Creating a Periodic Timer

You can use a periodic timer to provide repeated signals to your process.

The following example illustrates a periodic timer with a delay of a second and a repeating interval of ten milliseconds. It also configures a thread function as the timer expiry notification using `SIGEV_THREAD`.

The following example code performs the following tasks:

1. Creates a notification function (thread function) that must be invoked after timer expiry.
2. Sets the thread priority to **255** using the thread scheduling parameters (`struct sched_param`). This ensures that the thread function has the highest priority when it is invoked as a result of a timer expiry.
3. Creates a timer based on the current system time (`CLOCK_REALTIME`) and a notification function (`struct sigevent sig`) that must be invoked when the timer expires.
4. Defines the input values for timer_settime(). The key input values are the timer value (`in.it_value.tv_sec = 1;`) and the interval (`in.it_interval.tv_nsec = 100000000;`). The periodic timer will expire after a second and then invoke the notification function every one-tenth of a second until it is destroyed.
5. Starts the periodic timer using timer_settime().
6. Uses `sleep(2)` to pause the program execution for two seconds before destroying the timer.

```
#include <time.h>
#include <stdio.h>
#include <signal.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>
static int i = 0;

//Thread function to be invoked when the periodic timer expires
void sighler (union sigval val)
    {
    printf("Handler entered with value :%d for %d times\n", val.sival_int, ++i);
    }
int main()
    {
    int Ret;

    pthread_attr_t attr;
    pthread_attr_init( &attr );

    struct sched_param parm;
    parm.sched_priority = 255;
    pthread_attr_setschedparam(&attr, &parm);

    struct sigevent sig;
    sig.sigev_notify = SIGEV_THREAD;
    sig.sigev_notify_function = sighler;
    sig.sigev_value.sival_int =20;
    sig.sigev_notify_attributes = &attr;

    //create a new timer.
    timer_t timerid;
    Ret = timer_create(CLOCK_REALTIME, &sig, &timerid);
    if (Ret == 0)
```

```
        {
        struct itimerspec in, out;
        in.it_value.tv_sec = 1;
        in.it_value.tv_nsec = 0;
        in.it_interval.tv_sec = 0;
        in.it_interval.tv_nsec = 100000000;
        //issue the periodic timer request here.
        Ret = timer_settime(timerid, 0, &in, &out);
        if(Ret == 0)
            sleep(2);
        else
            printf("timer_settime() failed with %d\n", errno);
        //delete the timer.
        timer_delete(timerid);
        }
    else
    printf("timer_create() failed with %d\n", errno);
    return Ret;
    }
```

The output of the above program is:

```
Handler entered with value :20 for 1 times
Handler entered with value :20 for 2 times
Handler entered with value :20 for 3 times
Handler entered with value :20 for 4 times
Handler entered with value :20 for 5 times
Handler entered with value :20 for 6 times
Handler entered with value :20 for 7 times
Handler entered with value :20 for 8 times
```

Note: Ideally, in the preceding output the handler must have entered **10** times. This is not the case on the Symbian platform as there is some latency due to the timer emulation solution and the underlying Symbian platform clock resolution.

Last updated January 28th, 2010