System Programming

11. Signal (2)

Seung-Ho Lim

Dept. of Computer & Electronic Systems Eng.

sigprocmask() (1)

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

- set a signal set mask (for blocking or unblocking a signal(s))
  - if a signal is blocked, the signal handling is postponed until the signal is unblocked
- parameters
  - *how* : blocking or unblocking (see the next page)
  - *set* : signal set used for blocking or unblocking, type:sig_set_t
  - *oset* : the calling process's old signal set mask
- return
  - 0 if OK
  - -1 on an error

sigprocmask() (2)

- how description

how	*description*
SIG_BLOCK	Add the input signal set to the signal set mask of the process (for blocking)
SIG_UNBLOCK	Remove the input signal set from the signal set mask of the process (for nonblocking)
SIG_SETMASK	Replace the current signal set with the input signal set mask in the signal set mask of the process (for blocking)

Mask set handling : sigemptyset( )

- Just clear the signal set mask argument for set manipulation,
- This is not a set of the signal set mask of a process!

```
#incldue <signal.h>
// for initializing of a set
int sigemptyset (sigset_t *set);
    input:
        - set : signal mask
    return:
        - normal : 0
        - error : -1
```

Mask handling : sigfillset( ), sigaddset( )

- Set all signal set mask, // just set manipulation

int sigfillset (sigset_t *set);

input:

- set : signal mask set

return:

- normal : 0, error : -1

- Add a signal to a signal mask set, // just set manipulation

int sigaddset (sigset_t *set, int signo);

input:

- set : signal mask set
- signo : signal number

return:

- normal : 0, error : -1

Mask handling: sigdelset( ), sigismember( )

- Delete a signal from a signal mask set, // just set manipulation

int sigdelset (sigset_t *set, int signo);

input:- set : signal mask

- signo : signal mask

return: - normal : 0, error : -1

- Ask if a signal is in a signal mask set. // just set manipulation

int sigismember (sigset_t *set, int signo);

input: - set : signal mask

- signo : signal mask

return: - normal : 1 if signo is in the set, 0 if not,

- error: -1

sigpending()

```
#include <signal.h>

int sigpending (sigset_t *set);
```

- get the signal set of pending-signals
  - i.e. signals that have been delivered but not processed yet.
- parameter
  - set : signal mask structure to store the pending signals
- return
  - 0 if OK
  - -1 on an error

sigsuspend()

```
#include <signal.h>
int sigsuspend (const sigset_t *set);
```

- atomic operation for both setting mask and pause
  - temporarily replaces the signal mask of the calling process with the given mask set
  - pause (suspends) the process until delivery of a signal
- return when the signal is caught (delivered)
  - then `sigsuspend()` returns after the signal handler returns
  - the signal mask is restored to the state before the call to `sigsuspend()`.
  - but, if the signal terminates the process, then `sigsuspend()` does not return. (e.g. when use the default handler)

Usage of sigsuspend()(1)

sigsuspend.c

```
..      // defines
void sig_catch (int sig_no)
{
        printf("sig_catch, %d\n",sig_no);
}
int main()
{
        int pid;
        sigset_t mysigset, oldsigset;
        // empty the set
        sigemptyset(&mysigset);// empty the set
        // add SIGUSR1 to the set
        sigaddset(&mysigset, SIGUSR1);
        // set a user-defined signal handler
        signal(SIGUSR1, sig_catch);
        // block SIGUSR1
        sigprocmask(SIG_BLOCK, &mysigset, oldsigset);
```

```
 if ((pid = fork()) == 0) {  // child
        // to prevent the early handling before pause()
        sigsuspend(&oldsigset); // unblock and pause!

        printf("Child wake up\n");
        exit(0);
 } else {  // parent
        sleep(1);
        kill (pid, SIGUSR1);  // send the SIGUSR1 to the child
        wait();
 }
}
```

sleep()

```
#include <unistd.h>
unsigned int sleep (unsigned int seconds);
```

- parameter
  - *seconds* : waiting time in second, i.e. the process will be blocked for the seconds
- return
  - 0
  - time left to the wakeup time if the call is interrupted by a signal handler
- when being unblocked, a SIGALRM happens (same as alarm())
  - So be careful in using the sleep() and alarm() together! (do not use them together)
- cf.
  - ***nanosleep** (nano-sec);*
  - ***usleep** (micro-sec);*

