

Clase Auxiliar IV

Prof: L. Mateu

Aux: M. Leyton

27 de agosto de 2004

1. Inserción en un Árbol Binario desordenado

Suponiendo la siguiente estructura de datos,

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

y utilizando nSystem implemente:

1. void insert(Node *node, int inf, int *pINSERTED, nSem sem); Creando tantas tareas como nodos para insertar paralelamente en el árbol.
2. Además dibuje un diagrama de threads para indicar una posible condición “race”.

2. Ejercicio

Varios threads pueden esperar hasta que ocurra un cierto evento invocando el procedimiento esperar(). Un único thread notifica la ocurrencia del evento invocando el procedimiento notificar(). Si un thread invoca esperar() y el evento ya ocurrió, el thread continúa de inmediato (sin esperar). Un programador escribió la siguiente implementación basada en semáforos.

```
int ocurrio=FALSE;
int contEsperan=0;
nSem sem; /* =nMakeSem(0); en el nMain */

void esperar(){
    if(!ocurrio){
        contEsperan++;
        nWaitSem(sem);
    }
}

void notificar(){
    ocurrio=TRUE;
    while(contEsperan>0){
        contEsperan--;
        nSignalSem(sem);
    }
}
```

Utilizando un diagrama de threads, muestre que esta solución es incorrecta.

Escriba la solución correcta para este problema. Emplee sólo semáforos en su solución.

3. Ports con Semáforos

Utilizando semáforos de nSystem, se desea implementar un sistema de Ports. Los Ports funcionan de manera similar a los mensajes:

- Port makePort(); Crea una nueva estructura de datos para manejo de mensajes al estilo Port.
- void portSend(Port *p, void *msg); Envía un mensaje al Port. Esta función puede ser llamada varias veces antes que portReceive. La llamada a esta función es bloqueante.
- void* portReceive(Port *p); Retorna la tarea de mayor antigüedad enviada al Port. Si no existen mensajes, esta función se queda bloqueada hasta que un mensaje sea enviado. Al llamar a esta función se debe desbloquear la tarea que envía el mensaje con portSend.

4. Solución Inserción Paralela

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

//Protegiendo la insercion
//Contiene Problema de Race
void insert(Nodo* n, int inf,
           int *pINSERTED, nSem sem){

    if(*pINSERTED) return; //RACE CONDITION

    nWaitSem(sem);
    if(n->left==NULL || n->right==NULL){
        *pINSERTED=TRUE;
        if(n->left==NULL)
            n->left= makeNode(inf);
        else
            n->right= makeNode(inf);

        nSignalSem(sem);
        return
    }

    //Esto puede ser mejorado al igual que la busqueda,
    //con level y recursividad
    nSignaSem(sem);
    return nEmitTask(insert,n->left,inf,pINSERTED,sem) ||
           nEmitTask(insert,n->right,inf,pINSERTED, sem);
}

//Protegiendo la insercion
void insert(Nodo* n, int inf, int *pINSERTED, nSem sem){

    nWaitSem(sem);
    if(*pINSERTED) {
        nSignalSem(sem);
        return;
    }

    if(n->left==NULL || n->right==NULL){
        *pINSERTED=TRUE;
        if(n->left==NULL)
            n->left= makeNode(inf);
        else
            n->right= makeNode(inf);

        nSignalSem(sem);
        return
    }

    //Esto puede ser mejorado al igual que la busqueda,
    //con level y recursividad
    nSignaSem(sem);
    return nEmitTask(insert,n->left,inf,pINSERTED,sem) ||
           nEmitTask(insert,n->right,inf,pINSERTED, sem);
}
```

```
nSem leídoMsg; //Bloquea emisor hasta un receive
nSem vacíoMsg; //Bloquea emisor si ya hay un mensaje

void *msg;
} Port;

Port* makePort(){
    Port *new_port=
        (Port*) nMalloc(sizeof(*new_port));

    new_port->hayMsg=nMakeSem(0);
    new_port->leídoMsg=nMakeSem(0);
    new_port->vacíoMsg=nMakeSem(1);
}

void portSend(Port *p, void *msg){

    nWaitSem(p->vacíoMsg);

    p->msg=msg;

    nSignalSem(p->hayMsg);
    nWaitSem(p->leídoMsg);
}

void *portReceive(Port *p){

    void *retval;

    nWaitSem(p->hayMsg);

    retval=p->msg;

    nSignalSem(p->leídoMsg);
    nSignalSem(p->vacíoMsg);

    return retval;
}
```

5. Solución Ejercicio

```
nSem mutex = nMakeSem(1);

void esperar(){
    nWaitSem(mutex);
    if(!ocurrio){
        conEsperan++;
        nSignalSem(mutex);
        nSignalSem(sem);
    }
    nSignalSem(mutex)
}

void notificar(){
    nWaitSem(mutex);
    ocurrio=TRUE;
    while(conEsperan>0){
        conEsperan--;
        nSignalSem(sem);
    }
    nSignalSem(mutex);
}
```

6. Solución Ports con Semáforos

```
typedef struct Port{
    nSem hayMsg; //Bloquea lector si no hay mensajes
```