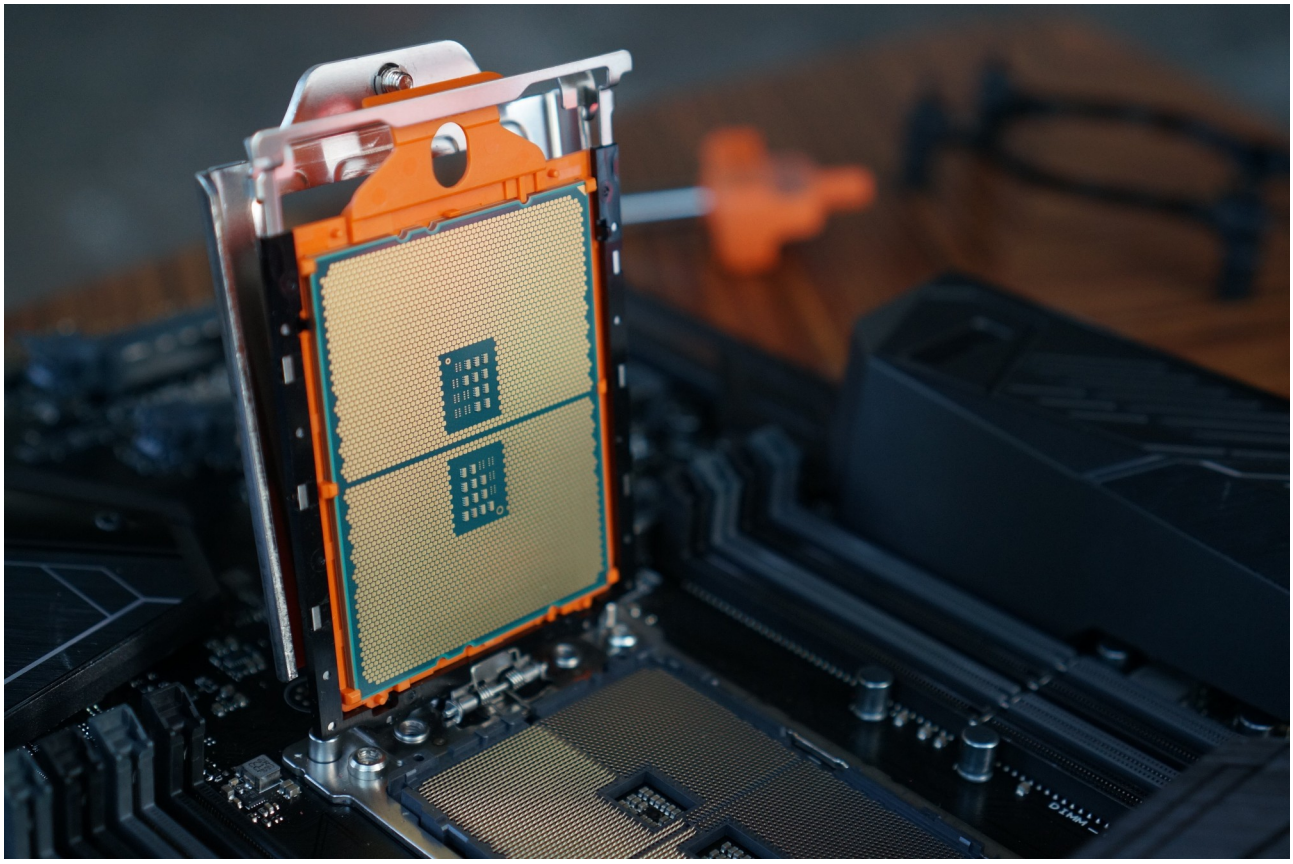
Introducción a trabajo con hilos

Emilio Cabo Gomis



Índice

PARA SEGUIR ESTE TRABAJO ES NECESARIO SABER ALGO DE C

¿Porqué los programadores toman tanto tiempo en programar programas que empleen procesos que tomen ventaja de la ejecución multihilo?