System call and a signal (1)

- If a signal is delivered when a system call (e.g. `read()`) is being performed?
  - i.e., a signal is delivered when a process is blocked because of I/O in a system call.
- In this case, what is the next action of the process?
  - should the process continue to wait the I/O completion in a blocked state by postponing the signal handling?
  - or should the process do the signal handling first and then waits the I/O completion?

System call and a signal (2)

- Type 1
  - after the waiting I/O has been completed, the signal is handled.
  - when I/O completion is guaranteed such as disk I/O (disk file `read`())
  - linux's blocked state = ***TASK_UNINTURRUPTIBLE***
- Type 2
  - the process is waken up (***be ready***) and the do the signal handling first, (this means the system call read() is interrupted.)
  - linux's blocked state = ***TASK_INTURRUPTIBLE***
  - after the signal handling,
    - linux default: recall the I/O system call (ex: read(), getchar(), etc.)
    - other versions (UINX): ***error return*** from the I/O system call
- For both cases,
  - If **`siginterrupt`** (signal_no, TRUE/FALSE) is called with TRUE, the I/O system call is interrupted (error return) after the signal handling.

Example (1)

alarm-getchar.c

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

#define TIMEOUT     3           // login time limit = 3 sec. After alarm, ring a bell.
#define MAXTRIES    5           // retry login five times when timeout
#define LINESIZE    100         // login name/passwd buffer size
#define CTRL_G      '\007'      // bell
#define TRUE                    1
#define FALSE                   0

volatile int timed_out;         // set when an alarm occurs
char myline [LINESIZE];         // character buffer
void sig_catch(int);            // alarm signal handler
```

Example (2)

```
char *quickreply(char *prompt) {
int main()
{
        quickreply("login-name:");
}

char *quickreply(char *prompt) {
        void (*was)(int);
        int ntries, i;
        char *answer;

        was = signal (SIGALRM, sig_catch);
        siginterrupt (SIGALRM, TRUE);

      // set error return when a signal occurs
        for (ntries = 0; ntries <MAXTRIES; ntries++) {  // retry loop
            timed_out = FALSE;
            printf("\n%s > ",prompt);
            fflush(stdout);

            alarm(TIMEOUT);
```

```
        for (i = 0; i < LINESIZE; i++) {
                if ((myline[i] = getchar()) < 0)
                    break;  // error return by alarm
                if (myline[i] == '\n') {
                    myline[i] = 0;
                    break;  // end of line input
                }
        }

        // normal case or alarm case here

        alarm(0);  // reset the alarm

        if (!timed_out) { // normal case
                printf("%s",myline);
                break;
    }   // end of retry loop
    // normal or fail 5 times
    answer = myline;
    signal(SIGALRM,was);
    return(ntries == MAXTRIES ? ((char *) 0) : answer);
}
```

Example (3)

```
void sig_catch (int sig_no)
{
      timed_out = TRUE;
      putchar (CTRL_G);  // ring a bell
      fflush (stdout);       // insure that the bell-ring

      // reinstall the user defined signal handler
      signal (SIGALRM, sig_catch);
}
```

Result

```
#./alarm-getchar

login-name: >
login-name: >
login-name: >
login-name: >
login-name: >

#./alarm-getchar

login-name: >
login-name: >
login-name: > shlim
shlim
```

System Programming

signel-Timer

Seung-Ho Lim

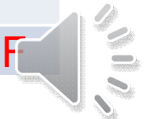
Dept. of Computer & Electronic Systems Eng.

Interval Timer

