Next Up Previous
Next: Tracer **Up:** MIRACLE How-To **Previous:** CrLayMsg

Timers

If you are learning ns2, timers are often an obstacle. They concern with the ability of calling classes methods in consequence of an event schedule. The difficult raises when trying to find out in which order methods are called. If timers are involved, you can't expect to find nested calls of methods until the one you are interested in. In this section it is explained how to understand this behavior.

A TimeHandler, as name suggests, is an object created to manage time, and is defined in the common/timer-handler(.cc,.h) files of your ns2 distribution. This handler is in charge to schedule the execution of events during the simulation. The time the handler refers to is the simulation time, not depending on the time your machine uses to process the code.

A timer is like a state machine and is charachterized by three states. The first is `TIMER_IDLE` which means that timer can manage an event. The next state is `TIMER_PENDING`, describing a status in which the timer is waiting to manage an event, and so it can't accept new events. This means that the timer has accepted to manage an event to be scheduled in the future and it is busy, waiting for this event. The last state is `TIMER_HANDLING` in which the event is processed, calling the `expire()` function. After processing the event, timer returns in idle state.

In cbr/cbr-module(.cc,.h) of the MIRACLE distribution you can find a simple timer example, exploited to manage CBR packets transmission.

In the header file we define a new TimerHandler object. It is NOT possible to use TimerHandler class as is, because each timer has to handle its events in a particular fashion, which depends on the content of `expire()` function.

```
/**
 * Timer used by CbrModule to schedule packet transmission
 */
class SendTimer : public TimerHandler
{
public:
  SendTimer(CbrModule *m) : TimerHandler() { module = m; }

protected:
  virtual void expire(Event *e);
  CbrModule* module;
};
```

We defined the constructor, the `expire()` function and a reference to the module the timer refers to.

On the other side, when defining CbrModule, we have to add `SendTimer_` as a friend class, in order to make it able to properly manage events. In this case `expire()` function has to call a CbrModule private method to activate packet sending processes.

```
  friend class SendTimer;
```

It is also necessary to define (in CbrModule) the timer itself, to have a reference to the object to whom submit events.

```
  SendTimer sendTmr_;  /*timer which schedules packet transmissions*/
```

In CbrModule.cc file, when you define the module constructor you have also to activate Timer constructor. This one has a reference to **this** to definitively associate `SendTimer` to this `CbrModule`. If you look at the definition of timer constructor, you'll see that it only associates a CbrModule to the object (the one which submits events) to be managed by the timer.

```
CbrModule::CbrModule()
  : sendTmr_(this),
```

At the top of the same file there is the definition of `expire()` function. When timer switchs to the `TIMER_HANDLING` state, it refers to this particular set of actions. When a CbrModule event expires, the function `transmit()` is called. Note that this function is defined in `CbrModule`, so Timer class has to be declaredas a friend of Module one. The `module` attribute refers to `CbrModule`, as explained before (see the constructor).

```
void SendTimer::expire(Event *e)
{
  module->transmit();
}
```

Now we are ready to schedule an event. `TimerHandler` class offers two methods to do this. They work in a similar fashion: both of them schedule an event but, while one is made to be called occasionally (i.e., when the timer is not busy, `TIMER_IDLE`), the second is created to work in an always pending status, in order to consecutively recall `expire()` function. If timer is always in `TIMER_PENDING` status, no other (sporadic) events can be managed by the timer. The former is called `sched()` and the latter is called `resched()`. The `sched()` function works only if timer is `TIMER_IDLE` while `resched()` can potentially cancel a previously scheduled `TIMER_PENDING` event and to schedule its new event. In this way the timer is maintaned always busy for those methods which don't call `resched()` function. If your method instead, has the privileged `resched()` call you can access and modify the events.

In `CbrModule` we want to periodically transmit a packet. To achieve this result we use `resched()` function.

```
void CbrModule::start()
{
  ...
  sendTmr_.resched(period_);
}
```

At the end of `transmit()` method we reschedule again an event in order to create a ``self-updating'' loop. In this case, at the end of expire function, we schedule the timer for next `period_` instants, maintaining the loop.

Next Up Previous

Next: Tracer **Up:** MIRACLE How-To **Previous:** CrLayMsg

nsmiracle-users mailing list