Socket Programming signal and timer

CSEE, Handong Univ.

Jong-won Lee

ljw@handong.edu

Signals

A signal (software interrupt)

- A notification to a process that an event has occurred.

Signals can be sent

- By one process to another process (or itself)
- By the kernel to a process

Signals

Signal disposition (action associated with a signal)

- We can provide a function that is called whenever a specific signal occurs. This function is called a *signal handler.*
- We can *ignore* a signal by setting its disposition to `SIG_IGN`.
- We can set the *default* disposition for a signal by setting its disposition to `SIG_DFL`.

Signal types

Signal name	Description	Default action
SIGALRM	Time-out occurs (alarm)	terminate
SIGCHLD	Change in status of a child	ignore
SIGINT	Terminal interrupt character (^C)	terminate
SIGKILL	Termination	terminate
SIGPIPE	Write to pipe with no readers	terminate
SIGSTOP	Stop (^S)	Stop process

Signal Handling

Signal() function

- not recommended. Use sigaction() instead of this function.
- Define the action associated with a signal

```
#include <signal.h>

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

- *signum:* the name of a signal
- *func :* the address of a function to be called when the signal occurs, SIG_IGN or SIG_DFL
- *Return* : the previous disposition of the function if OK, SIG_ERR on error

Signal()

Example

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

void timeout(int sig)
{
        if(sig==SIGALRM)
                puts("Time out!");

        alarm(2);
}
void keycontrol(int sig)
{
        if(sig==SIGINT)
                puts("CTRL+C pressed");
}
```

```
int main(int argc, char *argv[])
{
        int i;
        signal(SIGALRM, timeout);
        signal(SIGINT, keycontrol);
        alarm(2);

        for(i = 0; i < 3; i++)
        {
                puts("wait...");
                sleep(100);
        }
        return 0;
}
```

Signal()

Example

```
[ljw@localhost Chapter10]$ ./signal1
wait...
Time out!
wait...
^CCTRL+C pressed
wait...
Time out!
[ljw@localhost Chapter10]$
```

Signal Handling

Sigaction() function

- Define the action associated with a signal

```
#include <signal.h>

int sigaction(int signum, const struct sigaction * act,
              struct sigaction * oldact);
```

- *signum:* the name of a signal
- *act :* the new action
- *oldact* : the previous action

Signal Handling

Struct sigaction

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

- *sa_handler* : the address of a function to be called when the signal occurs
- *sa_sigaction:* If SA_SIGINFO is specified in sa_flags, then sa_sigaction (instead of sa_handler) specifies the signal-handling function

-*sa_mask :* specifies a set of signals that will be blocked when the signal handler is called.

-*sa_flags* : options. Basically 0.

Signal Handling

Handling a signal mask

```
#include <signal.h>
int sigaddset(sigset_t *set, int signo);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigdelset(sigset_t *set, int signo);
int sigismember(sigset_t *set, int signo);
          //return 1 if true
```

```
#include <signal.h>
int sigprocmask(int how, sigset_t *set,
                sigset_t *oldset);
//how: SIG_BLOCK, SIG_UNBLOCK, SIG_SETMASK
```

sigaction()

Example 1

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

void timeout(int sig)
{
        if(sig == SIGALRM)
                puts("Time out!");
        alarm(10);
}

void keycontrol(int sig)
{
        if(sig == SIGINT)
                puts("CTRL+C pressed!");
}
```

```
int main(int argc, char *argv[])
{
        int i;
        struct sigaction act;

        act.sa_handler = timeout;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        sigaction(SIGALRM, &act, 0);

        act.sa_handler= keycontrol;
        sigemptyset(&act.sa_mask);
        act.sa_flags=0;
        sigaction(SIGINT, &act, 0);kk
```

sigaction()

Example 1

```
        for(i = 0; i < 3; i++)
        {
                puts("wait...");
                sleep(100);
        }
        return 0;
}
```

```
[ljw@localhost Chapter10]$ ./sigaction1
wait...
^CCTRL+C pressed!
wait...
Time out!
wait...
Time out!
[ljw@localhost Chapter10]$
```

High Resolution Timer

Functions

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

int timer_create(clockid_t clockid,
    struct sigevent *restrict sevp, timer_t *restrict timerid);
```

- *clock_id*: the clock that the timer uses to measure time
  : CLOCK_REALTIME, CLOCK_PROCESS_CPUTIME_ID, CLOCK_THREAD_CPUTIME_ID …
-*servp:* points to a sigevent structure that specifies how the caller should be notified when the timer expires. when the signal handler is called.
-*timer_id*: timer ID created.