```
#include <sys/time.h>
int setitimer
(int which, const struct itimerval *value, struct itimerval *oval);
int getitimer (int which, struct itimerval *oval);
```

- an interval timer generates a SIGALRM signal *periodically*
  - used to implement a periodic job
- parameters
  - *which* : timer type
  - *value* : new interval (for setitimer)
  - *oval* : current(old) timer interval
    - may be NULL for setitimer
- return
  - 0 if OK or -1 on an error (with errno)

which	description
ITIMER_REAL	Time in real: SIGALRM
ITIMER_VIRTUAL	Time in user mode: SIGVALRM
ITIMER_PROF	Process running time (user mode + kernel mode): SIGPROF



itimerval strucure

```
struct itimerval {
     struct timeval it_interval; // periodic interval after the 1st alarm
     struct timeval it_value;    // first interval
}

struct timeval {
     long tv_sec;  // seconds
     long tv_usec; // micro seconds
}

/*
  it_interval = 0;    // for one time itimer
  it_value = 0;      // to turn off the itimer
*/
```

Interval Timer Example

itimer.c

```
#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void alarm_handler (int signo)
{
   printf (" Timer hit\n");
    // do the periodic job here
}
```

```
int main()
{
    struct itimerval delay;
    int ret;
    signal (SIGALRM, alarm_handler);
    delay.it_value.tv_sec = 5;  // first alarm
    delay.it_value.tv_usec = 0;
    delay.it_interval.tv_sec = 1;  // periodic
    delay.it_interval.tv_usec = 0;
    ret = setitimer ( ITIMER_REAL, &delay, NULL);
    if (ret) {
      perror (" setitimer ");
      exit(0);
    }
    while (1) {
       pause();
    }
}
```

Results

```
#gcc –o itimer timer.c
# ./itimer
 Timer hit
  Timer hit
  Timer hit
  Timer hit
  Timer hit
  Timer hit
  Timer hit
  Timer hit
  Timer hit
  Timer hit
```

POSIX Timer (Advanced)

- The POSIX Timer is a more advanced & controllable timer
  - for realtime applications
- See manual pages (e.g. $ man timer_create)
  - `timer_create`
  - `timer_settime`
  - `timer_gettime`
  - `timer_getoverrun`
  - `timer_delete`
- Note
  - Instead of the SIGALRM, another signal can be specified to be used. (SIGRTMIN ~ SIGRTMAX)
  - Instead of a signal hander, a handler thread can be specified to be used.
  - at compile time, **"-lrt"** must be added. (rt library)

POSIX Timer (Advanced)

```
#include <sys/time.h>
int timer_create(clockid_t clockid, struct sigevent *restrict evp,
timer_t *restrict timerid);
```

- create a POSIX per-process timer
- parameters
  - *clockid* : timer clock id
  - *evp* : timer event
  - timeid : timer id for the created timer
- return
  - 0 if OK or -1 on an error (with errno)

clockid	description
CLOCK_REALTIME	A settable system-wide real-time clock
CLOCK_MONOTONIC	A nonsettable monotonically increasing clock
…	…

POSIX Timer (Advanced)

```
#include <time.h>

int timer_settime(timer_t timerid, int flags,
                  const struct itimerspec *new_value,
                  struct itimerspec *old_value);
int timer_gettime(timer_t timerid, struct itimerspec *curr_value);
```

- arm/disarm and fetch state of POSIX per-process timer
- parameters
  - *timerid* : timer id
  - *Flags : time settings features*
  - *new_value* : timer expiration time
  - *Old_value* : old expiration time
- return
  - 0 if OK or -1 on an error (with errno)

itimerspec strucure

```
struct timespec {
           time_t tv_sec;                /* Seconds */
           long   tv_nsec;               /* Nanoseconds */
};

struct itimerspec {
           struct timespec it_interval;  /* Timer interval */
           struct timespec it_value;     /* Initial expiration */
};
```

sigaction()