La explicación es fácil; la práctica no lo es tanto.

Diferentes tipos de ejecución del programa:

Tendremos que diferencias entre ejecución concurrente, simultánea y secuencial.

- Secuencialmente. Algo se realiza de manera secuencial cuándo se realiza una tarea detrás de otra.
- Un proceso se realiza de manera simultánea cuando se ejecuta al mismo tiempo pero en distintas ubicaciones.
- Un proceso se realiza de manera concurrente cuándo se realiza en distinto tiempo en el mismo sitio.

Por ejemplo un programa concurrente.
Si queremos multiplicar 6 * 3 y luego sumar 3 + 2 podemos realizarlo de la siguiente manera

1. Declaramos e inicializamos tres variables. Dos que contengan los respectivos operandos y otra que contenga los respectivos resultados.
2. Realizamos las operaciones
3. Imprimimos el resultado en pantalla.
4. Sobrescribimos los valores de los operandos de la primera operación e introducimos los de la segunda operación.

Ejecución concurrente

Pongamos que tenemos un programa en el que se realicen dos operaciones.
Vamos a programarlo y mostrar el resultado.

```
#include <stdio.h>

int main () {
   int x = 6;
   int y = 3;
   int z, inumOp = 0;

   z =  x * y;
   printf("====================\n\tOperacion %d\n====================\n",
inumOp + 1);
   printf("Resultado de la operación %d= %d\n", inumOp + 1, z);
   printf("Direccion de la variable x: %p\n Direccion de la variable y: %p\n", &x, &y);

   inumOp++;
   //redefinicion de las variables
   x = 3;
   y = 5;
   z =  x + y;
```

```
    printf("====================\n\tOperacion %d\n====================\n",
inumOp + 1);
    printf("Resultado de la operación %d= %d\n", inumOp + 1, z);
    printf("Direccion de la variable x: %p\n Direccion de la variable y: %p\n", &x, &y);
    return 1;
}
```

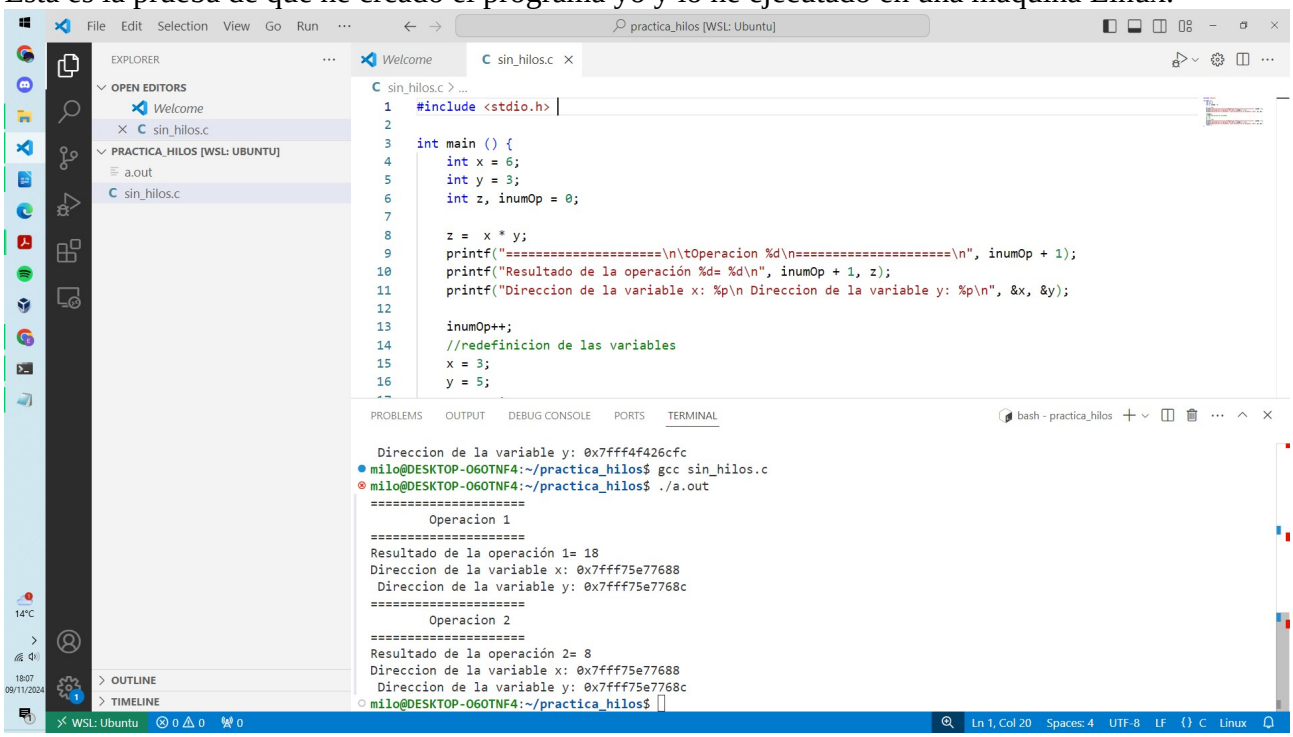
En este programa hemos tenido que emplear. Unicamente tres variables para dos operaciones.

