Real-Time Programming TI00AA55

Lecture 11 - 14.04.2015

Jarkko.Vuori@metropolia.fi

Real time signals 1

- The conventional signals are not queued and the signal handler cannot take parameters other than the signal number
- Current POSIX.1 incorporates also earlier POSIX.1b (real-time facilities) that defines real-time signals. They have additional features like:
    1. Queueing of signals
    2. Passing information to the signal handler
- Let's recall the definition of structure sigaction:

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

- The last member is a so called real-time signal handler. It is an option to the first member. Real time signal handler is used, if the flag SA_SIGINFO is set in the member sa_flags. Then the prototype of the handler must be:

```
void handler(int signo,
              siginfo_t *siginfo,
              void *context);
```

- Remark. The context parameter is not currently used
- A programmer can use real-time signals for his own purposes
- On the other hand, some new real-time APIs use them

Real time signals 2

- Type `siginfo_t` (second parameter) is defined as follows:

```
typedef struct {
   int si_signo;
   int si_code;
   union sigval si_value
} siginfo_t;
```

- The member `si_code` of structure `siginfo_t` can have the following values:
  - SI_USER
  - SI_QUEUE
  - SI_TIMER
  - SI_ASYNCIO
  - SI_MESGQ

- Value SI_USER means that signal was sent with the conventional functions kill, raise or abort. In this case the member si_value has no meaning
- Value SI_QUEUE means that signal was sent with the new sigqueue function (see page 5)
- Value SI_TIMER means, that signal was sent because of the expiration of timer (so called real time timer, see page 14)
- Value SI_ASYNCIO means that signal was sent because of completion of asynchronous i/o
- Value SI_MESGQ means that message has arrived to the message queue

Real time signals 3

- Real time signal handler can take application specific information in the member `si_value` of the parameter siginfo
  - The type of this member is union that makes it possible to pass int data or void* data
  - Actually the latter provides possibility to pass the pointer to what ever information to the signal handler
  - In this way we can avoid using of global variable when passing information between main process and signal handler
- The type union `sigval` (the type of member `si_value` of the structure `siginfo_t`) is defined as follows:

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

- If signal was sent because of expiration of the timer, the parameter value that is received is determined when the timer was created with the function `timer_create` (see page 9)
- If the signal was sent because of completion of an asynchronous i/o, the parameter value contains the information that was set when i/o was initiated with the `aio_read` or `aio_write` function call
- If the signal was sent because a message has arrived to the message queue, the parameter value contains the information that was set with the function `mq_notify` (see lecture 13)

Real time signals 4

- To send a real time signal from one process to another we need a new function
  - because it is not possible to pass parameter using function kill
  - Another reason is that the queueing system of signals does not work with kill function
- The new signal sending function is `sigqueue`
  - It guarantees that signals are queued
  - It must be used for real time signals
- Remember too that the signal queueing is working only if the signal handler for that signal is set with the function `sigaction` and with the flag SA_SIGINFO in the `sa_flags` member of `struct sigaction`
- Real time signals have their own signal numbers between constants SIGRTMIN – SIGRTMAX
- The prototype of the function `sigqueue` is
  `int sigqueue(pid_t pid, int signo, const union sigval value);`
- Remark 1. If _POSIX_REALTIME_SIGNALS is defined, the implementation supports real time signals
- Remark 2. The programmer can send whatever value (int or void*) to the real time signal handler
- Remark 3. It is dangerous to pass a pointer as a parameter to a signal handler from another process, because process cannot access the address space of another process

Clocks and interval timers

- Clock means a counter, that is incremented at fixed time intervals
- Interval timer means a counter that kernel decrements at fixed intervals based on the clock and sends a signal when counter becomes 0, i.e. when a specified time has elapsed
- Unix systems have many different clocks and timers
- We have learned only most elementary clock that is requested by `time` function and most elementary timer that is requested by `alarm` function
  - This timer sends SIGALRM signal
- The disadvantage of this clock is a poor resolution (1s)
- The disadvantages of this timer is a poor resolution (1s) and the dependency on the SIGALRM
  - It does not restart automatically and you cannot have several timers at the same time
- Next we learn the most “advanced” and latest clocks and timers that POSIX.1-2008 recommends
- New timer API uses real-time signals

