signals API in Linux

March 30, 2017

signal

C89, C99, POSIX:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- handler: SIG_IGN, SIG_DFL или указатель на функцию

signal

C89, C99, POSIX:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- handler: SIG_IGN, SIG_DFL или указатель на функцию
- При вызове обработчика сигнала:
  - сбросить обработчик и не заблокировать сигнал до конца обработчика (original UNIX, System V)

signal

C89, C99, POSIX:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- handler: SIG_IGN, SIG_DFL или указатель на функцию
- При вызове обработчика сигнала:
  - сбросить обработчик и не заблокировать сигнал до конца обработчика (original UNIX, System V)
    - сигнал может прийти снова до того, как процесс успеет восстановить обработчик

signal

C89, C99, POSIX:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- handler: SIG_IGN, SIG_DFL или указатель на функцию
- При вызове обработчика сигнала:
  - сбросить обработчик и не заблокировать сигнал до конца обработчика (original UNIX, System V)
    - сигнал может прийти снова до того, как процесс успеет восстановить обработчик
  - не сбрасывать обработчик и заблокировать сигнал до конца обработчика; после этого разблокировать сигнал (BSD)

signal

C89, C99, POSIX:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- handler: SIG_IGN, SIG_DFL или указатель на функцию
- При вызове обработчика сигнала:
  - сбросить обработчик и не заблокировать сигнал до конца обработчика (original UNIX, System V)
    - сигнал может прийти снова до того, как процесс успеет восстановить обработчик
  - не сбрасывать обработчик и заблокировать сигнал до конца обработчика; после этого разблокировать сигнал (BSD)
    - иногда в результате обработки хочется заблокировать сигнал

signal

C89, C99, POSIX:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- handler: SIG_IGN, SIG_DFL или указатель на функцию
- При вызове обработчика сигнала:
  - сбросить обработчик и не заблокировать сигнал до конца обработчика (original UNIX, System V)
    - сигнал может прийти снова до того, как процесс успеет восстановить обработчик
  - не сбрасывать обработчик и заблокировать сигнал до конца обработчика; после этого разблокировать сигнал (BSD)
    - иногда в результате обработки хочется заблокировать сигнал
  - сейчас в Linux:
    - signal syscall: семантика, как в System V
    - signal glibc wrapper: семантика, как в BSD

sigaction

```
int sigaction(int signum,
              const struct sigaction * act,
              struct sigaction * oldact);
struct sigaction {
   void     (*sa_handler)(int);
   void     (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t   sa_mask;
   int        sa_flags;
};
```

sigaction

```
int sigaction(int signum,
              const struct sigaction * act,
              struct sigaction * oldact);
struct sigaction {
   void     (*sa_handler)(int);
   void     (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t   sa_mask;
   int        sa_flags;
};
```

- Флаги `sa_flags`:
  - `SA_SIGINFO` — использовать `sa_sigaction`
  - `SA_RESETHAND` — сбросить обработчик при вызове
  - `SA_RESTART` — перезапустить прерванный блокирующий системный вызов (как в BSD-шном `signal(2)`) (работает не для всех системных вызовов)

sigaction

```
int sigaction(int signum,
              const struct sigaction * act,
              struct sigaction * oldact);
struct sigaction {
   void     (*sa_handler)(int);
   void     (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t   sa_mask;
   int        sa_flags;
};
```

- Флаги `sa_flags`:
  - `SA_SIGINFO` — использовать `sa_sigaction`
  - `SA_RESETHAND` — сбросить обработчик при вызове
  - `SA_RESTART` — перезапустить прерванный блокирующий системный вызов (как в BSD-шном `signal(2)`) (работает не для всех системных вызовов)
  - `SA_NOCLDSTOP` — не генерировать `SIGCHLD` при остановке дочернего процесса
  - `SA_NOCLDWAIT` — не превращать дочерние процессы в зомби

sigaction

```
struct sigaction sa;
memset(&sa, 0, sizeof(struct sigaction));

sigemptyset(&sa.sa_mask);
sa.sa_handler = handler;
sigaction(SIGUSR1, &sa, NULL);
```

sigaction

```
struct sigaction sa;
memset(&sa, 0, sizeof(struct sigaction));

sigemptyset(&sa.sa_mask);
sa.sa_handler = handler;
sigaction(SIGUSR1, &sa, NULL);
```

- Для SIGKILL и SIGSTOP нельзя установить свои обработчики

Информация о сигнале

```
struct siginfo_t {
    int si_code;
    int si_pid;
    int si_uid;
    /* ... */
};
```

- `si_code`:
  - `SI_USER` — сигнал был послан кем-то через `kill`, выставлены `si_pid` и `si_uid`
  - `SI_KERNEL` — сингал был послан ядром
  - при `SIGFPE`:
    - `FPE_INTDIV` — целочисленное деление на ноль
    - `FPE_INTOVF` — переполнение целого числа
    - `FPE_FLTDIV` — деление на ноль числа с плавающей точкой
    - ...
  - для разных случаев много разных значений

realtime signals

- отправка сигналов

```
int sigqueue(pid_t pid, int sig,
             const union sigval value);
union sigval {
    int   sival_int;
    void *sival_ptr;
};
```

- обработка сигналов — `rt_sigaction`
  - `sigaction` glibc wrapper вызывает именно его

Все есть файл

```
int signalfd(int fd, const sigset_t * mask, int flags);
```

- Linux-specific