Real Time Clock & timers

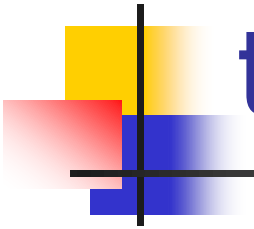
- Timers can be used to send a signal to a process after a specified period of time has elapsed.

- **Timers may be used in one of two modes: one-shot or periodic:**
  - when a **one-shot timer** is set up, a value time is specified. When that time has elapsed, the operating system sends the process a signal and deletes the timer.

  - when a **periodic timer** is set up, both a value and an interval time are specified. When the value time has elapsed, the operating system sends the process a signal and reschedules the timer for interval time in the future. When the interval time has elapsed, the OS sends another signal and again reschedules the timer for interval time in the future. This will continue until the process manually deletes the timer.

- By default a timer will send the SIGALRM signal. If multiple timers are used in one process, however, there is no way to determine which timer sent a particular SIGALRM. Therefore, an alternate signal, like SIGUSR1, may be specified when the timer is created.

POSIX.4 Real Time Clock & timers

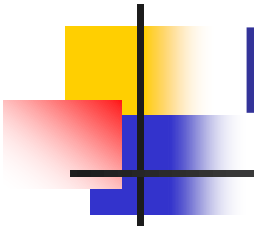


Resolution of timers:

- Timers are maintained by the operating system, and they are only checked periodically. A timer that expires between checks will be signaled (and rescheduled if periodic) at the next check. As a result, a process may not receive signals at the exact time(s) that it requested.

- The period at which the timers are checked, called the **clock resolution**, is operating system and hardware dependent (~1 msec in PC without add-on high resolution timers). The actual value can be determined at runtime by calling clock_getres() on the system-wide real-time clock (CLOCK_REALTIME).

- According to POSIX, there is at least 1 real-time clock (CLOCK_REALTIME)



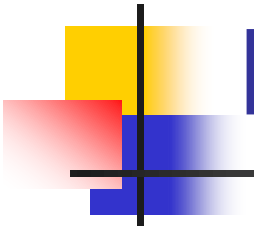
Real Time Clock & timers

Resolution of timers:

- For example, suppose the OS checks the timers every 10 milliseconds and a process schedules a periodic timer with value = 5 milliseconds and interval = 21 milliseconds.

- **Quiz**: Will the timer periodically expire every 21 milliseconds?

- **Answer**: If the period of timer is not an exact multiple of the granularity of underlying clock (see man 7 time), then the interval will be rounded up to the next multiple.



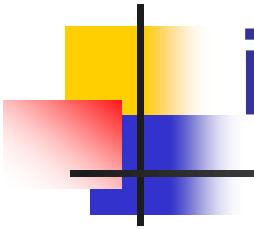
Real Time Clock & timers

High Resolution Timers (require hardware support):

- How to use high resolution timers?

- There are no special requirements, except a recent glibc, to make use of high resolution timers. When the high resolution timers are enabled in the (Linux) kernel, then nanosleep, itimers and posix timers provide the high resolution mode without changes to the source code.

High Resolution Timers in Linux



See man 7 time:

- Before Linux 2.6.21, the accuracy of timer and sleep system calls (see below) was limited by the resolution of software clock (~1-10 msec)

- Since kernel 2.6.21, Linux supports high-resolution timers (HRTs), optionally configurable via CONFIG_HIGH_RES_TIMERS. On a system that supports HRTs, the accuracy of sleep and timer system calls is no longer constrained by the software clock, but instead can be as accurate as the hardware allows (microsecond accuracy is typical of modern hardware). You can determine whether high-resolution timers are supported by checking the resolution returned by a call to clock_getres(2) or looking at the "resolution" entries in /proc/timer_list.

Real Time Clock & timers

Operations:

- Create_timer() is used to create a new timer. As with clock_getres(), the system-wide real-time clock (CLOCK_REALTIME) should be used. The following code shows how to create a timer that sends the default SIGALRM signal.

```
timer_t timer1;

// create a new timer that sends the default SIGALARM signal

if (timer_create (CLOCK_REALTIME, NULL, &timer1) != 0) {
    perror("timer create");
    exit(1);
}
```

NULL specifies that default SIGALARM will be delivered!

Timers: struct sigevent

- If a different signal needs to be sent on timer expiration, then the second argument of timer_create takes a pointer to struct sigevent to specify a different signal to be sent.

```
struct sigevent {
    int             sigev_notify;     /* notification type */
    int             sigev_signo;      /* signal number */
    union sigval    sigev_value;      /* Extra data to be delivered
                                         with a real-time signal!     */
}
```

- sigev_notify
  - SIGEV_NONE - No notification from timer
  - SIGEV_SIGNAL - Send a signal

Real Time Clock & timers

Operations:

- The following code shows how to create a timer that sends the SIGUSR1 signal:

```
timer_t timerid;
struct sigevent se;

// Zero out the data structure and configure it for using SIGUSR1 signal
memset(&se, 0, sizeof(se));
se.sigev_signo = SIGUSR1;
se.sigev_notify = SIGEV_SIGNAL;

// Create a new timer that will send the SIGUSR1 signal
if (timer_create(CLOCK_REALTIME, &se, &timerid) != 0)
{
    perror("Failed to create timer”);
    exit(1);
}
```

Real Time Clock & timers

- The timer_settime() function is used to schedule a timer. The struct itimerspec definition taken from /usr/include/linux/time.h is seen here.
- The **it_value** member sets the time until the timer first expires. If it is set to 0, the timer will never go off. The **it_interval** member sets the period of the timer after it first expires. If it is set to 0, the timer will be one-shot.

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

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

Real Time Clock & timers

- Example of scheduling timer1 (created in a preceding example) to go off in 2.5 seconds, and then every 100 milliseconds thereafter.

```
struct itimerspec timervals;

// The it_value member sets the time until the timer first goes off (2.5 seconds).
// The it_interval member sets the period of the timer after it first goes off (100 ms).
timervals.it_value.tv_sec = 2;                          // 2 seconds
timervals.it_value.tv_nsec = 500000000;// 0.5 seconds (5e8 nanoseconds)
timervals.it_interval.tv_sec = 0;                       // 0 seconds
timervals.it_interval.tv_nsec = 100000000;              // 100 milliseconds (1e8 nanoseconds)

// Schedule the timer
if (timer_settime(timerid, 0, &timervals, NULL) != 0)
{
        perror("Failed to start timer");
        exit(1);
}
```

Pointer to old itimerspec!

It specifies a "relative timer": it does not use absolute time!

POSIX Real Time Clocks

- POSIX defines the structure of time representation. There is at least 1 real time clock. We can check the current time with **clock_gettime** and check the clock resolution with **clock_getres**. See the following example (compile with -lrt):

```
#include  <time.h>
struct timespec  current_time, clock_resolution;
return_code = clock_gettime(CLOCK_REALTIME, &current_time);
//check return_code...
Printf("current time in CLOCK_REALTIME is %ld sec, %ld nsec \n",
    current_time.tv_sec,              // the second portion
    current_time.tv_nsec);            // the fractional portion in nsec representation

return_code  =  clock_getres(CLOCK_REALTIME, &clock_resolution)

//check return_code...
printf("CLOCK_REALTIME's resolution is %ld sec, %ld nsec \n",
    clock_resolution.tv_sec, clock_resolution.tv_nsec);
```