Системные вызовы семейства exec

Прототипы функций (1)

```
int execl(const char *,
   const char *, …);
int execlp(const char *,
   const char *, …);
int execle(const char *,
   const char *, …, char * const[]);
```

Прототипы функций (2)

```
int execv(const char *,
   char * const[]);
int execvp(const char *,
   char * const[]);
int execve(const char *,
   char * const[], char * const[]);
```

execl()

Список параметров

Строго задается абсолютное либо относительное имя

Окружение сохраняется

execlp()

Список параметров

Поиск программы осуществляется в соответствии с переменной окружения PATH

Окружение сохраняется

execle()

Список параметров

Строго задается абсолютное либо относительное имя

Окружение копируется из параметров системного вызова

execv()

Вектор параметров

Строго задается абсолютное либо относительное имя

Окружение сохраняется

execvp()

Вектор параметров

Поиск программы осуществляется в соответствии с переменной окружения PATH

Окружение сохраняется

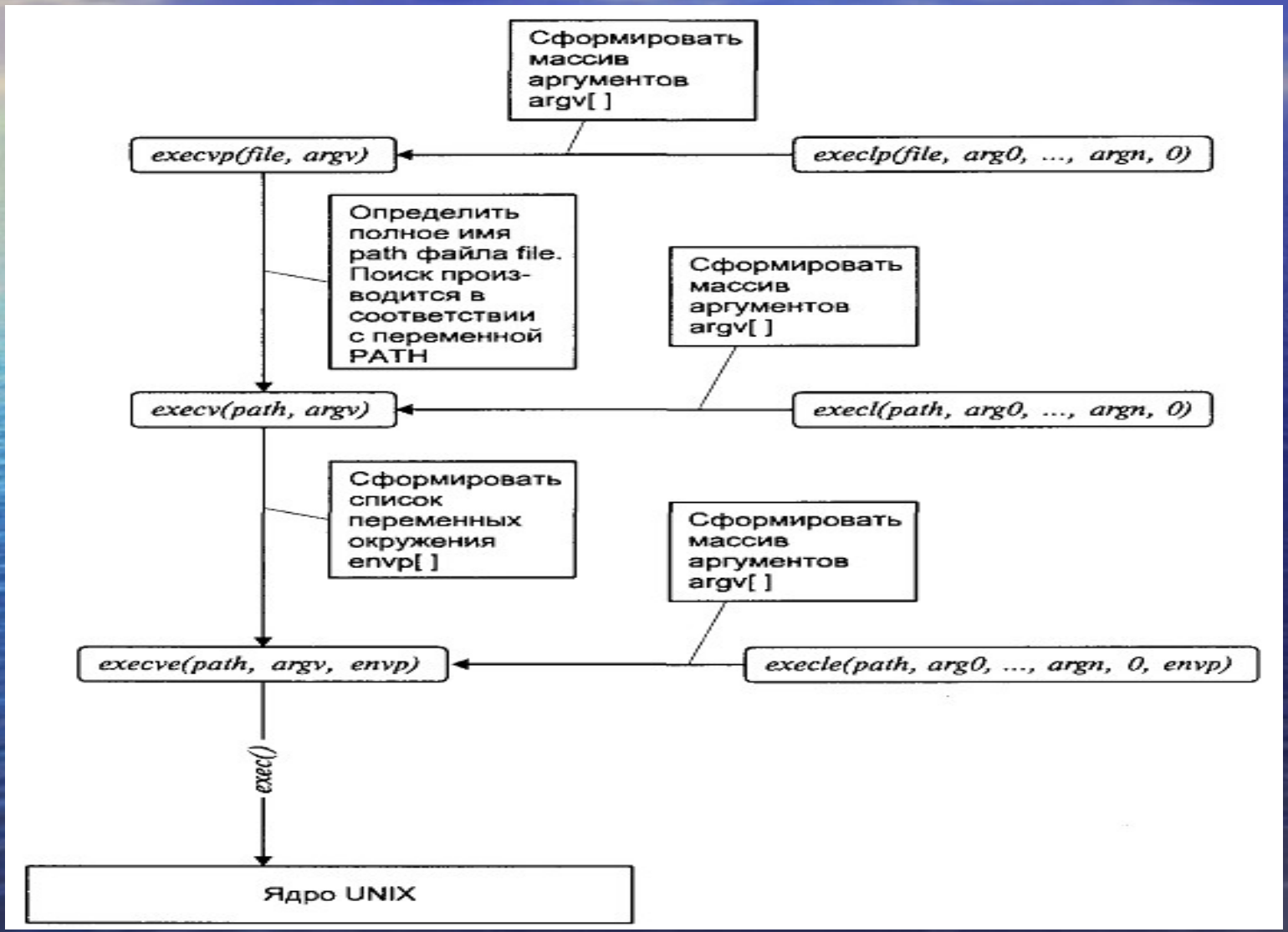
execve()

