Hugo Cisneros Amengual
Alicia García Yuen

COMPUTACION ALTO RENDIMIENTO
LAB POSIX-THREADS

Control version and revisions.

Ed.	Date	Change	Done by	Change
0.1	19/10/2024	DOCUMENTATION	HC. AG	N/A
1	20/10/2024	Revision	HC. AG	N/A

*N/A -> Does not apply

AG -> AG

HC -> Hugo Cisneros

Índice

1. Introducción

Este documento contiene la explicación y modo de funcionamiento de tres programas con hilos, los tres programas pertenecen al laboratorio LAB-Posix_Threads. La práctica se compone de tres ejercicios principales programados en C con hilos que se compararan cada uno con su versión y ejecución tradicional sin hilos. Se incluirá su tiempo de ejecución y uso de memoria como información de referencia a la hora de comparar.

Se mostrará el enunciado y su ejemplo, una breve explicación del código (la principal esta comentada en los programas) y una conclusión final para cada uno.

Pinchando aqui, es donde se encuentra el link a nuestro github, donde se ve el código entero subido.

2. Primer Ejercicio: Imprimir con hilos

ENUNCIADO

Escribe un programa en el que el hilo principal solicite un número y dos cadenas de texto. Crea dos tareas (hilos) que impriman en la pantalla el ID de la tarea y las cadenas de texto un número de veces igual al número introducido en el hilo principal. El hilo principal debe esperar a que terminen las tareas y luego repetir toda la operación.

Ejemplo aclaratorio previo:

Input a number : 2
Input first string: What happen?
Input second string: Nothing happen
thread (1026): 1 What happen?
thread (1026): 2 What happen?
thread (2051): 1 Nothing happen
thread (2051): 2 Nothing happen
Input a number:

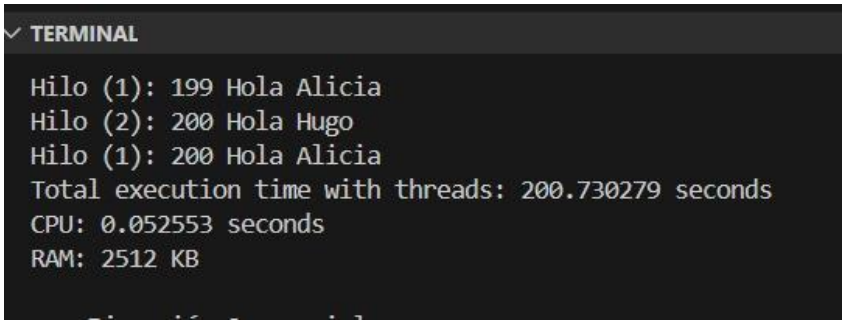
Explicación código con hilos:

Comenzamos pidiéndole al usuario el índice que será el número de veces que imprimiremos los strings y pidiendo los dos strings para la ejecución. Luego creamos dos hilos, cada uno imprimirá una de las cadenas de texto tantas veces como el número proporcionado al principio. Cada hilo ejecuta la función **print_string** que imprime cada cadena según el hilo y un contador que muestra cuántas veces se ha repetido. Los hilos al funcionar de forma paralela esperan a que el otro termine antes el repetir el proceso y así continuamente hasta acabar el índice indicado.

Comparación y conclusión código sin hilos y con hilos:

```
Secuencial (2): 195 Hola Hugo
Secuencial (2): 196 Hola Hugo
Secuencial (2): 197 Hola Hugo
Secuencial (2): 198 Hola Hugo
Secuencial (2): 199 Hola Hugo
Secuencial (2): 200 Hola Hugo
Tiempo de ejecución sin hilos: 401.491358 segundos
CPU: 0.132503 seconds
RAM: 2512 KB
```

Como podemos observar, el código en secuencial y con tiempos de espera (simulando trabajo) tarda 400 segundos.



En diferencia al anterior, el código con hilos y con cargas de tiempo tarda la mitad (200 segundos menos).

Esta diferencia se debe a que mientras que el código sin hilos tarda un segundo en imprimir una línea, mientras que con hilos en ese mismo segundo imprime dos.

3. Segundo Ejercicio: Sumar dos matrices

ENUNCIADO

Escribe un programa para sumar dos matrices de 3x3. Solicita los elementos de las matrices en el programa principal y crea 3 tareas, cada una sumará una fila con otra. Luego, imprime la matriz resultante en el programa principal.

