CC4302 - Sistemas Operativos
Auxiliar 1

Profesor: Luis Mateu
Auxiliar: Diego Madariaga

19 de marzo de 2019

1. nSystem

El siguiente es un programa de ejemplo en nSystem:

```
#include "nSystem.h"

int escritor(int num, int espera);

int nMain() {
    nTask tareas[3];
    int i;
    for (i = 0; i < 3; ++i)
        tareas[i] = nEmitTask(escritor, i, i * 200);
    for (i = 0; i < 3; ++i)
        nWaitTask(tareas[i]);
    nPrintf("Fin ejemplo\n");
}

int escritor(int num, int espera) {
    int i = 5;
    while (i > 0) {
        nPrintf("thread %d: %d\n", num, i);
        nSleep(espera);
        i--;
    }
}
```

Pregunta: ¿Qué sucede si en el ejemplo usamos `sleep` en vez de `nSleep`?

Respuesta: se bloqueará el proceso Unix completo, es decir el proceso que contiene a nSystem y todos los threads, en vez de detenerse sólo el thread que invoca `sleep`.

2. Buscar en un árbol

El primer caso es una búsqueda en profundidad:

```
int buscarSeq1(Nodo *node, int num) {
    if (node == NULL)
        return FALSE;
    else if (node->valor == num)
        return TRUE;
    else
        return buscarSeq1(node->izq, num) || buscarSeq1(node->der, num);
}
```

El segundo caso es crear una tarea por cada nodo:

```
int buscarSeq2(Nodo *node, int num) {
    if (node == NULL) {
        return FALSE;
    } else if (node->valor == num) {
        return TRUE;
    } else {
        nTask task1 = nEmitTask(buscarSeq2, node->izq, num);
        nTask task2 = nEmitTask(buscarSeq2, node->der, num);
        int r1 = nWaitTask(task1);
        int r2 = nWaitTask(task2);
        return r1 || r2;
    }
}
```

Notar que es un error hacer `return nWaitTask(task1) || nWaitTask(task2)` porque si `nWaitTask(task1)` retorna verdadero no ejecutará la segunda parte del OR y por tanto podría no esperar por `task2` y podría finalizar todo el programa estando `task2` aún en ejecución.

El tercer caso es:

```
int buscarSeq3rec(Nodo *node, int num, int *FOUND) {
if (*FOUND) {
return TRUE;
} else if (node == NULL) {
return FALSE;
} else if (node->valor == num) {
*FOUND = TRUE;
return TRUE;
} else {
nTask task1 = nEmitTask(buscarSeq3rec, node->izq, num, FOUND);
nTask task2 = nEmitTask(buscarSeq3rec, node->der, num, FOUND);
int r1 = nWaitTask(task1);
int r2 = nWaitTask(task2);
return r1 || r2;
}
}

int buscarSeq3(Nodo *node, int num) {
int FOUND = FALSE;
int res = buscarSeq3rec(node, num, &FOUND);
return res;
}
```

Notar que en este caso `FOUND` es una variable única compartida por todos los threads que eventualmente puede ser escrita de forma simultánea por varios threads (si se encuentra `num` en distintos lugares del árbol al mismo tiempo). Sin embargo en este caso no implica un problema porque todos estarán escribiendo el mismo valor `TRUE` y la asignación de una constante es una operación atómica. Notar que este es uno de los pocos casos en que es posible leer y escribir una variable global sin tener problemas, sin embargo **la gran mayoría de las veces es necesario usar algún mecanismo de sincronización para leer y modificar una variable global** como usar semáforos, monitores, etc. lo que será parte de las próximas clases auxiliares.

3. Ejemplo de I/O

El siguiente es un programa de ejemplo, que lee desde la entrada estándar un número n, y luego escribe n archivos en paralelo con una cantidad variable de líneas.

```
#include <stdio.h>
#include <stdlib.h>
#include "nSystem.h"

#define MIN_LINEAS 100000
#define MAX_LINEAS 150000
```

```
int escritor(int num);

int nMain() {
    nSetTimeSlice(1);
    nSetNonBlockingStdio();
    nPrintf("Cantidad de archivos? ");
    char c[4];
    nRead(0, c, 3);
    int cant = atoi(c);
    if (cant <= 0)
        return;
    srand(time(0));
    nTask tareas[cant];
    int i;
    for (i = 0; i < cant; ++i) {
        tareas[i] = nEmitTask(escritor, i);
    }
    for (i = 0; i < cant; ++i) {
        nWaitTask(tareas[i]);
    }
}

int escritor(int num) {
    char archiv[30];
    sprintf(archiv, "archivo%d.txt", num);
    int fd = nOpen(archiv, O_CREAT | O_WRONLY | O_TRUNC, 0644);
    nFprintf(fd, "Archivo %d\n\n", num);
    int lineas = MIN_LINEAS + (rand() % (MAX_LINEAS - MIN_LINEAS));
    nPrintf("Escribiendo archivo %s de %i lineas\n", archiv, lineas);
    int i;
    for (i = 1; i <= lineas; ++i) {
        nFprintf(fd, "linea %i: %i\n", i, rand());
    }
    nClose(fd);
    nPrintf("Archivo %s finalizado\n", archiv);
}
```