Московский Физико-Технический Институт
Физтех-школа Аэрокосмических технологий
Институт Аэромеханики и Летательной Техники

МФТИ

ФАКТ

Архитектура Компьютера и Операционные Системы. Часть 2. Основы операционных систем

Лекция 10: Иерархия процессов. Сигналы

Новиков Андрей Валерьевич
д.ф.-м.н.

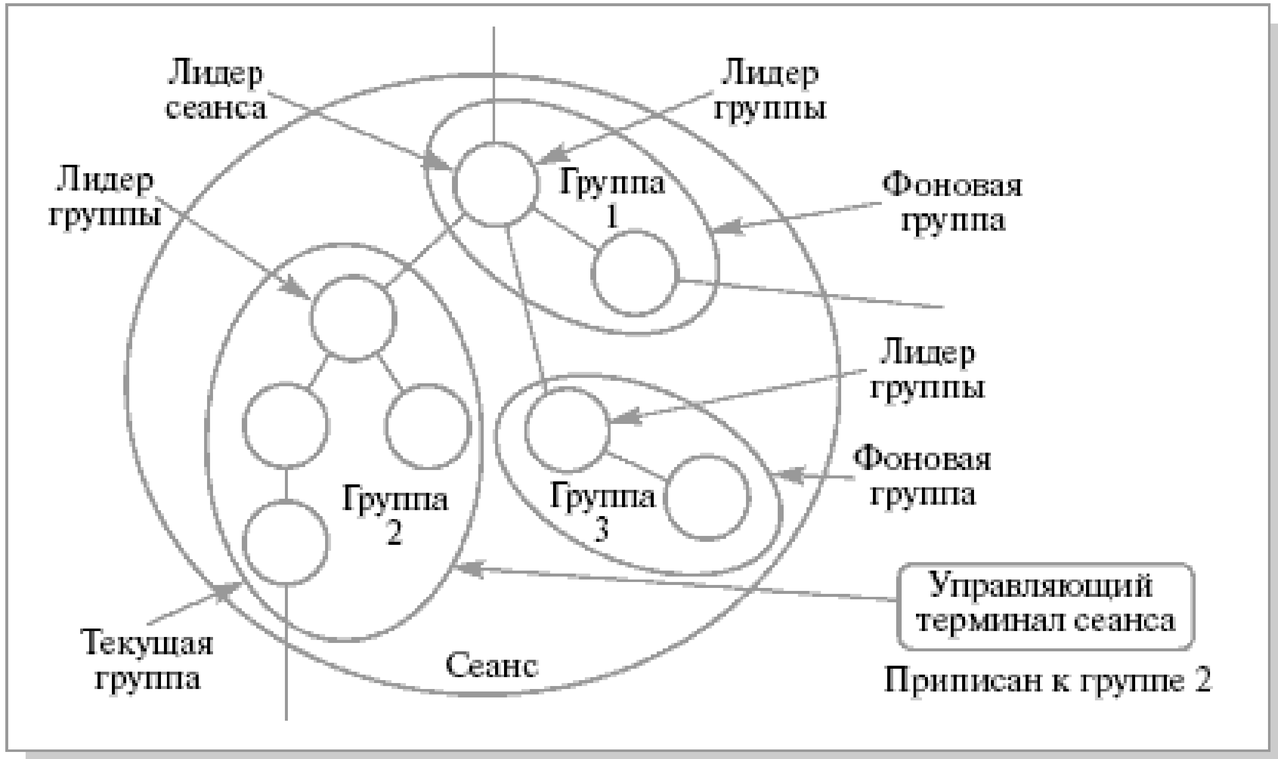
Жуковский

Иерархия процессов

группы/семьи и сеансы/кланы

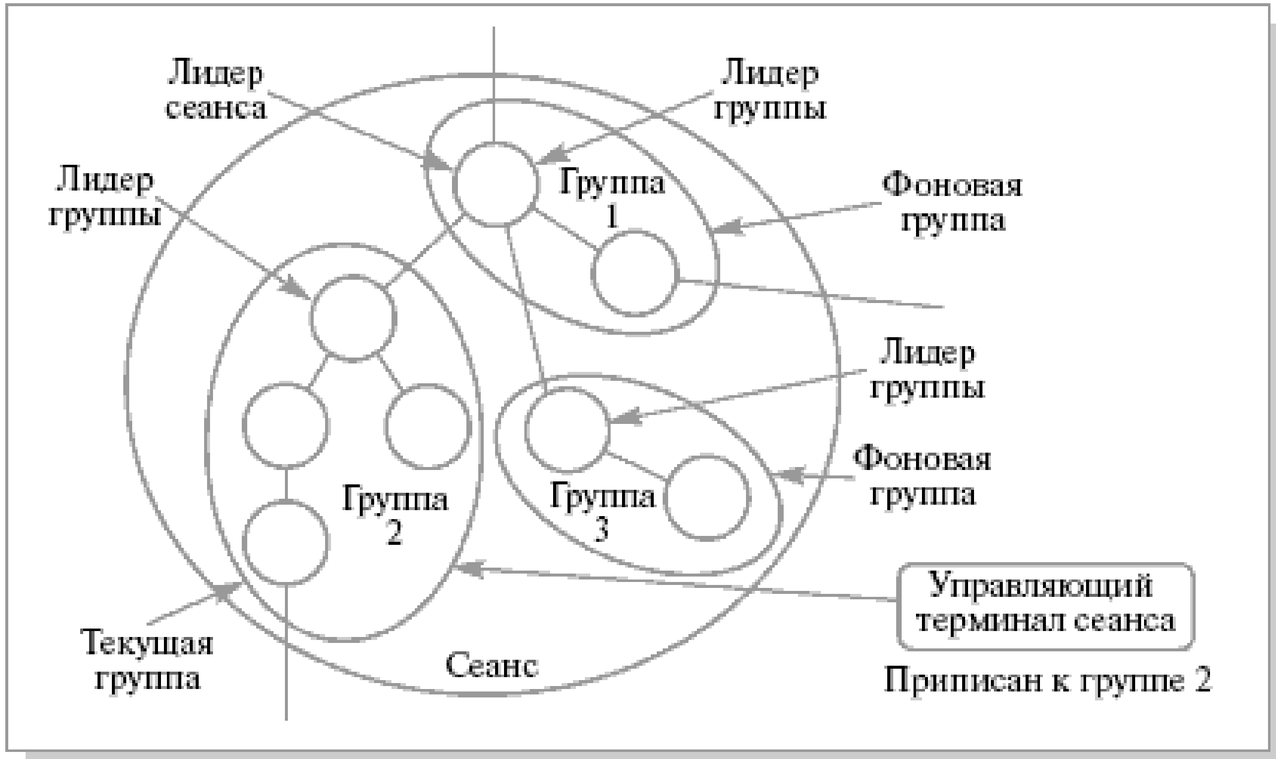
Иерархия процессов в ОС

- Все процессы связаны родственными отношениями и образуют **генеалогическое дерево** или лес из таких деревьев. Все эти деревья принято разделять на группы процессов, или семьи.
- **Группа процессов** включает в себя один или более процессов и существует, пока в группе присутствует хотя бы один процесс.
  - Процесс обязательно включен в какую-нибудь группу
  - При рождении процесс попадает в ту же группу, в которой находится его родитель
  - Процессы могут мигрировать из группы в группу
  - Многие системные вызовы могут быть применены ко всем процессам в некоторой группе.
  - **Лидер группы** – процесс с номером группы



