POSIX TIMERS implementation in RTLinux

Authors: J. Vidal, F. Gonzálvez, I. Ripoll.

POSIX Timers

POSIX timers allows a mechanism that can notify a thread when the time as measured by a particular clock has reached or passed a specified value, or when a specified amount of time has passed.

Facilities supported by POSIX timers that are desirable for real-time operating systems:

- Support for additional clocks.
- Allowance for greater time resolution (modern timers are capable of nanosecond resolution; the hardware should support it)
- Ability to use something other than SIGALARM to indicate timer expiration (in particular, a POSIX.4 real-time extended signal would be nice)

Therefore POSIX timers allows greater time resolution, implementation-defined timers, and more flexibility in signal delivery.

POSIX interface to timers

```
/* One-Shot and Repeating Timers */

 int timer_create(clockid_t clockid, struct sigevent *restrict evp,  timer_t *restrict timer_id);

int timer_delete(timer_t *timer_id);

int timer_settime(timer_t timer_id, int flags, const struct itimerspec *new_setting, struct itimerspec *old_setting);

int timer_gettime(timer_t timer_id, struct itimerspec *expires);

int timer_getoverrun(timer_t timer_id);
```

Creating a timer

Here is a simple and portable way of creating a timer:

```
#include <signal.h>

#include <time.h>

#define A_DESCRIPTIVE_NAME 13

int err;

struct sigevent signal_specification;

timer_t created_timer; /* Contains the ID of the created timer */
```

```
/* What signal should be generated when this timer expires ? */
signal_specification.sigev_signo= RTL_SIGUSR1;
signal_specification.sigev_value.sival_int = A_DESCRIPTIVE_NAME


err=timer_create(CLOCK_REALTIME, &signal_specification, &created_timer);
```

The code snipped creates a timer based upon the system clock called CLOCK_REALTIME. CLOCK_REALTIME exists on all POSIX.4-conformant systems, so you can count on it. A machine may define other clocks for you, corresponding perhaps to extra, dedicated hardware resources on your particular target machine. The POSIX.4 conformance statement should indicate what clocks are available on a particular system.

The evp argument, if non-NULL, points to a sigevent structure. This structure, allocated by the application, defines the asynchronous notification to occur as specified in Signal Generation and Delivery when the timer expires. If the evp argument is NULL, the effect is as if the evp argument pointed to a sigevent structure with the sigev_notify member having the value SIGEV_SIGNAL, the sigev_signo having a default signal number, and the sigev_value member having the value of the timer ID. If you want to specify a particular signal to be delivered on timer expirations, use the struct sigevent, as defined in <signal.h>:

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

This structure contains three members. sigev_notify is a flag value that specifies what sort of notification should be used upon timer expirtation--signals, nothing, or something else. Currently, only two values are defined for sigev_notify: SIGEV_SIGNAL means to send the signal described by the remainder of the struct sigevent, and SIVEV_NONE means to send no notification at all upon timer expiration.

The third parameter is where the system stored th ID of the created timer. You'll need this ID in order to use the timer.

Setting a timer.

Once you have a timer ID, you can set that timer, as in the following example:

```
#include <time.h>


struct itimerspec new_setting, old_setting;


 new_setting.it_value.tv_sec=1;
 new_setting.it_value.tv_nsec=0;
 new_setting.it_interval.tv_sec=0;
 new_setting.it_interval.tv_nsec=100*1000;

err=timer_settime(created_timer, 0, &new_setting, &old_setting);
```

This example sets the interval timer to expire in 1 second, and every 100.000 nanoseconds thereafter. The old timer setting is returned in the structure old_setting. With the second parameter, you tell the system to interpret the interval timer setting as an absolute (TIMER_ABSTIME) or as a relative setting, like in the above example.

Two timer types are required for a system to support realtime applications:

- One-shot: A one-shot timer is a timer that is armed with an initial expiration time, either relative to the current time or at an absolute time (based on some timing base, such as time in seconds and nanoseconds since the Epoch). The timer expires once and then is disarmed. With the specified facilities, this is accomplished by setting the it_value member of the value argument to the desired expiration time and the it_interval member to zero.
- Periodic: A periodic timer is a timer that is armed with an initial expiration time, again either relative or absolute, and a repetition interval. When the initial expiration occurs, the timer is reloaded with the repetition interval and continues counting. With the specified facilities, this is accomplished by setting the it_value member of the value argument to the desired initial expiration time and the it_interval member to the desired repetition interval.