Ejemplo aclaratorio previo:

$$(A+B)[i,j] = A[i,j] + B[i,j]$$

$$\begin{bmatrix}1 & 3 & 2\\1 & 0 & 0\\1 & 2 & 2\end{bmatrix} + \begin{bmatrix}1 & 0 & 5\\7 & 5 & 0\\2 & 1 & 1\end{bmatrix} = \begin{bmatrix}1+1 & 3+0 & 2+5\\1+7 & 0+5 & 0+0\\1+2 & 2+1 & 2+1\end{bmatrix} = \begin{bmatrix}2 & 3 & 7\\8 & 5 & 0\\3 & 3 & 3\end{bmatrix}$$

Explicación código con hilos:

En este programa almacenaremos la información que contienen los hilos en una estructura igual que el ejercicio anterior. En **ThreadData** almacenamos la información necesaria para que el hilo pueda sumar según la fila y las matrices que involucra. Las matrices **mat1** y **mat2** serán las que se suman y la matriz resultado será **result.** Creamos tres hilos y cada uno de ellos sumará una fila de las matrices ya que son 3*3. Cada hilo ejecutará la función **sumar_fila** donde se suman los elementos de la fila especificada de las matrices **mat1 y mat2** según el hilo y se almacena en **result.** Añadimos una función **gettimeofday** para saber el tiempo que tartda en ejecutarse y una función memory_use para saber el consumo de memoria por el sistema.

Comparación y conclusión código sin hilos y con hilos:

```
6 7 8
Matriz resultante:
2 25 6
3 7 9
10 12 14
Tiempo de ejecución: 15 segundos
```

El código con hilos tarda 15 segundos, aunque se le descuenta tiempo de pedida de datos. En general tarda más ya que ha estado más tiempo creando códigos.

```
Resultado de la suma (A + B):
2 4 6
8 10 12
14 16 18

Tiempo de ejecución: 0.0000 segundos.
Memoria usada (RSS):
VmRSS:      1536 kB
```

Este código sin hilos tarda menos ya que la amtriz es menor y no tiene que crear hilos.

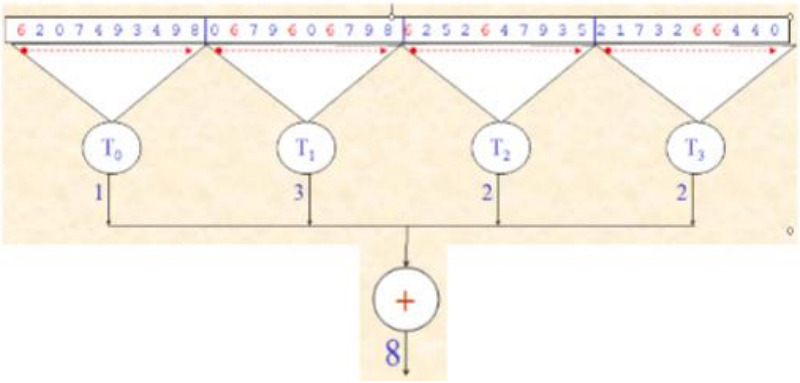
A la conclusión que llegamos es que, ya que la matriz tieen que ser 3x3 por lo que dice el enunciado, creemos que si se aumentase el tamaño de la amtriz, se vería una diferencia a favor del programa con hilos, ya que estos aumentan su eficacia en cuanto más columnas tenga la matriz.

4. Tercer Ejercicio: Encontrar elemento en vector

ENUNCIADO

Escribe un programa con hilos para encontrar el número de ocurrencias de un número determinado en un vector. Lee un vector de 20 elementos y divídelo en cuatro partes. Cada tarea debe encontrar las ocurrencias del número en su parte correspondiente. Imprime los resultados en el hilo principal. Supón que estamos buscando el número 6. Usa mutex para la exclusión mutua cuando las tareas estén utilizando el vector.

Ejemplo aclaratorio previo:



Explicación código con hilos:

En este programa como valor del tamaño del vector asignaremos 20. Dividiremos el vector en 4 partes y por último el número buscado estará en taget_number. Comenzamos pidiendole al usuario los 20 valores del vector y almacenaremos toda la información para el hilo en ThreadData, luego cada hilo llama a la función contar_ocurrencias y cuenta cuantas veces aparece el número objetivo en la parte del vector asignada. Esperamos a que todos los hilos acaben con la función pthread_join y finalmente sumamos los resultados de cada hilo y se imprime el número total final.

Utilizaremos clock para medir el tiempo que tardan los hilos.

Comparación y conclusión código sin hilos y con hilos:

```
5
6
7
89

2
2
2
Introduce el número a buscar: 2
Se han encontrado 2 en el array: 5
El tiempo de ejecución fue: 0.000821 segundos
```

```
TERMINAL
4
4
Ocurrencias de 4 en la parte 1: 1
Ocurrencias de 4 en la parte 2: 0
Ocurrencias de 4 en la parte 3: 4
Ocurrencias de 4 en la parte 4: 5

Total de ocurrencias de 4 en el vector: 10
El tiempo de ejecución fue: 0.000032 segundos
```

El código con hilos tarda más que sin hilos ya que el tamaño es muy pequeño. Como queremos respetar el enunciado, el tamaño se fija a 20. Sin embargo, creemos que si el tamaño fuese considerablemente más grande (200), los hilos irían ás rápido.