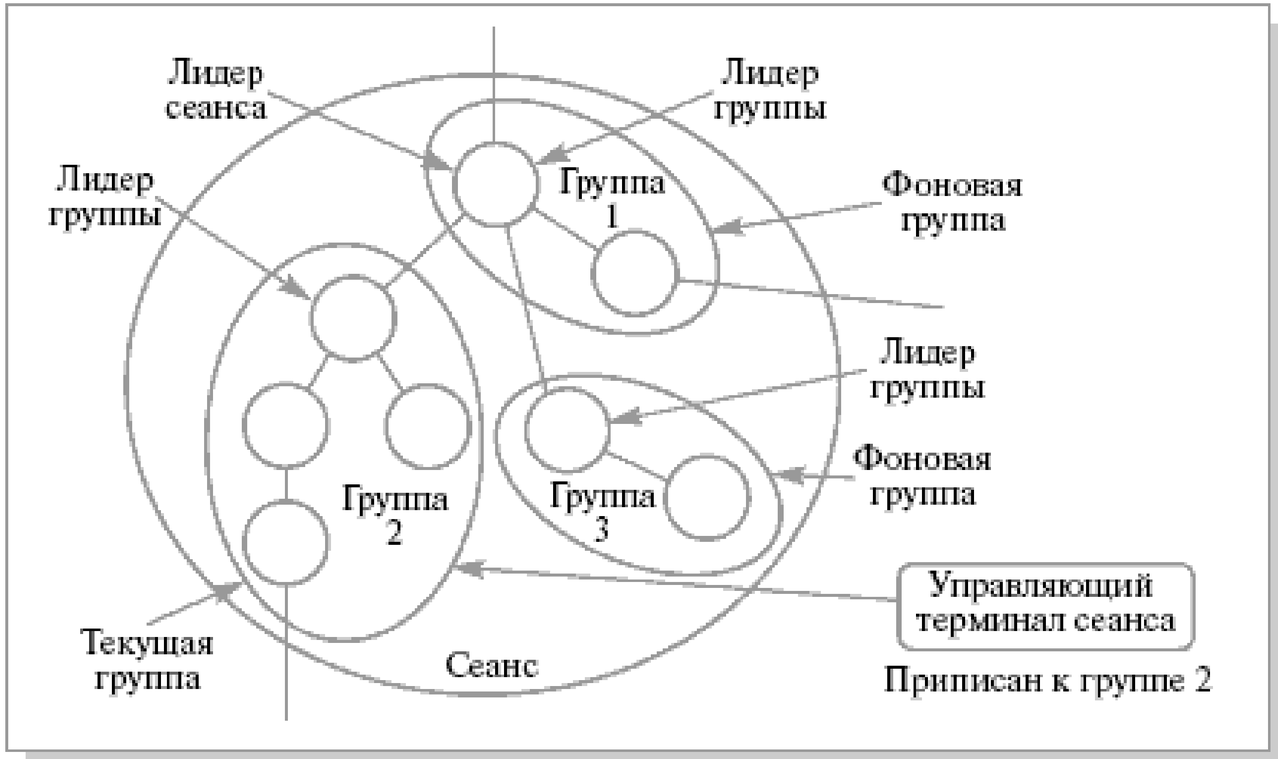
Иерархия процессов в ОС

- Группы процессов объединяются в **сеансы** («кланы семей»).
  - Понятие сеанса изначально было введено в UNIX для логического объединения групп процессов, созданных в результате каждого входа и последующей работы пользователя в системе.
  - С каждым сеансом может быть связан терминал, называемый **управляющим терминалом сеанса**, через который обычно и общаются процессы сеанса с пользователем.
  - Сеанс может иметь **0 или 1** управляющий терминал.
  - Терминал не может быть управляющим для нескольких сеансов.



