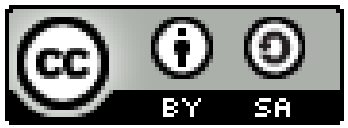
Сигналы

Асинхронный вид IPC

Предназначение, генерация, обработка



Синхронное и асинхронное взаимодействие

Синхронное взаимодействие

Синхронное взаимодействие в заданное программой время, и определяется её кодом. Например, чтение из файла на начнётся раньше, чем процесс вызовет системный вызов read. Данные через сокет будут переданы также по запросу из программы — ни раньше, ни позже.

В окружении программы могут происходить события, на которые требуется немедленная реакция, а не тогда, когда программы сама соизволит заняться взаимодействием.

Асинхронное взаимодействие

Асинхронное взаимодействие происходит в произвольные моменты выполнения программы, без явной привязки к её коду.

Способ асинхронного взаимодействия — сигналы

Одним из способов уведомления процесса о событиях, которые происходят асинхронно по отношению к выполняющеюся программе, является механизм сигналов.
Основные действия при работе с сигналами:

- Сигнал можно **послать** процессу, тем самым потребовать от процесса реакции на событие. Источником сигнала может быть другой процесс, тот же процесс, или же ядро системы.
- Сигнал можно **обработать**, тем самым выполнить заданный код при получении сигнала. Среди возможных действий — завершиться от необработанного сигнала, проигнорировать, выполнить произвольные действия. Для каждого сигнала существует **действий по умолчанию** — то, что произойдёт, если программист не определил действий при приходе сигнала.

Информация, передаваемая сигналом

Сигналы **не** предназначены для передачи большого объёма информации. Их основная задача — уведомление о событии.
Наиболее часто доступной (и полезной) информацией является **номер сигнала** . Для многих событий этот номер зарезервирован. Вручную можно послать любой сигнал. Для переносимости программ этим номерам даны символические имена в заголовочных файлах. Например:

```
#define SIGHUP 1
```

Использование просто чисел может сделать программу непереносимой!
Сигналы с номерами 1–32 предназначены для обычного использования. Сигналы с большими номерами (33–64) предназначены для реализаций задач реального времени.

Обзор основных сигналов
SIGHUP

Предназначение

Сигнал SIGHUP (1) посылается ядром при разрыве связи с управляющим терминалом или смерти контролирующего процесса. Часто используется как указание процессу-демону перечитать свою конфигурацию.

Действие по умолчанию

Аварийное завершение

Обзор основных сигналов
SIGINT

Предназначение

Сигнал SIGINT (2) посылается ядром активному процессу на терминале при нажатии определённой комбинации клавиш — обычно **Ctrl-C**. Это стандартный способ снятия выполняющейся консольной задачи.

Действие по умолчанию

Аварийное завершение

Обзор основных сигналов
SIGQUIT

Предназначение

Сигнал SIGQUIT (3) посылается ядром активному процессу на терминале при нажатии определённой комбинации клавиш — обычно **Ctrl-**. Этот способ снятия необходим тогда, когда надо исследовать, чем занималась программа в момент снятия с выполнения.

Действие по умолчанию

Аварийное завершение с созданием дампа памяти процесса (см. ulimit, gdb).

Обзор основных сигналов
SIGABRT

Предназначение

Сигнал SIGABRT посылается при вызове специальной функции **abort()** ; Вызов этой функции ставят в те места программы, выполнения которых не должно быть при нормальной работе.

Действие по умолчанию

Аварийное завершение с созданием дампа памяти процесса.

Обзор основных сигналов
SIGILL

Предназначение

Сигнал SIGILL (4) посылается ядром при попытке выполнить процессором несуществующий инструкции. Обычно происходит при передаче управления не в код (или в неподходящее место кода), а также при выполнении кода, откомпилированного для более нового процессора на старом.

Действие по умолчанию

Аварийное завершение с созданием дампа памяти процесса.

Обзор основных сигналов
SIGKILL

Предназначение

Сигнал SIGKILL (9) используется для **безусловного** снятия процесса с выполнения. Следует использовать, когда другие средстве не помогают.
Также используется при завершении работы операционной системы, для завершения процессов, не умерших нормальным способом.
Этот сигнал нельзя игнорировать или перехватить!

Действие по умолчанию

Аварийное завершение!

Обзор основных сигналов
SIGSEGV

Предназначение

Сигнал SIGSEGV (11) посылается ядром при нарушении защиты памяти. Например, при попытке изменить память только для чтения, попытке доступа к несуществующей памяти (nullptr), попытке выполнить что-то в сегменте, в котором выполнение запрещено . . . . “Любимый” сигнал программистов.

Действие по умолчанию

Аварийное завершение с созданием дампа памяти процесса.