Esto lo hemos hecho sobrescribiendo el valor de cada variable a medida que las instrucciones se han ido ejecutando.

Este ha sido el resultado de la ejecución del programa

```
====================
      Operacion 1
====================
Resultado de la operación 1= 18
Direccion de la variable x: 0x7fff75e77688
 Direccion de la variable y: 0x7fff75e7768c
====================
      Operacion 2
====================
Resultado de la operación 2= 8
Direccion de la variable x: 0x7fff75e77688
Direccion de la variable y: 0x7fff75e7768c
```

Esta es la prueba de que he creado el programa yo y lo he ejecutado en una máquina Linux.



Es una conexión mediante WSL

¿Porqué querríamos ejecutar estas instrucciones simultáneamente?

Porque si bien si tenemos un tiempo T al llegar en

t1
t2
t3
.

.
.
TN

Tendremos el mismo resultado
Dado el mismo tiempo TN ejecutaremos habremos hecho más operaciones que de otra manera.

Ejecución simultánea

Para realizar una ejecución simultánea tendremos que emplear algo llamado hilos.

Para ello emplearemos la librería <pthread.h>

Y este es el código resultante

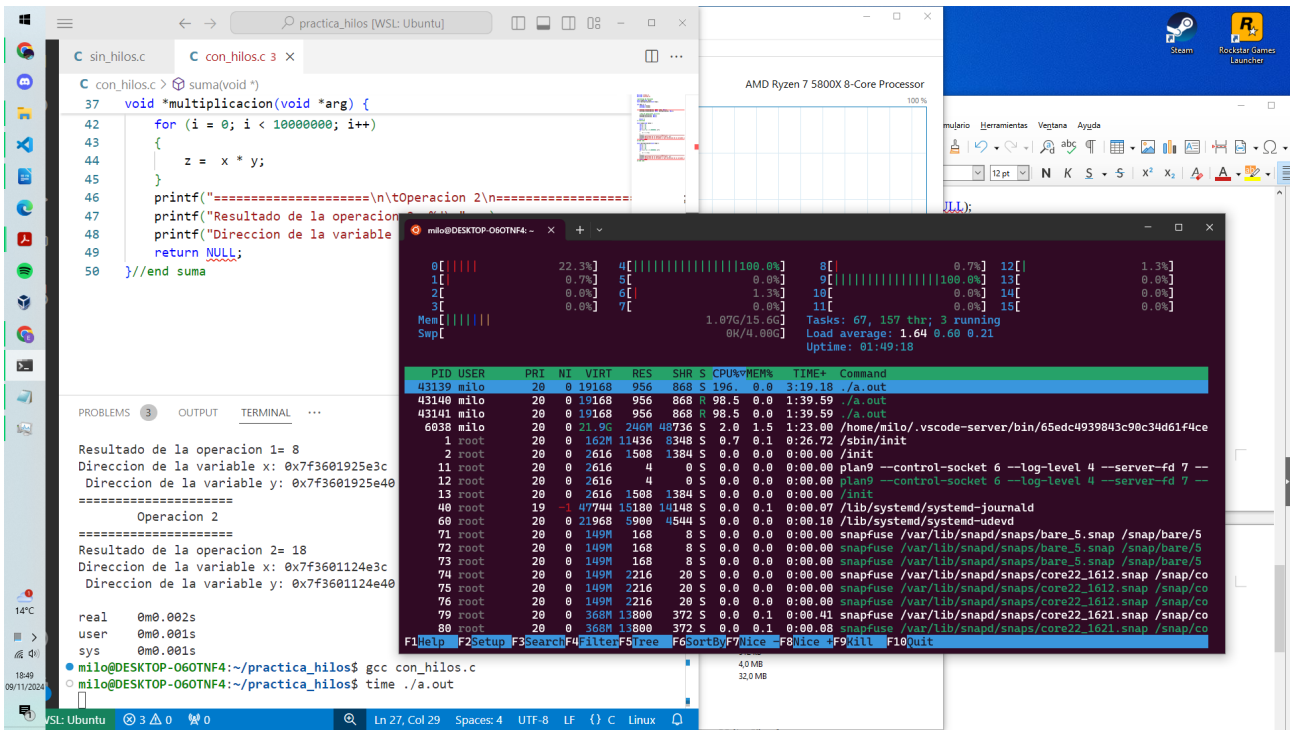
```
#include <stdio.h>
#include <pthread.h>

//prototipo de funciones
void *suma(void *arg);
void *multiplicacion(void *arg);

int main () {
   pthread_t hilo1;
   pthread_t hilo2;

   pthread_create(&hilo1, NULL, suma, NULL);
   pthread_create(&hilo2, NULL, multiplicacion, NULL);

   //Fin de operaciones con hilos
   pthread_join(hilo1, NULL);
   pthread_join(hilo2, NULL);

  return 0;
} //end main

void *suma(void *arg) {
   int x = 3;