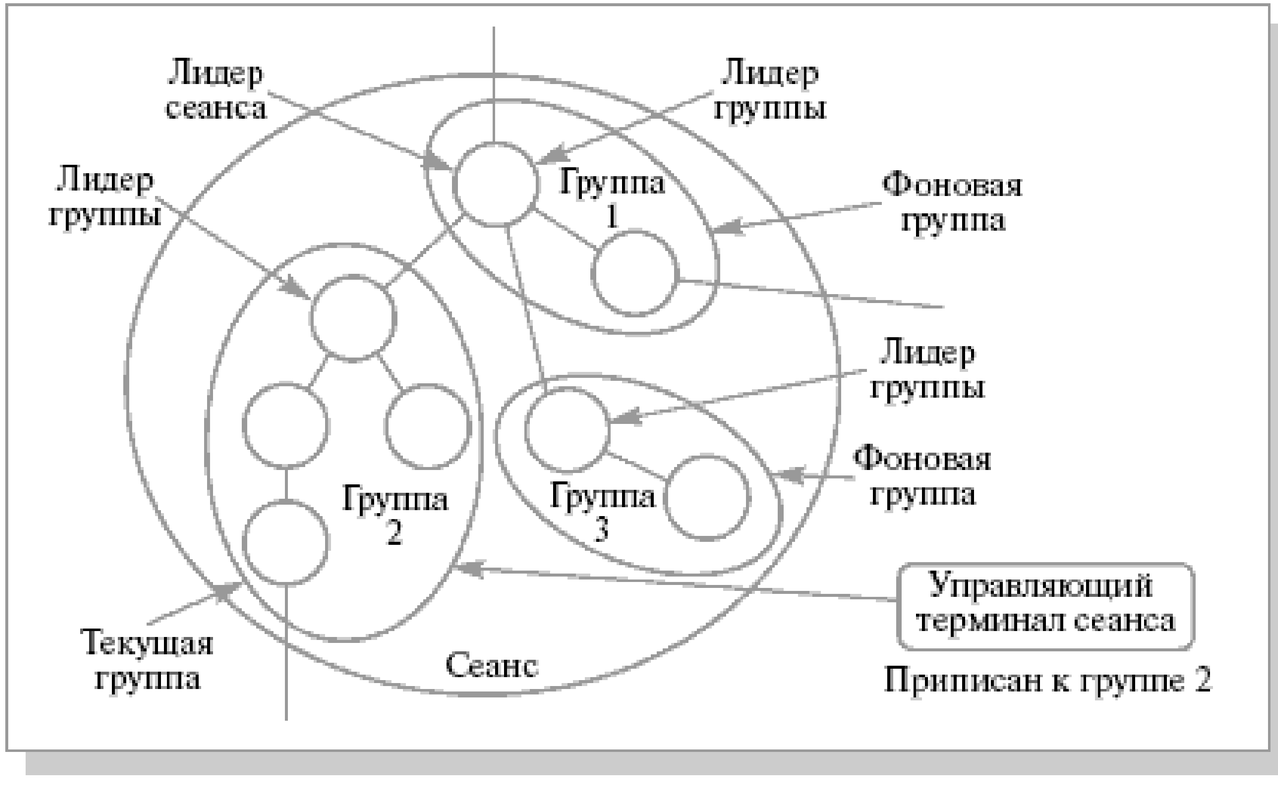
Управляющий терминал

- Управляющий терминал (если есть) обязательно приписывается к некоторой группе процессов в сеансе. Такая группа процессов называется **текущей группой процессов** для данного сеанса.
- Все процессы, входящие в текущую группу процессов, могут совершать операции ввода-вывода, используя управляющий терминал.
- Все остальные группы процессов сеанса называются фоновыми группами, а процессы, входящие в них – **фоновыми процессами**.
- При попытке ввода-вывода фонового процесса через управляющий терминал этот процесс получит сигналы, которые стандартно приводят к прекращению работы процесса.
- Передавать управляющий терминал от одной группы процессов к другой может только лидер сеанса.



