Взаимоотношения между процессами. Сигналы.

Операционные системы 2011/12

Татьяна Романова

17 сентября 2011 г.

План на сегодня

- Терминалы.
- Группы процессов.
- Сессии.
- Концепция сигналов.
- Надежные и ненадежные сигналы.
- Обзор средств для работы с сигналами.

Материалы

- Ю. Вахалия, UNIX изнутри, глава 4.
- Р. Стивенс, С. Паго, UNIX. Профессиональное программирование, главы 10-11.

Взаимоотношения процессов

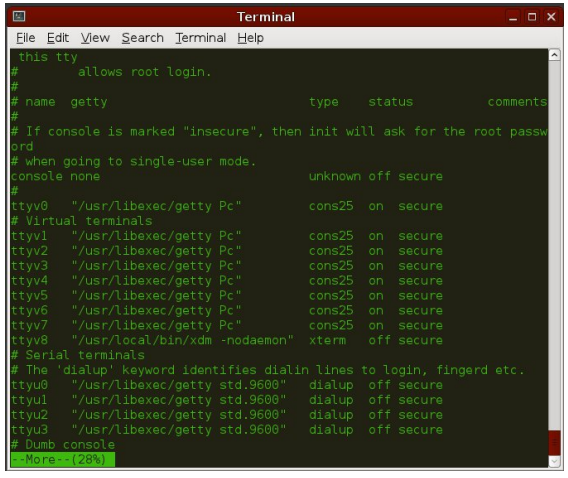
Что мы уже знаем о взаимоотношениях процессов?

- Между процессами существует связь родитель-потомок (и создание потомка — единственный способ создания процесса).
- Существует «прародитель» init (PID 1).
- Родитель извещается, когда потомок завершает работу.
- Функции wait и waitpid могут ждать заврешения потомка.

Что мы хотим узнать?

- Как связаны терминал и работающие процессы?
- Из-за чего и как может система остановить работу процесса?
- Может ли один процесс «общаться» с другим, и как?

Терминалы



- Устройство ввода/вывода, также принимающее управляющие команды (скорость передачи, обработка спец. символов, сброс буферов).
- Бывают локальные, удаленные, текстовые, графические терминалы.
- Эмулятор терминала (gnome-terminal) открывает файл устройства (псевдотерминал /dev/pts/0) и запускает shell, связывая его ввод и вывод с открытым файлом. Аналогично ведут себя сетевые сервисы (telnetd).

Начало работы

Пример начала работы во FreeBSD

- Процесс init читает файл /etc/ttys:

```
ttyv7      "/usr/libexec/getty Pc"
ttyv8      "/usr/local/bin/xdm -nodaemon"
```

- Для каждого терминала запускается программа, которая связывает дескрипторы 0, 1 и 2 с устройством.
- Далее выводится приглашение логина, считываются данные и запускается процесс login
- login считывает пароль, шифрует его и сравнивает с записью в файле паролей.
- При корректном входе login
  - изменяет домашний каталог и владельца терминала;
  - инициализирует среду окружений (переменные HOME, PATH, SHELL, USER);
  - меняет UID и GID и запускает командную оболочку.

Сигналы

Сигналы — программные прерывания, один из способов межпроцессного взаимодействия.

Несколько сигналов, с которыми мы встречались:

- SIGHUP — разрыв терминальной линии;
- SIGINT — введен символ прерывания (Ctrl+C);
- SIGKILL — завершить процесс;
- SIGSEGV — ошибка сегментации (некорректное обращение к памяти);

Сигнал посылается процессу с помощью вызова kill(pid_t pid, int sig).

Группы процессов

```
tr@samanka: $ find . | grep freq.txt
./dict/format/data/freq.txt
./diploma/data/freq.txt
^C
```

Какому процессу будет отправлен SIGINT?
Необходимо, чтобы по Ctrl+C завершались оба процесса.

Группа процессов — это один или несколько процессов, выполняющие одно задание и принимающие сигналы от одного и того же терминала.
Функции:

- pid_t getpgid(pid_t pid) — возвращает идентификатор группы процессов процесса pid;
- pid_t setpgid(pid_t pid, pid_t pgid) — устанавливает идентификатор группы процессов процесса pid; процесс может это сделать для себя и для дочерних процессов.

Сеансы

Группы процессов объединяются в сеансы.

- В одном сеансе может одновременно работать несколько групп.
- Только одна из таких групп является управляющей группой терминала (группой переднего плана, ей доступны ст. ввод и ст. вывод).
- Остальные группы являются фоновыми.
- Сигналы прерывания передаются только группе переднего плана.
- Сигнал разрыва связи SIGHUP посылается только лидеру сессии.

Управление заданиями

Задание — набор процессов, возможно, объединенных в конвеер.

Пример задания переднего плана:

```
find /usr/include -type f -name '*.h' | xargs wc -l | sort -k 1,1 -nr
```

- Ввод и вывод связан с терминалом
- Могут получать от терминала сигналы
  - SIGINT(Ctrl+C),
  - SIGSTOP (Ctrl+Z),
  - SIGQUIT

Фоновые задания

Пример, два фоновых задания, три фоновых процесса:

```
find . -name *.c | grep lab & make &
```

- Не могут читать со ст. ввода, приостанавливаются до команды fg.
- Пишут на ст. вывод, но можно запретить (stty tostop).
- Информация об изменении состояний фоновых процессов выводится только после появления приглашения.

