Код

sigset_t

Целое число - набор сигналов - битовая маска

sigfillset

`int sigfillset(sigset_t *set);` - инициализирует набор, содержащий все сигналы
Возврат: в случае успеха 0, иначе -1 и errno
Ошибки: нет

sigemptyset

`int sigemptyset(sigset_t *set);` - инициализирует пустой набор сигналов
Возврат: в случае успеха 0, иначе -1 и errno
Ошибки: нет

sigaddset

`int sigaddset(sigset_t *set, int sig);` - добавить в набор
Возврат: в случае успеха 0, иначе -1 и errno
Ошибки: EINVAL недопустимый или неподдерживаемый sig

pthread_sigmask

`int pthread_sigmask(int how, const sigset_t *set, sigset_t *oldset);` - изменить или извлечь маску.
Возврат: в случае успеха 0, иначе номер ошибки
Ошибки:

- EFAULT set или oldset указывает на место вне выделенной памяти
- EINVAL или how некорректный, или ядро не поддерживает размер, переданный в sigsetsize

write

`ssize_t write(size_t count, int fd, const void buf[count], size_t count);` - записывает count байтов из buffer в файл, на который указывает файловый дескриптор fd.

Мы используем write потому что printf не является сигналобезопасной, а значит мы не можем использовать его и в функциях, и в обработчике. Подробнее в man 7 signal-safety.
Возврат: в случае успеха количество записанных байт, иначе -1 и errno
Ошибки:

- EAGAIN

- EWOULDBLOCK
- EBADF
- EDESTADDRREQ
- EDQUOT
- EFAULT
- EFBIG
- EINTR
- EINVAL
- EIO
- ENOSPC
- EPERM
- EPIPE

struct sigaction и sigaction

```
struct sigaction {
    void (*sa_handler)(int); /* Адрес обработчика */
    sigset_t sa_mask; /* Сигналы, блокируемые во время вызова обработчика */
    int sa_flags; /* Флаги, контролирующие активацию обработчика */
    void (*sa_restorer)(void); /* Не для использования в приложениях */
};
```

Поле sa_restorer является служебным, нужен для восстановления контекста в той точке, на которой процесс был прерван. В sa_handler можно записать SIG_DFL (по умолчанию), SIG_IGN (игнорирование) и указатель на адрес handler. sa_flags можно указать:

- SA_SIGINFO - обработчик принимает 3 аргумента, а не 1.
- ...

`int sigaction(int sig, const struct sigaction *act, struct sigaction *oldact);` - это системный вызов для изменения действия, выполняемого процессом при получении определенного сигнала.

sig может быть любым допустимым сигналом, кроме SIGKILL и SIGSTOP. Устанавливается новое действие из act, а в oldact записывается предыдущее.

Возврат: в случае успеха 0, иначе -1 и errno
Ошибки:

- EFAULT act или oldact указывает на место вне выделенной памяти
- EINVAL sig невалидный или попытка изменить SIGKILL и SIGSTOP

sigwait

`int sigwait(const sigset_t *set, int *sig);` - ожидает доставки сигнала, входящего в набор set, затем возвращает его внутри sig. Если с помощью sigwait() сигнал ожидается сразу несколькими

потоками, он сможет быть доставлен только одному из них; при этом невозможно предсказать, какому именно.

Приостанавливает выполнение текущего потока до тех пор, пока сигнал из set не появится в pending. Чтобы он попал в pending, его нужно сначала заблокировать с sigprocmask. Функция получает сигнал, удаляет его из pending и возвращает его номер в sig.

Возврат: в случае успеха 0, иначе -1 и errno
Ошибки: EINVAL set содержит недопустимый сигнал

Теория

Общее о сигналах

Сигнал - это оповещение процесса о том, что произошло некое событие. Иначе говоря это способ межпроцессного взаимодействия.

Системный вызов kill (pid, sig) -> sys_kill регистрирует сигнал в структуре процесса в очереди сигналов. Но выполняется он не сразу, а только когда процесс обратится к ядру по средством системного вызова: ядро скажет, что у него висит сигнал и запустит соответствующий handler. То есть у нас есть генерация сигнала и доставка в процесс. Промежуток между генерацией и доставкой называется ожиданием (pending).

Все handler по умолчанию хранятся в блоке .text. А указатели хранятся в структуре процесса. Если handler не создан, то будем использовать handler по умолчанию. Также в структуре процесса (впрочем и в структуре потока) есть поля pending (висящие), ignored, blocked - соответственно ожидающие, игнорируемые и заблокированные сигналы. Они реализованы как 32 битные маски, т.к. сигналы от 1 до 31.

Типы сигналов и действия по умолчанию подробно написаны в man 7 signal.

