

# **CS241 Operating Systems**

## **CPU Scheduling (6)**

Klara Nahrstedt

Lecture 15

2/22/2006

---

# Content

- XSI Interval Timers
- Sleep functions
- Real-time Signals
- Timers
  - POSIX TRM Interval Timers
- Summary

# Administrative Notes

- MP2 on scheduling is running
- Quiz 4 on Friday, 2/24/06
- Material presented in this lecture covers material in the book R&R 9.2, 9.3, 9.4 and 9.5

# POSIX:XSI Interval Timers

- Generates a signal after a time interval repeatedly – periodic

```
struct timeval it_value; /* time until next expiration*/  
struct timeval it_interval; /* value to reload into the timer */
```

Timeval struct has fields for seconds and microseconds

```
#include <sys/time.h>
```

```
int getitimer(int which, struct itimerval *value);  
int setitimer(int which, const struct itimerval *restrict value,  
              struct itimerval *restrict ovalue);
```

# ltimers

- `it_interval = 0` then timer won't restart

# Sleep Functions

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

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

sleep interacts with SIGALRM

# Nanosleep

```
#include <time.h>  
int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

Resolution of CLOCK\_REALTIME is of order 10ms

Does not affect SIGALRM

# Print asterix every 2 seconds

```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
/* ARGSUSED */
static void myhandler(int s) {
    char aster = '*';
    int errsav;
    errsav = errno;
    write(STDERR_FILENO, &aster, 1);
    errno = errsav;
}
```

```

static int setupinterrupt(void) { /* set up myhandler for SIGPROF */
    struct sigaction act;
    act.sa_handler = myhandler;
    act.sa_flags = 0;
    return (sigemptyset(&act.sa_mask) || sigaction(SIGPROF, &act,
NULL));
}

/* Create signal handler */
static int setupitimer(void) /* set ITIMER_PROF for 2-second intervals */
    struct itimerval value;
    value.it_interval.tv_sec = 2;
    value.it_interval.tv_usec = 0;
    value.it_value = value.it_interval;
    return (setitimer(ITIMER_PROF, &value, NULL));
}

/* Create timer to generate signals */

```

```
int main(void) {
    if (setupinterrupt()) {
        perror("Failed to set up handler for SIGPROF");
        return 1;
    }
    if (setupitimer() == -1) {
        perror("Failed to set up the ITIMER_PROF interval timer");
        return 1;
    }
    for ( ; ; );           /* execute rest of main program here */
}
```

Set up signal and handler to create \*s. Set up interval timer to create signals

# POSIX:XSI Interval Timer used to time function (R&R: P. 320)

```
#include <stdio.h>
#include <sys/time.h>
#define MILLION 1000000L
void function_to_time(void);
int main(void) {
    long difftime;
    struct itimerval ovalue, value;
    ovalue.it_interval.tv_sec = 0;
    ovalue.it_interval.tv_usec = 0;
    ovalue.it_value.tv_sec = MILLION;           /* a large number */
    ovalue.it_value.tv_usec = 0;
    if (setitimer(ITIMER_VIRTUAL, &ovalue, NULL) == -1) {
        perror("Failed to set virtual timer");
        return 1;
    }
}
```

# POSIX:XSI Interval Timer used to time function (R&R:P.320)

```
function_to_time();          /* timed code goes here */
if (getitimer(ITIMER_VIRTUAL, &value) == -1) {
    perror("Failed to get virtual timer");
    return 1;
}
diftime = MILLION*(ovalue.it_value.tv_sec - value.it_value.tv_sec)
         + ovalue.it_value.tv_usec - value.it_value.tv_usec;
printf("The function_to_time took %ld microseconds or %f seconds.\n",
       difftime, difftime/(double)MILLION);
return 0;
}
```

# Real Time Signals

POSIX:XSI and POSIX:RTS have expanded signal-handling capabilities

1. To include the **queuing of signals**
2. To include **passing information to signal handlers**

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

```
struct sigaction {  
    void (*sa_handler) (int); /* SIG_DFL, SIG_IGN, or pointer to function  
    sigset_t sa_mask; /*additional signals to be blocked */  
                        /* during execution of handler*/  
    int sa_flags; /* special flags and options*/  
    void(*sa_sigaction) (int, siginfo_t *, void *); /* realtime handler*/  
  
/* Handler looks like*/  
void func(int signo, siginfo_t * info, void *context);
```

# SIGINFO\_T structure

```
int si_signo; /* signal number */
int si_code; /* cause of signal */
union sigval si_value; /* signal value*/
```

## Causes of signal (si\_code):

- SI\_USER (kill or raise generated signal),
- SI\_QUEUE (sigqueue caused signal)
- SI\_TIMER (interval timer caused signal)
- others (SI\_ASYNCIO, SI\_MESGQ)

si\_value defined only in POSIX: RTS and if si\_code is SI\_QUEUE, SI\_TIMER, ...

union sigval is defined as

```
int sival_int; /*integer or pointer can be transmitted to the signal handler
                by the generator of the signal */
```

```
void *sival_ptr;
```

# POSIX:RTS signal queuing

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

