CC4302 - Sistemas Operativos
Auxiliar 1 - Introducción a nSystem

Profesor: Luis Mateu
Auxiliar: Diego Madariaga

Marzo 2020

1. nSystem

Durante el curso usaremos nSystem, el cual permite crear threads dentro de un solo proceso Unix. No usaremos otras librerías (como *pthreads*) porque al estudiar el *scheduling* de tareas veremos como está hecho nSystem y le haremos algunas modificaciones.

Una vez descargado[1] pueden compilarlo siguiendo las instrucciones dentro del archivo README. Para recompilar, no olvide ejecutar `make clean` antes de `make`.

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--;
    }
}
```

Con `nEmitTask` se ejecuta el método `escritor` en un thread independiente retornando un descriptor del thread iniciado. Con `nWaitTask` se bloquea la ejecución de un thread a la espera de que la tarea termine, y cuando la tarea finaliza el valor de retorno de `nWaitTask` corresponde al valor retornado por el método iniciado con `nEmitTask`.

Notar que el ejemplo no tiene `main` si no que el método principal es `nMain`, esto es porque el `main` lo proveerá nSystem, quien primero creará el entorno para usar threads y luego ejecutará nuestro `nMain`. Por tanto, para compilar este archivo no funcionará el uso de `gcc ejemplo.c` (arrojará el

[1] `https://users.dcc.uchile.cl/~lmateu/CC41B/`

mensaje *undefined reference to 'main'*). Notar que varias de las funciones comúnmente usadas tienen una versión para nSystem como `nSleep`, `nPrintf`, `nOpen`, `nClose`, `nFprintf`, `nMalloc`, etc. para que sea usada por cada thread sin interferir con los otros threads activos.

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

Para compilar el código primero definamos la variable de ambiente `NSYSTEM` (en bash `export NSYSTEM=/home/.../nSystem` o en tcsh `setenv NSYSTEM /home/.../nSystem`) y luego:

- Opción 1: crear un archivo .o y luego incluirlo con nSystem para generar el ejecutable:

```
gcc -c ejemplo.c -I$NSYSTEM/include
gcc ejemplo.o $NSYSTEM/lib/libnSys.a -o ejemplo
```

- Opción 2: usar alguno de los Makefile que están en los ejemplos de nSystem.

```
make APP=aux1p1
```

Si abrimos un Makefile veremos que se compone de reglas (escritas antes de un dos puntos), y para poder aplicar cada regla se deben cumplir primero sus condiciones (escritas después de los dos puntos). Cada condición puede corresponder a otras reglas del Makefile o a nombres de archivos en el mismo directorio.

Si una condición corresponde a una regla entonces se debe ejecutar ésta primero (la cual también debe cumplir sus condiciones), y si corresponde a un archivo, éste debe existir y su fecha de modificación debe ser más reciente que sus dependencias (en caso contrario debe aplicar la regla correspondiente para recompilarlo).

Una vez que se cumplen todas las condiciones de una regla se ejecutan las instrucciones bajo ella, notar que estas instrucciones *deben* estar precedidas de un tabulador y no de espacios en blanco. En el código fuente de esta auxiliar se incluye un Makefile de ejemplo.

2. Buscar en un árbol

Dada la siguiente estructura de datos de un árbol binario:

```
typedef struct Nodo {
    int valor;
    struct Nodo* izq;
    struct Nodo* der;
} Nodo;
```

Implementar los siguientes métodos que buscan un número dentro del árbol (los valores en el árbol no tienen ningún orden específico):

- `int buscarSeq1(Nodo *node, int num)` Busca `num` en el árbol haciendo un recorrido en profundidad.
- `int buscarSeq2(Nodo *node, int num)` Busca `num` en el árbol haciendo un recorrido simultáneo en cada rama creando tantos threads como nodos en el árbol.
- `int buscarSeq3(Nodo *node, int num)` Como el anterior, pero con la optimización que la ejecución de todas las tareas termina cuando alguna encuentra el elemento buscado.

3. Ejemplo de I/O

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