- Return value: 0 on success, -1 on failure

High Resolution Timer

Example of timer creation

```
int createTimer ( timer_t *timerID,
                  int sec, int msec )
{
    struct sigaction        sa;
    struct sigevent         te;
    struct itimerspec       its;

    /* Set up signal handler. */
    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = timer;     // timer handler
    sigemptyset(&sa.sa_mask);

    if (sigaction(SIGRTMIN, &sa, NULL) == -1)
    {
        printf("sigaction error\n");
        return -1;
    }
```

```
 /* Set and enable alarm */
    te.sigev_notify = SIGEV_SIGNAL;
    te.sigev_signo = SIGRTMIN;
    te.sigev_value.sival_ptr = timerID;
    timer_create(CLOCK_REALTIME, &te, timerID);

    its.it_interval.tv_sec = sec;
    its.it_interval.tv_nsec = msec * 1000000;
    its.it_value.tv_sec = sec;
    its.it_value.tv_nsec = msec * 1000000;

    timer_settime(*timerID, 0, &its, NULL);

    return 0;
}
```

pecifies the clock that the new timer uses to measure time.
pecifies the clock that the new timer uses to measure time.

High Resolution Timer

Functions

```
#include <time.h>

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

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

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

High Resolution Timer

Functions

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

- timer_settime() arms or disarms the timer identified by timerid. The new_value argument specifies the new initial value and the new interval for the timer.
- If new_value->it_value specifies a nonzero value, then timer_settime() arms (starts) the timer, setting it to initially expire at the given time. (If the timer was already armed, then the previous settings are overwritten.) If new_value->it_value specifies a zero value, then the timer is disarmed.
- The new_value->it_interval field specifies the period of the timer. If this field is nonzero, then each time that an armed timer expires, the timer is reloaded from the value specified in new_value->it_interval. If new_value->it_interval specifies a zero value, then the timer expires just once, that is one-shot timer.

High Resolution Timer

Functions

```
int timer_gettime(timer_t timerid, struct itimerspec *curr_value);
```

- timer_gettime() returns the time until next expiration, and the interval, for the timer specified by timerid, in the buffer pointed to by curr_value.
- The time remaining until the next timer expiration is returned in curr_value->it_value.
- If the value returned in curr_value->it_value is zero, then the timer is currently disarmed.
- If the value returned in curr_value->it_interval is zero, then this is a "one-shot" timer.

pecifies the clock that the new timer uses to measure time.
pecifies the clock that the new timer uses to measure time.

High Resolution Timer

Functions

```
#include <time.h>

int timer_delete(timer_t timerid);
```

- timer_delete() deletes the timer whose ID is given in timerid.
  If the timer was armed at the time of this call, it is disarmed before being deleted.

- Return value: 0 on success, -1 on failure

High Resolution Timer

Example

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

sigset_t mask;
timer_t timerID;

// timer handler
void timerHandler()
{
    printf("time-out\n");
}
```

```
void keycontrol(int sig)
{
    struct itimerspec       its;

    if (sig == SIGINT)
        printf("CTRL+C pressed!\n");
   //  change to one-shot timer
    its.it_interval.tv_sec = 0;
    its.it_interval.tv_nsec = 0;
    its.it_value.tv_sec = 5;
    its.it_value.tv_nsec = 0;
    timer_settime(timerID, 0, &its, NULL);
}
```

pecifies the clock that the new timer uses to measure time.

High Resolution Timer

Example

```
int createTimer( timer_t *_timerID, I
                 nt sec, int msec )
{
    struct sigevent         te;
    struct itimerspec        its;
    struct sigaction        sa;

    /* Set up signal handler. */
    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = timerHandler;
    sigemptyset(&sa.sa_mask);

    if (sigaction(SIGRTMIN, &sa, NULL) == -1)
    {
        printf("sigaction error\n");
        exit(1);
    }
```

```
    /* Set and enable alarm */
    te.sigev_notify = SIGEV_SIGNAL;
    te.sigev_signo = SIGRTMIN;
    te.sigev_value.sival_ptr = _timerID;
    timer_create(CLOCK_REALTIME, &te, _timerID);

    its.it_interval.tv_sec = sec;
    its.it_interval.tv_nsec = msec * 1000000;
    its.it_value.tv_sec = sec;
    its.it_value.tv_nsec = msec * 1000000;
    timer_settime(*_timerID, 0, &its, NULL);

    return 0;
}
```

High Resolution Timer

Example

```
int main()
{
    struct sigaction act;

    createTimer(&timerID, 3, 0);

    act.sa_handler = keycontrol;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGINT, &act, 0);

    for(i = 0; i < 10; i++)
    {
            puts("wait...");
            sleep(100);
     }
     return 0;
}
```