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

Erase una vez, en un lejano país, un rey que poseía en su reino una pequeña y valiosísima colección de árboles, que había heredado de los viajes de sus antepasados a otros reinos lejanos. Para proteger los árboles de los ladrones, el rey ordenó que se construyese una valla a su alrededor. La tarea recayó sobre el mago del reino, hombre sabio y prudente.

Desafortunadamente, el mago se dio cuenta con gran presteza de que el único material disponible para construir la valla era la madera de los propios árboles. Dicho de otro modo, era necesario cortar algunos árboles para construir la valla alrededor de los restantes. Lógicamente, para no provocar la ira del rey y perder su empleo y su cabeza en el empeño, el mago quiso minimizar el valor de los árboles que tuviesen que ser cortados. Se retiró a su torre y estuvo allí hasta que encontró la mejor solución posible al problema planteado por el rey. La valla fue construida y todos vivieron felices.

En esta práctica, se desarrollará una aplicación concurrente en C basada en los mecanismos de concurrencia de POSIX (pthreads) que resuelva el problema al que se enfrentó el mago de nuestra historia.

Indicaciones.

La resolución de este problema se puede abordar siguiendo una estrategia simple en la que se generan y evalúan todas las combinaciones de n elementos cogidos de r en r (siendo r el número de árboles a cortar, $1 \leq r \leq n-1$), se calcula la longitud de la valla necesaria para cerrar los n-r árboles restantes y se comprueba que esta sea una solución factible si la madera de los r árboles cortados es suficiente para construir la valla. Para cada solución factible se evalúa si ésta minimiza o no el valor de los árboles cortados de la mejor solución encontrada hasta el momento con objeto de guardarla o descartarla, según sea el caso.

La generación de combinaciones de elementos es uno de esos ejemplos clásicos que se resuelven mediante un algoritmo recursivo. Para calcular la longitud de la valla es necesario calcular el convex hull (ver Figura 1) de un conjunto de puntos Q (que se define como el menor polígono convexo P para el que cada punto de Q está en un lado del polígono P o en su interior). En [1][2] se describen diferentes algoritmos para calcular convex hulls.

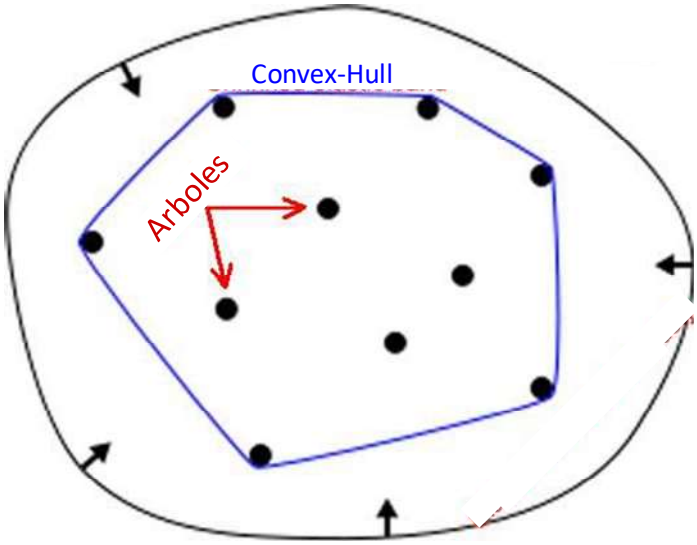

Figura 1. Convex-Hull para calcular la cerca de árboles.

En esta primera versión no se utiliza mecanismos de sincronización, de esta forma hay que diseñar la aplicación concurrente de forma que los diferentes hilos de ejecución puedan funcionar de forma independiente, sin comunicación ni sincronización.

Para probar y depurar el programa debéis preparar algunos ficheros de ejemplo sencillos con pocos árboles (n < 8). Junto con estos ficheros de pequeño tamaño, se debe preparar también algún ejemplo mayor que requiera un mayor tiempo de procesamiento secuencial para evaluar la escalabilidad de solución propuesta.

El programa concurrente puede recibir 3 parámetros, el fichero de entrada con la especificación del bosque de árboles, el número de threads que hay que utilizar para resolver el problema de optimación y el fichero de salida en donde se escribirá solución encontrada (este parámetro es opcional, si no se especifica la solución se escribe en el fichero Valla.res):

Sintaxis: CalcArboles <Fichero_Entrada> `<Max_Threads>` [<Fichero_Salida>]

- **Datos de entrada.**

Los datos de entrada del problema están almacenados en un archivo donde se describirá un bosque hipotético. La primera línea contiene un número entero n, $2 \leq n \leq 32$, que corresponde al número de árboles en el bosque. Los árboles se identifican mediante un valor entero correlativo de 1 a n. Cada una de las siguientes n líneas del fichero contiene 4 enteros xi, yi, vi, li que describen un árbol en particular. La posición del árbol en el plano está definida por (xi, yi), su valor es vi y la longitud de la valla que se puede construir con la madera de ese árbol es li. Los valores de vi y li están entre 0 y 10000.

- **Datos de salida.**

El programa debe calcular el subconjunto de árboles que, usando su madera, permite encerrar a los restantes árboles dentro de una única valla. El objetivo es lograr encontrar el subconjunto que requiere un menor coste. Si existe más de un subconjunto con el coste mínimo, hay que escoger el que tenga el menor número de árboles. Por simplicidad, se considerará que los árboles tienen diámetro cero.

La salida del programa debe identificar cada uno de los árboles a cortar, la longitud de la madera sobrante, el valor del conjunto de árboles cortados y el valor de los árboles restantes.

Ejemplo 1

Entrada	Salida
6 0 0 8 3 1 4 3 2 2 1 7 1 4 1 2 3 3 5 4 6 2 3 9 8	Cortar árboles: 2 4 5 Madera sobrante: 3.16 Valor árboles cortados: 9 Valor árboles restantes: 24

Ejemplo 2

Entrada	Salida
3 3 0 10 2 5 5 20 25 7 -3 30 32	Cortar árboles: 2 Madera sobrante: 15.00 Valor árboles cortados: 20 Valor árboles restantes: 40

Funciones involucradas.

En la práctica podéis utilizar las siguientes funciones de c, entre otras:

- `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);`

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**, cuando se procese ficheros grandes (*EjecuciónEjemploGordo.txt*).

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 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, partiendo de una nota base de 5:

- 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 una correcta identación
  - Descomposición Funcional (Código modular y estructurado)
  - Control de errores.
  - Test unitarios para verificar la correcta ejecución de la lógica de la aplicación.

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

Versiones Secuenciales

Junto con el enunciado se os proporciona una versión secuencial en C para resolver el problema descrito.

- Compilar CalcArboles.c
  - gcc CalcArboles.c –o CalcArboles –I. –lm
- Ejecutar con ejemplo pequeño (generando resultados en Ejemplo1.res)
  - ./CalcArboles ./ConjuntoPruebas/Ejemplo1.dat Ejemplo1.res
- Ejecutar con ejemplo pequeño (generando resultados en EjemploGordo.res)
  - ./CalcArboles ./ConjuntoPruebas/EjemploGordo.dat EjemploGordo.res

Referencias

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "**Introduction to Algorithms**", Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0262032937.

[2] "**Convex Hull**", http://softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm

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: CalcArbolesConcurrent.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 Palotes" utilizará el nombre de fichero: `PiruloPalotesPRA1.tgz`

Fecha de entrega

Entrega a través de Sakai el 16 de octubre, entrega presencial en la clase de grupo de laboratorio durante la semana del 28 de octubre al 31 de noviembre de 2019.