Clocks 1

- Conventional functions `time`, `gettimeofday`, `clock` and `times` can be used to measure different times (wall clock time and processor time (see page 15)
- The latest **POSIX.1-2008 recommends** the following method. Clocks have a type `clockid_t`
  - A clock can be system wide or process wide clock
- There are the following clock ids and corresponding clocks in the system:
  - CLOCK_REALTIME ("wall clock time")
  - CLOCK_MONOTONIC (monotonic time)
  - CLOCK_PROCESS_CPUTIME_ID (proc cpu time)
  - CLOCK_THREAD_CPUTIME_ID (thread cpu time)
- CLOCK_REALTIME and CLOCK_MONOTONIC are system wide
  - Others are process or thread wide
- Everybody can read, but only super user can set these clocks
- The data type `struct timespec` represents times
- The time is even more accurate than in the `struct timeval`
  - The data member `tv_nsec` indicates nanoseconds
- The definition of `struct timespec` is:

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

Note that the `tv_nsec` is long in order to ensure that 10^9 fits the variable
  - because there are 10^9 nanoseconds in one second
  - Long is at least 32 bit in all implementations (range ≈ $-2 \cdot 10^9$ .. $2 \cdot 10^9$)

Clocks 2

- Functions that are used to handle these clocks are:
```
int clock_settime(clockid_t clock_id,
const struct timespec *tp);
clock_gettime(clockid_t clock_id,
struct timespec *tp);
clock_getres(clockid_t clock_id,
struct timespec *tp);
```
- Return value is 0, if the function returns successfully
- Return value is -1, if the function execution has failed
- CLOCK_REALTIME and CLOCK_MONOTONIC measure time (in nanoseconds) since the Epoch 1.1.1970 at 00:00:00.
- The value that the function `clock_gettime` has produced in the member `tv_sec` of it's parameter can be used in a similar way than the value produced by the function `time`
- Example
```
struct timespec now;
clock_gettime(CLOCK_REALTIME, &now);
printf("%s", ctime(&now.tv_sec));
```
- CLOCK_PROCESS_CPUTIME_ID and CLOCK_THREAD_CPUTIME_ID measure the cpu time (in nanoseconds) used by the process or thread since the process or thread was started
- Remark 3. To be able to use these functions you need **to link real time library** (gcc needs –lrt)

TMR Interval timers

- The latest POSIX.1-2008 recommends timers represented here (TMR timers)
- They provide a possibility to create several timers in the application
- Timers are based on the clocks (page 7)
- The user can choose, what clock the timer is based on
- The structure `struct itimerspec` is needed when TMR timers are used. The definition is as follows:

```
struct itimerspec {
  struct timespec it_value;//timer expiration
  struct timespec it_interval;//timer period
};
```

- Functions you need to use these timers:

```
int timer_create(clockid_t clock_id,
                 struct sigevent *evp,
                 timer_t *timerid);
int timer_settime(timer_t timerid,
                  int flags,
          const struct itimerspec *value,
          struct itimerspec *value);
int timer_gettime(timer_t timerid,
               struct itimerspec *value);
```

- Because realtime signals are usually used with TMR timers lets recall the definition of the following structures:
  - struct sigaction
  - siginfo_t
  - struct sigevent
  - union sigval

The definitions of these types are on the page 12

Example of TMR interval timer

```
// Prototype of the signal handler
void rt_handler(int signo, siginfo_t *info, void* p ) {
    printf("Timer expired\n");
}

int main(void) {
  timer_t timerid;
  struct sigevent sevent;
  struct sigaction sa;
  struct itimerspec timerstruct;
  int i;

  // Install real time signal handler
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = SA_SIGINFO;
  sa.sa_sigaction = rt_handler;
  sigaction(SIGRTMAX, &sa, NULL);
  // Construct a structure that specifies what signal is generated
  // and how signal handler is called
  sevent.sigev_notify = SIGEV_SIGNAL;
  sevent.sigev_signo = SIGRTMAX;
  sevent.sigev_value.sival_int = 2; // integer parameter
  // Create a timer instance
  timer_create(CLOCK_REALTIME, &sevent, &timerid);
  // Set the times for a timer
  timerstruct.it_value.tv_sec = 2;       // First time after 2 secs
  timerstruct.it_value.tv_nsec = 0;
  timerstruct.it_interval.tv_sec = 0;   // Do not repeat
  timerstruct.it_interval.tv_nsec = 0;  // Do not repeat
  timer_settime(timerid, 0, &timerstruct, NULL);
  // Do what ever
  for (i = 0 ; i < 6 ; i++) {
    sleep(1);
  }
  return 0;
}
```

