threads)



Процесс	Поток
Независим	Является частью процесса
Имеет собственное адресное пространство	Делит одно адресное пространство с другими потоками процесса,
Имеет свое состояние (переменные)	В рамках процесса используют общее состояние (переменные)
Взаимодействуют через специальные механизмы связи	Быстро взаимодействуют

ps –LC sh ← получить список потоков для команд «sh».

LWP – идентификатор потока.

```
sh-4.4$ ps -LC sh
  PID   LWP TTY          TIME CMD
   15    15 pts/0    00:00:00 sh
   22    22 pts/1    00:00:00 sh
```

Сигналы

Обмен сигналами – механизм межпроцессного взаимодействия, позволяет реагировать процессам на события и оповещать друг друга о их наступлении.

Стандартные сигналы Linux обознаются целыми числами, начиная от единицы.

`kill -l` ← • получить список доступных сигналов;

`man 7 signal` ← • подробная информация о сигналах.

SIGHUP (1) - генерируется при возникновении проблемы с управляющим терминалом или завершении управляющего процесса;

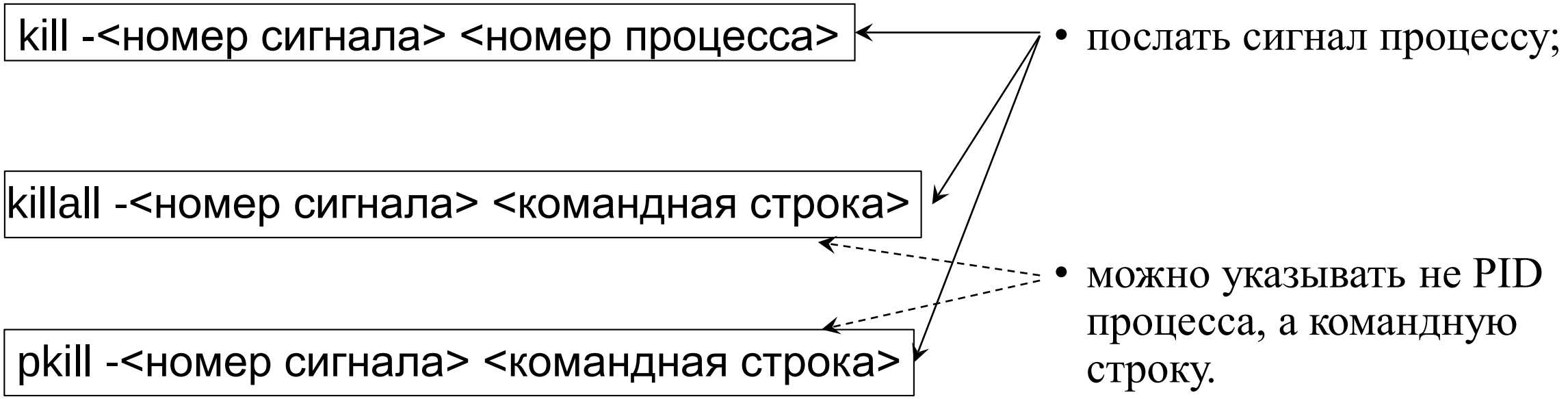
SIGINT (2) - сигнал, посылаемый при нажатии [CTRL] + [c]. Завершает работу процесса в терминале;

SIGQUIT (3) - генерируется при нажатии пользователем комбинации клавиш выхода [Ctrl] + [d];

SIGKILL (9) - немедленное прекращение работы процесса без действий по очистке памяти;

SIGTERM (15) - сигнал по умолчанию для завершения программ.

Передача сигналов



pkill разрешает использовать произвольную часть командной строки процесса в качестве аргумента. Поэтому, опаснее, чем killall (можно непреднамеренно завершить некоторые процессы, для которых часть командной строки совпала с введенным шаблоном).

Перехват и обработка сигналов в bash

`trap <command> <signal>` ← • перехватить сигнал и отреагировать на него;

`trap -p` ← • вывести список ловушек;

`trap <signal>` ← • удалить ловушку.

Пример

```
$ trap "echo Oops" INT
$ trap -p
trap -- 'echo Oops‘
$ <Ctrl>+<C>
Oops
```

Какие процессы открыли файл?

fuser
lsof

- определить какие процессы открыли заданный файл.

lsof ~/1.txt
- определить, какой процесс открыл файл 1.txt;

lsof +D ~/dir/
- вывести список всех открытых файлов в директории ~/dir (рекурсивно);

lsof +d ~/dir/
- вывести список всех открытых файлов только в самой директории ~/dir, но не глубже;

lsof -c bash
- найти все файлы, открытые процессом(-ами) с заданным именем (bash);

lsof -p 10989
- найти все файлы, открытые процессом с PID=10989.