```
int sigqueue(pid_t pid, int signo, const union sigval value);
```

- ❑ **sigqueue extends kill function that permits signals to be queued**
- ❑ Multiple signals created by kill may not be queued but they are if created by sigqueue
- ❑ Signo should be non-zero. If zero, will check for errors – can use this to see if pid valid, see p322
- ❑ SA\_SIGINFO is in sa\_flags of struct sigaction, then signal queued, otherwise signal sent at least once, but it is not queued.

# Send a queued signal to a process

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    int pid;
    int signo;
    int sval;
    union signal value;
    if (argc != 4) {
        fprintf(stderr, "Usage: %s pid signal value\n",
            argv[0]);
        return 1;
    }
}
```

# Send a queued signal to a process

```
pid = atoi(argv[1]);
    signo = atoi(argv[2]);
    sval = atoi(argv[3]);
    fprintf(stderr, "Sending signal %d with
        value %d to process %d\n",
        signo, sval, pid);
    value.sival_int = sval;
    if (sigqueue(pid, signo, value) == -1) {
        perror("Failed to send the signal");
        return 1;
    }
    return 0;
}
```

# POSIX:TMR Interval Timers

- struct itimerspec
  - struct timespec it\_interval; /\* timer period\*/
  - struct timespec it\_value; /\* timer expiration\*/
- timer\_create creates timers --- they are NOT inherited on fork!!!

# timer\_create

```
#include <signal.h>
#include <time.h>
int timer_create(clockid_t clock_id, struct sigevent *restrict evp,
                timer_t *restrict timerid);
/* evp is signal event generated when timer expires, 0=default*/
struct sigevent {
    int          sigev_notify /*notification type*/
    int          sigev_signo /*signal number*/
    union sigval sigev_value; /*signal value*/
};

Union sigval {
    int          sival_int;
    void         *sival_ptr;
};
```

# timer\_delete

```
#include <time.h>
int timer_delete(timer_t timerid);

int timer_getoverrun(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
    const struct itimerspec *value, struct itimerspec *ovalue);
```

# Display Message every 2 seconds (R&R:P.328)

```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#define BILLION 1000000000L
#define TIMER_MSG "Received Timer Interrupt\n"
/* ARGSUSED */
static void interrupt(int signo, siginfo_t *info, void *context) {
    int errsave;
    errsave = errno;
    write(STDOUT_FILENO, TIMER_MSG, sizeof(TIMER_MSG) - 1);
    errno = errsave;
}
```

Define action to do on signal

```
static int setinterrupt() {
    struct sigaction act;
    act.sa_flags = SA_SIGINFO;
    act.sa_sigaction = interrupt;
    if ((sigemptyset(&act.sa_mask) == -1) ||
        (sigaction(SIGALRM, &act, NULL) == -1))
        return -1;
    return 0;
}
```

Set actions to do for SIGALRM

```
static int setperiodic(double sec) {
    timer_t timerid;
    struct itimerspec value;
    if (timer_create(CLOCK_REALTIME, NULL, &timerid) == -1)
        return -1;
    value.it_interval.tv_sec = (long)sec;
    value.it_interval.tv_nsec = (sec - value.it_interval.tv_sec)*BILLION;
    if (value.it_interval.tv_nsec >= BILLION) {
        value.it_interval.tv_sec++;
        value.it_interval.tv_nsec -= BILLION;
    }
    value.it_value = value.it_interval;
    return timer_settime(timerid, 0, &value, NULL);
}
```

Create an interval timer that creates SIGALM's

```
int main(void) {
    if (setinterrupt() == -1) {
        perror("Failed to setup SIGALRM handler");
        return 1;
    }
    if (setperiodic(2.0) == -1) {
        perror("Failed to setup periodic interrupt");
        return 1;
    }
    for ( ; ; )
        pause();
}
```

Pause causes program to sleep for a real-time amount.... Different from asterix program because of real-time

# Using a POSIX:TMR to time function

```
#include <stdio.h>
#include <time.h>
#define MILLION 1000000L
#define THOUSAND 1000
void function_to_time(void);

int main(void);
    long difftime;
    struct itimerspec nvalue, ovalue;
    timer_t timeid;

    if (timer_create(CLOCK_REALTIME, NULL, &timeid) == -1) {
        perror("Failed to create a timer based on CLOCK_REALTIME");
        return 1;
    }
```

# tmrtimer.c (R&R P329)

```
ovalue.it_interval.tv_sec = 0;
ovalue.it_interval.tv_nsec = 0;
ovalue.it_value.tv_sec = MILLION;
ovalue.it_value.tv_nsec = 0;
(timer_settime(timeid, 0, &ovalue, NULL);
Function_to_time());
timer_gettime(timeid, &nvalue);
difftime = MILLION*(ovalue.it_value.tv_sec -
    nvalue.it_value.tv_sec) + (ovalue.it_value.tv_nsec -
    nvalue.it_value.tv_nsec)/THOUSAND;
return 0;
}
```

# Summary

- Timers
  - POSIX:TMR
  - POSIX: XSI