
Signals

System Programming

2019 여름 계절학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

- A long and thorough look at Unix signals
- The earlier implementations of signals
- POSIX.1 *reliable-signal* concept and all the related functions

□ Signal

- Software interrupts: a way of handling asynchronous events, e.g. Ctrl-C
- 15 signals in Version 7, 31 signals in SVR4/4.4BSD, FreeBSD 5.2.1, Mac OS X 10.3, and Linux 2.4.22, and 38 signals for Solaris 9
- `<signal.h>`

□ Conditions to generate a signal

- Terminal-generated signals, e.g. DELETE key (SIGINT)
- Hardware exceptions such as divide by 0, invalid memory reference (SIGSEGV), and the like
- `kill(2)` and `kill(1)`
- Software conditions, e.g. when out-of-band data arrives over a network connection (SIGURG), when a process writes to a pipe after the reader has terminated (SIGPIPE), and when an alarm clock expires (SIGALRM).

❑ Disposition (or action) of the signal

- Ignore the signal
 - SIGKILL and SIGSTOP can never be ignored.
 - Ignoring some signals, e.g. SIGFPE and SIGSEGV, results in undefined program behaviors.
- Catch the signal
- Default action
 - For most signals, it is to terminate the process

❑ Figure 10.1 Unix System signals

- “terminate+core” means that a memory image of the process is left in the file named `core`.

❑ SIGABRT

- Generated by `abort` function

❑ SIGALRM

- When `alarm` or `setitimer` function expires

❑ SIGBUS

- An implementation-defined hardware fault

❑ SIGCHLD

- When a child terminates or stops

❑ SIGCONT

- Sent to a stopped process when it is continued

❑ SIGEMT

- An implementation-defined hardware fault

❑ SIGFPE

- An arithmetic exception, such as divide-by-0, floating point overflow, and so on

❑ SIGHUP

- Sent to the controlling process if a disconnect is detected by the terminal interface
- Sent to each process in the foreground process group if the session leader terminates

❑ SIGILL

- When an illegal hardware instruction is execu

❑ SIGINFO

- Sent to all processes in the foreground process group when we type the status key (often Ctrl-T)

❑ SIGINT

- Sent to all processes in the foreground process group in case of the interrupt key (often DELETE or Ctrl-C)

❑ SIGIO

- To indicate an asynchronous I/O event

❑ SIGIOT

- To indicate implementation-defined hardware fault

❑ SIGKILL

- Can't be caught or ignored. A sure way to kill any process.

Signal Concepts

❑ SIGPIPE

- Generated when we write to a pipeline (a socket) when the reader (the other end) has terminated

❑ SIGPOLL

- When a specific event occurs on a pollable device

❑ SIGPROF

- When a profiling interval timer (set by the `setitimer`) expires

❑ SIGPWR

- On a system with a UPS, to instruct the `init` process to shutdown everything
- System V's `powerfail` and `powerwait` in `inittab` file

❑ SIGQUIT

- Sent to all processes in the foreground process group in case of the terminal quit key (often Ctrl-backslash)

❑ SIGSEGV

- To indicate an invalid memory reference

Signal Concepts

❑ SIGSTOP

- To stop a process, can't be caught or ignored

❑ SIGSYS

- To signal an invalid system call

❑ SIGTERM

- By the `kill(1)` command (by default)

❑ SIGTRAP

- An implementation-defined hardware fault

❑ SIGTSTP

- Sent to all processes in the foreground process group in case of the terminal suspend key (often Ctrl-Z)

❑ SIGTTIN

- When a background process tries to read from its controlling terminal

❑ SIGTTOU

- When a background process tries to write to its controlling terminal

❑ SIGURG

- To notify that an urgent condition has occurred, or in case of out-of-band data on a network connection

❑ SIGUSR1/SIGUSR2

- A user-defined signal for use in application programs

❑ SIGVTALRM

- When a virtual interval timer (set by `setitimer`) expires

❑ SIGWINCH

- When a window size (associated with (pseudo) terminal) is changed

❑ SIGXCPU/SIGXFSZ

- If soft CPU time limit / soft file size limit is exceeded

signal Function

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

```
void (*signal(int signo, void (*func) (int))) (int);
```

- *signo* in Figure 10.1
- *func*: SIG_IGN, SIG_DFL, or a signal handler
- It returns the pointer of the previous signal handler.

```
typedef void      Sigfunc(int);  
Sigfunc *signal(int, Sigfunc *);
```

□ Figure 10.2

```
$ ./a.out &  
[1]      4720  
$ kill -USR1 4720  
received SIGUSR1  
$ kill -USR2 4720  
received SIGUSR2  
$ kill 4720  
[1] + Terminated
```

*start process in background
job-control shell prints job number and process ID
send it SIGUSR1*

send it SIGUSR2

*now send it SIGTERM
./a.out &*

signal Function

Figure 10.2

```
static void      sig_usr(int);      /* one handler for both signals */
int
main(void)
{
    if (signal(SIGUSR1, sig_usr) == SIG_ERR)
        err_sys("can't catch SIGUSR1");
    if (signal(SIGUSR2, sig_usr) == SIG_ERR)
        err_sys("can't catch SIGUSR2");
    for ( ; ; )
        pause();
}

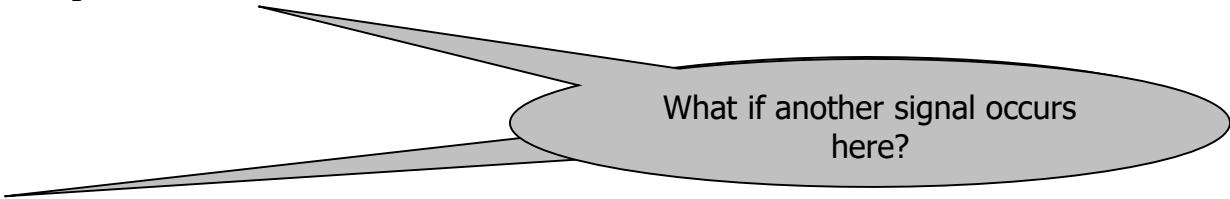
static void
sig_usr(int signo)      /* argument is signal number */
{
    if (signo == SIGUSR1)
        printf("received SIGUSR1\n");
    else if (signo == SIGUSR2)
        printf("received SIGUSR2\n");
    else
        err_dump("received signal %d\n", signo);
}
```

