

Clase Auxiliar III

Prof: L. Mateu

Aux: M. Leyton

20 de agosto de 2004

1. Pool de Impresoras con Mensajería

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. Búsqueda en un Árbol Binario

Suponiendo la siguiente estructura de datos,

```
typedef struct Node{  
    int inf;  
    struct Node *left;  
    struct Node *right;  
} Node;
```

y utilizando nSystem implemente:

1. `int buscarSeq(Node *node, int inf)`; Creando tantas tareas como nodos para recorrer el árbol en paralelo.
2. `int buscarSeq(Node *node, int inf, int level)`; Creando tareas mientras la profundidad del árbol no supere algún nivel, y luego recorriendo el árbol recursivamente.
3. `int buscarSeq(Node *node, int inf, int level, int *pFOUND)`; Como el anterior, pero con una optimización: La ejecución de todas las tareas termina cuando alguna encuentra el elemento buscado.

Observación: Un árbol binario \neq búsqueda binaria

3. Solución Pool de Mensajes

```
#include "nSystem.h"
#define POOLSIZE 30
#define PEDIR -1

/***** Variables globales *****/
nTask pooltask;
int impresoras[POOLSIZE], prim_libre, prim_ocupado;

/***** Funciones Cliente *****/
int pedirImpresora() { return nSend(pooltask, PEDIR ); }

void entregarImpresora(int id){ nSend(pooltask, ENTREGAR); }

/***** Funciones Basicas Pool *****/
int poolDar(){
    if(impresoras[prim_libre]!=-1){
        int retval=impresoras[prim_libre];
        impresoras[prim_libre]=-1;
        prim_libre= (prim_libre + 1) % POOLSIZE;

        return retval;
    }
    else return -1;
}

void poolRecibir(int id){
    //Verificacion de integridad
    if (impresoras[prim_ocupado] == -1) {
        impresoras[prim_ocupado] = id;
        prim_ocupado=(prim_ocupado + 1)%POOLSIZE;
        return 0;
    }
    return -1;
}

/***** Servidor con manejo de cocurrencia *****/
void poolMain(){
    nTask emisor;
    FifoQueue *fq = newFifoQueue();

    while(TRUE){
        int idimp= *(int *) nReceive(&emisor, -1);

        if(idimp==PEDIR){ //PEDIR

            int retval=poolDar();
            if(retval!=-1)
                addFifoQueue(fq, emisor);
            else nReply(emisor, retval);
        }
        else { // ENTREGAR

            int rc=poolRecibir(idimp);
            nReply(emisor, rc);

            //vemos si hay tareas esperando
            if(fifoQueueSize(fq)>0){
                int retval=poolDar(); //falta verificar retval ==0

                //desbloqueamos al la tarea bloqueada
                nReply(getFirst(fq), 0);
            }
        }
    }
}

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

    prim_libre=0; prim_ocupado=0;
    pooltask= nEmitTask(poolMain);
}
```

4. Solución Búsqueda en Árbol Binario

```
/* Creo tantas tareas como nodos para recorrer el arbol en paralelo */
int BuscarSeq(Node *node, int inf) {
    if (node==NULL) return FALSE;
    else if (inf == node->inf) return TRUE;
    else {
        nTask task1 = nEmitTask(BuscarSeq, node->left, inf);
        nTask task2 = nEmitTask(BuscarSeq, node->right, inf);

        //Que problema presenta la siguiente linea
        return nWaitTask(task1) || nWaitTask(task2);
    }
}

/* Creo tantas tareas como procesadores. Una vez que complete mi nivel
 * de procesadores, recorro secuencialmente */
int BuscarSeq(Node *node, int inf, int level) {
    if (node==NULL) return FALSE;
    else if (inf == node->inf) return TRUE;
    else {
        if (level <= 0) {
            return BuscarSeq(node->left, inf, level) ||
                BuscarSeq(node->right, inf, level);
        }
        else {
            nTask task1 = nEmitTask(BuscarSeq, node->left, inf, level-1);
            nTask task2 = nEmitTask(BuscarSeq, node->right, inf, level-1);
            int r1 = nWaitTask(task1);
            int r2 = nWaitTask(task2);
            return r1 || r2;
        }
    }
}

/* Cuando encuentre el elemento las demas tareas terminan su ejecucion. */
int BuscarSeq(Node *node, int inf, int level, int *pFOUND) {
    if (node==NULL) return FALSE;
    else if (*pFOUND) return TRUE;
    else if (inf == node->inf) {
        *pFOUND = TRUE;
        return TRUE;
    }
    else {
        if (level <= 0) {
            return BuscarSeq(node->left, inf, level, pFOUND) ||
                BuscarSeq(node->right, inf, level, pFOUND);
        }
        else {
            nTask task1 = nEmitTask(BuscarSeq, node->left, inf, level-1, pFOUND);
            nTask task2 = nEmitTask(BuscarSeq, node->right, inf, level-1, pFOUND);
            int r1 = nWaitTask(task1);
            int r2 = nWaitTask(task2);
            return r1 || r2;
        }
    }
}
```