Сигналы

Системные вызовы и библиотеки Unix SVR4

Иртегов Д.В.

ФФ/ФИТ НГУ

Электронный лекционный курс подготовлен в рамках реализации

Программы развития НИУ-НГУ на 2009-2018 г.г.

ЦЕЛИ РАЗДЕЛА

- Описать сигналы и причины их возникновения
- Научить обрабатывать сигналы, когда они возникают
- Описать воздействие сигналов на различные системные вызовы.

СИГНАЛЫ

- могут быть посланы одним процессом другому процессу
- могут быть посланы от ядра к процессу
- сообщают о внешнем событии или ошибке
- обрабатываются получающим процессом

ТИПЫ СИГНАЛОВ

сигнал	**реакция**	**событие**
SIGHUP	exit	обрыв линии (см. termio(7))
SIGINT	exit	прерывание (см. termio(7))
SIGQUIT	core	завершение (см. termio(7))
SIGILL	core	неправильная инструкция
SIGTRAP	core	прерывание трассировки
SIGABORT	core	аборт
SIGEMT	core	команда EMT (программное прерывание)
SIGFPE	core	арифметическая особая ситуация
SIGKILL	exit	принудительное завершение ("убийство")
SIGBUS	core	ошибка шины
SIGSEGV	core	нарушение сегментации
SIGPIPE	exit	разрыв конвейера

ТИПЫ СИГНАЛОВ

SIGALRM	exit	будильник
SIGTERM	exit	программный сигнал прерывания от kill
SIGCLD	ignore	изменение состояния подпроцесса
SIGPWR	ignore	сбой питания
SIGSTOP	stop	остановка (сигналом)
SIGTSTP	stop	остановка (пользователем) (см. termio(7))
SIGCONT	ignore	продолжение
SIGTTIN	stop	ожидание ввода с терминала(см. termio(7))
SIGTTOU	stop	ожидание вывода на терминал (см.termio(7))
SIGVTALRM	exit	сигнал виртуального таймера
SIGPROF	exit	сигнал таймера профилирования
SIGXCPU	core	исчерпался лимит времени (см. getrlimit(2))
SIGXFSZ	core	выход за пределы длины файла (см. getrlimit(2))

ПОЛУЧЕНИЕ СИГНАЛА

- получающим процессом выполняются действия по обработке сигнала
- возможные реакции:
  - SIG_DFL - реакция по умолчанию
  - SIG_IGN - проигнорировать сигнал
  - адрес функции - перехватить сигнал

Signal(2)

ИСПОЛЬЗОВАНИЕ

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

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

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

signal

- успех - предыдущая реакция на сигнал
- неуспех - SIG_ERR и errno установлена

sigset

- успех - SIG_HOLD если сигнал заблокирован, иначе предыдущая реакция на этот сигнал
- неуспех - SIG_ERR и errno установлена

Signal(2)

ИСПОЛЬЗОВАНИЕ

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

handler_t sigset (int sig, handler_t handler);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

signal

- успех - предыдущая реакция на сигнал
- неуспех - SIG_ERR и errno установлена

sigset

- успех - SIG_HOLD если сигнал заблокирован, иначе предыдущая реакция на этот сигнал
- неуспех - SIG_ERR и errno установлена

signal и sigset

- Обработчик, установленный через signal(2), сбрасывает реакцию на сигнал на SIG_DFL
- Обработчик, установленный через sigset(2), блокирует сигнал на время своей работы, и разблокирует после возврата

Signal(2) - продолжение

ИСПОЛЬЗОВАНИЕ

```
#include <signal.h>
int sigrelse(int sig);
int sigignore(int sig);
int sigpause(int sig);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - 0

неуспех - -1 и errno установлено

ГЕНЕРАЦИЯ СИГНАЛОВ

- из пользовательских программ
  - kill (2)
  - sigsend(2)
  - alarm(2)
- из ядра
  - от клавиатуры
  - от ошибок программирования

kill(2)

ИСПОЛЬЗОВАНИЕ

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - 0

неуспех - -1 и errno установлена

sigsend(2)

ИСПОЛЬЗОВАНИЕ

```
#include <sys/types.h>
#include <sys/signal.h>
#include <sys/procset.h>
int sigsend (idtype_t idtype,
  id_t id, int sig);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - 0

неуспех - -1 и errno установлена.

