Clase Auxiliar 2

Prof: L. Mateu
Aux: J. Bustos

1. Introducción a nSystem

De acuerdo a la documentación de nSystem, este es un:

> *...sistema de procesos livianos para Unix con fines pedagógicos. El sistema consiste de unos cientos de líneas de código en C (la mitad son comentarios) que implementan:*
>
> - *Creación y destrucción de procesos livianos (tareas). Estos procesos comparten un mismo espacio de direcciones (están dentro de un solo proceso UNIX).*
> - *Paso de Mensajes para la sincronización de tareas.*
> - *Entrada/Salida no bloqueante para el proceso UNIX.*
> - *Un scheduler muy simple. Permite implementar administración preemptive y non-preemtive.*

En terminos prácticos, nSystem es una librería que permite administrar threads dentro de un solo proceso pesado Unix. Para poder utilizar nSystem se deben llevar a cabo los siguientes pasos:

1. Bajar nSystem desde Ucursos *http://ucursos.ing.uchile.cl*. El cual estará disponible a mas tardar con el enunciado de la tarea 1.
2. Descomprimirlo y configurar la variable de ambiente `$NSYSTEM` con el `PATH` de nSystem. Por ejemplo:
   - En *tcsh*: `setenv NSYSTEM /home/micarpeta/nSystem`
   - En *bash*: `export NSYSTEM=/home/micarpeta/nSystem`
3. Compilar nSystem

   ```
   cd $NSYSTEM/src
   make
   ```

   Observe que por defecto los `Makefiles` de nSystem estan conigurados para Linux. Para utilizalro en Solaris debe comentar y descomentar las lineas respectivas.
4. Compilar el programa que utiliza nSystem de la siguiente manera:

   ```
   gcc -I$NSYSTEM/include/ pruebat1.c tarea1.c $NSYSTEM/lib/libnSys.a -o prueba
   ```

   Donde `tarea1.c` es la tarea desarrollada por Uds y `pruebat1.c` es el archivo de prueba para la tarea. Como observación y a modo que preparen los `Makefiles` de entrega para las tareas, lo anterior es equivalente a:

   ```
   gcc -c -I$NSYSTEM/include/ tarea1.c
   gcc -c -I$NSYSTEM/include/ pruebat1.c
   gcc -I$NSYSTEM/include/ pruebat1.o tarea1.o $NSYSTEM/lib/libnSys.a -o prueba
   ```

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 profundida del árabol 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 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;
    }
  }
}
```