CC41B Sistemas Operativos
Control 1 – Semestre Primavera 2007
Prof.: Luis Mateu

Pregunta 1

```
typedef struct Nodo {
  char *llave, *def;
  struct Node *izq, *der;
  nSem semIzq, semDer;
} Nodo;

typedef struct {
  Nodo *raiz;
  nSem semRaiz;
} ConcDict;

void addDef(ConcDict *dict,
     char *llave, char *def)
{
  insertar(dict->semRaiz,
           &dict->raiz,
           llave, def);
}
```

```
void insertar(nSem sem, Nodo **ppnodo,
        char *llave; char *def) {
  Nodo *pnodo;
  if (*ppnodo==NULL) {
    *ppnodo= crearNodoHoja(llave,
                    def); /* dado */
  }
  else {  /*** Supuesto: la llave  ***
           *** no está en el arbol ***/
    pnodo= *ppnodo;
    if (strcmp(llave, pnodo->llave)<0) {
      insertar(pnodo->semIzq,
               &pnodo->izq, llave, def);
    }
    else {
      insertar(pnodo->semDer,
               &pnodo->der, llave, def);
}}}
```

El programa anterior es la implementación incompleta de un diccionario concurrente en base a un árbol de búsqueda binaria. El procedimiento `crearNodoHoja` es dado e inicializa correctamente todos los campos. Esto incluye la creación de los semáforos que contiene el nodo, cada uno con un ticket. Cuando se crea el diccionario, se crea automáticamente el semáforo para la raíz en la estructura `ConcDict`, aún cuando el diccionario esté vacío. Suponga que no existe una operación para eliminar definiciones en el diccionario.

(a) Haga un diagrama de threads que muestre que el diccionario puede quedar en un estado inconsistente cuando 2 threads llaman a `addDef` concurrentemente. (2 puntos.)

(b) Complete el programa de más arriba agregando la líneas que faltan. Ud. necesita garantizar la exclusión mutua cuando 2 threads están trabajando en el mismo nodo, pero Ud. *debe* permitir que continúen en paralelo una vez que trabajan en ramas distintas del árbol. El código dado es correcto, sólo agregue las líneas que faltan para la sincronización. La declaración de los semáforos y su paso como parámetros es una ayuda para que resuelva el problema. (4 puntos.)

Importante: Implementar la exclusión mutua a nivel del árbol completo tiene 0 puntos.

Pregunta 2

La siguiente es una implementación secuencial del juego de la vida.

```
int **mat, **newMat; /* se inicializan en el nMain */
int step;

void lifeGame(int n, int k) {
  int i, j;
  for (step= 0; step<k; step++) { /* ciclo externo */
    for (i= 0; i<n; i++)
      for (j= 0; j<n; j++)
        newMat[i][j]= compute(mat, i, j); /* dado */
    swap(&mat, &newMat); /* dado */
  }
}
```

El programa parte con una Matriz M (mat en el programa) y calcula las matrices M_0, M_1, M_2, ..., M_{k-1}. Cada matriz es de n x n. El valor de cada elemento de la matriz M_{step} depende únicamente de M_{step-1}. Este hecho simplifica la paralelización del procedimiento. El procedimiento `compute` es dado y su implementación no es relevante en esta pregunta.

Paralelice `lifeGame` haciendo uso de P threads adicionales. Para ello haga que cada thread construya n/p filas (o columnas) de la nueva matriz. Suponga que n es múltiplo de p. Observe que en cada instante todos los threads deben trabajar en la construcción de la misma matriz M_{step}, es decir el mismo valor para `step`, porque de otro modo su programa podría no entregar el mismo resultado que la versión secuencial. Esto significa que al final de cada iteración de `step`, se debe esperar a que todos los threads completen la construcción de su respectiva submatriz, y luego comenzar una nueva iteración.

Restricciones:

- Ud. debe crear solo p threads adicionales (el cliente no acepta el sobrecosto adicional de crear más de p threads).
- Resuelva el problema utilizando nSystem y los *monitores* de nSystem.