SYSTEM PROGRAMMING: SIGNALS

Onur Tolga Şehitoğlu

METU, CEng 536 Lecture notes
October, 2006

Signals

- Software interrupts
- The most primitive way of interprocess communication
- System generated, terminal interaction or process generated.
- A signal value is an integers in the range 1-32 or 1-64.(1..

Signal	Explanation	Default action
SIGALRM	timer expired (alarm)	terminate
SIGBUS	hardware fault	terminate+core
SIGCANCEL	threads library internal use	ignore
SIGCHLD	change in status of child	ignore
SIGCONT	continue stopped process	continue/ignore
SIGFPE	arithmetic exception	terminate+core
SIGHUP	hangup	terminate
SIGILL	illegal instruction	terminate+core
SIGINT	terminal interrupt character	terminate
SIGIO	asynchronous I/O	terminate/ignore
SIGKILL	termination	terminate
SIGLWP	threads library internal use	ignore
SIGPIPE	write to pipe with no readers	terminate
SIGPOLL	pollable event (poll)	terminate
SIGPROF	profiling time alarm (setitimer)	terminate
SIGPWR	power fail/restart	terminate/ignore
SIGQUIT	terminal quit character	terminate+core
SIGSEGV	invalid memory reference	terminate+core
SIGSTOP	stop	stop process
SIGSYS	invalid system call	terminate+core
SIGTERM	termination	terminate
SIGTRAP	hardware fault	terminate+core
SIGTSTP	terminal stop character	stop process
SIGTTIN	background read from control tty	stop process
SIGTTOU	background write to control tty	stop process
SIGUSR1	user-defined signal	terminate
SIGUSR2	user-defined signal	terminate

Handling a Signal

- Processes can alter default behaviour of signals (except `SIGKILL`). Processes can register special functions called *handlers* in order to implement their own action when a signal is received.
- `#include<signal.h>`
  `void (*signal(int` *`signo`*`, void (*`*`func`*`)(int)))(int);`
  returns the previos disposition or error.
- *`func`* value can be either `SIG_IGN`, `SIG_DFL` or a function address for ignoring the signal, setting default behaviour for the signal and defining the handler respectively.
- When the signal *signo* is received, execution is interrupted the corresponding action is taken, then the execution may continue from where it is left.
- Signal number is passed as the integer value to the handler function.

- System calls may be interrupted when they take too long (like terminal input). Earlier systems return an error for the system call (EINTR). Current system restart most of such system calls, so signal handling is almost transparent.
- Most signals are generated by the system. Also superuser processes and process owners other process can generate/send signals: `#include <signal.h>`
  `int kill(pid_t pid, int signo);`
  `int raise(int signo);`
- There are four different conditions for the pid argument to kill.

`pid > 0`	The signal is sent to the process whose process ID is pid.
`pid == 0`	The signal is sent to all processes whose process group ID equals the process group ID of the sender
`pid < 0`	The signal is sent to all processes whose process group ID equals the absolute value of pid
`pid == -1`	SVR4 and 4.3+BSD sends signal to all processes for superuser calls. Otherwise all processes of the owner.

Alarm and Pause

- Process set timers:
```
#include<unistd.h>
unsigned int alarm(unsigned int seconds);
int pause(void);
```
- `alarm()` Sets an alarm clock scheduled to send `SIGALRM` after *`seconds`* seconds to current process.
- `pause(void)` waits until the current process receives a signal.

Reliable Signals

- Old Unix system signals were unreliable. Race conditions: getting another signal while current signal is being handled?
- All signals were delivered, ignored, handled or lost.
- Current signal implementation is reliable and more functional. Signals can be blocked. System keeps blocked signals in the pending state and delivers them when signal is unblocked. Restart of the system calls can be controlled.

Signal Sets

- Bitset manipulation functions in order to deal signal sets:

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);

int sigismember(const sigset_t *set, int signo);
```

- `sigemptyset` initialize the set to emptyset, `sigfillset` initialize the set to universal set (all signals included). `sigaddset` and `sigdelset` adds *`signo`* to set and subtracts it from the set respectively. `sigismember` is for member check.

Blocked Signals

- The set of blocked signals can be received and set by using `sigprocmask` system call.

```
#include <signal.h>
int sigprocmask(int how, const sigset_t * set,
                sigset_t * oset);
```

- *how* is either `SIG_BLOCK`, `SIG_UNBLOCK`, or `SIG_SETMASK`. If *curr* is the current set:

```
SIG_BLOCK : curr=curr ||set
SIG_UNBLOCK : curr=curr && ¬set
SIG_SETMASK : curr=set
```

- In order to inspect but not change the current mask: *set* is given as NULL.
- In order to change but not inspect old value: *oset* is given as NULL. If both non-NULL, mask is changed then old mask returned.

- In order to get list of pending (blocked but received) signals: `#include <signal.h> int sigpending(sigset_t *set);`
- Fills the current set of pending signals into *set*.
- Pending signals are delivered as soon as the signal is unblocked.

Handling the signal

- `#include <signal.h>`
  `int sigaction(int` *`signo`*`, const struct sigaction *` *`act`*`,`
  `struct sigaction *` *`oact`*`);`
- `sigaction` **structure:**

```
struct sigaction {
  void      (*sa_handler)(int);   /* addr of signal h
                                  /* or SIG_IGN, or S
  sigset_t sa_mask;               /* additional signa
  int      sa_flags;              /* signal options*/
};
```

```
#include "apue.h"
Sigfunc * signal(int signo, Sigfunc *func)
{ struct sigaction    act, oact;
    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if (signo == SIGALRM) {
#ifdef SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT;
#endif
    } else {
#ifdef  SA_RESTART
        act.sa_flags |= SA_RESTART;
#endif
    }
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```