For both of these types of timers, the time of the initial timer expiration can be specified in two ways:
1.- Relative (to the current time)
2.- Absolute

Time Value Specification Structures

Many of the timing facility functions accept or return time value specifications. A time value structure timespec specifies a single time value and includes at least the following members:

Member Type	***Member Name***	***Description***
time_t	tv_sec	Seconds.
long	tv_nsec	Nanoseconds.

The tv_nsec member is only valid if greater than or equal to zero, and less than the number of nanoseconds in a second (1000 million). The time interval described by this structure is (tv_sec * 10^9 + tv_nsec) nanoseconds.

A time value structure itimerspec specifies an initial timer value and a repetition interval for use by the per-process timer functions. This structure includes at least the following members:

Member Type	***Member Name***	***Description***
struct timespec	it_interval	Timer period.
struct timespec	it_value	Timer expiration.

If the value described by it_value is non-zero, it indicates the time to or time of the next timer expiration (for relative and absolute timer values, respectively). If the value described by it_value is zero, the timer shall be disarmed.

If the value described by it_interval is non-zero, it specifies an interval which shall be used in reloading the timer when it expires; that is, a periodic timer is specified. If the value described by it_interval is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is specified.

Design guidelines

In RTLinux timer_id is implemented as a pointer to the timer struct. This allows to allocate and freeing memory dinamically (only when it is needed) and an efficient acces to timers structure. What we got is a list of timers ordered by thread owner priority. This allows an efficient implementation of find_preemptor, so it isn't necessary to run all the list each time. The main dissadvantage is that timer_create & timer_delete must be called from init_module & clean_module respectivelly.

Timers expirations and expirations notifications are done at the scheduler. This allows to check all the timers quickly and generate the correspondent signals for the owner thread. When that thread gets the CPU, the signal will be delivered. In ONE-SHOT mode (default mode in RTLinux) timers expirations values are considered for the preemptor seek.

Implementation issues

Right now, following files of the RTLinux version 3.2 has been modified/added.
Modifications are in-crusted with CONFIG_OC_PTIMERS.
In schedulers directory:
    rtl_sched.c, rtl_timer.c.
In include directory:
    rtl_sched.h, rtl_timer.h, include/rtl_time.h, include/posix/time.h

The monitor directory comes with a patched version of the scheduler that informs about tasks activations and executions times. Also provides a program (reader) to get kernel information.

BUGS 22-11-02

timespec_add_ns:

The macro timespec_add-ns available in include/rtl_time.h is implemented as:

```
#define timespec_add_ns(t,n) do { \
        (t)->tv_nsec += n;  \
        timespec_normalize(t); \
} while (0)
```

and timespec_normalize :

```
#define timespec_normalize(t) {\
        if ((t)->tv_nsec >= NSECS_PER_SEC) { \
                (t)->tv_nsec -= NSECS_PER_SEC; \
                (t)->tv_sec++; \
        } else if ((t)->tv_nsec < 0) { \
                (t)->tv_nsec += NSECS_PER_SEC; \
                (t)->tv_sec--; \
        } \
      }
```

What should happen if the result of (t)->tv_nsec += n; is bigger than two seconds. Clearly, this will lead to an invalid time specification having tv_nsec field a value bigger of NSECS_PER_SEC (1000*1000*1000). Also overflow could happen if the result is bigger than 2^31 (2147483648).

The alternative solution is to implement timespec_normalize as:

```
#define TWOSECONDS (NSECS_PER_SEC*2)
#define timespec_add_ns(t,n) do { \
  long long aux=(t)->tv_nsec+(n);\
  \
  if ((aux > TWOSECONDS) || (aux < -TWOSECONDS)) /*check overflow*/ {\
    (t)->tv_nsec +=((n) % NSECS_PER_SEC) ; \
    (t)->tv_sec += ((n) / NSECS_PER_SEC); \
  } else {  (t)->tv_nsec=aux; }\
  \
  timespec_normalize(t); \
}  while (0)
```

The file timespec_add_ns_bug.c placed in the directory examples/bug tests both implementations.