   int y = 5;
   int z = 0;
```

```
    int i = 0;
    for (i = 0; i < 2000000000; i++)
    {
        z =  x + y;
    }
    printf("====================\n\tOperacion 1\n====================\n");
    printf("Resultado de la operacion 1= %d\n", z);
    printf("Direccion de la variable x: %p\n Direccion de la variable y: %p\n", &x, &y);
    return NULL;
}//end suma

void *multiplicacion(void *arg) {
    int x = 6;
    int y = 3;
    int z = 0;
    int i = 0;
    for (i = 0; i < 2000000000; i++)
    {
        z =  x * y;
    }
    printf("====================\n\tOperacion 2\n====================\n");
    printf("Resultado de la operacion 2= %d\n", z);
    printf("Direccion de la variable x: %p\n Direccion de la variable y: %p\n", &x, &y);
    return NULL;
}//end suma
```

En el momento de la ejecución si vemos el programa HTOP

En mi caso estoy empleando un procesador Ryzen 5800X con 8 núcleos y 16 hilos.

Debido a que estas operaciones estan empleando 2 hilos y tardan en ejecutarse podremos ver que hilos estan empleándose para la ejecución.



En mi caso se están empleando el 4 y el 9 para ejecutar este programa

Este es el output de mi programa

==========

Operacion 1

==========

Resultado de la operacion 1= 8

Direccion de la variable x: 0x7fab54b84e38

Direccion de la variable y: 0x7fab54b84e3c

==========

Operacion 2

==========

Resultado de la operacion 2= 18

Direccion de la variable x: 0x7fab54383e38

Direccion de la variable y: 0x7fab54383e3c

Debido a que he empleado variables distintas es normal que la dirección de memoria que he empleado cambie.

