Timer derivation

Saturday, January 7, 2017 11:02 AM

Synopsis

```
#include <sys/timerfd.h>

int timerfd_create(int clockid, int flags);

int timerfd_settime(int fd, int flags,
                    const struct itimerspec *new_value,
                    struct itimerspec *old_value);

int timerfd_gettime(int fd, struct itimerspec *curr_value);
```

Screen clipping taken: 1/7/2017 11:02 AM

https://linux.die.net/man/2/timerfd_create

http://stackoverflow.com/questions/12463554/timer-library-in-c

Since you are running Linux, I would recommend using the built in POSIX timer API's.

```
int timer_create(clockid_t clockid, struct sigevent *sevp, timer_t *timerid);
```

Here is a link to some documentation showing how to use POSIX timers which provide support for callback functions.

Regarding multiple timers in a process, the documentation says this:

```
   A program may create multiple interval timers using timer_create().

   Timers are not inherited by the child of a fork(2), and are disarmed and
   deleted during an execve(2).

   The kernel preallocates a "queued real-time signal" for each timer created
   using timer_create().  Consequently, the number of timers is limited by the
   RLIMIT_SIGPENDING resource limit (see setrlimit(2)).
```

Note that POSIX timers can be used in a threaded application by setting up notification using SIGEV_THREAD_ID as shown below:

```
The sevp.sigev_notify field can have the following values:

       SIGEV_NONE
              Don't asynchronously notify when the timer expires.  Progress of the
              timer can be monitored using timer_gettime(2).

       SIGEV_SIGNAL
              Upon timer expiration, generate the signal sigev_signo for the process.
              See sigevent(7) for general details.  The si_code field of the
              siginfo_t structure will be set to SI_TIMER.  At any point in time, at
              most one signal is queued to the process for a given timer; see
              timer_getoverrun(2) for more details.

       SIGEV_THREAD
              Upon timer expiration, invoke sigev_notify_function as if it were the
              start function of a new thread.  See sigevent(7) for details.

       SIGEV_THREAD_ID (Linux-specific)
              As for SIGEV_SIGNAL, but the signal is targeted at the thread whose ID
              is given in sigev_notify_thread_id, which must be a thread in the same
              process as the caller.  The sigev_notify_thread_id field specifies a
              kernel thread ID, that is, the value returned by clone(2) or gettid(2).
              This flag is only intended for use by threading libraries.
```