Обзор основных сигналов
SIGPIPE

Предназначение

Сигнал SIGPIPE (13) посылается ядром процессу, который пишет в канал (pipe, named pipe, socket), в том случае, если закрыт дескриптор для чтения.

Действие по умолчанию

Аварийное завершение

Обзор основных сигналов
SIGALRM

Предназначение

Сигнал SIGALRM (14) посылается ядром процессу, который заказал приход этого сигнала через заданное кол-во секунд с помощью системного вызова alarm:

```
unsigned int alarm(unsigned int seconds);
```

Действие по умолчанию

Аварийное завершение

Используется для ограничения времени выполнения программ, а также для реализации функции sleep().

Обзор основных сигналов
SIGTERM

Предназначение

Сигнал SIGTERM (15) используется для “нормального” принудительного завершения процессов, работающих в фоновом режиме. Обычный способ завершения работы программ-демонов. Также посылается консольной командой завершения процесса, если не указан номер сигнала.

Действие по умолчанию

Аварийное завершение

Обзор основных сигналов
SIGCHLD

Предназначение

Сигнал SIGCHLD (17) посылается ядром процессу-родителю при смерти потомка.

Действие по умолчанию

Игнорирование. Но на самом деле реализация системного вызова wait использует обработку этого сигнала.

Обзор основных сигналов
SIGUSR1, SIGUSR2

Предназначение

Сигналы SIGUSR1, SIGUSR2 (10, 12 . . .) предназначены для произвольного использования пользователем. Стандарт не резервирует никаких системных событий для этих сигналов.

Действие по умолчанию

Игнорирование

Обзор основных сигналов
SIGSTOP

Предназначение

Сигнал SIGSTOP (19) используется для приостановки (“замораживания”) процесса. Посылается также ядром активному процессу на терминале при нажатии “Ctrl-Z”.
Этот сигнал нельзя игнорировать или перехватить!

Действие по умолчанию

Останов процесса.

Обзор основных сигналов
SIGCONT

Предназначение

Сигнал SIGCONT (18) предназначен для продолжения выполнения процесса, остановленного с помощью SIGSTOP.

Действие по умолчанию

Продолжение выполнения процесса.

Команда bash “fg” позволяет продолжить выполнения процесса, остановленного по “Ctrl-z” (SIGSTOP) в качестве активного процесса переднего плана (foregound), а команда “bg” — продолжает выполнение команды в фоновом режиме.

Обзор основных сигналов
SIGTRAP

Предназначение

Сигнал SIGTRAP () посылается ядром при отладке процесса и срабатывании отладочной ловушки.

Действие по умолчанию

Аварийное завершение с созданием дампа памяти процесса. Но чаще этот сигнал обрабатывается не самим процессом, а отладчиком.

.

Системный вызов для посылки сигнала
kill

Для того, что бы послать произвольный сигнал из процесса, можно использовать системный вызов kill:

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

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

Аргумент **pid** указывает, кому надо послать сигнал. Если pid >0 — конкретный PID процесса. Если $=0$ — то всем процессам из своей группы. Если $=-1$ — всем доступным процессам, кроме 1. Если pid <0 — то всем процессам из группы $|\text{pid}|$.
Аргумент **sig** задаёт номер сигнала, или 0 — только проверить возможность.
Возвращает -1 в случае ошибки, и 0 — в противном случае.

Команда kill

Для посылки сигналов из командной строки есть одноимённая команда:

```
kill [-s signal|-p] [-q sigval] [-a] [--] pid...

kill -l [signal]
```

Позволяет послать заданный сигнал (или SIGTERM), а также узнать список существующих сигналов.

Функция raise

Что бы послать сигнал самому себе, можно использовать функцию raise:

```
#include <signal.h>

int raise(int sig);
```

На самом деле в процессе с одной нитью выполнения это эквивалентно

```
kill( getpid(), sig );
```

Также сигналы можно послать с помощью killpg, tkill, sigqueue, pthread_sigqueue.

Установка обработчика сигнала

Если не задать действие при приходе сигнала, выполняется действие по умолчанию. Можно задать свой обработчик сигнала системным вызовом signal:

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum,
      sighandler_t handler);
```

Аргумент signum — номер сигнала, для которого пишем обработчик.
handler — адрес нового обработчика, причём можно использовать макросы **SIG_IGN** — игнорировать сигнал, и **SIG_DFL** — обработчик по умолчанию.

Установка обработчика сигнала
продолжение

В случае ошибки возвращает **SIG_ERR** , а в случае успеха — адрес старого обработчика. После этого вызова, при приходе сигнала signum будет вызываться заданный обработчик.

Один обработчик может обрабатывать несколько разных сигналов, т.к. он получает номер сигнала в качестве аргумента.

Следует писать обработчики как можно более короткие. Есть список функций, которые безопасно вызывать в обработчике сигнала.

Достоинства и недостатки kill и signal

Достоинства

- Простота.
- Существуют даже в до-POSIX системах.

Недостатки

- В стандарте не указано, что должно происходить, если при обработке сигнала будет получен другой или такой же.
- Нет возможности отложить обработку сигнала — только игнорировать или немедленно обработать.
- Нет возможности передать или получить дополнительную информацию при обработке сигнала.

Замаскировать набор сигналов — sigprocmask

У каждого процесса есть **сигнальная маска** — набор битов, указывающих обработка каких сигналов заблокирована. Системный вызов **sigprocmask** позволяет управлять этой маской.

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

how — задаёт действие: SIG_BLOCK — добавить набор указанных сигналов к заблокированным, SIG_UNBLOCK — убрать указанные сигналы из списка заблокированных, SIG_SETMASK — установить маску в указанное значение.
set — указатель на объект типа sigset_t, указывающий набор сигналов (если не 0).
oldset — если не 0, но по этому адресу будет записана старая сигнальная маска.
Возвращает: 0 — Ok, −1 — ошибка.

Функции для работы с сигнальной маской

Список

Для удобства работы и переносимости программ есть набор функций для работы с sigset_t:

```
int sigemptyset(sigset_t *set);

int sigfillset(sigset_t *set);

int sigaddset(sigset_t *set, int signum);

int sigdelset(sigset_t *set, int signum);

int sigismember(const sigset_t *set, int signum);
```

Все эти функции в качестве первого аргумента получают указатель на объект типа sigset_t, с которым функция будет работать.

Функции для работы с сигнальной маской

Описание

sigemptyset — создаёт пустую сигнальную маску — все биты сброшены;

sigfillset — создаёт полностью заполненную маску, все биты установлены;

sigaddset — добавляет заданные сигналы к маске;

sigdelset — убирает заданные сигналы из маски;

sigismember — проверяет, установлен ли в маске бит, соответствующий сигналу.

Функция sigismember возвращает 0, если сигнал не принадлежит маске, 1 — если принадлежит, -1 — в случае ошибки.
Остальные возвращают 0 — Ok, -1 — ошибка.

Набор сигналов, ожидающих обработки — sigpending

Если какие-либо сигналы были заблокированы, можно узнать, какие из них приходили за это время.

```
#include <signal.h>

int sigpending(sigset_t *set);
```

Этот системный вызов устанавливает в set биты, соответствующие пришедшим, но заблокированным сигналам.

Возвращает: 0 — Ok, −1 — ошибка.

Сигнальная маска наследуется при **fork** , и сохраняется при **execve** .

Современный метод установки обработчика сигнала

sigaction

Недостатков системного вызова **signal** лишён вызов **sigaction**:

```
int sigaction(int signum,
              const struct sigaction *act,
              struct sigaction *oldact);
```

signum — номер сигнала.
act — указатель на структуру, описывающий новый обработчик. Если 0 — обработчик не изменяется.
oldact — указатель на структуру, в которую будет сохранена информация о старом обработчике. Если 0 — информация не сохраняется.
Возвращает: 0 — Ok, −1 — ошибка.

Описание struct sigaction

Для пользователя структура sigaction выглядит так:

```
struct sigaction {
    void        (*sa_handler)(int);                 // union?
    void        (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t    sa_mask;
    int         sa_flags;
    void        (*sa_restorer)(void); // unused
};
```

sa_handler — указатель на обработчик в стиле signal (получает 1 аргумент).

sa_sigaction — указатель на обработчик в новом стиле (получает 3 аргумента).

sa_mask — сигнальная маска, описывающая, какие ещё сигналы будут заблокированы во время работы обработчика.

sa_flags — набор флагов.

Описание struct sigaction

Флаги

Поле **sa_flags** может содержать следующие флаги:
SA_SIGINFO — установить обработчик, получающий 3 аргумента (в стиле sigaction): номер сигнала, указатель на структуру siginfo_t, описывающую сигнал, и указатель на описание контекста.
SA_NODEFER — не блокировать сам сигнал на время его обработки (по умолчанию блокируется).
SA_NODEFER — восстановить старый обработчик после одного срабатывания нового (синоним — SA_ONESHOT).
SA_NOCLDSTOP — не получать уведомления, если signum == SIGCHLD, и потомок остановлен по SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU.
Могут быть ещё SA_NOCLDWAIT, SA_ONSTACK, SA_RESTART . . .

Расширенный вариант kill — sigqueue

```
int sigqueue(pid_t pid, int sig,
             const union sigval value);

union sigval {
  int   sival_int;
  void *sival_ptr;
};
```

Первые 2 аргумента и возвращаемое значение полностью аналогичны аргументам *kill*. Последний аргумент позволяет передать дополнительную информацию в процесс, обрабатывающий сигнал. Принимающий процесс должен задать обработчик с тремя аргументами, и использовать поле si_value структуры siginfo_t.