SISTEMAS CONCURRENTES Y PARALELOS
PRÀCTICA 2

Presentación

Objetivos

Esta práctica tiene como objetivo la sincronización de una aplicación concurrente multi-hilo utilizando los mecanismos de sincronización de phtreads y java. Para su realización se pondrán en práctica los conceptos de sincronización presentados en la asignatura.

Hay que entregar obligatoriamente la versión de pthreads de la práctica y la versión en Java**.**

Presentación de la práctica

Hay que presentar los archivos con el código fuente realizado. Toda la codificación se hará exclusivamente en lenguaje C y Java.

Enunciado de la práctica

Esta práctica tiene como base la aplicación *manfut* concurrente realizada en la primera práctica de la asignatura. En esta segunda práctica nos vamos a centrar en dotar a la aplicación de los mecanismos de sincronización oportunos que permita una ejecución correcta, determinista y eficiente de la misma.

Sincronización

En cada una de las versiones de la aplicación (pthreads y java) se deberá implementar las siguientes sincronizaciones y modificaciones:

- **Evitar las condiciones de carrera**. Se tendrá que analizar todas las operaciones que pueden generar condiciones de carrera e implementar las secciones críticas que se consideren oportunas para garantizar un funcionamiento correcto y determinista de la aplicación. En la versión de java se tiene que lograr que todas las clases implementadas sean thread-safe.

  En esta versión no sé calculara un óptimo parcial por cada hilo, si no un óptimo parcial global a todos los hilos.

- **Threads de mensajes**. Implementar un hilo adicional que será el encargado de imprimir todos los mensajes por pantalla del resto de hilos[1]. Para que sea más eficiente, los mensajes se escribirán de 100 en 100 en el dispositivo (pantalla/consola). Hay que implementar una cola/lista para guardar los mensajes hasta que se puedan imprimir, garantizando que la función que utilizan los hilos para enviar el mensaje sea lo más rápida posible.

- **Espera resultado final.** En esta versión no se permite la utilizar los *join* para esperar a que los hilos finalicen y se pueda mostrar el resultado. Hay que utilizar un mecanismo de sincronización para que el hilo principal espere a que se haya calculado el resultado.

- **Estadísticas**. La aplicación calculará una serie de estadísticas locales a cada objeto/hilo y las globales durante la ejecución de cada una de las etapas:

[1] En esta versión de la aplicación se tiene que activar todos los mensajes de la versión original.

- Número de Combinaciones evaluadas.
- Número de combinaciones no válidas.
- Coste promedio de las combinaciones válidas
- Puntuación promedio de las combinaciones válidas.
- Mejor combinación (desde el punto de vista de la puntuación).
- Peor combinación (desde el punto de vista de la puntuación)

Al final de la ejecución de la aplicación se mostrarán tanto las estadísticas finales de cada hilo, como las estadísticas globales.

- **Progreso**. La aplicación mostrará cada vez que cada hilo haya evaluado M combinaciones las estadísticas parciales hasta ese momento. M se pasará como parámetro a la aplicación (M=25000 por defecto).

Al implementar las sincronizaciones se tienen que utilizar todos los mecanismos de sincronización que hemos visto en clase. En la versión pthread hay que utilizar: Semáforos, *Mutex*, Barreras y variables de condición. En la versión java hay que utilizar obligatoriamente: *synchronized*, variables de condición y semáforos.

Funciones involucradas.

En la práctica podéis utilizar las siguientes funciones de c:

- `int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);`
- `int pthread_mutex_lock(pthread_mutex_t *mlock)`
- `int pthread_mutex_unlock(pthread_mutex_t *mlock)`
- `pthread_mutex_destroy(&lock);`
- `int pthread_barrier_init(pthread_barrier_t *restrict barrier, const pthread_barrierattr_t *restrict attr, unsigned count);`
- `int pthread_barrier_wait(pthread_barrier_t *barrier);`
- `int pthread_barrier_destroy(pthread_barrier_t *barrier);`
- `int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *attr);`
- `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mlock);`
- `int pthread_cond_signal(pthread_cond_t *cond);`
- `int pthread_cond_broadcast(pthread_cond_t *cond);`
- `int pthread_cond_destroy(pthread_cond_t *cond);`

