



Presentación

Objetivos

Esta práctica tiene como objetivo el diseño e implementación de una aplicación concurrente multi-hilo utilizando los hilos y los mecanismos de sincronización de pthreads. 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 y sincronización de POSIX (pthreads) que resuelva el problema al que se enfrentó el mago de nuestra historia.

Indicaciones resolución problema.

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.

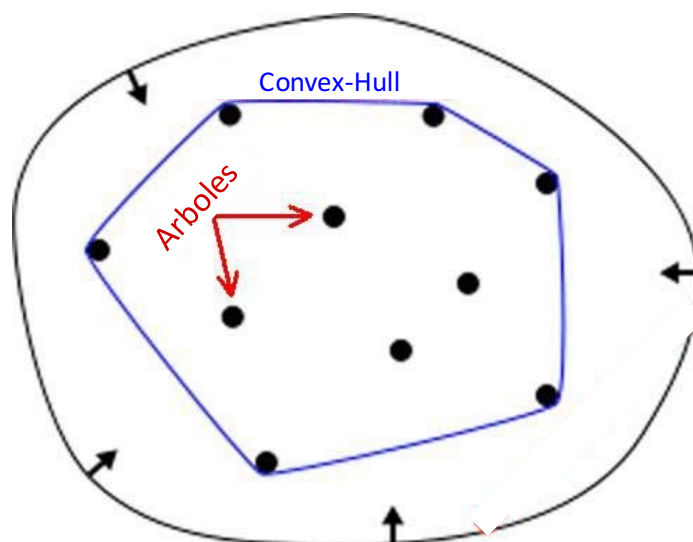


Figura 1. Convex-Hull para calcular la cerca de árboles.

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 optimización 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 x_i , y_i , v_i , l_i que describen un árbol en particular. La posición del árbol en el plano está definida por (x_i, y_i) , su valor es v_i y la longitud de la valla que se puede construir con la madera de ese árbol es l_i . Los valores de v_i y l_i 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.



SISTEMAS CONCURRENTES Y PARALELOS

PRÀCTICA 3

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

Sincronización

En la aplicación se deberá implementar las siguientes sincronizaciones:

- **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.

Es obligatorio el cálculo de un óptimo parcial global compartido por todos los hilos.

- **Espera resultado final.** No se permite 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 calcular una serie de estadísticas locales a cada objeto/hilo y las globales durante cada S combinaciones evaluadas por cada hilo:

- Número de Combinaciones evaluadas.
- Número de combinaciones válidas.
- Número de combinaciones no válidas.
- Mejor combinación válida (desde el punto de vista del coste de los árboles)
- Peor combinación válida (desde el punto de vista del coste de los árboles)
- Mejor combinación válida (desde el punto de vista del número de árboles talados)
- Peor combinación válida (desde el punto de vista del número de árboles talados)

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



SISTEMAS CONCURRENTES Y PARALELOS

PRÀCTICA 3

Al implementar las sincronizaciones se tienen que utilizar todos los mecanismos de sincronización que hemos visto en clase. Hay que utilizar: Semáforos, *Mutex*, y variables de condición.



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);`
- `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_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);`



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 la práctica, partiendo de una nota base de 10 (9 puntos implementación y 1 punto el informe):

- Aspectos que se evalúan:
 - Ejecución Secuencial (Susp)
 - Ejecución no determinista ([-1.0p,-5.0p])
 - Condiciones Carrera (-2.0p)
 - Prestaciones (-1.0p, +0.5p)
 - Número incorrecto de hilos (-0.5p)
 - Control Errores: cancelación hilos (-0.5p)
 - Descomposición desbalanceada (Versión óptima) (-0.5p)
 - Descomposición parcial: falta asignar parte del trabajo (-0.5p)
 - Optimo parcial global (-0.5p)
 - Espera Activa (-1.0p)
 - Estadísticas (-0.75p)
 - Esperar Resultado final (-0.5p)
 - No utilizar Mutex (-0.75p)
 - No utilizar Variables de condición (-0.75p)
 - No utilizar Semáforos (-0.75p)
- Evaluación del informe de la práctica:
 - Sin informe (-1.0p)
 - Informe con incoherencias en los tiempos sin justificar (-0.25p)
 - Informe: análisis poco riguroso (-0.25p)
 - No se explica el diseño de la solución propuesta (-0.25p)
 - Descripción de los problemas encontrados y como se han solventado (-0.25p)
- Estilo de codificación:
 - El código no está correctamente comentado (-0.25p)
 - No utilizar una correcta tabulación y organización del código (-0.25p)
 - Sin descomposición funcional (el código no es modular, ni está estructurado) (-0.25p)
 - Sin control de errores (-0.25p)

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





versión Secuencial

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 pracl.tgz fichero1 fichero2 ...
```



SISTEMAS CONCURRENTES Y PARALELOS

PRÀCTICA 3

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 **14 de diciembre a las 23:55h**, entrega presencial la semana del 15 al 18 de diciembre de 2025.

A partir del sábado 13 de diciembre no se admiten consultas sobre la práctica.