Таблица 20.1. Сигналы Linux

Имя	Номер сигнала	Описание	SUSv3	По умолчанию
SIGABRT	6	Аварийно завершить процесс	+	Ядро
SIGALRM	14	Время таймера реального времени истекло	+	Заверш.
SIGBUS	7 (SAMP=10)	Ошибка доступа к памяти	+	Ядро
SIGCHLD	17 (SA=20, MP=18)	Дочерний процесс завершен или остановлен	+	Игнор.
SIGCONT	18 (SA=19, M=25, P=26)	Продолжить, если завершен	+	Прод.
SIGEMT	н/опр (SAMP=7)	Аппаратная ошибка		Заверш.
SIGFPE	8	Арифметическое исключение	+	Ядро
SIGHUP	1	Потеря соединения	+	Заверш.
SIGILL	4	Недопустимая инструкция	+	Ядро
SIGINT	2	Прерывание с терминала	+	Заверш.
SIGIO/ SIGPOLL	29 (SA=23, MP=22)	Возможен ввод/вывод	+	Заверш.
SIGKILL	9	Императивное завершение	+	Заверш.

...

Что такое обработчик сигнала?
Обработчик сигнала - это функция, вызываемая при получении указанного сигнала процессом. Активация обработчика сигнала может в любое время прервать ход выполнения программы. Ядро осуществляет вызов обработчика от имени процесса, а после возобновляет выполнение с той точки, где процесс был прерван.

Что мы можем сделать с полученным сигналом?
По умолчанию обработчик сигнала, который может:

- Завершить процесс
- Игнорировать сигнал
- Приостановиться
- Возобновить выполнение после приостановки

Что такое диспозиция сигнала?
Это изменение действия, выполняемого при его получении: действие по умолчанию, игнорирование, обработчик сигнала. Мы сообщаем ядру, что хотим использовать обработчик. Нельзя сделать так, чтобы сигнал завершал процесс или сбрасывал дамп ядра.

Почему нельзя использовать printf?
Реентерабельные функции - функции, которые могут одновременно безопасно выполняться

несколькими потоками в рамках одного процесса. Безопасно значит, что функция достигает ожидаемого результата, не зависимо от других потоков.

Функция является нереентерабельной, если она обновляет глобальные или статические структуры данных. Например: malloc работает с динамической памятью, **printf и scanf используют для операций статические структуры данных. Именно поэтому при использовании printf можно увидеть странный вывод, повреждение данных или вовсе аварийное завершение программы**.

Однако, по стандарту функция является небезопасной, только когда инициация обработчика сигнала прерывает выполнение небезопасной функции и если сам обработчик также вызывает небезопасную функцию. То есть достаточно, чтобы обработчик вызывал только реентерабельные функции.

Подробнее о signal-safety функциях можно почитать в man 7 signal-safety.

Отдельно про потоки

Ключевые особенности:

- Назначение сигналов распространяется на весь процесс. Если любому потоку придет сигнал остановки или завершения, ее выполнят все потоки процесса.
- Все потоки в процессе разделяют одно и то же действие для каждого сигнала. Если у одного потока есть обработчик для SIGINT, то обработчик можно будет вызвать из любого потока. Если поток решит игнорировать сигнал, то этот сигнал будет игнорироваться всеми потоками.
- Сигнал может быть отправлен как процессу в целом, так и отдельному потоку. Сигнал направлен в поток, если:
  - он сгенерирован при выполнении аппаратной инструкции (SIGBUS, SIGSEGV)
  - SIGPIPE при попытке записи в поврежденный конвейер
  - отправлен с помощью pthread_kill или pthread_sigqueue, которые позволяют потокам одного процесса общаться между собой сигналами
- Все сигналы, сгенерированные другими механизмами направлены на весь процесс.
- Когда сигнал доставляется многопоточному процессу, у которого есть подходящий обработчик, ядро наугад выбирает один поток и доставляет сигнал ему.
- Маска сигнала относится к каждому отдельному потоку, нет понятия глобальной маски. Таким образом программа может контролировать потоки, которые должны получить сигнал.
- Ядро ведет учет сигналов как для процесса в целом, так и для каждого его потока.

Изменение масок сигналов потока
Новый поток наследует от своего создателя копию сигнальной маски.

`int pthread_sigmask(int how, const sigset_t *set, sigset_t *oldset);` - изменить или извлечь маску

Отправка сигнала потоку
`int pthread_kill(pthread_t thread, int sig);` - отправляет сигнал sig потоку в том же процессе.

Она реализована с применением системного вызова tgkill(tgid, tid, sig), подробнее в tgkill(2).

Обработка асинхронных сигналов
Рекомендуется:

- Заблокировать любые асинхронные сигналы в главной программе до создания потоков, тогда каждый поток унаследует копию сигнальной маски.
- Создаем отдельный поток, который принимает входящие сигналы с помощью sigwait().

Теперь асинхронные сигналы принимаются синхронно. То есть выделенный поток может безопасно работать с разделяемыми ресурсами и вызывать функции, не адаптированные к работе с асинхронными сигналами.

`int sigwait(const sigset_t *set, int *sig);` - ожидает доставки сигнала, входящего в набор set, затем возвращает его внутри sig.