Clase Auxiliar 7

Prof: L. Mateu
Aux: Javier Bustos

2 de octubre de 2006

En la clase de hoy veremos las impelmentaciones reales de Mensajes y Monitores en nSystem.

1. Mensajes

```
#include "nSysimp.h"
#include "nSystem.h"

/*********************************************************
 * Epilogo
 *********************************************************/

static int pending_sends=0;
static int pending_receives=0;

void MsgEnd()
{
  if ( pending_sends!=0 || pending_receives!=0)
  {
    nFprintf(2, "\nNro. de tareas bloqueadas en un nSend: %d\n",
                        pending_sends);
    nFprintf(2, "Nro. de tareas bloqueadas en un nReceive: %d\n",
                        pending_receives);
} }

/*********************************************************
 * nSend, nReceive y nReply
 *********************************************************/

int nSend(nTask task, void *msg)
{
  int rc;

  START_CRITICAL();
  pending_sends++;
  { nTask this_task= current_task;

    if (task->status==WAIT_SEND || task->status==WAIT_SEND_TIMEOUT)
    {
      if (task->status==WAIT_SEND_TIMEOUT)
        CancelTask(task);
      task->status= READY;
      PushTask(ready_queue, task); /* En primer lugar en la cola */
    }
    else if (task->status==ZOMBIE)
      nFatalError("nSend", "El receptor es un ``zombie''\n");

    /* En nReply se coloca ``this_task'' en la cola de tareas ready */
    PutTask(task->send_queue, this_task);
```

```
    this_task->send.msg= msg;
    this_task->status= WAIT_REPLY;
    ResumeNextReadyTask();

    rc= this_task->send.rc;
  }
  pending_sends--;
  END_CRITICAL();

  return rc;
}

void *nReceive(nTask *ptask, int timeout)
{
  void *msg;
  nTask send_task;

  START_CRITICAL();
  pending_receives++;
  { nTask this_task= current_task;

    if (EmptyQueue(this_task->send_queue) && timeout!=0)
    {
      if (timeout>0)
      {
        this_task->status= WAIT_SEND_TIMEOUT;
        ProgramTask(timeout);
        /* La tarea se despertara automaticamente despues de timeout */
      }
      else this_task->status= WAIT_SEND; /* La tarea espera indefinidamente */

      ResumeNextReadyTask(); /* Se suspende indefinidamente hasta un nSend */
    }

    send_task= GetTask(this_task->send_queue);
    if (ptask!=NULL) *ptask= send_task;
    msg= send_task==NULL ? NULL : send_task->send.msg;
  }
  pending_receives--;
  END_CRITICAL();

  return msg;
}

void nReply(nTask task, int rc)
{
  START_CRITICAL();

    if (task->status!=WAIT_REPLY)
      nFatalError("nReply","Esta tarea no espera un ``nReply''\n");

    PushTask(ready_queue, current_task);

    task->send.rc= rc;
    task->status= READY;
    PushTask(ready_queue, task);

    ResumeNextReadyTask();

  END_CRITICAL();
}
```

2. Monitores

```
#include "nSysimp.h"
#include "fifoqueues.h"

typedef struct
{
  nTask owner;
  Queue mqueue;
  FifoQueue wqueue;
}
  *nMonitor;

typedef struct
{
  nMonitor mon;
  FifoQueue wqueue;
}
  *nCondition;

#define NOVOID_NMONITOR

#include <nSystem.h>
#include <stdio.h>

static void ReadyFirstTask(Queue queue);

nMonitor nMakeMonitor()
{
  nMonitor mon= (nMonitor)nMalloc(sizeof(*mon));
  mon->owner= NULL;
  mon->mqueue= MakeQueue();
  mon->wqueue= MakeFifoQueue();
  return mon;
}

void nDestroyMonitor(nMonitor mon)
{
  DestroyQueue(mon->mqueue);
  DestroyFifoQueue(mon->wqueue);
  nFree(mon);
}

void nEnter(nMonitor mon)
{
  START_CRITICAL();

  if (mon->owner!=NULL)
  {
    if (mon->owner==current_task)
      nFatalError("nEnter", "Trying to own the same monitor twice\n");
    current_task->status= WAIT_MON;
    PutTask(mon->mqueue, current_task);
    ResumeNextReadyTask();
  }

  mon->owner= current_task;

  END_CRITICAL();
}

void nExit(nMonitor mon)
```

```
{
  START_CRITICAL();

  if (mon->owner!=current_task)
    nFatalError("nExit", "This thread does not own this monitor\n");
  mon->owner= NULL;

  PushTask(ready_queue, current_task);
  ReadyFirstTask(mon->mqueue);
  ResumeNextReadyTask();

  END_CRITICAL();
}

void nWait(nMonitor mon)
{
  START_CRITICAL();

  if (mon->owner!=current_task)
    nFatalError("nWait", "This thread does not own this monitor\n");
  mon->owner= NULL;
  current_task->status= WAIT_COND;
  PutObj(mon->wqueue, current_task);
  ReadyFirstTask(mon->mqueue);
  ResumeNextReadyTask();

  mon->owner= current_task;

  END_CRITICAL();
}

void nNotifyAll(nMonitor mon)
{
  START_CRITICAL();

  if (mon->owner!=current_task)
    nFatalError("nNotifyAll", "This thread does not own this monitor\n");

  while (!EmptyFifoQueue(mon->wqueue))
  {
    nTask task= (nTask)GetObj(mon->wqueue);
    task->status= WAIT_MON;
    PushTask(mon->mqueue, task);
  }

  END_CRITICAL();
}

nCondition nMakeCondition(nMonitor mon)
{
  nCondition cond= (nCondition)nMalloc(sizeof(*cond));
  cond->mon= mon;
  cond->wqueue= MakeFifoQueue();
  return cond;
}

void nDestroyCondition(nCondition cond)
{
  DestroyFifoQueue(cond->wqueue);
  nFree(cond);
}
```

```
void nWaitCondition(nCondition cond)
{
  START_CRITICAL();

  if (cond->mon->owner!=current_task)
    nFatalError("nNotifyAll", "This thread does not own this monitor\n");

  cond->mon->owner= NULL;
  current_task->status= WAIT_COND;
  PutObj(cond->wqueue, current_task);
  ReadyFirstTask(cond->mon->mqueue);
  ResumeNextReadyTask();

  cond->mon->owner= current_task;

  END_CRITICAL();
}

void nSignalCondition(nCondition cond)
{
  nTask task;
  START_CRITICAL();

  if (cond->mon->owner!=current_task)
    nFatalError("nSignalCondition", "This thread does not own this monitor\n");

  task= (nTask)GetObj(cond->wqueue);
  if (task!=NULL)
  {
    task->status= READY;
    PushTask(ready_queue, task);
  }

  END_CRITICAL();
}

static void ReadyFirstTask(Queue queue)
{
  nTask task= GetTask(queue);
  if (task!=NULL)
  {
    task->status= READY;
    PushTask(ready_queue, task);
} }
```