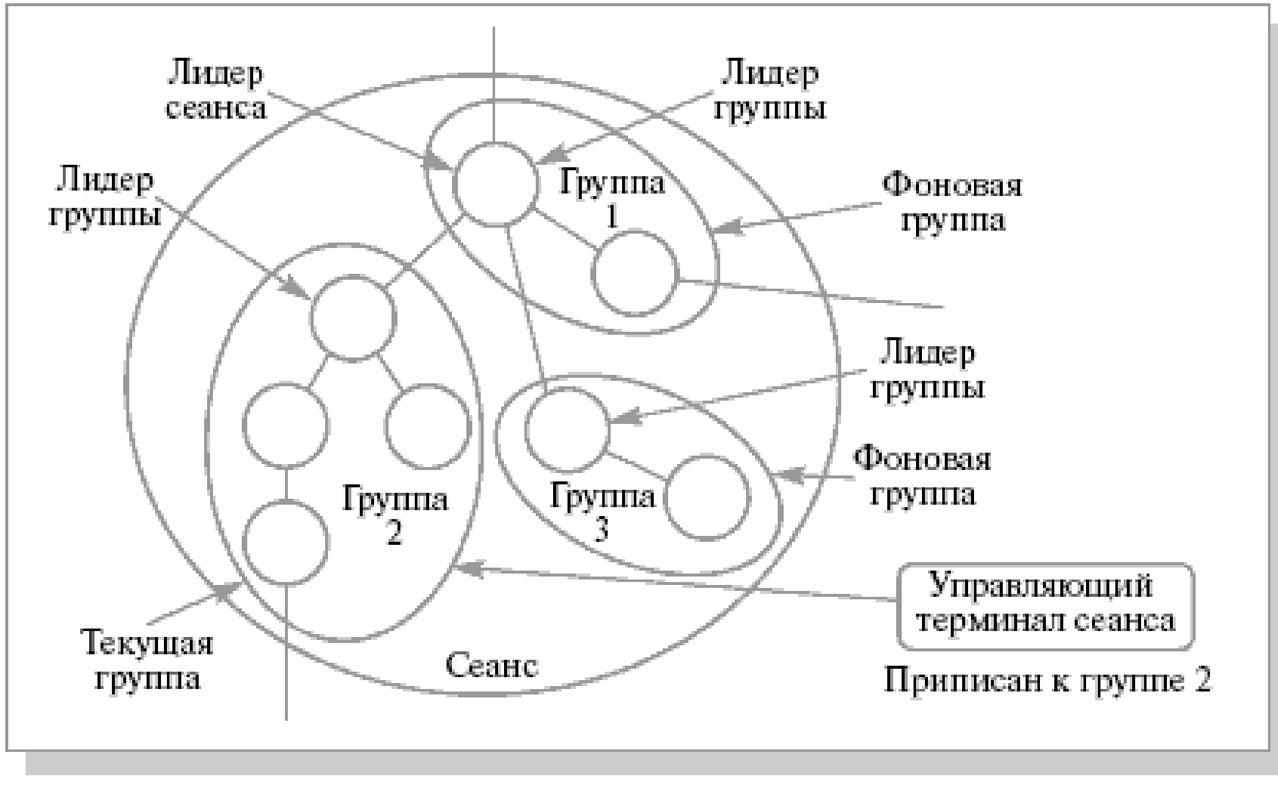
Управляющий терминал

- Процессы, входящие в текущую группу сеанса, могут получать сигналы, инициируемые нажатием определенных клавиш на терминале
- SIGINT при нажатии клавиш <ctrl> и <c>
- SIGQUIT при нажатии клавиш <ctrl> и <4>.
- Стандартная реакция на эти сигналы – завершение процесса (с образованием core файла для сигнала SIGQUIT).



Лидер сеанса

- При завершении работы лидера сеанса все процессы из текущей группы сеанса получают сигнал SIGHUP (*hang up* — отбой, прерывание линии).
- SIGHUP по умолчанию приводит к завершению процесса.
- После завершения лидера сеанса в нормальной ситуации работу продолжат только фоновые процессы.



Номер группы процессов (вариант 1)

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpgid(pid_t pid);
```

Системный вызов возвращает идентификатор группы процессов для процесса, задаваемого идентификатором.

- **pid** – идентификатор процесса, номер группы которого требуется узнать
  допустим только pid себя самого или процесса из своего сеанса.
- Возвращаемое значение:
  pgid > 0 – идентификатор группы
  -1 – ошибка, расшифровка в **errno**

Номер группы процессов (вариант 2)

```
#include <sys/types.h>
#include <unistd.h>


pid_t getpgrp(void);
```

Системный вызов возвращает идентификатор группы процессов, к которому относится текущий процесс.

- Возвращаемое значение:
  pgid > 0 – идентификатор группы
  -1 – ошибка, расшифровка в **errno**

Изменение группы (вариант 1)

```
#include <sys/types.h>
#include <unistd.h>

int setpgid(pid_t pid,
            pid_t pgid);
```

Системный вызов setpgid служит для перевода процесса из одной группы процессов в другую, а также для создания новой группы процессов.

- pid – идентификатором процесса, который нужно перевести в другую группу; только сам, либо прямой ребёнок
- pgid – идентификатором группы процессов, в которую предстоит перевести этот процесс.

Если pid == pgid создаётся новая группа первоначально из одного процесса.

- Возвращаемое значение:
  0 – успех; -1 – ошибка (расшифровка в **errno**)

Изменение группы (вариант 2)

```
#include <sys/types.h>
#include <unistd.h>

int setpgrp(void);
```

Системный вызов setpgrp создаёт новую группу процессов и переводит в неё текущий процесс, который становится лидером группы.

- Возвращаемое значение:
  0 – успех; -1 – ошибка (расшифровка в **errno**)

Номер сеанса