Absolute TMR interval timer

- The new counter value is loaded “automatically” by the kernel when the counter reaches the zero value
- However there is still an error that is called a drift
- To fully eliminate the drift TMR interval timer can use absolute time mode
- The second parameter (flags) of the function `timer_settime` has two options: 0 and TIMER_ABSTIME
- If the value of TIMER_ABSTIME is used, the structure members are interpreted as absolute time value
- See the example program in the web (interval_timer_abs.c) that demonstrates how absolute times are used to generate a signal in one second intervals without a drift

Structure definitions

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

```
struct sigevent {
  int sigev_notify;        //notification type
  int sigev_signo;         //signal number
  union sigval sigev_value;//signal value
};
```

```
typedef struct {
  int si_signo;
  int si_code;
  union sigval si_value
} siginfo_t;
```

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

More time functions and improvements

- We have learned earlier about different time functions:

```
struct tm *localtime(const time_t *timep);
char *asctime(const struct tm *timep);
char *ctime(const time_t *timep); // time zone
struct tm *gmtime(const time_t *timep);
```

- The problem with these functions is that they are not reentrant
  - Function is called reentrant if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution
- To fix the problem POSIX:TSF extension defines corresponding thread safe functions:

```
struct tm *localtime_r(const time_t *timep, struct tm *result);
char *asctime_r(const struct tm *timep, char *buff);
char *ctime_r(const time_t *timep, char *buff);  // time zone
struct tm *gmtime_r(const time_t *timep, struct tm *result);
```

gettimeofday

- There is also the function `gettimeofday` "between" `time` and `clock_gettime`. It uses `struct timeval`:

```
struct timeval {
  time_t tv_sec;  // seconds since Epoch
  time_t tv_usec; // and microseconds
};
<sys/times.h>
int gettimeofday(struct timeval *tp, void
*tzp);
```

  (returns 0)
- Remark 1. This type is same that was used as a timeout in the function `select` (the last parameter)
  - In that context the time was used to express time interval (not the absolute time)
- Remark 2. The value in the member `tv_sec` after calling `gettimeofday` can be used in similar way than the value produced by the function `time`
- Example

```
struct timeval now;
gettimeofday(&now, NULL);
printf("%s", ctime(&now.tv_sec));
```

- Remark. The new version of POSIX.1-2008 **recommends rather clock_gettime** (see page 7)

Conventional clocks 1

- ANSI C defines a function clock
  ```
  <time.h>
  clock_t clock(void);
  ```
- The type of the reading of this conventional clock is `clock_t`
- This function returns the **cpu time** of the process that is used so far
  - POSIX requires that time unit should be a micro second (CLOCKS_PER_SEC is 1000000)
- UNIX specification defines another function `times`, that can be used to measure different process times: total time (wall clock time), user cpu time and system cpu time
  - The sum of the last two times is the total cpu time
- The prototype of the function is:
  ```
  <times.h>
  clock_t times(struct tms *buff);
  ```
- Return value is the total time, and the structure `struct tms` is defined as follows:
  ```
  struct tms {
    clock_t tms_utime; //user cpu time used so far
    clock_t tms_stime;  //system cpu time used
    clock_t tms_cutime; //user cpu time of children
    clock_t tms_cstime; //system time of children
  }
  ```
- The function returns the value of tick counter since the booting
  - Ticks per second can be determined with the function `sysconf(_SC_CLK_TCK)`

Conventional clocks 2

- Remark. The two functions on the previous page `clock` and `times` both return the time of type `clock_t`
  - However, these times mean different things and even time units are different
  - The function `clock` returns the processor time of the process and the function `times` return the wall clock time
  - The time units are explained on the right
- ISO C
  `clock_t clock(void);`
  CPU-time (microseconds)
  The unit is specified with the constant CLOCKS_PER_SEC (is 1000000)
- UNIX
  `clock_t times(struct tms *buff);`
  Wall clock time (1 / 100 seconds or ticks)
  - The unit is specified with the value returned by the function `sysconf` _SC_CLK_TCK as parameter (is 100)