Id_type

idtype	Получающий процесс (процессы)
P_PID	процесс, чей PID равен id
P_PGID	все процессы, чей pgif равен id
P_SID	все процессы, чей ID сессии равен id
P_UID	все процессы, чей EUID равен id
P_GID	все процессы, чей EGID равен id
P_CID	все процессы, чей класс планирования равен id[см. priocntl(2)]
P_ALL	все процессы, id игнорируется
P_MYID	вызывающий процесс

alarm(2)

ИСПОЛЬЗОВАНИЕ

```
#include <unistd.h>
unsigned alarm (unsigned sec);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

время, оставшееся до сигнала

setjmp(3C)

ИСПОЛЬЗОВАНИЕ

```
#include <setjmp.h>
int setjmp (jmp_buf env);
void longjmp (jmp_buf env, int val);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

setjmp(3) возвращает 0, когда встречается в выполняемой последовательности кодов

setjmp(3) возвращает ненулевое значение, если вызов longjmp(3) привел к переходу к месту вызова setjmp(3)

pause(2)

ИСПОЛЬЗОВАНИЕ

```
#include <unistd.h>
int pause (void);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

-1 и errno установлена

mysleep

```
static jmp_buf env;
mysleep(int seconds) {
   void sigcatch(int), (*astat)(int);
   int unslept = seconds;
   astat = signal(SIGALRM, sigcatch);

   if (setjmp(env) == 0) {
    alarm(seconds);
    pause();
   }
   unslept = alarm(0);
   signal(SIGALRM, astat);
   return(unslept);
}
static void sigcatch(int sig) { longjmp(env, 1); }
```

sleep(3C)

ИМЯ

sleep - задержать исполнение на заданный интервал времени

ИСПОЛЬЗОВАНИЕ

```
#include <unistd.h>
unsigned sleep (unsigned seconds);
```

ОПИСАНИЕ

Не оказывает воздействия на обработку или доставку ни одного сигнала. Безопасно для использования в многопоточной программе.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

оставшееся "недоспанное" время

sigsetops(3C)

ИСПОЛЬЗОВАНИЕ

```
#include <signal.h>
int sigemptyset(sigset_t * set);
int sigfillset(sigset_t * set);
int sigaddset(sigset_t * set, int signo);
int sigdelset(sigset_t * set, int signo);
int sigismember(sigset_t * set, int signo);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - sigismember: 1 если истинно, 0 если ложно;

остальные функции: 0

неуспех - -1 и errno установлена

sigpending(2)

ИСПОЛЬЗОВАНИЕ

```
#include <signal.h>
int sigpending(sigset_t * set);
```

sigprocmask(2)

ИСПОЛЬЗОВАНИЕ

```
#include <signal.h>
int sigprocmask(int how,
  sigset_t *set,sigset_t *oset);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - 0

неуспех - -1 и errno установлена

sigprocmask how

SIG_BLOCK - Множество сигналов, на которое указывает set, будет добавлено к текущей маске сигнала.

SIG_UNBLOCK - Множество set будет удалено из текущей маски.

SIG_SETMASK - Текущая маска будет заменена на set.

sigaction(2)

ИСПОЛЬЗОВАНИЕ

```
#include <signal.h>
int sigaction(int sig,
  const struct sigaction *act,
  struct sigaction *oact);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - 0

неуспех - -1 и errno установлена

struct sigaction

void (*sa_handler)(); - Адрес функции обработки сигнала, SIG_IGN или SIG_DFL

sigset_t sa_mask - Маска сигналов, которые должны быть заблокированы, когда вызывается функция обработки сигнала.

int sa_flags - Флаги, управляющие доставкой сигнала.

sa_flags

A_ONSTACK - Используется для обработки сигналов на альтернативном сигнальном стеке.

SA_RESETHAND - Во время исполнения функции обработки сбрасывает реакцию на сигнал к SIG_DFL; обрабатываемый сигнал при этом не блокируется.

SA_NODEFER - Во время обработки сигнала сигнал не блокируется.

SA_RESTART - Системные вызовы, которые будут прерваны исполнением функции обработки, автоматически перезапускаются.

SA_SIGINFO - Используется для доступа к подробной информации о процессе, исполняющем сигнальный обработчик, такой как причина возникновения сигнала и контекст процесса в момент доставки сигнала.

SA_NOCLDWAIT - Подавляет создание процессов-зомби.

SA_NOCLDSTOP - Подавляет генерацию SIGCHLD, когда порожденные процессы останавливаются или возобновляются.

СИГНАЛЫ ДЛЯ УПРАВЛЕНИЯ ЗАДАНИЯМИ

Имя	**Значение**	**Умолчание**	**Событие**
SIGSTOP	23	Stop	Остановка (сигналом)
SIGTSTP	24	Stop	Остановка (пользователем)
SIGCONT	25	Ignore	Продолжение исполнения
SIGTTIN	26	Stop	Остановка при вводе с терминала
SIGTTOU	27	Stop	Остановка при выводе на терминал