```
#include <sys/types.h>
#include <unistd.h>

pid_t getsid(pid_t pid);
```

Возвращает идентификатор сеанса для указанного процесса.

- **pid** – идентификатор процесса, номер сеанса которого требуется узнать
  если pid==0, то предполагается текущий процесс.
- Возвращаемое значение:
  sid > 0 – номер сеанса
  -1 – ошибка, расшифровка в **errno**

Номер сеанса

```
#include <sys/types.h>
#include <unistd.h>

int setsid(void);
```

Создаёт

- новую группу, состоящую только из процесса, который его выполнил (он становится лидером новой группы)
- новый сеанс, идентификатор которого совпадает с идентификатором процесса, сделавшего вызов. Такой процесс называется **лидером сеанса**.

Этот системный вызов может применять только процесс, не являющийся лидером группы.

❑ Возвращаемое значение:

0 – успех, -1 – ошибка (расшифровка в **errno**)

Механизм сигналов в ОС

управление процессами

Управление процессами

- Процессы независимы друг от друга и могут не общаться с «внешним» миром.
- Как управлять такими «замкнувшимися» процессами?
  Например, как уведомить об исключительной ситуации? Или просто приказать/попросить завершиться?
- ...
- Сгенерировать внутри процесса псевдо прерывание!
  и тем самым заставить его приостановить текущую работу и обработать прерывание
- В Unix этот механизм называют «послать процессу **сигнал**»

Идея классического механизма прерываний (interrupt)

- Процессор, после оправки запроса ввода-вывода к устройству, ожидает результата.
- 1. Процессор периодически сам опрашивает регистр состояния контроллера устройства на наличие бита занятости – **polling**
- 2. У процессора имеется специальный вход, на который устройство по готовности выставляет сигнал запроса прерывания **interrupt request** (само или через контроллер прерываний). При этом
  - Процессор по окончании текущей команды НЕ переходит к следующей
  - Сохраняет состояние
  - Выполняет команды по фиксированным адресам для обработки прерывания.
  - После обработки восстанавливает предыдущее состояние
  - Выполнение нормальных команд продолжается.

Механизм исключений (exception)

- Процессор при возникновении в текущей команде исключительной ситуации (деление на ноль, защита памяти, неверный адрес, ...) поступает так же как при прерывании:
  - По окончании текущей команды НЕ переходит к следующей.
  - Сохраняет состояние, которое было до выполнения ошибочной команды.
  - Выполняет команды по фиксированным адресам для обработки исключения.
  - После обработки исключения может восстановить предыдущее состояние и продолжить или не возвращается.

Механизм программных прерываний (trap, software interrupt)

- При «алгоритмической» необходимости прервать текущую последовательность команд процессора (например, переключить режим выполнения с пользовательского на ядерный в системном вызове) используют механизм аналогичный прерываниям:
  - Процессор по окончании текущей команды НЕ переходит к следующей
  - Сохраняет состояние.
  - Выполняет специальные команды по фиксированным адресам.
  - После обработки может восстановить выполнение отложенной команды.

Обработка прерываний и исключений – идея сигналов

- *Аппаратные прерывания* от устройств ввода-вывода производит сама *операционная система*, не доверяя работу с системными ресурсами процессам пользователя.
- *Исключительные ситуации* и некоторые *программные прерывания* возможно обрабатывать в *пользовательском процессе* через механизм *сигналов*.
- **Сигнал** — асинхронное уведомление пользовательского **процесса** (НЕ процессора) о каком-либо событии, один из основных способов взаимодействия между процессами.
- Получение процессом сигнала выглядит как возникновение прерывания. Процесс прекращает регулярное исполнение, и управление передается обработчику сигнала. По окончании обработки сигнала процесс может возобновить регулярное исполнение.

Типы сигналов

- Типы сигналов и способы возникновения регламентированы в ОС
- Сигналы задаются номерами 1 – 31 и именуются как SIG* (SIGTERM, SIGKILL, …)
- Процесс может получить сигнал от:
  - hardware (при возникновении исключительной ситуации);
  - терминала (при нажатии определенной комбинации клавиш);
  - другого процесса, выполнившего системный вызов передачи сигнала;
  - операционной системы (при наступлении некоторых событий);
  - системы управления заданиями.

сигнальное средство связи между процессами

Некоторые сигналы POSIX 1/2

Название	Код	Действие по умолчанию	Описание	Тип
SIGHUP	1	Завершение	Закрытие терминала	Уведомление
SIGINT	2	Завершение	Сигнал прерывания (Ctrl-C с терминала	Управление
SIGQUIT	3	Завершение с дампом памяти	Сигнал «Quit» (Ctrl-\ с терминала)	Управление
SIGILL	4	Завершение с дампом памяти	Недопустимая инструкция процессора	Исключение
SIGTRAP	5	Завершение с дампом памяти	Ловушка трассировки или брейкпоинт	Отладка
SIGABRT	6	Завершение с дампом памяти	Сигнал, посылаемый функцией abort()	Управление
SIGFPE	8	Завершение с дампом памяти	Ошибочная арифметическая операция	Исключение
SIGKILL	9	**Завершение**	**Безусловное завершение**	**Управление**
SIGBUS	10	Завершение с дампом памяти	Неправильное обращение в физическую память	Исключение
SIGSEGV	11	Завершение с дампом памяти	Нарушение при обращении в память	Исключение

Некоторые сигналы POSIX 2/2

Название	Код	Действие по умолчанию	Описание	Тип
SIGSYS	12	Завершение с дампом памяти	Неправильный системный вызов	Исключение
SIGPIPE	13	Завершение	Запись в разорванное соединение (пайп, сокет)	Уведомление
SIGTERM	15	Завершение	Сигнал завершения (сигнал по умолчанию для утилиты kill)	Управление
SIGUSR1	16	Завершение	Пользовательский сигнал № 1	Пользовательский
SIGUSR2	17	Завершение	Пользовательский сигнал № 2	Пользовательский
SIGCHLD	18	Игнорируется	Дочерний процесс завершен или остановлен	Уведомление
SIGSTOP	23	Остановка процесса	Остановка выполнения процесса	Управление
SIGCONT	25	Продолжить выполнение	Продолжить выполнение ранее остановленного процесса	Управление

Реакция процесса на сигнал

- Принудительно проигнорировать сигнал
- Произвести обработку по умолчанию:
  - проигнорировать
  - остановить процесс (перевести в состояние ожидания до получения другого специального сигнала)
  - завершить работу с образованием core файла или без него.
- Выполнить обработку сигнала, заданную пользователем:
  - пользовательская реакция на сигнал устанавливается специальным системным вызовом;
  - реакцию на некоторые сигналы запрещено изменять, они всегда обрабатываются по умолчанию (например 9 – SIGKILL).
  - при fork() все установленные реакции наследуются.

Отправка сигналов процессам

Консольная команда