Вектор параметров

Строго задается абсолютное либо относительное имя

Окружение копируется из параметров системного вызова

Семейство функций exec()



main

int main(int argc, char * argv[], char * envp[]);

- argc – количество аргументов командной строки
- argv – вектор аргументов командной строки
- envp – вектор окружения

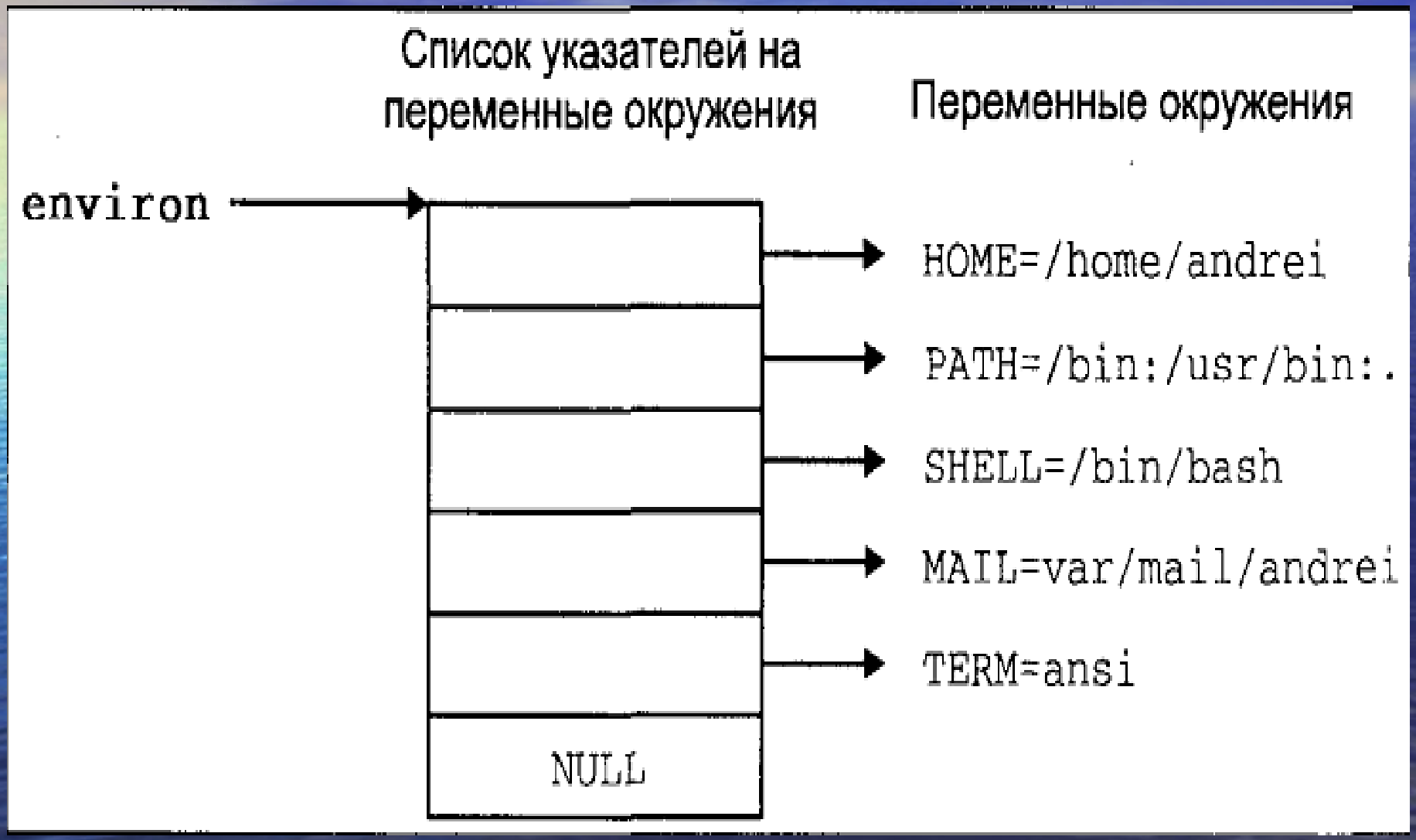
Окружение процесса

Набор текстовых строк вида:

переменная=значение

Обычно используются для передачи текстовой информации между процессами.

