Presentación

Objetivos

Esta práctica tiene como objetivo el diseño e implementación de una aplicación concurrente multi-hilo utilizando phtreads. Para su realización se pondrán en práctica muchos de los conceptos presentados en esta asignatura.

Presentación de la práctica

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

Enunciado de la práctica

El objetivo de la práctica es implementar una versión concurrente de un problema de optimización (problema de la mochila).

Se quiere solucionar un problema de optimización de recursos para un director deportivo de un equipo de fútbol. El objetivo es que a partir de un presupuesto para fichajes se seleccione la mejor plantilla de jugadores (sólo tendremos en cuenta los jugadores titulares) en función del coste y puntuación de los diferentes jugadores disponibles en el mercado de fichajes.

Debido a que la solución de este problema puede requerir la utilización de todos los recursos de la máquina, proponemos desarrollar una solución concurrente multi-hilo para mejorar su tiempo de ejecución.

Habrá que implementar dos versiones de la práctica: una en C utilizando pthreads y otra en java utilizando los threads de Java. Implementar la aplicación para que soporte un número variable de threads para solventar el problema. Por defecto, se utilizarán 2 threads.

Problema optimización (versión secuencial)

El fichero manfut.c contiene el código para calcular la plantilla óptima de forma secuencial. El programa recibe dos parámetros obligatorios: el presupuesto de fichajes (en millones de $) y un archivo (mercado *.txt) con un listado de los jugadores disponibles:

- `Manfut 200 mercat25j.csv` `# Obtiene la combinación óptima con un presupuesto millones y un mercado de fichajes con 16 jugadores (mercat25j.csv).`

El método utilizado para calcular la plantilla óptima consiste en evaluar todas las posibles combinaciones de jugadores disponibles en el mercado. El programa calcula el coste y la puntuación total de cada plantilla, guardando en todo momento la plantilla con mayor puntuación y que no supere el presupuesto establecido.

Las posibles combinaciones se obtienen codificando la plantilla con un número binario donde tendremos X bits por cada posición de la plantilla[1]. El número de bits X requeridos para cada posición dependerá del número de jugadores disponibles en el mercado (por ejemplo, si en el

[1] Para acotar la complejidad del problema, supondremos una plantilla de futbol-7, con 1 portero, 3 defensas, 2 medios y un delantero.

mercado hay 4 porteros, se utilizarán 2 bits. Si hay 16 defensas, se utilizarán 4 bits). El número total de bits requerido se obtiene:

$$B = \sum_{i=0}^{Pos} Bits_i * Puestos_i$$

que nos define el número total de combinaciones existentes (2^B).

Por ejemplo, la codificación que tendríamos para un equipo de futbol-7 con las siguientes posiciones (1 portero, 3 defensas, 2 medios y 1 delantero) y los siguientes jugadores en el mercado de fichajes (4 porteros, 8 defensas, 8 medios y 4 delanteros) sería:

$$Del_1 Del_1 M_1 M_1 M_1 M_2 M_2 M_2 D_1 D_1 D_1 D_2 D_2 D_2 D_3 D_3 D_3 P_1 P_1 = 19\ bits$$

Y una posible combinación podría ser 00-001-000-010-001-000-00 (16928), y representaría una plantilla con el primer delantero del mercado, el primer y el segundo medio, los tres primeros defensas y el primer portero.

Al finalizar el cálculo, el programa muestra por pantalla la información sobre la mejor plantilla encontrada.

Problema optimización (versión concurrente)

El objetivo de la versión concurrente es mejorar el tiempo de cálculo del mejor equipo. Para ello se distribuirá el trabajo a realizar (combinaciones de equipos a evaluar) entre un conjunto de threads para que se realice de forma concurrente/paralela.

Para esto, tenéis que analizar el código secuencial localizando los hot-sports (puntos calientes) en donde se concentra el cómputo de la aplicación secuencial. Para ello tenéis que analizar el código o bien utilizar una herramienta de profiling (gnuprof, por ejemplo). Una vez detectado en que parte del código se concentra el trabajo que realiza la aplicación, hay que ver cómo se puede distribuir entre múltiples threads de forma que se procese concurrentemente.