En la práctica podéis utilizar las siguientes clases y métodos de java:

- Palabra clave *synchronized*
- *clase java.util.concurrent.locks.condition:*
  - *signal()*
  - *void signalAll()*
  - *void await()*

- *clase java.lang.Object:*
  - *void notify()*
  - *void notifyAll()*
  - *void wait()*
- *clase java.util.concurrent.Semaphore:*
  - *void acquire()*
  - *void acquire(int permits)*
  - *void release()*
  - *void release(int permits)*

Análisis prestaciones

Calcular el tiempo de ejecución de la versión concurrente sincronizada y compararlo con el de la versión concurrente sin sincronizar y con la versión secuencial. Discutir los resultados obtenidos.

Analizar también, cómo evoluciona el tiempo de ejecución de la nueva versión en función del número de hilos utilizados en la aplicación concurrente. Entregar un pequeño informe en donde se muestren (preferiblemente de forma gráfica) y expliquen los resultados obtenidos.

Indicar en el informe las características del hardware en donde habéis obtenido los resultados (especialmente el número de procesadores/núcleos).

Evaluación

La evaluación de la práctica se realizará en función del grado de eficiencia/concurrencia logrado.

Se utilizarán los siguientes criterios a la hora de evaluar la práctica, partiendo de una nota base de 10 para esta versión:

- Condiciones Carrera (-2.0p)
- No es thread-safe (-1.0p)
- Thread mensajes (-1.0p)
- Estadísticas (-1.0p)
- Progreso (-1.0p)
- Esperar Resultado final (-0.5p)
- No utilizar Mutex (-1.0p)
- No utilizar Barreras (-1.0p)
- No utilizar Variables de condición (-1.0p)
- No utilizar Semáforos (-1.0p)
- No liberar mecanismos de sincronización (-0.5p)

- Evaluación del informe de la práctica:

  + Informe completo, con gráficos y conclusiones correctas (+0.25p, +1.0p)

  - Sin informe (-1.5p)

  - Informe con incoherencias en los tiempos sin justificar (-0.5p)

- Informe: análisis poco riguroso (-0.5p)
- No se explica el diseño de la solución de sincronización propuesta (-0.5p)
- No se describe los problemas encontrados y como se han solventado (-0.5p)

Formato de entrega

MUY IMPORTANTE: La entrega de código que no compile correctamente, implicará suspender TODA la práctica.

No se aceptarán prácticas entregadas fuera de plazo (salvo por razones muy justificadas).

La entrega de esta práctica es obligatoria para todos los miembros del grupo.

Comenzar vuestros programas con los comentarios:

```
/* ------------------------------------------------------------
Práctica 2.
Código fuente: manfut.c
Grau Informàtica
NIF i Nombre completo autor1.
NIF i Nombre completo autor2.
------------------------------------------------------------ */
```

Para presentar la práctica dirigiros al apartado de Actividades del Campus Virtual de la asignatura de Sistemas Concurrentes y Paralelos, ir a la actividad "Práctica 1" y seguid las instrucciones allí indicadas.

Se creará un fichero tar/zip con todos los ficheros fuente de la práctica, con el siguiente comando:

```
$ tar -zcvf prac2.tgz fichero1 fichero2 ...
```

se creará el fichero "prac1.tgz" en donde se habrán empaquetado y comprimido los ficheros "fichero1", fichero2, y ...

Para extraer la información de un fichero tar se puede utilizar el comando:

```
$ tar -zxvf tot.tgz
```

El nombre del fichero tar tendrá el siguiente formato: "Apellido1Apellido2PRA2.tgz". Los apellidos se escribirán sin acentos. Si hay dos autores, indicar los dos apellidos de cada uno de los autores separados por "_". Por ejemplo, el estudiante "Perico Pirulo Palotes" utilizará el nombre de fichero: `PiruloPalotesPRA2.tgz`

Fecha de entrega

Entrega a través del Campus Virtual el 10 de enero a las 23:55.