Unreliable Signals

❑ Unreliable signals in earlier versions of the Unix System

- Signals could get lost
- The action for a signal was reset to its default action each time the signal occurred.
- Unable to turn a signal off when it is not wanted (i.e. no signal blocking)

```
int sig_int_flag;
main()
{
    int sig_int();
    ...
    signal(SIGINT, sig_int);
    ...
    while (sig_int_flag == 0)
        pause();
    ...
}
sig_int()
{
    signal(SIGINT, sig_int);
    sig_int_flag = 1;
}
```



What if another signal occurs here?

❑ With earlier Unix systems, if a process caught a signal while being blocked in a “slow” system call, the system call was interrupted. It returned an error with `errno` set to `EINTR`.

❑ **Slow system calls**

- reads from and write to certain file types (pipes, terminal devices, and network devices)
- opens of files that block until some condition occurs
- `pause` and `wait`
- certain `ioctl` operations
- some of the IPC functions (Chapter 15)

❑ We now have to handle the error return explicitly.

again:

```
if ( (n = read(fd, buf, BUFSIZE)) < 0) {  
    if (errno == EINTR)  
        goto again; /* just an interrupted system call */  
    /* handle other errors */  
}
```

❑ Automatic restarting of certain interrupted system calls
under 4.2BSD

– `ioctl`, `read`, `readv`, `write`, `writv`, `wait`, and `waitpid`

- ❑ A signal is *generated, delivered, or pending*.
- ❑ If a signal is *blocked*, and if its action is either SIG_DFL or to catch the signal, then the signal remains *pending* until the process unblocks the signal or change the action to SIG_IGN.
- ❑ What if a blocked signal is generated more than once before the signal is unblocked?
 - Most Unix systems do not *queue* signals (i.e. deliver the signal once.)
- ❑ No order in which different signals are delivered to a process.
- ❑ *Signal mask* that defines the set of signals blocked.

kill and raise Functions

```
#include <signal.h>
int kill(pid_t pid, int signo)
int raise(int signo);
```

❑ kill sends a signal to a process or a group of processes

- $pid > 0$
 - Sent to the process whose process ID is pid .
- $pid == 0$
 - Sent to all processes whose $pgid$ equals the $pgid$ of the sender.
- $pid < 0$
 - Sent to all processes whose $pgid$ equals the absolute value of pid .
- $pid == -1$
 - Sent to all process for which the sender has permission to send a signal
- Permission to send a signal
 - The real or effective UID of the sender has to equal the real or effective UID of the receiver. (If `_POSIX_SAVED_IDS` is supported, then the receiver's saved set-user-ID is checked instead of its effective UID.)

❑ raise sends a signal to itself.

alarm and pause Functions

```
#include <unistd.h>
```

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

- ❑ When the timer expires, SIGALRM is generated.
- ❑ It returns 0 or number of seconds until previously set alarm

- *Only one alarm clock per process.* If there is a not-yet-expired clock for the process, the remaining seconds is returned.

```
#include <unistd.h>
```

```
int pause(void) ;
```

- ❑ **pause** suspends the calling process until a signal is caught. (it returns -1 with `errno` set to `EINTR`).

sigaction Function

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *act,
              struct sigaction *oact);

struct sigaction {
    void      (*sa_handler) (int); /* addr of signal handler, */
                                   /* or SIG_IGN, or SIG_DFL */
    sigset_t  sa_mask;             /* additional signals to block */
    int       sa_flags;            /* signal options */
    /* alternate handler */
    void      (*sa_sigaction) (int, siginfo_t *, void *);
}
```

sigaction Function

- ☐ A replacement of the `signal` function.
- ☐ The `sa_mask` specifies a set of signal that are added to the signal mask of the process. When the signal-catching function returns, the signal mask is reset to its previous value.
- ☐ OS includes the signal being delivered in the signal mask when the handler is invoked.
- ☐ The signal handler remain installed until explicitly changed.

- ☐ `sa_flags`
 - `SA_INTERRUPT`, `SA_NOCLDSTOP`, `SA_NOCLDWAIT`, `SA_NODEFER`,
`SA_ONSTACK`, `SA_RESETHAND`, `SA_RESTART`, `SA_SIGINFO`

abort Function

```
#include <stdlib.h>
```

```
void abort(void);
```

- ❑ It sends SIGABRT to the process to allow it to perform any cleanup before terminating.
- ❑ **abort** overrides the blocking or ignoring of the signal by the process (POSIX.1).
 - If the process doesn't terminate itself from this signal handler, POSIX states that, when the signal handler returns, `abort` terminates the process.
- ❑ **POSIX.1** requires that if the call to **abort** terminates the process, then it shall have the effect of calling `fclose` on all open standard I/O streams.

sleep Function

```
#include <unistd.h>
```

```
unsigned int sleep(unsigned int seconds) ;
```

❑ The calling process is suspended until either

- The amount of wall clock time specified has elapsed, or
- A signal is caught by the process and the signal handler returns

❑ In the case of a signal being caught, the return value is the number of unslept seconds.

Thank you for your attention !!

Q and A