A la hora de realizar el diseño concurrente, tenéis que tener en cuenta que en esta primera versión no se van a utilizar mecanismos de sincronización. Por lo tanto, para garantizar que la aplicación concurrente es determinista no se debe utilizar variables compartidas que puedan ser modificadas simultáneamente por mas de un hilo, lo cual podría provocar condiciones de carrera e incoherencias en la aplicación.

Al programa concurrente se le pasará los mismos parámetros que a la versión secuencial más un parámetro adicional para definir el número de hilos de ejecución:

Sintaxis: `ConcManfut <inversión> <mercado_jugadores> <num_threads>`

Funciones involucradas.

En la práctica podéis utilizar las siguientes funciones de c:

- *`int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);`*
- *`int pthread_join(pthread_t thread, void **retval);`*

- *void pthread_exit(void *retval);*
- *int pthread_cancel(pthread_t thread);*

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

- *clase java.lang.Thread*
  - `run()`
  - `start()`
  - `join()`
  - `interrupt()`
  - `isAlive()`
  - `currentThread()`
  - `sleep(long millis)`

Análisis prestaciones

Calcular el tiempo de ejecución de la versión concurrente y compararlo con el de la versión secuencial. Discutir los resultados obtenidos. El objetivo de la práctica es que **la versión concurrente sea más rápida que la versión secuencial**.

Analizar también, el tiempo de ejecución de la versión concurrente en función del número de hilos. Entregar un pequeño informe en donde se muestren (preferiblemente de forma gráfica) y se 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 las dos implementaciones de la práctica (C y Java), partiendo de una nota base de 5 para cada una de las versiones:

- Aspectos que se evalúan para cada una de las versiones:
  - Ejecución Secuencial (Susp)
  - Ejecución no determinista ([-1.0p,-5.0p])
  - Prestaciones (-1.0p, +0.5p)
  - No mejora el tiempo de la versión secuencial (-2.0p)
  - Número incorrecto de hilos (-0.5p)
  - Join hilos (-0.5p)
  - Control Errores: cancelación hilos (-0.5p)

- Condiciones de carrera (-0.5p)
- No liberar memoria: (-0.25p,-0.5p)
- Descomposición desbalanceada (Versión óptima) (-0.5p)
- Descomposición parcial: falta asignar resto división (-0.5p)

- Evaluación del informe de la práctica:

  + Informe completo, con gráficos y conclusiones correctas (+0.25p,+1.0p)
  - Sin informe (-1.0p)
  - Informe con incoherencias en los tiempos sin justificar (-0.5p)
  - Informe: análisis poco riguroso (-0.5p)
  - No se explica el diseño de la solución propuesta (-0.5p)
  - Descripción de los problemas encontrados y como se han solventado.

- Estilo del código.
  - Comentarios
  - Utilizar un correcto sangrado (*indentation*)
  - Descomposición Funcional (Código modular y estructurado)
  - Control de errores.

Se puede tener en cuenta criterios adicionales en función de la implementación entregada.

Versiones Secuenciales

Junto con el enunciado se os proporciona las versiones secuenciales en C y Java que implementan el problema de optimización. Podéis utilizar dichas versiones para implementar las versiones concurrentes que se os pide en esta práctica.

Versión en C

La versión secuencial en C se denomina manfut.c. La versión adaptada para la práctica la tenéis disponible en el archivo comprimido Secencial_C.zip que se os pasa en la actividad.

Para poder utilizar la versión en C será necesario compilar los archivos utilizando el gcc:

- `gcc manfut.c -o manfut -lm`

Para poder ejecutar la versión secuencial, únicamente tenéis que especificar el ejecutable, el presupuesto de fichajes y el nombre del fichero de entrada que contiene el mercado de jugadores disponible:

- `./manfut 300 mercat15j.csv`

También podéis obtener el tiempo requerido para la ejecución si utilizáis el comando time:

- `time ./manfut 300 mercat15j.csv`

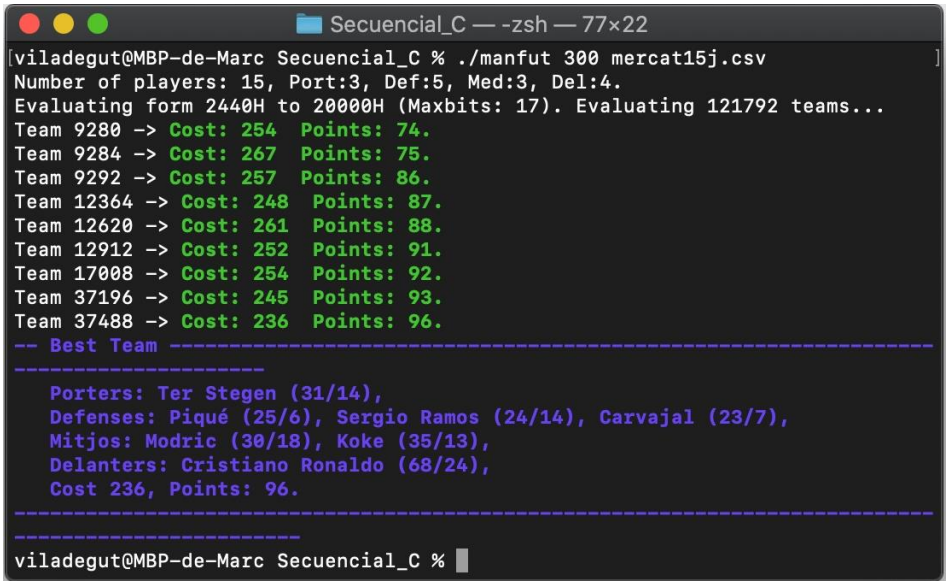

Figura 1. Ejecución secuencial del programa Manfut desarrollado en C.

Con estos ficheros también se os adjunta un conjunto de ficheros de entrada en el directorio Test, estos ficheros os pueden servir para probar la aplicación.

Versión en Java

La versión secuencial en Java se denomina Manfut. Para poder utilizar la versión en Java, tenéis que descomprimir el archivo y compilarlo utilizando el siguiente comando:

- `javac -cp "." *.java`

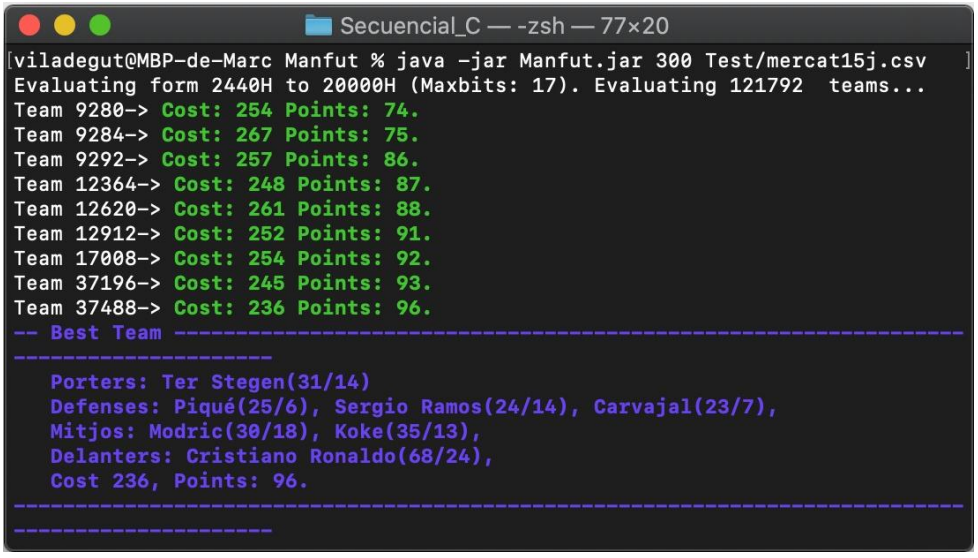

Figura 2. Ejecución secuencial del programa Manfut desarrollado en Java.

Para ejecutar la versión secuencial, ejecutáis la clase Manfut, especificando el presupuesto a invertir y el nombre del fichero de entrada que contiene el mercado de jugadores disponible:

- `javac -cp "." Manfut 300 mercat15j.csv`
- `time java -jar Manfut.jar 300 mercat15j.csv`

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

Comenzar vuestros programas con los comentarios:

```
/* ------------------------------------------------------------
Práctica 1.
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 prac1.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 2929Palotes" utilizará el nombre de fichero: `PiruloPalotesPRA1.tgz`

Fecha de entrega

Entrega a través del Campus Virtual el 29 de noviembre, entrega presencial en la clase de grupo de laboratorio durante la semana del 23 de noviembre al 29 de noviembre de 2020.