Переменные окружения



Переменные окружения

- PATH – пути поиска исполнимых файлов
- HOME – домашний каталог пользователя
- PS1 – приглашение Shell
- TERM – тип терминала

Системные вызовы (1)

char * getenv(const char * name);

вернуть значение переменной

int putenv(char * string);

поместить строку в окружение, параметр string станет частью окружения процесса

Системные вызовы (2)

int setenv(const char * name,
  const char * value, int overwrite);

добавить / изменить значение переменной окружения

int unsetenv(const char * name);

удалить переменную

Системные вызовы (3)

int clearenv();

полностью очистить окружение – глобальная переменная environ получит значение NULL

Идентификаторы пользователя

uid – реальный идентификатор пользователя

euid – эффективный идентификатор пользователя

suid – сохраненный идентификатор пользователя

uid

Определяет «истинного» владельца процесса, того на чей счет относятся потребляемые процессом ресурсы.

euid

Определяет права доступа к объектам системы (файлы (особенности Linux), процессы, IPC-средства.

suid

Предоставляет возможность переключения euid между значениями uid и suid.

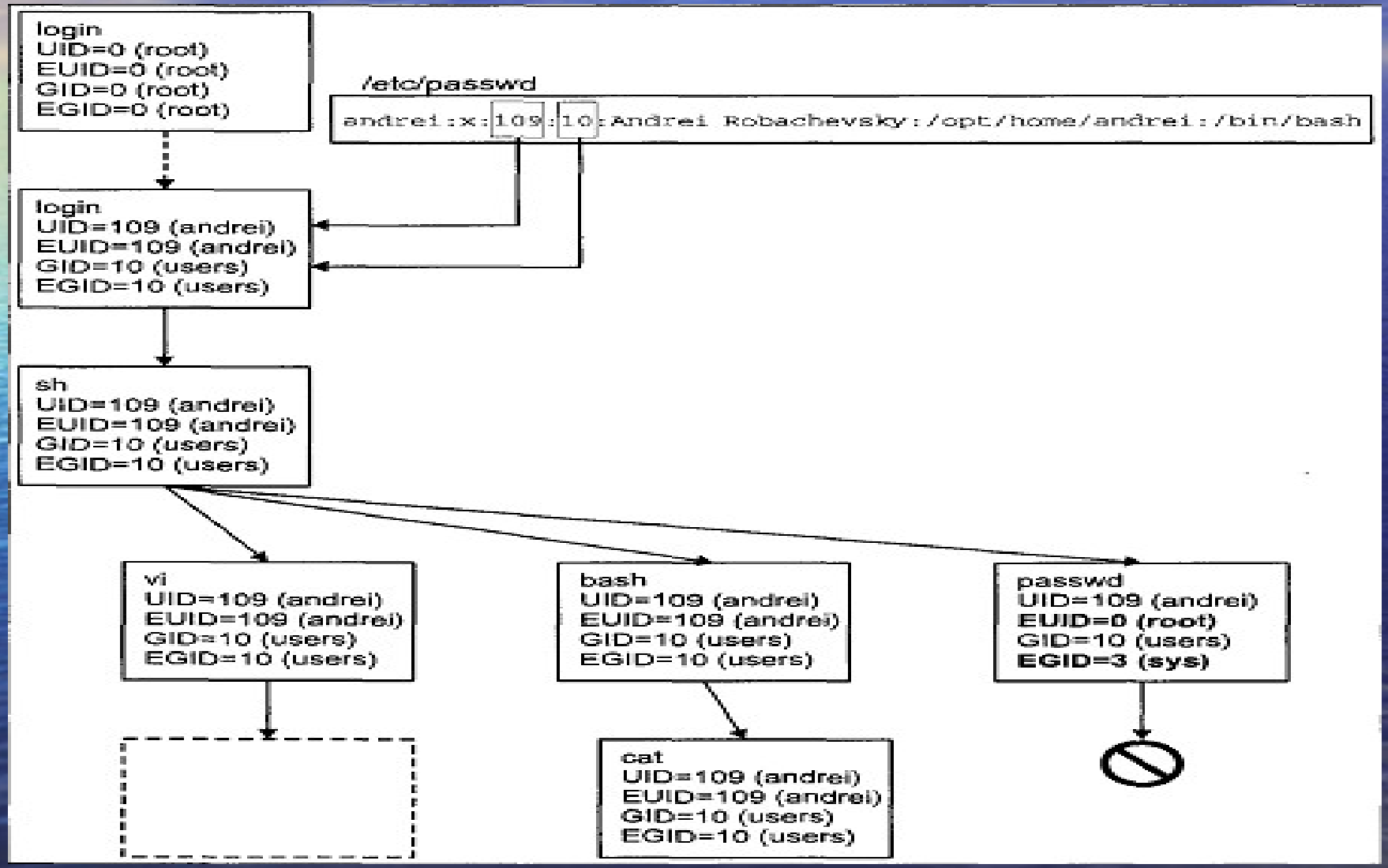
В большинстве случаев все идентификаторы процесса, связанные с пользователями – uid, euid и suid, совпадают.