¿Esto ha mejorado el rendimiento de nuestro pequeño programa?

La respuesta es un rotundo si.

Al emplear hilos hemos reducido el tiempo de ejecución de nuestro programa a la mitad.

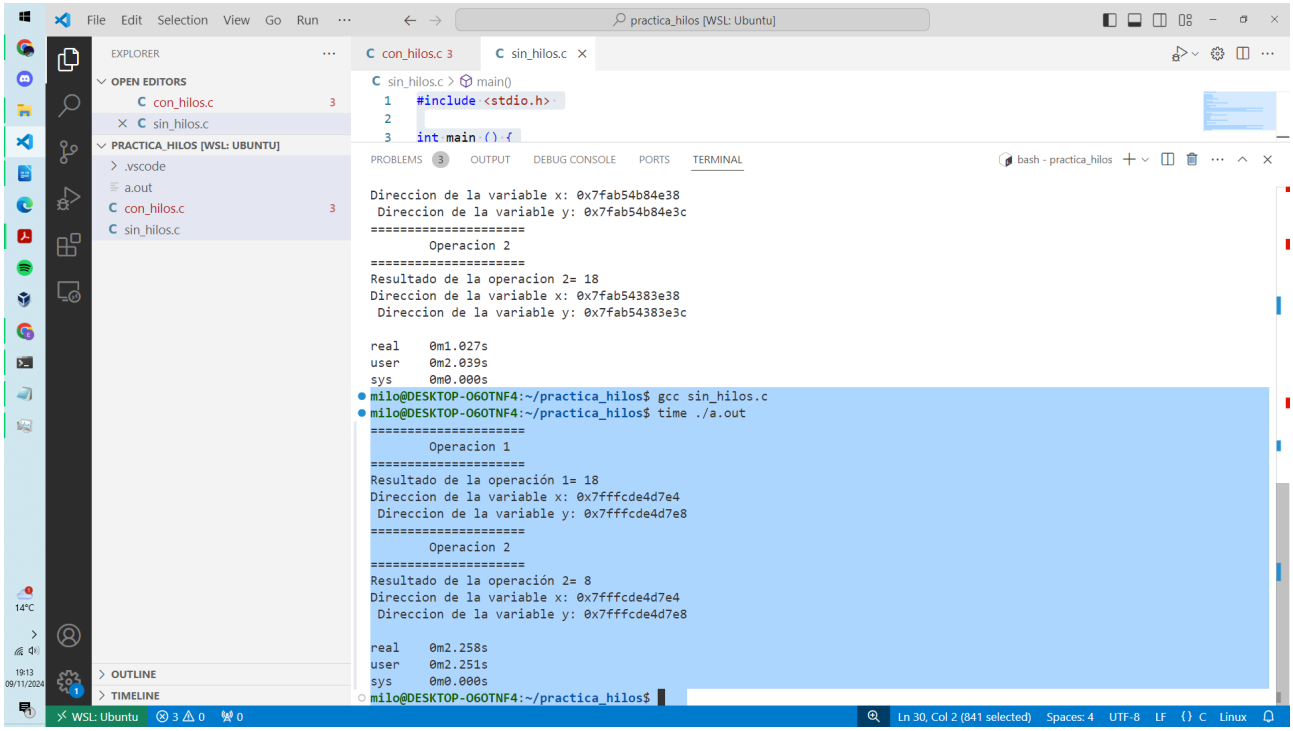
Para ello primero realizamos unas modificaciones en el programa para que tenga loops al igual que el programa del principio. Sino el programa no tendrá la complejidad suficiente.

