Clase Auxiliar 3

Prof: L. Mateu
Aux: Javier Bustos

21 de agosto de 2006

1. Pool de Impresoras con Semáforos

Utilizando mensajería de nSystem, se le pide a Ud. diseñar un mecanismo para administrar un pool de impresoras concurrentes de tamaño `POOLSIZE`. Un thread puede pedir una impresora con `int pedirImpresora()`. Una vez que termine puede liberar la impresora para que otros la utilicen mediante: `void entregarImpresora(int id)`; Suponga que:

- Existen las siguientes variables globales:

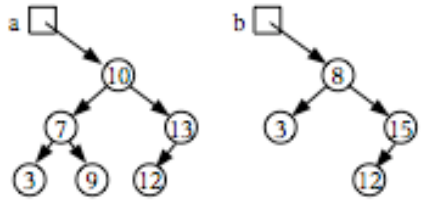
```
nTask pooltask;
int impresoras[POOLSIZE], prim_libre, prim_ocupado;
```

- El método `void initPool();` es llamado al inicio del programa para inicializar las variables globales.
- Ud. cuenta con la implementación de un FifoQueue.

2. P2 C1 2004

Se tienen dos árboles de búsqueda binaria referenciados por las variables globales a y b. Se desea desplegar en pantalla en orden ascendente el contenido de ambos árboles.

Por ejemplo para los 2 árboles de la izquierda, la salida de su programa debe ser:




```
3 3 7 8 9 10 12 12 13 15
```

La estructura de cada nodo está dada por:

```
typedef struct Nodo
{
  struct Nodo *izq, *der;
  int valor;
} Nodo;
```

Restricciones: la solución debe tomar tiempo proporcional al número de nodos de ambos árboles y ocupar a lo más un espacio en memoria proporcional a la máxima altura de ambos árboles.

Resuelva este problema en nSystem utilizando 2 tareas adicionales. Cada tarea recorre recursivamente uno de los árboles, imprimiendo ordenadamente los enteros almacenados a la tarea principal por medio de monitores. No olvide terminar adecuadamente ambas tareas con `nWaitTask`.

3. Soluciones

3.1. P1 : Pool de impresoras

```
#include "nSystem.h"
#define POOLSIZE 30

/****** Variables globales ******/
int impresoras[POOLSIZE];
nSem pool_sem,mutex;

/***** Funciones Cliente *****/
int pedirImpresora() {
 nWaitSem(pool_sem);
 return poolDar();
}

void entregarImpresora(int id){
 poolRecibir(id);
 nSignalSem(pool_sem);
}

/***** Funciones Basicas Pool ******/
int poolDar(){
    int i;
    int retval = -1;

    nWaitSem(mutex);
    for (i=0; i<POOLSIZE; i++)
{
if (!impresoras[i])
{
impresoras[i] = 1;
retval = i;
break;
};
     }
    nSignalSem(mutex)
    return retval;
}

void poolRecibir(int id){

    nWaitSem(mutex);  // Es necesario en escritura?
    impresoras[id] = 0;
    nSignalSem(mutex);
}

/****** Inicializador de variables globales ********/
void initPool(){
    for(int i=0; i<POOLSIZE ;i++)
        impresoras[i]=0;

    pool_sem = nMakeSem(POOLSIZE);
    mutex = nMakeSem(1);
}
```

3.2. P2: Control 1 2004

```
/*

Usando la siguiente API para monitores en
nSystem.

void nEnter(nMon m);
void nExit(nMon m);
void nWait(nMon m);
void nNotifyAll(nMon m);

En Java:                  En nSystem
synchronized(m) {         nEnter(m);
  while (... cond ...)    while (... cond ...)
    m.wait();               nWait(m);
  ...                     ...
  m.notifyAll();          nNotifyAll(m);
}                         nExit(m);

*/

/* Esta solucion tiene un problema con respecto a que arbol empieza imprimiendo */
int lastInformed = MAXINT;
nMon m;
```

```
void InOrder(Node n)
{
if (n == null) return;
if (n -> izq != null) InOrder(n->izq); // vamos a la izq

/* He aqui toda la logica del asunto */
nEnter(m);
while (n -> valor > lastInformed) nWait(m);

/* Estoy dentro del area de exclusion mutua */
nPrintf (''%d '', n -> valor);
lastInformed = n->valor;
nNotifyAll(m);

/* Ya hicimos lo que debiamos, ahora seguimos */
if (n -> der != null) InOrder(n->der); // vamos a la der
}

/* Solucion 2:
 * Mejor le informamos al ente central y que el decida:
 * /

int lastA = MAXINT;
int lastB = MAXINT;

/* Con las siguientes flags informamos si pueden seguir o no */
int flagA = 1;
int flagB = 1;

/* con las siguientes si terminaron o no */
int finishA = 0;
int finishB = 0;

nMon m;

void ImprimeA()
{
inOrderA(a);
nEnter(m);
flagA=0;
finishA = 1;
nNotifyAll(m);
}

void ImprimeB()
{
InOrderB(b);
nEnter(m);
flagB = 0;
finishB = 1;
nNotifyAll(m);
}

void InOrderA(Node n)
{
if (n == null) return;
if (n -> izq != null) InOrderA(n->izq); // vamos a la izq

/* He aqui toda la logica del asunto */
nEnter(m);
while (flagA) nWait(m);

/* Estoy dentro del area de exclusion mutua */
lastA = n->valor;
flagA = 0;
nNotifyAll(m);

/* Ya hicimos lo que debiamos, ahora seguimos */
if (n -> der != null) InOrderA(n->der); // vamos a la der
}

void InOrderB(Node n)
{
if (n == null) return;
if (n -> izq != null) InOrderB(n->izq); // vamos a la izq

nEnter(m);
while (flagB) nWait(m);
lastB = n->valor;
flagB = 0;
nNotifyAll(m);

if (n -> der != null) InOrderB(n->der); // vamos a la der
}
```

```
void nMain()
{
/*
Inicializaciones varias del monitor
*/

nTask t1,t2;

t1 = nEmitTask(ImprimeA);
t2 = nEmitTask(ImprimeB);

while (!finishA && !finishB)
    {
    nEnter(m);
    while (!flagA && !flagB && !finishA && !finishB)) nWait(m);

    /* Esperamos que ambos hayan escrito y ninguno terminado */
    if (!finishA && !finishB))
        {
        if (lastA < lastB) {nPrintf(''%d '',lastA);flagA=1};
        else {nPrintf(''%d '',lastB);flagB=1};
        }
    nNotifyAll(m);
    }

/* Alguien termino! */

while (!finishA || !finishB)
    {
    nEnter(m);
    while (!flagA && !flagB) nWait(m);
    if (finishA && !finishB)  {nPrintf(''%d '',lastB);flagB=1};
    else (!finishA && finishB) {nPrintf(''%d '',lastA);flagA=1};
    nNotyfyAll(m);
    };

nWaitTask(t1);
nWaitTask(t2);
}
```