Delays (sleeping)

- Different sleep functions:

A. `unsigned sleep(unsigned seconds);`
   - Problems:
     1. resolution only one second
     2. uses SIGALRM signal

B. `int nanosleep(const struct timespec *requested, struct timespec *remaining);`
   - This function has no interaction with the SIGALRM
   - Resolution is nanosecond

- Both functions return, if time has elapsed or signal is delivered and signal handler returns

- Example (Delay of 0.5 seconds)

```
struct timespec delay =
{ 0, 500000000 };
nanosleep(&delay, NULL);
```

- Remark 1. If signals are delivered to the process and you want to make sure that the required delay really happens, you need to write a loop using the principle in much the same way we did with the sleep function
- Remark 2. There is also function `usleep`, that takes the delay as microseconds
  - The delay must not be a second or longer
  - This means that parameter needs to be less that 1 000 000
  - For example `usleep`(500000) causes 0.5 second delay
- The use of this function is not recommended

POSIX:XSI Interval timers 1

- POSIX:XSI also defines interval timers
- There are three different kind of interval timers:
  - ITIMER_REAL (based on the "wall" clock)
  - ITIMER_PROF (based on the cpu time)
  - ITIMER_VIRTUAL (based on the system time)
- The time intervals are expressed with the structures itimerval:

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

- Note that the time is defined in the accuracy of microseconds because `struct timeval` is used
- Function `getitimer` can be used to read the current value of interval timer and `setitimer` can be used to start and stop the interval timer

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

POSIX:XSI interval timers 2

- When timer is set using the function `setitimer`, it expires when the time that was set in the member `it_value` of parameter `value` has elapsed
  - Kernel decrements this parameter member according the rule given as the first parameter (which)
  - When time in the member `it_value` becomes zero, kernel sends a signal SIGALRM to the process
- Kernel loads automatically the member `it_value` with the member `it_interval`, so that timer starts from the begin without any interaction from the user program
- The process can query the current value of the counter (member `it_value`) at any time with the function `getitimer`
- Remark. The new version of POSIX.1-2008 **recommends rather TMR interval timer** (see page 9)

Clocks (summary)

- POSIX:TMR
- CLOCK_REALTIME
  - Measures wall clock time
- CLOCK_MONOTONIC
  - Measures wall clock time using monotonic clock
  - (Time changes have no effect)
- CLOCK_PROCESS_CPUTIME_ID
  - Measures total cpu-time of the process
- CLOCK_THREAD_CPUTIME_ID
  - Measures total cpu-time of the thread
- Use structure `struct timespec` (seconds and nanoseconds)
- Functions: `clock_settime`, `clock_gettime` and `clock_getres`

Timers (summary)

- POSIX:TMR
  - It is possible to create several timers
  - You can choose the clock which the timer is based on
  - The structure `struct itimerspec` is used that contains two members each of which is of type `struct timespec` (seconds and nanoseconds)
  - Functions: `timer_create`, `timer_settime` and `timer_gettime`
  - Function `timer_create` produces timer id
  - User can specify the signal (real time signal)
- POSIX:XSI
  - ITIMER_REAL
    - Is based on the wall clock
  - ITIMER_PROF
    - Is based on the cpu-time
  - ITIMER_VIRTUAL
    - Is based on the system time
  - Structure `struct itimerval` is used, that contains two members, each of which is of type `struct timeval` (seconds and microseconds)
  - Functions: `getitimer` and `setitimer`
  - Signal SIGALRM is generated
  - Conventional: Function alarm and signal SIGALRM

Conventional time functions (summary)

- `time`
  - Resolution is second
- `gettimeofday`
  - Resolution is micro second
  - Takes structure `struct timeval` as a parameter
- `clock`
  - Returns cpu-time (unit is microsecond, CLOCKS_PER_SEC)
- times
  - Produces user cpu time and system cpu time in the structure struct tms
  - Returns wall clock time (unit is tick, sysconf(_SC_CLK_TCK)

```
struct tms {
  clock_t tms_utime;
  clock_t tms_stime;
  clock_t tms_cutime;
  clock_t tms_cutime;
};
```