

Signals and Timers

CS241 Discussion Section

Week 6

2/26/07 – 3/02/07

Outline

- SMP5 preview
- Signals and threads
- `pause()`, `sigwait()`
- POSIX:TMR timers
- `clock_gettime()`
- `nanosleep()`

SMP5 preview

Much like in SMP4, you will be implementing a scheduler

However, this one's a real preemptive scheduler

It uses timers and signals to interrupt running threads

Review: signals

Asynchronous notification to a process indicating some action should be taken

Sending signals to a process:

```
kill -<signal> <pid>  
int kill(pid_t pid, int sig);
```

We can signal individual threads, too:

```
int pthread_kill(thread_t tid, int sig);
```

Process Signal Masks

Setting SIGINT to be blocked

```
if ((sigemptyset(&set) == -1) ||
    (sigaddset(&set, SIGINT) == -1))
    perror("Failed init signal set");
else if
    (sigprocmask(SIG_BLOCK, &set, &oldset) == -1)
    perror("Failed to block SIGINT");
```

SIG_BLOCK adds set to current mask

oldset will store the previous signal mask

Thread Signal Masks

`pthread_sigmask () :`

Takes same parameters as `sigprocmask`

Only affects the signal mask of a single thread

Signal mask is inherited on thread creation

Problem 1 (ds6-p1.c)

Another “hello world” variant

Prints out “Hello” and “World” periodically

Change the code to stop the hello thread when receiving `SIGUSR1` and the world thread when receiving `SIGUSR2` .

Hint 1: you should not have to change the signal handler

Hint 2: what about the main thread?

Signal Handlers

Allow us to change what happens when a signal is received

```
void handler(int signo) { ... }
struct sigaction act;

act.sa_flags = 0;
act.sa_handler = handler;
// additional signals blocked in the handler
sigemptyset(&act.sa_mask);
sigaction(SIGUSR1, &act, NULL);
```

sa_handler vs. sa_sigaction

We can get additional information about the signal

```
void handler(int signo, siginfo_t* info, void*  
    context);  
act.sa_flags = SA_SIGINFO;  
// fill sa_sigaction instead of sa_handler  
act.sa_sigaction = handler;
```

Extra information contains, e.g., the source of the
signal (`info->si_code`):

`SI_USER` – user-created signal (with `abort`, `kill`, etc.)

`SI_TIMER` – a POSIX:RTS timer expired

etc.

Problem 2 (ds6-p2.c)

Windows users will sometimes press Ctrl-C (for copy to clipboard), inadvertently killing a UNIX process.

Let's change the SIGINT handler to kill the process only if the user *really* means it!

I.e., presses Ctrl-C three times within a 5-second "tick"

And let's only count signals sent by the kernel (based on keyboard input)

```
info->si_code == SI_KERNEL
```

pause()

Waits for *any* signal that is not blocked/ignored

But if a signal is generated before pause() is called, pause() will never see it

If we use the sigmask to block the signal until pause() is called, it will be queued until we remove it

However, pause() will just sit there waiting for the signal that is blocked; it will never check the queue

In summary: pause() only returns if called before the signal is generated!

sigwait()

Takes as parameter a sigset corresponding to which signals it should wait for

You should block the signals first

sigwait() will remove a signal from the queue that is in its sigset

Must also pass a pointer to an integer for it to store signal that was removed

```
sigwait(sigset_t *set, int *signo);
```

Problem 3 (ds6-p3.c)

Counting signals

Use `sigwait()` to count how many times a process receives `SIGUSR1` or `SIGUSR2`

Don't forget to block them first!

Timers

We will be using `POSIX:TMR` timers

Send the `SIGALRM` signal to the process

If we set up a signal handler for `SIGALRM`, we have a programmable timer!

!! Signals sent for timers or interrupts need to be unblocked for the thread that will be receiving them !!

Accessing the clock

The POSIX:TMR extension allows us to get and set time from the real-time clock

```
struct timespec {  
    time_t tv_sec; /* seconds */  
    long tv_nsec; /* nanoseconds */ }
```

Timers need two of these: one for how long from now to begin, another for how often to generate the interrupt (timer interval)

```
struct itimerspec {  
    struct timespec it_interval; /* period */  
    struct timespec it_value; }
```

Setting a timer

Create the timer

```
timer_t timerid;  
timer_create(CLOCK_REALTIME, NULL, &timerid);
```

Set up the structs (fire first at 5 s, then every 2.5 s)

```
struct itimerspec value;  
value.it_interval.tv_sec = 2;  
value.it_interval.tv_nsec = 500000000L;  
value.it_value.tv_sec = 5;  
value.it_value.tv_nsec = 0;
```

Start the timer

```
timer_settime(timerid, 0, &value, NULL);
```

Resetting (or disabling) a timer

How do we turn off a timer?

Simple: just “set” it to 0

We can also restart a timer

Just call `timer_settime` with the same parameters as before; the timer will be reset

Or pass in a different time to change its behavior

Using timers effectively

(SMP5 advice)

POSIX:TMR timers by default send SIGALRM

Use struct sigevent to change this if needed

Setup a signal handler to catch that signal

Use infinite `pause` or `sched_yield` loops to put a thread to sleep while it waits for the timer

If a thread terminates while the timer is still active, the signal may be delivered to a different thread

Problem 4 (ds6-p4.c)

Let's see how timers are used in this simple example program

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

```
gcc -o p4 ds6-p4.c -lrt
```

Timing your code

`clock_gettime()` fills in a struct `timespec` with elapsed time in nanoseconds since the epoch (Jan 1, 1970)

Difference between two structs can time a function/action

Useful to keep track of how long threads are waiting or executing

```
struct timespec tend, tstart;
clock_gettime(CLOCK_REALTIME, &tstart);
function_to_time();
clock_gettime(CLOCK_REALTIME, &tend);
double timediff = tend.tv_sec - tstart.tv_sec
    + ((double)tend.tv_nsec - tstart.tv_nsec)/1e9;
```

Problem 5 (ds6-p5.c)

Time how long 1e5, 1e6, etc. iterations of an empty for loop take to execute

Time how long 1e5, 1e6, etc. sched_yields take to execute

Recap

How to generate and block signals

How to modify signal handlers

How to initialize and start a timer

How to time your code