

Clase Auxiliar VII

Prof: L. Mateu

Aux: M. Leyton

28 de septiembre de 2004

1. Mensajería nSystem (sin timeout)

Implemente la mensajería de nSystem sin timeout.

- `int nSend(nTask task, void *msg)`
- `void *nReceive(nTask *ptask, int timeout)`
- `void nReply(nTask task, int rc)`

Observe que nReceive aun puede recibir el parametro timeout. Si `timeout == 0` nReceive no se debe quedar bloqueado esperando un mensaje.

2. Scheduling con dos prioridades

Se quiere implementar un *Scheduling* con dos prioridades: HIGH y LOW. Para ello se quiere agregar una primitiva que se llame `nSetPri(int priority)`, que cambia la prioridad a la tarea que lo invoca.

Modifique el descriptor de tarea, y los programas que maneja la cola de tareas, de modo que deje de ser una Queue y sea una PriQueue.

- `Queue MakePriQueue(); /* El constructor */`
- `void PutPriTask(Queue queue, nTask task); /* Agrega una tarea al final */`
- `void PushPriTask(Queue queue, nTask task); /* Agrega una tarea al principio */`
- `nTask GetPriTask(Queue queue); /* Extrae una tarea */`
- `int EmptyPriQueue(Queue queue); /* Verdadero si la cola esta vacia */`
- `int PriQueueLength(Queue queue); /* Entrega el largo de la cola */`
- `int QueryPriTask(Queue queue, nTask task); /* Verdadero si task esta en la cola */`
- `void DeletePriTaskQueue(Queue queue, nTask task); /*Borra la tarea de queue*/`

3. Solucion Mensajería

```
#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 (sin timeout)
 *****/

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

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

        /* Si la tarea destino ya esta esperando
         * por el mensaje */
        if (task->status==WAIT_SEND )
        {
            task->status= READY;
            PushTask(ready_queue, task); /* En primer lugar en la cola */
        }

        /* 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(); /* Cedemos la ejecucion a la siguiente
                                tarea READY, mientras espero el nReply */

        /*Al retornar del ResumeNextReadyTask() ya me hicieron un nReply */
        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)
        {
            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();
}

```

4. Solución Scheduling

```

typedef struct Task
{
    ...

    int pri; /* HIGH, LOW */
} *nTask;

void nSetPri(int pri){
    START_CRITICAL();
    current_task->pri=pri;
    END_CRITICAL();
}

```

```

/*****
 * Colas para Scheduling de CPU con Prioridades: PriQueue
 *****/
#define TYPE_PRIQUEUE 3

typedef struct PriQueue{
    int type;
    PriQueue high;
    PriQueue low;
}

PriQueue MakePriQueue()
{
    PriQueue queue= (PriQueue) nMalloc(sizeof(*queue));
    queue->type= TYPE_PRIQUEUE;
    queue->high=MakeQueue();
    queue->low=MakeQueue();
    return queue;
}

void PutPriTask(PriQueue queue, nTask task)
{
    if(task->pri == LOW)
        PutTask(queue->low, task);
    else
        PutTask(queue->high, task);
}

void PushPriTask(PriQueue queue, nTask task)
{
    if(task->pri == LOW)
        PushTask(queue->low, task);
    else
        PushTask(queue->high, task);
}

nTask GetPriTask(PriQueue queue)
{
    if(!EmptyQueue(queue->high))
        return GetTask(queue->high);
    else
        return GetTask(queue->low);
}

int QueryPriTask(PriQueue queue, nTask query_task)
{
    return QueryTask(queue->high, query_task) ||
        QueryTask(queue->low, query_task);
}

void DeletePriTaskPriQueue(PriQueue queue, nTask task)
{
    if(task->pri == HIGH)
        DeleteTaskQueue(queue->high, task);
    else
        DeleteTaskQueue(queue->low, task);
}

int EmptyPriQueue(PriQueue queue)

```

```

{
    return EmptyQueue(queue->high) &&
           EmptyQueue(queue->low);
}

int PriQueueLength(PriQueue queue)
{
    return QueueLength(queue->high) +
           QueueLength(queue->low);
}

void DestroyPriQueue(PriQueue queue)
{
    DestroyQueue(queue->high);
    DestroyQueue(queue->low);
    nFree(queue);
}

```