Сигналы

Сигналы

- Средство асинхронного взаимодействия процессов
- Посылаются:
  - Одним процессом другому процессу
  - Ядром ОС процессу для индикации событий, затрагивающих процесс
  - Ядром ОС процессу в ответ на некорректные действия самого процесса
  - Процессом самому себе

Виды сигналов

- Асинхронные — могут поступить процессу и вызвать обработку в произвольный момент времени
- Синхронные — поступают процессу и вызывают обработку в определенные моменты времени (например, в результате вызова kill, или в результате попытки выполнения некорректной инструкции)

Стандартные сигналы

- Взаимодействие процессов (асинхронное):
  - SIGINT — завершение работы процесса, посылается при нажатии Ctrl-C
  - SIGTERM — завершение работы процесса
  - SIGKILL — завершение работы процесса (нельзя предотвратить)
  - SIGQUIT — завершение работы процесса с выдачей core dump
  - SIGUSR1, SIGUSR2 — произвольного назначения (определяется пользователем)
  - SIGSTOP — приостановка работы процесса (нельзя предотвратить)

Стандартные сигналы

- Ядро процессу (асинхронные)
  - SIGHUP — отключение от терминала
  - SIGALRM — срабатывание таймера
  - SIGCHLD — завершение работы сыновнего процесса

Стандартные сигналы

- Ядро процессу в ответ на ошибочное действие (синхронные)
  - SIGILL — недопустимая инструкция
  - SIGFPE — ошибка вычислений с плавающей точкой (обычно — целочисленное деление на 0)
  - SIGSEGV — ошибка доступа к памяти
  - SIGPIPE — запись в канал, закрытый на чтение

Стандартные сигналы

- Процесс самому себе (синхронные сигналы)
  - SIGABRT

- Перечислены не все сигналы. См. список: man 8 signal
- Процесс может послать другому процессу любой сигнал (в т. ч., например, SIGSEGV)
- Все сигналы, посылаемые одним процессом другому, асинхронны
- Сигналы, посылаемые процессом самому себе синхронны

Способы обработки сигнала

- Стандартная реакция на сигнал (реакция по умолчанию)
  - Завершение работы процесса (большинство сигналов)
  - Завершение работы процесса с записью core dump (SIGSEGV, SIGABRT...)
  - Пустая реакция (ничего не делать) (SIGCHLD)
  - Приостановка работы процесса (SIGSTOP)
- Игнорирование сигнала (кроме SIGSTOP, SIGKILL)

Способы обработки сигнала

- Пользовательская обработка — назначение функции, которая будет вызвана для обработки поступившего сигнала
- Функция обработчик:
  void handler(int signal);
- Не возвращает значения, принимает номер сигнала, который обрабатывает — одна и та же функция-обработчик может использоваться для обработки нескольких сигналов

Специальные сигналы

- Сигналы SIGKILL и SIGSTOP не могут быть перехвачены, заблокированы или проигнорированы
  - SIGKILL снимает процесс с выполнения ВСЕГДА!

Посылка сигнала

- Системный вызов kill

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

- pid > 0 — посылка указанному процессу
- pid == 0 — посылка всем процессам текущей группы
- pid == -1 — посылка всем процессам, которым процесс имеет право послать сигнал
- pid < -1 — посылка всем процессам в группе -pid

Доставка сигнала

Процесс 1000

```
kill(1234, 5);
```

Процесс 2000

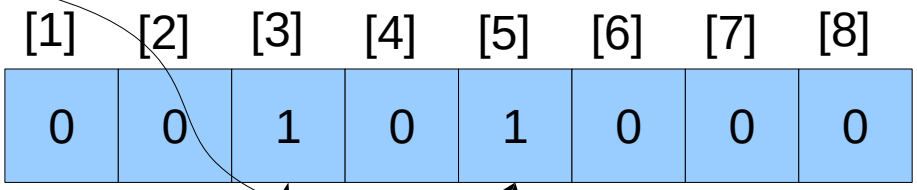
```
kill(1234, 3);
```

Процесс 2009

```
kill(1234, 5);
```

Процесс pid == 1234

Маска сигналов, ожидающих доставки (pending)



Доставка сигнала (I этап)

- Во множестве сигналов, ожидающих доставки, устанавливается соответствующий бит (возможно, он уже был установлен)
- Если процесс был заблокирован из-за ожидания ввода-вывода, он переносится в очередь процессов, готовых к выполнению, и настраиваются действия пост-обработки

Доставка сигнала (II этап)

- При запуске процесса из очереди готовых процессов на выполнение проверяется множество сигналов, ожидающих доставки и не заблокированных
- Из таких сигналов выбирается некоторый сигнал (обычно с минимальным номером) и доставляется процессу:
  - Удаляется из множества сигналов, ожидающих доставки
  - Производится либо стандартная обработка, либо вызов пользовательской функции

Доставка сигнала (II этап)

- При вызове пользовательской функции:
  - Производится настройка стека для обеспечения продолжения работы процесса после завершения обработчика
  - Модифицируется множество заблокированных процессом сигналов (только на время работы обработчика)

Слияние сигналов

- От посылки сигнала процессу до начала выполнения функции обработки может пройти некоторое время
- За это время один и тот же сигнал может быть послан процессу несколько раз
- Обработчик сигнала будет вызван **только один раз**
- Сигналы нельзя использовать там, где требуется учет количества поступлений

Установка обработки сигнала

```
typedef void (*sighnd_t)(int);
sighnd_t signal(int signum, sighnd_t hnd);
```