Если не совпадают, то это suid-программа.

При выполнении exec

Если на запускаемый исполнимый файл (не скрипт) установлен бит смены идентификатора пользователя (suid-бит), то euid и suid процесса устанавливаются в uid владельца данного файла.

Наследование пользовательских идентификаторов



Файл

-rwsr-xr-x 1 max group 3765 Jan 19 20:45 prog1

Процесс до exec

uid=579 (user), euid=579 (user), suid=579 (user)

Процесс после exec(“prog1”, …)

uid=579 (user), euid=521 (max), suid=521 (max)

Системные вызовы

int getuid(); - получить uid

int geteuid(); - получить euid

int setuid(int); - установить uid, euid, suid в одно значение

int seteuid(int); - установить euid

Программы, использующие suid-бит

su

sudo

passwd

mount

umount

ping

traceroute

Сигналы

Назначение сигналов

Простейший способ межпроцессного взаимодействия.

Предназначены для информирования процесса о наступлении какого-либо события.

Надежные и ненадежные

Отправка сигнала

Доставка сигнала

Свойства сигналов

Уникальный номер (обычно в диапазоне от 1 до 32) (номера больше 32 используются для сигналов реального времени)

Уникальное имя (определено в файле signal.h) (kill -l)

Причины отправки

По инициативе процесса – системный вызов

int kill(int, int);

По инициативе ядра

По инициативе ядра

Нажатие на управляющем терминале определенных клавиш (драйвер терминала)

Аппаратные особые ситуации

Программные состояния (будильники)

Доставка сигнала (диспозиция)

- По умолчанию (обычно завершение процесса с созданием или без создания core-файла)
- Игнорирование сигнала
- Перехват сигнала (вызов собственного обработчика)

Когда core-файл не создается

- Исполняемый файл процесса имеет установленный suid-бит и реальный владелец процесса не совпадает с владельцем файла
- Нет права записи в текущий каталог процесса
- Размер core-файла превышает лимит

Диспозиция по умолчанию

Процесс завершается (переходит в состояние зомби) и в младший байт статуса завершения записывается номер сигнала.

Перехват сигнала

Системный вызов

```
void (*signal(int sig, void (*disp)(int))) (int);
```

Функция-обработчик

```
void disp(int sig);
```

Сигналы (1)

SIGKILL (9)

SIGSTOP

SIGCONT

Сигналы (2)

SIGINT (2)

SIGQUIT (3)

SIGTSTP
SIGTTIN
SIGTTOU

Сигналы (3)

SIGBUS

SIGSEGV

SIGFPE

SIGILL

SIGSYS

Сигналы (4)

SIGHUP (1)

SIGTERM (15)

SIGPIPE

Сигнал посылается процессу при некорректной работе с каналом или сокетом.

Более подробно будет рассмотрен в теме «каналы».

SIGCHLD

Сигнал посылается процессу-родителю при завершении любого из его сыновей.

Явное игнорирование сигнала говорит системе, что процесс не будет с помощью wait пытаться получить статус завершения.

SIGUSR1, SIGUSR2

Нет специального действия, закрепленного за этими сигналами. Поэтому процессы могут использовать их для взаимной синхронизации.

SIGALRM

Процесс может с помощью системного вызова alarm() зарегистрировать будильник. По истечении заданного времени процессу будет послан данный сигнал.

