Práctica 3: Threads en espacio de usuario.

Diseño de Sistemas Operativos

Pablo Castro Valiño (pablo.castro1@udc.es)

Curso 2012-2013

Índice

1. Introducción

La utilización de threads a día de hoy es una práctica tremendamente común y fundamental para el desarrollo de la informática actual. La necesidad de aumentar la paralelización en los procesos ha llevado a la industria a hacer especial hincapié en el desarrollo de este campo de la computación.

Nuestra práctica en concreto consiste en una implementación rudimentaria de threads en espacio de usuario, los cuales utilizaremos en la parte básica para realizar el cálculo simultáneo del factorial de tres números diferentes y en la parte optativa, para la realización de un programa que realice doble buffering, en una primera parte de manera síncrona y en una segunda de manera asíncrona.

2. Procedimiento para la realización de la práctica.

2.1. Realización de un factorial en 3 threads.

Para la realización de esta parte de práctica se ha creado el fichero **factorial.c** (4) con el total contenido de la misma. En él encontramos diferenciadas las funciones que implementan los diferentes threads (*t0, t1, t2*), la función *yield* que realiza el intercambio entre ellos y la función *adjust_frame* que funciona como función auxiliar para recolocar el *stack pointer* cuando se realiza un salto.

A continuación vamos a comentar el funcionamiento de las diferentes funciones:

t0, t1, t2 El comportamiento de estas funciones es tremendamente simple, se limitan a realizar el cálculo del factorial del número que reciben, de forma recursiva y haciendo continuas llamadas a *adjust_frame* para cambiar entre los hilos.

adjust_frame Es la función que únicamente llama a *Yield* y realiza un return.

Se utiliza únicamente para conseguir que cuando se realice un salto y se modifique el puntero base de la pila, la cima de la pila quede en el lugar donde queremos.

En circunstancias normales, el puntero de la cima de la pila va a quedarse muy arriba, de forma que si introducimos esta función antes de volver a nuestro código, al realizar *return* de ella, se colocará el puntero

cima de pila en donde está el base actual y el base se colocará en la dirección adecuada del frame anterior, y de esta forma se continuará ejecutando correctamente el código de la función.

Yield Es la función que realiza la inicialización de los threads y que cambia entre ellos.

Lo primero que hace es guardar el frame anterior y la dirección de retorno y comprobar si es la primera ejecución de cada uno de los threads, para en ese caso, lanzarlos.

También se selecciona mediante un bucle el thread hacia el que vamos a cambiar y se comprueba que dicho thread elegido no ha terminado, en caso contrario se selecciona el siguiente.

En el caso de que todos hayan terminado, se coloca la variable **ct** a 3 para indicar que se debe de terminar y volver al frame de la función *adjust_frame* que se lanzó desde el *Main*, y por lo tanto terminar la ejecución del programa.

2.2. Doble buffering

Para la realización de esta parte de práctica se ha creado el fichero **copia_sincro.c** (5) con el total contenido de la misma.

En él encontramos diferenciadas las funciones que implementan los diferentes threads (*t0, t1*), la función *yield* que realiza el intercambio entre ellos y la función *slow* que funciona como función auxiliar realizando cálculos inservibles para realentizar la ejecución.

El funcionamiento del programa es muy simple, se abre el fichero de tres modos distintos, el primero para crear el archivo en el que se realizará la lectura y escritura en caso de que no exista, la segunda en modo lectura y la tercera en modo escritura.

El thread *t0* realiza la lectura del fichero en un buffer común mientras que *t1* se encarga de la escritura de dicho buffer en el fichero, de forma que cuando terminemos se habrá leido del fichero el valor introducido, 5 veces y se habrá vuelto a escribir en el fichero. De esta forma el fichero **entrada.txt** contrendrá 'VALOR' 6 veces, el que hemos introducido desde el *Main* para inicializar el fichero y los 5 siguientes que hemos leido y reescrito.

3. Índice de archivos adjuntos.

1. **Makefile** Se ha introducido un fichero *Makefile* para agilizar la compilación.

   - **make all** Compila todos los ficheros.
   - **make fact** Compila los ficheros que realizan el factorial.
   - **make sinc** Compila el fichero que realiza el doble buffering.
   - **make clean** Elimina todos los ficheros ejecutables.

2. **tests/programa1.c**
   Programa básico realizado en clase para testear el limite del stack.

3. **tests/programa2.c**
   Programa básico realizado en clase para la realización de los threads, que únicamente escribe la letra 'a' o 'b' en función del thread en el que se encuentre, y lo hace de manera alternativa.

4. **factorial.c**
   Programa que realiza el factorial de forma recursiva en tres threads distintos. Para ello se sube por el stack haciendo llamadas recursivas en cada thread, de forma que el ultimo frame de cada uno, el valor del número del que se quiere hacer el factorial va a ser 0 y luego vuelve a bajar intercalando unos threads con otros para realizar el cálculo.

5. **copia_sincro.c**
   Programa que crea dos threads, uno que escribe en un fichero y el otro que lee.