- SIG_IGN — игнорировать сигнал
- SIG_DFL — установить обработку по умолчанию
- Иначе задается функция-обработчик
- Возвращается старая обработка сигнала
- Обработку сигналов SIGKILL, SIGSTOP изменить нельзя

Особенности обработки сигналов в разных системах

- Переустановка обработчика: в System V обработчик сигнала сбрасывается на обработку по умолчанию, в BSD и Linux — нет

System V

```
void hnd(int signo)
{
    signal(signo, hnd);
    // …
}
```

BSD, Linux

```
void hnd(int signo)
{
    // …
}
```

Если в System V непрерывно посылать процессу сигналы при высокой загрузке системы, процесс не успеет восстановить обработчик сигнала и ядро снимет процесс с выполнения — DOS (Denial of Service) атака

Особенности обработки сигналов в разных системах

- Блокирование сигнала: в BSD и Linux на время обработки сигнала этот сигнал блокируется, в System V не блокируется
- При высокой загрузке системы в System V поток сигналов может привести к повторному входу в обработчик сигнала, что может привести к ошибке

Особенности обработки сигналов в разных системах

- Перезапуск системных вызовов: если сигнал был доставлен в процесс в тот момент, когда процесс находится в состоянии ожидания:
  - В System V системный вызов завершается с ошибкой EINTR, эту ошибку в процессе необходимо обработать и при необходимости перезапустить системный вызов
  - В BSD, Linux системный вызов перезапускается автоматически
  - Не перезапускаются: sleep, pause, select

Обработка сигналов в Linux

- На время выполнения обработчика сигнала повторное поступление сигнала блокируется. Если сигнал поступил в это время, обработчик сигнала будет перезапущен как только доработает до конца.
- Настройки обработки сигналов сохраняются при вызове обработчика. Обработчик достаточно установить один раз.
- Если процесс ожидал обмена данными (вызовы read, write, open, accept, wait, …), после завершения обработчика процесс продолжит ожидание обмена.
- Если процесс ожидал прихода сигнала (sleep, pause, usleep, nanosleep, sigsuspend, select, pselect, …), после завершения обработчика обработка системного вызова завершится

Особенности обработки сигналов в разных системах

- Схема обработки сигналов BSD и Linux более удобна для программиста и более надежна
- В дальнейшем будем предполагать, что используется эта схема
- Системный вызов sigaction позволяет устанавливать обработчик произвольным образом комбинируя свойства

Обработчики сигналов

- В обработчиках сигналов можно использовать только асинхронно-безопасные (async signal safe) стандартные функции
- Большинство системных вызовов (включая fork и exec) — асинхронно-безопасные
- Функции работы с динамической памятью (new, delete, malloc, free), функции работы с потоками (fopen, fprintf, <<) - **не асинхронно-безопасные**

Системно-зависимые особенности signal(2)

- На время обработки (выполнения обработчика сигнала) повторный вызов текущего обработчика может блокироваться или не блокироваться
- Обработчик может сбрасываться после запуска или не сбрасываться
- Некоторые системные вызовы (read, write, accept...) могут перезапускаться или завершаться с errno == EINTR
- Вызов sigaction(2) позволяет управлять этим

Безопасная обработка сигналов

- Безопаснее всего в обработчике сигнала устанавливать глобальный флаг поступления сигнала, который обрабатывать в основной программе
- Для этого требуются доп. средства управления сигналами

```
volatile sig_atomic_t sigint_flag;
void hnd(int s)
{
  sigint_flag = 1;
}
```

Volatile, sig_atomic_t

- Ключевое слово volatile обозначает, что значение переменной может измениться «неожиданно» для компилятора программы
- Компилятор не должен пытаться оптимизировать обращения к переменной (например, загружая ее на регистр)
- Тип sig_atomic_t — это целый тип (обычно int), для которого гарантируется атомарная запись и чтение

Множества сигналов

// очистка множества

```
void sigemptyset(sigset_t *pset);
```

// заполнение множества

```
void sigfillset(sigset_t *pset);
```

// добавление сигнала в множества

```
void sigaddset(sigset_t *pset, int signo);
```

// удаление сигнала из множества

```
void sigdelset(sigset_t *pset, int signo);
```

Блокирование сигналов

- Если сигнал заблокирован, его доставка процессу откладывается до момента разблокирования

```
int sigprocmask(int how, const sigset_t *set,
                sigset_t *oldset);
```

- SIG_BLOCK — добавить сигналы к множеству блокируемых
- SIG_UNBLOCK — убрать сигналы из множества блокируемых
- SIG_SETMASK — установить множество

Ожидание поступления сигнала

```
int sigsuspend(const sigset_t *mask);
```

- На время ожидания сигнала выставляется множество блокируемых сигналов mask
- После доставки сигнала восстанавливается текущее множество блокируемых сигналов

Отображение сигналов на файловые дескрипторы

- Системный вызов signalfd(2) позволяет создать файловый дескриптор, работая с которым можно получать уведомления о поступлении сигналов
  - signalfd — создает файловый дескриптор
  - select/poll/epoll — ожидание события (прихода сигнала)
  - read — ожидание прихода сигнала и получения информации о нем
  - close — закрытие

Стратегия корректной работы с сигналами

- Функции-обработчики сигналов устанавливают флаг поступления сигнала
- Программа выполняется с заблокированными сигналами
- Сигналы разблокируются только на время ожидания прихода сигнала (с помощью sigsuspend или pselect) или используется signalfd а сигналы не нужно разблокировать