

## Сигналы. Программные каналы.

IPC - Inter process communication System V.

- Межпроцесс. обм. не гарантирует общ. конкурентич.

Стандарт POSIX.1 FIPS

Portable operating system Interface ... Unix

Federal Information processing standard.

Разработан орг. инжиниром ст.д. СМФ (?)

X/Open Portability Guide (XPG3, 1984 - XPG4)

Средства межпроцессорного взаимодействия:

1. Сигналы
2. Программные каналы / message / mail - ор
3. Семафоры.
4. Очереди сообщений.
5. Регистрируемые файлы
6. RPC - remote procedure call

### Сигналы

В Linux есть все!

В класс. Unix не больше 20ти сигналов.

(21!) Сигнал — средство меж. проц. взаимодействия, инициирует процесс о событии.

Все сигналы / типы сигналов предопределены.

В windows - события.

Механизм сигналов позволяет процессам реагировать на события, которые не могут произойти внутри самого процесса - сигнал.  
событие, вне-асинхронное событие.

Процесс может реагировать на полуц. сигнал. Реакция процесса на тревоживший его сигнал зависит от того, как сам процесс определил свою реакцию на полуц. сигнал.

Может:

- 1) Пренебречь
- 2) Отреагировать по умолчанию (специф. опр. в ядре.)
- 3) Опред. свою реакцию на полуц. сигн.

#define NSIG 20  
#define SIGHUP 1 → разрыв связи с терминалом.  
— “ — SIGINT 2 (лад. демон)  
— “ — SIGQUIT 3 → заверш. с терминалом.  
— “ — SIGKILL 9 → сил. выг. kill, выг. в процессе.  
— “ — SIGSEGV 11 → выход из процесса с ошибкой ассиста адресации  
Размер, т.к. в течение процессом может быть динамическое АП.

Но один процесс не может опр. в АП другого. Потому контролировать - отслеживать выход процесса из процесса его АП.

- “ — **SIGSYS** 12 → ошибки ввода-вывода
  - “ — **SIGPIPE** 13 → данные в канал есть, читать неки.  
**недопустимый сигнал.**
  - “ — **SIGALARM** 14 — можно уст. опр. моменту с помощью alarm => сигнал системного таймера
  - “ — **SIGTERM** 15 — kill в течение ком. строки
  - “ — **SIGUSER1** 16 { можно опр. пользователем.
  - “ — **SIGUSER2** 17 }
  - “ — **SIGCLD** 18 — сигнал, сопровожд. заверш. него проц. недавно.
  - “ — **SIGPWR** 19 — падение напряжения  
— реакция на сигнал по умолч.
- #define SIG\_SETFL (int (\*)()) 1  
#define SIG\_IGN (int (\*)()) 2
- сигнал будет игнорироваться.

Системный вызов kill: **kill**, **signal** погаже лучше

**Системный вызов kill**

int kill(int pid, int sig); kill(pid, sig);  
- сигнал sig будет передан процессу с pid.

**Процесс, отсл. в группе** = процесс одной группы **может** получать один и тот же сигнал.

$\angle = 1 :$  —  $\emptyset$  — сигнал SIG будет пропущен  
все процессы с gpid = gpid процесса, который kill.  
кроме (pid0, i).  
реакции М.З. проигнорированы.

- - 1 - сигнал SIG\_DYING несет  
приведение к виду вид  
приведен, борж. kill.

kill (getpid(), SIGALARM); - если. будет ненорм  
сигнал приведен, борж. kill.

### Системный борж signal()

void (\*signal (int sig, void (\*handler)(int)))(int);

Пример

```
#include <iostream.h>
#include <signal.h>

int sig_handler (int sig_num) {
    signal (sig_num, sig_handler);
    cout << "catch sig" << sig_num << endl;
}

int main (void)
{
    signal (SIGTERM, sig_handler);
    signal (SIGINT, SIG_IGN);
    signal (SIGSEGV, SIG_DFL);
    pause ();
}
```

Э способа фиксирует приведен ид толь  
кодома сигналы.

sig\_handler - собственный обработчик сигнала.

- не есть стандартный POSIX.1, опр. в ANSI C.
- возвращается указатель на стековый обр. сигнала.

```

#include <signal.h>
int main (void) {
    void (*old_handler)(int)=signal(SIGINT,SIG_IGN);
    //если true
    signal(SIGINT,old_handler);
}
...
    (иначе SIG_DFL)

```

Это имеет смысл, когда на один и тот же сигнал  
программист хочет различать различные реакции в процессе выполнения.

```

int sigaction (int sig-номер,
               struct sigaction *action,
               struct sigaction *old-action);
    - систем. вызов (б. Linux и Posix)

```

Небходимо иметь полную структуру sigaction.  
Родсп. указывает на старый обработчик.

Зачем ус. собр. обр.? С помощью его можно  
менять механизм ход выполнения  
процесса - синхронно, асинхронно.  
(бывш. соф.) (бывш. обр.)

Программа: ошибка в handler'е.

В POSIX: sigsetjmp() - ус. 1 или неки. т. переход  
сиг. в. siglongjmp() - обеспечивает переход на  
один из возможн. точек.

## Программное каналье.

Именованное

Несимметричное

Pipe - труба, по которой можно передавать данные.

```
int pipefd[2];  
pipe(pipefd);
```

Несимметричный - не именем озаглавлен, но есть дескриптор, в ред-те exec. fork(), => наслед. дескр. откр. озаглавл. Теряется именем.

пр-канал может взаимодействовать только родителями.

программной канал.

Именованной 'mkfifo' на Mac OS

'mknode' - р - на Linux.

- специальный озаглавл. (буква r)

буфер типа FIFO, создается в сис. области памяти, свидетельство об этом в том, что пр-канал получает идентификатор АП, потому что он - реагирует на сообщения из АП-АП пары сис.

Примеч., любой pipe будет иметь размер - 1024 (1024 байт).

Pipe передается только массив-дескрипторов.

Две реализации взаимодействия.

- в программном канале нельзя писать, если из него читают (и наоборот).

В случае И.И. блокируется. следует  
использовать прокси, избегающие его использования.

Труба будет разрезаться на 3<sup>х</sup> узлах:

- 1) В симметричной начинке, обе рукоятки зеркальны.
- 2) На диске (при перевороте обеих начинок),  
исп. стат. сим. волны для работы с крайними.
- 3) Если промежуток зажимов  $> 4096$  байт, то  
труба будет разрезана во времени, блокируется  
прежде до тех пор, пока данное не  
будет исправлено.

Функция I/O обеспечивает блокировку,  
если канал занят.

write, если есть место - успешно  
иначе - заблокирован.

read - заблокирует - блокирован, если ПК пытается

ПК - это потоковая модель передачи данных,

Перев демонизе, already - exec(),  
main, в некотором смысле он

поток, ps - stat - execve

Выполн + задание

в очередь



Демониз (Над next)