Пример

```
tr@cc:~$ vim &
[1] 4746
tr@cc:~$ ps -o pid,ppid,pgid,sid,comm,tty | cat
  PID  PPID  PGID   SID COMMAND     TT
 4735  4734  4735  4735 bash        pts/3
 4746  4735  4746  4735 vim         pts/3
 4747  4735  4747  4735 ps          pts/3
 4748  4735  4747  4735 cat         pts/3
```

```
tr@cc:~$ ps -o pid,ppid,pgid,sid,comm,tty
  PID  PPID  PGID   SID  COMMAND     TT
 4875  4874  4875  4875  bash        pts/4
 4883  4875  4883  4875  ps          pts/4
```

Сигналы

Генерирование сигналов

- Ввод управляющих символов с клавиатуры терминала.
- Аппаратные ошибки (деление на 0, ошибка доступа к памяти).
- Передача сигналов одним процессом другому (kill(2), kill(1)).
- Программно генерируемые события (SIGPIPE, SIGALRM).

Обработка сигналов

При получении сигналов возможны следующие действия:

- Игнорирование сигнала (кроме SIGSTOP, SIGKILL).
- Перехватывание сигнала (кроме SIGSTOP, SIGKILL).

  Пример:

  перехватывание сигналов SIGTERM, SIGINT для удаления временных файлов.
- Применить действие по умолчанию (обычно завершение процесса).

Создание собственных обработчиков

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

Создание собственных обработчиков

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

Функция signal принимает имя сигнала и адрес функции, которая будет вызвана при получении сигнала, и возвращает предыдущий обработчик.

Вместо адреса функции могут быть переданы константы SIG_DFL и SIG_IGN.

```
typedef void __sighandler_t(int);
#define SIG_ERR ((__sighandler_t *)-1)
```

Сигалы и fork/exec

- exec: при запуске нового процесса сбрасываются обработчики всех сигналов либо на SIG_DFL, либо на SIG_IGN (Почему?)
- fork: дочерний процесс наследует обработчики сигналов родительского процесса.

Пример

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
void sigquit(int sig) {
    signal(SIGQUIT,sigquit);
    printf("CHILD: My daddy is killing me!\n"), exit(0);
}
main() {
   int pid;
   if ((pid = fork()) == 0) {
      signal(SIGQUIT,sigquit);
      for(;;);
   }
   else {
      printf("\nPARENT: sending SIGQUIT\n\n");
      kill(pid,SIGQUIT);
      sleep(3);
   }
}
```

Сигналы и состояние процесса

1. Сигналы во время системных вызовов:
   - Прерываемые и непрерываемые системные вызовы.
   - Первые версии — выход с ошибкой EINTR, более поздние поддерживали автоматический перезапуск.

2. Реентерабельные функции
   - При выполнении некоторых функций нельзя переключаться на обработку сигнала
   - Это функции, работающие с глобальными или статическими структурами (ввод/вывод, malloc/free).

Ненадежные сигналы

После возникновения сигнала ядро сбрасывало действия, определенные по сигналу, на установленные по умолчанию.

```
void sigquit(int sig) {
    signal(SIGQUIT,sigquit); // reset
    printf("CHILD: My daddy is killing me!"), exit(0);
}
```

Состояние состязания: если сигнал придет повторно до переустановки — произойдет действие по умолчанию (выход).

Ненадежные сигналы (продолжение)

Нет возможности отключить сигнал на время.

Пример:

```
int sig_int_flag;
int sig_int() {
      signal(SIGINT, sig_int);
      sig_int_flag = 1;
}
int main() {
      signal(SIGINT, sig_int);
      while(sig_int_flag == 0)
            pause();
}
```

Проблема: если сигнал получен между проверкой флага и вызовом pause, процесс повиснет навсегда.

Надежные сигналы

Основные свойства:

- Постоянно установленные обработчики.
- Маскирование (блокирование) сигналов (защита критических областей кода).
- Атомарное разблокирование и ожидание (вызов sigsuspend).

Отправка и ожидание сигнала

- kill(pid_t pid, int signo) — посылает сигнал процессу или группе процессов.
- raise(int signo) — позволяет процессу послать сигнал себе.
- alarm — устанавливает таймер, по истечении которого будет сгенерирован SIGALRM.
- pause — останавливает процесс в ожидании сигнала.

Плохая реализация функции sleep

```
#include      <signal.h>
#include      <unistd.h>
static void sig_alrm(int signo)
{
    /* nothing to do, just return to wake up the pause */
}
unsigned int sleep1(unsigned int nsecs)
{
    if (signal(SIGALRM, sig_alrm) == SIG_ERR)
        return(nsecs);
    alarm(nsecs);           /* start the timer */
    pause();                /* next caught sig wakes us up */
    return(alarm(0));       /* turn off timer */
}

int main()
{
  sleep(2);
}
```

Работа с набором сигналов

- Набор сигналов — специальный тип sigset_t
- sigemptyset, sigfillset, sigaddset, sigdelset, sigismember — функции для работы с набором сигналов
- sigprocmask — позволяет получить и изменить занчение текущей маски сингалов.
- sigpending — возвращает набор сигналов, которые на данный момент ожидают выполнения.

Заключение

Итоги лекции

- Процессы в системе объединяются в группы для и формируют задания.
- Существуют фоновые задания и задания переднего плана.
- Группы процессов объединены в сессии, с сессией связан управляющий терминал.
- Сигналы — один из механизмов взаимодействия системы и процессов.
- Для большинства сигналов можно установить собственные обработчики.
- Ранние реализации групп, сессий и сигналов имели ряд проблем, которые были решены в поздних версиях.