```
$ kill [-signal] pid
$ kill -l
```

Посылает сигнал одному или нескольким процессам.

-signal сигнал, который должен быть доставлен; задаётся в числовой или символьной форме, например -9 или - KILL.

Если этот параметр опущен посылается сигнал SIGTERM.

- -l выводится список сигналов, существующих в системе, никакой сигнал не посылается

Отправка сигналов процессам

Консольная команда

```
$ kill [-signal] pid
$ kill -l
```

- pid — процесс или процессы, которым будут доставляться сигналы.
  n > 0 — сигнал конкретному процессу
  0 — сигнал всем процессам текущей группы в сеансе
  -1 — сигнал всем процессам, которые имеют UID текущего пользователя, т.е. все разрешённые процессы
  -n — сигнал всем процессам группы номер n.

Без прав суперпользователя послать сигнал можно только процессу, у которого эффективный UID совпадает с ID текущего пользователя.

Отправка сигналов процессам

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid,
         int signal);
```

Передаёт сигнал одному или нескольким процессам в рамках полномочий пользователя.

- pid – процесс или процессы, которым будут доставляться сигналы.
  - > 0 – сигнал конкретному процессу
  - 0 – сигнал всем процессам текущей группы
  - -1 - сигнал всем процессам с идентификатором текущего пользователя
  - n < -1 - сигнал всем процессам группы номер abs(n).
- sig – номер сигнала
  если sig == 0 то сигнал не посылается, но проверяется возможность
- Возвращает 0 при успехе, -1 при ошибке.