Appendix

Test programs

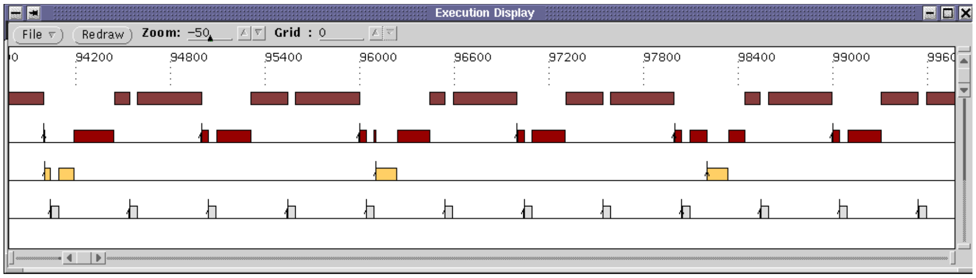
The directory example/timers contains 40 test programs, divided into three categories. In order to run them, please place in the desired directory, type make test and press enter. This will automatically load the test programs and show you the results. Next a brief description of each test directory is given. Further information can be found in the README file placed in each directory.

-**self_buid**: Self build tests. There are five test in this directory:

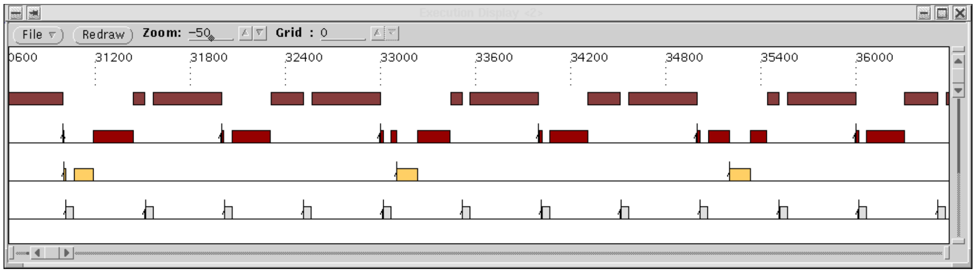
-**accuracity.c** : Measures timers resolution for both absolute and relative specs. The results vary deppending of the available hardware. In a PIII 500 MHZ with APIC, the resolution avilable is 10 microseconds for abosolutes timers and 20 microseconds for relative timers. On a K6 II 3D NOW 300 MHZ without APIC the error is doubled (20 us absolute timers & 39 us relative timers). This results have been taken with CLOCK_REALTIME and system clock on mode ONE-SHOT (default mode).

-**signals_bandwidth.c**: Measures RTLinux POSIX.1 signals bandwidth. This test deppends on the CPU speed. On a PIII 500 MHZ with APIC the bandwidth reaches values of 170 signals/milisecond. On a K6 II 3D NOW 300 MHZ without APIC bandwidth decreases until 66 signals / milisecond. This results have been taken with CLOCK_REALTIME and system clock on mode ONE-SHOT (default mode).

-**test.c**: Test timers real-time constraints. Schedules a set of tasks with timers + signals or with RTLinux API (pthread_make_periodic_np, pthread_wait_np). Also allows to prove timers with system clock on mode periodic and one shot.
This is the chronogram resulting of making the tasks periodic with the RTLinux API:



This is the chronogram resulting of making the tasks periodic with timers + signals:



Zoom: -50

Grid : 0

The monitor directory comes with a patched version of the scheduler that informs about tasks activations and executions times. Also provides a program (reader) to get kernel information.

-**timers.c**: Two threads programing the same timer.
-**simple_test.c**: Basic test for timers API.

-**posixtestsuite**: Open POSIX test suite timers tests, slighly modified to run on RTLinux. This tests are divided into four directories. Each one corresponding with the functionality to test (timer_create,timer_delete, timer_settime, timer_gettime).

-**hrt_test**: Linux high resolution timers project tests, slighly modified to run on RTLinux.

References.

Programming for the Real World -POSIX.4, Bill O. Gallmeister, 1995.

The Single UNIX® Specification, The Open Group, 1997

UNIX Programación Práctica. Kay A. Robbins, Steven Robbins. Prentice Hall.