```
#include <signal.h>
int sigaction
(int signo, const struct sigaction *act, struct sigaction *oldact);

/*
 struct sigaction {
   void (*sa_handler)(int):  /* address of handler */
   sigset_t sa_mask;  /* signals blocked during handler invocation*/
   int sa_flags;  /* flags controlling handler invocation */
   void (*sa_restorer)(void); /* NOT for application use */
 };
*/
```

- POSIX version for signal()
  - usually for multithreading or realtime applications
  - sig_flags and sa_restorer are set for MT and RT purposes

POSIX Timer Example (1)

posix-timer.c

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

#define SIGTIMER            (SIGRTMAX)
#define ONESHOTTIMER    (SIGRTMAX-1)

timer_t SetTimer(int signo, int sec, int mode);
void SignalHandler(int signo, siginfo_t * info, void *context);
timer_t timerid, oneshotTimer;

int main()
{
    struct sigaction sigact;
    sigemptyset(&sigact.sa_mask);
    sigact.sa_flags = SA_SIGINFO;
    sigact.sa_sigaction = SignalHandler;
```

POSIX Timer Example (2)

```
    // Set up sigaction to catch signal
    if (sigaction(SIGTIMER, &sigact, NULL) == -1) {
        perror("sigaction failed"); return -1;
    }
    if (sigaction(ONESHOTTIMER, &sigact, NULL) == -1)
    {
        perror("sigaction failed"); return -1;
    }

    // Establish a handler to catch CTRL+C and use it for exiting
    sigaction(SIGINT, &sigact, NULL);

    timerid = SetTimer(SIGTIMER, 1000, 1);
    oneshotTimer = SetTimer(ONESHOTTIMER, 5000, 0);
    while(1);
    return 0;
}
```

POSIX Timer Example (3)

```
timer_t  SetTimer(int signo, int sec, int mode)
{
      struct sigevent sigev;
      timer_t timerid;
      struct itimerspec itval;
      struct itimerspec oitval;

      // Create the POSIX timer to generate signo
      sigev.sigev_notify = SIGEV_SIGNAL;
      sigev.sigev_signo = signo;
      sigev.sigev_value.sival_ptr = &timerid;
```

```
      if (timer_create(CLOCK_REALTIME, &sigev, &timerid) == 0) {
            itval.it_value.tv_sec = sec / 1000;
            itval.it_value.tv_nsec
                  = (long) (sec % 1000) * (1000000L);
            if (mode == 1) { // periodic timer
                  itval.it_interval.tv_sec = itval.it_value.tv_sec;
                  itval.it_interval.tv_nsec = itval.it_value.tv_nsec;
            }
            else { // one shot timer
                  itval.it_interval.tv_sec = 0;
                  itval.it_interval.tv_nsec = 0;
            }
            if (timer_settime(timerid, 0, &itval, &oitval) != 0)   {
                  perror("time_settime error!");
                  return (timer_t) -1;
            } else {
                  printf("timer_create(%d) create!: Success“, timerid);
                  return timerid;
            }
      }
}
```

POSIX Timer Example (4)

```
SignalHandler(int signo, siginfo_t * info, void *context)
{
      if (signo == SIGTIMER) {
            puts("Periodic timer");
       }
       else if (signo == ONESHOTTIMER) {
            puts("One-short timer");
       }
       else if (signo == SIGINT) {
            timer_delete(oneshotTimer);
            timer_delete(timerid);
            perror("Ctrl + C catched!\n");
            exit(1);
        }
}
```

Results

```
# gcc -o posix-timer posix-timer.c –lrt
#./posix_time
timer(20525664)_create!: Success
timer(20526736)_create!: Success
Periodic timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
One-short timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
Periodic timer
^CCtrl + C catched!
```