POSIX.1b: Real-Time POSIX.1 Extensions

Timers

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

Basic Unix Timing Facilities

The problem:

- No guarantees for process scheduling
- What about background, periodic tasks?

```
while (1) {
  /* read from the device */
  read(fd, buf, nbytes);
  /* do something ... */
}
```

Basic Unix Timing Facilities

Seconds since the Epoch:

```
#include <time.h>
time_t time(time_t *tp);
```

Basic Unix Timing Facilities

Milliseconds since the Epoch:

- (but granularity is often much less than 1 msec!)

```
#include <sys/time.h>
int gettimeofday(struct timeval *tv);
```

Basic Unix Timing Facilities

One approach to periodicity:

- Poor accuracy
- Subject to drift
- Poor granularity

```
while(1) {
  sleep(sec);
  /* do something ... */
}
```

Basic Unix Timing Facilities

An attempt at improvement:

- Isn't likely to work!

```
while(1) {
  sleep(sec - s);
  /* do something ... */
}
```

Basic Unix Timing Facilities

`alarm (int secs)`

- Sends a `SIGALRM` when the timer expires
- One-shot timer
- Cancel alarm with `alarm(0)`
- Poor granularity

Basic Unix Timing Facilities

Setting a timer:

- No, you usually don't get microsecond resolution!
- Still subject to drift, due to poor granularity

```
struct itimerval interval;
...
interval.it_value.tv_sec = secs;
interval.it_value.tv_usec = usecs;
setitimer(ITIMER_REAL, &interval, NULL);
```

POSIX.1b Timing Facilities

Getting and setting the time of day:

- Note: most hardware doesn't support nanoseconds!

```
#include <time.h>
struct timespec {
  time_t tv_sec; /* seconds */
  long tv_nsec;  /* nanoseconds */
}
int clock_gettime(clockid_t clk_id,
                  struct timespec *tp);
int clock_settime(clockid_t clk_id,
                  const struct timespec *tp);
```

POSIX.1b Timing Facilities

Getting and setting the time of day:

- Note: most hardware doesn't support nanoseconds!

```
#include <time.h>
struct timespec current_time;
clock_gettime(CLOCK_REALTIME, &current_time);
clock_settime(CLOCK_REALTIME, &current_time);
```

POSIX.1b Timing Facilities

Available clocks:

- `CLOCK_REALTIME` — System-wide, real-time clock
- `CLOCK_MONOTONIC` — Time since an unspecified point

Not always supported:

- `CLOCK_PROCESS_CPUTIME_ID` — Per-process timer
- `CLOCK_THREAD_CPUTIME_ID` — Per-thread timer

POSIX.1b Timing Facilities

Determining clock resolution:

- Returns the smallest distinguishable time value

```
int clock_getres(clockid_t clk_id,
                 struct timespec *res);
```

POSIX.1b Timing Facilities

Short, imprecise delays:

- Sleeps for requested number of nanoseconds
- Exits early on receiving a signal
- If interrupted, returns time remaining

```
int nanosleep(const struct timespec *req,
              struct timespec *rem);
```

POSIX.1b Timing Facilities

```
struct timespec sleep_time;
struct timespec leftover_time;
int ret;

ret = nanosleep(&sleep_time, &leftover_time);
if (EINTR == ret) {
  /* we were interrupted */
  /* ... */
}
```

POSIX.1b Timing Facilities

Interval timers:

- POSIX.1b timers are dynamically created:

```
struct itimerspec {
  struct timespec it_value;
  struct timespec it_interval;
};

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

POSIX.1b Timing Facilities

One-shot alarm:

```
struct itimerspec when;

/* two seconds from now */
when.it_value.tv_sec = 2;
when.it_value.tv_nsec = 0;
when.it_interval.tv_sec = 0;
when.it_interval.tv_nsec = 0;

timer_settime(created_timer, 0, &when, NULL);
```

POSIX.1b Timing Facilities

Interval timer:

```
struct itimerspec when;

/* two seconds from now, every second thereafter */
when.it_value.tv_sec = 2;
when.it_value.tv_nsec = 0;
when.it_interval.tv_sec = 1;
when.it_interval.tv_nsec = 0;

timer_settime(created_timer, 0, &when, NULL);
```

POSIX.1b Timing Facilities

Send a different signal:

- Default is `SIGALRM`

```
/* signal handler */
void timer_expired(int signo) {...}

/* bind to SIGUSR1 */
struct sigaction sa;
sa.sa_handler = timer_expired;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
sigaction(SIGUSR1, &sa, NULL);
```

POSIX.1b Timing Facilities

Send a different signal:

- Default is `SIGALRM`

```
struct sigevent timer_event;
timer_t created_timer;
struct itimerspec expiration_time;

/* create a timer that sends SIGUSR1 */
timer_event.sigev_signo = SIGUSR1;
timer_create(CLOCK_REALTIME,
             &timer_event, &created_timer);
```

POSIX.1b Timing Facilities

Absolute time alarms:

- Use `clock_gettime()` or `mktime()` for seconds-since-Epoch

```
when.it_value.tv_sec = secs_since_epoch;
itimer_settime(created_timer,
               TIMER_ABSTIME, &when, NULL);
```

POSIX.1b Timing Facilities

Absolute time alarms:

- Use `clock_gettime()` or `mktime()` for seconds-since-Epoch

```
struct tm absolute;
struct timespec abs_time;
absolute.tm_hour = 23; /* 12:xx pm */
absolute.tm_min = 30;  /* xx:30 */
absolute.tm_mon = 8;   /* September */
...
abs_time.tv_sec = mktime(&absolute);
abs_time.tv_nsec = 0;
timer_settime(created_timer, 0, &abs_time, NULL);
```

POSIX.1b Timing Facilities

The time left on an interrupted timer:

```
struct itimerspec time_remaining;
timer_gettime(created_timer, &time_remaining);
```

POSIX.1b Timing Facilities

Timer overrun signals aren't queued:

- Rather, their occurrence is counted
- Overrun count is reset on each `timer_getoverrun()`

```
int n_overruns;
n_overruns = timer_getoverrun(created_timer);
```

POSIX.1b Timing Facilities

Timers can carry additional information:

- But only to POSIX.1b signal information handlers!

```
void posix1b_timer_handler(int sig,
         siginfo_t *extra, void *cruft)
{
  int n = extra->si_value.sival_int;
  ... or ...
  void *p = extra->si_value.sival_ptr;
  ...
}
```

POSIX.1b Timing Facilities

Timers can carry additional information:

```
struct sigaction sa;
/* hook a POSIX.1b handler to SIGALRM */
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_SIGINFO;
sa.sa_sigaction = posix1b_timer_handler;
if (sigaction(SIGALRM, &sa, NULL))
{
  perror("sigaction");
  exit(1);
}
```

POSIX.1b Timing Facilities

Timers can carry additional information:

```
struct sigevent timer_event;
timer_t created_timer;
time_event.sigev_notify = SIGEV_SIGNAL;
time_event.sgev_signo = SIGALRM;
time_event.sigev_value.sival_int = 0xdeadbeef;
timer_create(CLOCK_REALTIME, &timer_event,
             &created_timer);
```

POSIX.1b Timing Facilities

Timers can carry additional information:

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