Надежные сигналы

Дополнительные возможности:

- Блокирование доставки сигнала
- Рекурсивная обработка
- Получение дополнительной информации вместе с сигналом
- Управление блокирующими системными вызовами

Множество сигналов

Специальный тип

sigset_t

Внутренняя реализация этого типа не документируется (обработка с помощью специальных функций).

Библиотечные функции

```
int sigemptyset(sigset_t *);
int sigfillset(sigset_t *);
int sigaddset(sigset_t *, int);
int sigdelset(sigset_t *, int);
int sigismember(sigset_t *, int);
```

Управление маской сигналов

int sigprocmask(int how,
sigset_t * set, sigset_t * oset);

how

SIG_BLOCK

SIG_UNBLOCK

SIG_SETMASK

Работа с сигналами

int sigpending(sigset_t *);

получение сигналов, ожидающих доставки

int sigsuspend(sigset_t *);

int pause();

ожидание сигналов

Работа с диспозицией

```
int sigaction(int sig,
              const struct sigaction * act,
              struct sigaction * oact);
```

Структура sigaction

void (* sa_handler) ()

void (* sa_sigaction) (int, siginfo_t *, void *)

sigset_t sa_mask

int sa_flags

Флаги sa_flags

SA_SIGINFO

SA_RESETHAND (SA_ONESHOT)

SA_NODEFER

SA_RESTART

Доставка и обработка сигнала

Вызов функции ядра issig() от имени процесса приводит к действию по умолчанию или вызову функции sendsig(), запускающей обработчик сигнала. При этом обработчик выполняется в режиме задачи.

Функция ядра issig() (1)

- При возврате из режима ядра в режим задачи после обработки системного вызова или прерывания
- Перед переходом процесса в состояние сна с приоритетом допускающим прерывание сигналом

Функция ядра issig() (2)

- После пробуждения от сна с приоритетом, допускающим прерывание сигналом

Группы и сеансы

Группы и сеансы (1)

Идентификаторы

pgid – идентификатор группы процессов (не путать с gid)

sid – идентификатор сеанса (сессии)

Группы и сеансы (2)

Каждый сеанс включает одну или несколько групп. Не может группа включать процессы из разных сеансов. Лидер группы (сеанса) – процесс, у которого pid совпадает с pgid (sid). Сеанс может иметь управляющий терминал. Только одна группа сеанса текущая.

Группы и сеансы (3)

Если происходит зависание терминала, то лидеру сеанса посылается сигнал SIGHUP.

Если умирает лидер сеанса, то все процессы текущей группы получают сигнал SIGHUP.

Читать с терминала могут только процессы текущей группы.

Системные вызовы (группы)

```
pid_t getpgrp();

pid_t getpgid(pid_t);

int setpgrp();
```

Системные вызовы (сеансы)

```
pid_t getsid(pid_t);

pid_t setsid();
```

Текущие и фоновые группы, управляющий терминал

Листинг ps

```
$ ps -efj | egrep "PID|andy"
UID   PID  PPID  PGID   SID  C STIME    TTY      TIME CMD
andy 2436  2407  2435  2407  1 15:51:30 tty01    0:00 sort
andy 2431  2407  2431  2407  0 15:51:25 tty01    0:00 find / -name foo
andy 2407  2405  2407  2407  0 15:31:09 ttyO1    0:00 -sh
andy 2435  2407  2435  2407  0 15:51:30 ttyO1    0:00 cat
```

Процессы-демоны

Демоны

Специальные системные процессы, работающие длительное время, не связанные с терминалами. Управление обычно осуществляется при помощи сигналов. В крайнем случае убить и запустить заново с другими параметрами.

init

Самый главный процесс в системе. pid=1 Является отцом всех осиротевших процессов. Активно работает в момент начальной загрузки системы и при переходе с уровня на уровень. В остальное время спит. Смерть init приводит к состоянию kernel panic.

Последовательность шагов (1)

- Снять ассоциацию с управляющим терминалом (Демон не должен получать SIGHUP от терминала)
- Закрыть все открытые файлы (Демон может выводить сообщения только через syslog)

Последовательность шагов (2)

- Сменить текущий каталог на корневой (Демон не должен мешать размонтированию файловых систем)
- Выполнить fork, родительский процесс exit, код в тело дочернего процесса (ppid = 1)

Известные демоны

- inetd – сетевой суперсервер
- crond – демон расписания
- sendmail – демон почтовой службы
- httpd – демон web-сервера
- syslogd – демон системного журнала