Сигналы, каналы

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение, 2019

Содержание лекции

Введение

Сигнал (программное прерывание) -— это оповещение процесса о том, что произошло некое событие. Такие события могут происходить из-за пределов системы -– например, из-за введения пользователем символа прерывания — или возникать вследствие действий в программе или ядре.
Один процесс может отправить сигнал другому процессу. При таком использовании сигналы могут рассматриваться как технология синхронизации и являются примитивной формой межпроцессного взаимодействия.

Введение

Ядро отправляет процессу сигнал в следующих случаях

- Произошло аппаратное исключение. Это значит, что аппаратное обеспечение зафиксировало неверное состояние и оповестило об этом ядро, которое направило соответствующий сигнал затронутому процессу.
- Пользователь ввел один из специальных символов терминала, генерирующих сигналы.
- Произошло программное событие. Например, появился ввод из файлового дескриптора, сработал таймер, был завершен дочерний процесс.

Обработка сигнала

При получении сигнала процесс может выполнить следующие действия

- Игнорировать сигнал. Никакое действие не предпринимается. Существуют два сигнала, которые не могут быть проигнорированы: `SIGKILL` и `SIGSTOP`.
- Выполнить действие по умолчанию. Это действие зависит от того, какой сигнал отправляется.
- Вызвать обработчик сигнала. Ядро приостанавливает исполнение текущего кода процесса и переходит к ранее зарегистрированной функции. Затем процесс исполняет эту функцию. После выполнения обработчика, он переходит обратно в то место, где находился на тот момент, когда был захвачен сигнал.

Действия по умолчанию

По получении сигнала процесс по умолчанию выполняет одно из действий

- Сигнал игнорируется, то есть сбрасывается ядром и не влияет на процесс.
- Процесс завершается. Иногда это называют аварийным завершением процесса.
- Генерируется файл дампа ядра, и процесс завершается.
- Процесс останавливается.
- Выполнение процесса возобновляется.

Некоторые сигналы и действия по умолчанию

Сигнал	Описание	Действие по умолчанию
SIGABRT	Отправляется при вызове функции abort()	Завершиться с дампом ядра
SIGALRM	Отправляется функциями alarm() и setitimer().	Завершиться
SIGCHLD	Отправляется процессу при завершении одного из его потомков	Игнорировать
SIGCONT	Возобновление приостановленного процесса	Игнорировать
SIGINT	Пользователь ввел символ прерывания	Завершиться
SIGKILL	Сигнал завершения процесса	Завершиться
SIGPIPE	Запись данных в канал или сокет при отсутствии читателя	Завершиться
SIGQUIT	Пользователь ввел символ выхода	Завершиться с дампом ядра
SIGSEGV	Обращение к неверному адресу памяти	Завершиться с дампом ядра
SIGSTOP	Сигнал остановки процесса	Остановиться
SIGTERM	Сигнал завершения процесса	Завершиться

Отправка сигналов

Один процесс может отправить сигнал другому процессу с помощью системного вызова `kill()`.

```
#include <signal.h>
int kill(pid_t pid, int sig);
// Возвращает 0 при успешном завершении
// и -1 при ошибке
```

Аргумент `pid` идентифицирует один или несколько процессов, в которые отправляется сигнал `sig`.

- pid $>$ 0 – сигнал отправляет процессу с идентификатором pid.
- pid $=$ 0 – сигнал отправляется всем процессам группы, которой принадлежит вызывающий.
- pid $<$ -1 – сигнал отправляется всем процессам группы, идентификатор которой равен абсолютному значению pid.
- pid $=$ -1 – сигнал отправляется всем процессам, которым вызывающий процесс может отправлять сигналы (кроме init и самого себя).

Блокирование сигналов

Для каждого процесса ядро хранит сигнальную маску — набор сигналов, доставка которых в процесс временно заблокирована. Если в процесс отправляется заблокированный сигнал, то доставка этого сигнала откладывается до тех пор, пока сигнал не будет разблокирован путем удаления из сигнальной маски процесса.

Блокирование сигналов

Системный вызов sigprocmask() может использоваться для добавления и удаления сигналов из сигнальной маски.

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set,
                sigset_t *oldset);
// Возвращает 0 при успешном завершении или -1 при ошибке
```

Аргумент how определяет, как изменится сигнальная маска

- SIG_BLOCK – сигналы, переданные в аргументе set будут включены в сигнальную маску.
- SIG_UNBLOCK – сигналы, переданные в аргументе set будут исключены из сигнальной маски.
- SIG_SETMASK – сигнальной маской станут сигналы, указанные в аргументе set.

Наборы сигналов

Набор сигналов sigset_t инициализируется функцией sigemptyset() или sigfillset().

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t, *set);
// возвращают 0 при успешном завершении или -1 при ошибке
```

Для добавления и удаления сигналов из набора используются функции sigaddset() и sigdelset().

```
#include <signal.h>
int sigaddset(sigset_t *set, int sig);
int sigdelset(sigset_t *set, int sig);
// возвращают 0 при успешном завершении или -1 при ошибке
```

Задание обработчиков

Обработчик сигнала – это функция, вызываемая при получении указанного сигнала процессом.
Функция `sigaction()` позволяет устанавливать и получать обработчик сигнала.

```
#include <signal.h>
int sigaction(int sig, const struct sigaction *act,
              struct sigaction *oldact);
// Возвращает 0 при успешном завершении или -1 при ошибке
```

Аргумент `sig` означает сигнал, для которого устанавливается обработчик.
Аргумент `act` описывает политику обработки сигнала.
После установки нового обработчика, старый будет записан в `oldact`.

Структура sigaction

Важными полями в структуре sigaction являются следующие

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
};
```

- sa_handler – обработчик сигнала
- sa_mask – сигналы, блокируемые во время вызова обработчика
- sa_flags – параметры, контролирующие обработку сигнала

Ожидание сигнала

Вызов функции `pause()` приостанавливает выполнение процесса до тех пор, пока он не получит сигнал.

```
#include <unistd.h>
int pause(void);
// Всегда возвращает -1 с установкой errno в EINTR
```