Установка обработчика сигнала

```
#include <signal.h>


void (* signal(int sig,
      void (*handler)(int))
)(int);


typedef void (*handler_t)(int);
handler_t signal(int sig,
                 handler_t h);
```

Изменение реакции процесса на какой-либо сигнал.

- **sig** – номер сигнала, обработка которого меняется
- **handler** — новый способ обработки сигнала
  - указатель на пользовательскую функцию
  - SIG_DFL восстановления реакции процесса на этот сигнал по умолчанию
  - SIG_IGN игнорировать поступившие сигналы
- Возвращаемое значение:
  указатель на **предыдущую функцию** обработчика

Сигнал от процесса-потомка

- **SIGCHLD**
  Посылается ядром **автоматически** при изменении статуса дочернего процесса (завершён, приостановлен или возобновлен).
- Типичная обработка SIGCHLD предполагает считывание кода завершения дочернего процесса с помощью системного вызова **`waitpid()`**
- Если родительский процесс не считывает информацию о завершившемся дочернем, то тот становится **«зомби»** (**zombie** или **defunct** process)
- В стандарте POSIX.1-2001 разрешается на сигнал SIGCHLD установить реакцию **SIG_IGN**, которая считает код завершения дочернего процесса и «выбросит» его, так что зомби не образуются

Считывание информации о потомке

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid,
              int* status,
              int options);

pid_t wait(int* status);
/* == waitpid(-1,status,0); */
```

Ожидание изменения состояния процесса-потомка
И получение информации о таком потомке.

Сменой состояния считается: прекращение работы потомка, останов потомка по сигналу, продолжение работы потомка по сигналу.

- **pid** — идентификатор процесса-потомка
  - **–n** (n>1) — любой потомок из группы процессов с grpID = **n**
  - --1 — любой потомок
  - 0 — любой потомок из группы процессов с grpID = ppid, где ppid — номер «родителя», вызвавшего процесса
  - **n** > 0 — конкретный потомок с указанным ID

Считывание информации о потомке

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid,
              int* status,
              int options);

pid_t wait(int* status);
/* == waitpid(-1,status,0); */
```

- **status** — указатель на переменную, куда помещается информация о состоянии процесса-потомка **в кодированном виде**; для извлечения информации применяются спец. макросы (см. далее); может быть NULL, если информация не нужна

- **options** — определяет поведение системного вызова, объединение битовым ИЛИ (“|”) констант:
  WNOHANG — **не ждать** изменения состояния, только считывать код завершения

Считывание информации о потомке

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid,
              int* status,
              int options);

pid_t wait(int* status);
/* == waitpid(-1,status,0); */
```

- Возвращаемое значение:

  n>0 — ID процесса-потомка, о котором получена информация

  0 — существуют незавершившиеся потомки, а ожидание запрещено опцией WNOHANG

  -1 — ошибка + конкретная причина в **errno**, например, не существует процессов-потомков, соответствующих указанному PID

Расшифровка кода завершения

```
#include <sys/wait.h>

WIFEXITED(status)
WEXITSTATUS(status)

WIFSIGNALED(status)
WTERMSIG(status)
WCOREDUMP(status)
```

- WIFEXITED() — возвращает TRUE, если потомок нормально завершился, то есть вызвал системный вызов **`exit(rc)`** или вернулся из функции main()
- WEXITSTATUS() — возвращает код завершения retcode потомка, который тот указал в exit(retcode) или сделал `return retcode` из функции main()
- WIFSIGNALED() — возвращает TRUE, если потомок завершился из-за сигнала
- WTERMSIG() — возвращает номер сигнала, который привел к завершению потомка
- WCOREDUMP() — возвращает TRUE, если потомок создал дамп памяти (core dump)

Литература

1. В.Е. Карпов, К.А. Коньков - Основы операционных систем. Практикум // М.: Национальный открытый университет «ИНТУИТ», 2016 https://intuit.ru/studies/courses/2249/52/info
2. В.Е. Карпов, К.А. Коньков - Основы операционных систем // М.: Национальный открытый университет «ИНТУИТ», 2016