Modificaciones programa sin hilos

```
#include <stdio.h>

int main () {
   int x = 6;
   int y = 3;
   int z, inumOp = 0;

   int i = 0;
   for (i = 0; i < 2000000000; i++)
   {
      z =  x * y;
   }
   printf("====================\n\tOperacion %d\n====================\n",
inumOp + 1);
   printf("Resultado de la operación %d= %d\n", inumOp + 1, z);
   printf("Direccion de la variable x: %p\n Direccion de la variable y: %p\n", &x, &y);

   inumOp++;
   //redefinicion de las variables
   x = 3;
   y = 5;
   i = 0;
   for (i = 0; i < 2000000000; i++)
   {
      z =  x + y;
   }
   printf("====================\n\tOperacion %d\n====================\n",
inumOp + 1);
   printf("Resultado de la operación %d= %d\n", inumOp + 1, z);
   printf("Direccion de la variable x: %p\n Direccion de la variable y: %p\n", &x, &y);
   return 0;
}
```

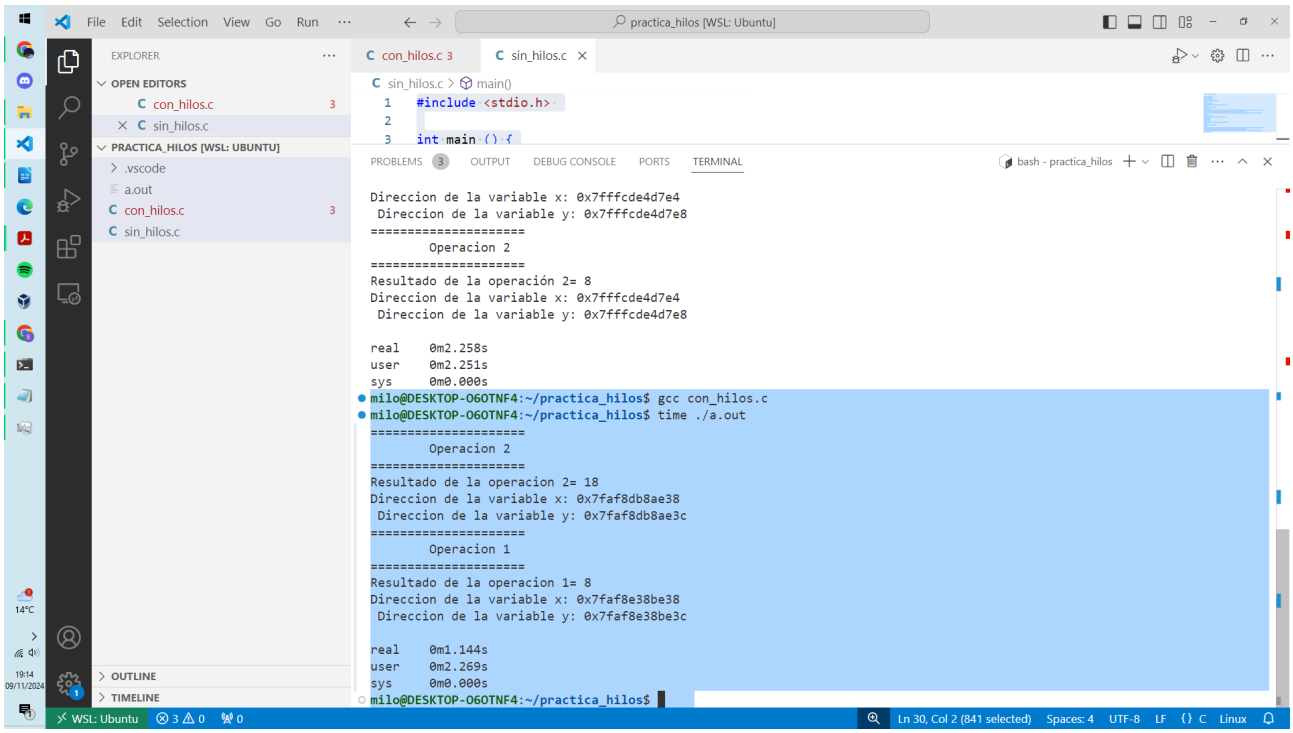
Compilamos y ejecutamos programa sin hilos

Empleamos la función time para ver cuanto tardamos.



Compilamos y ejecutamos programa con hilos

Realizamos lo mismo para el anterior programa.



El tiempo de ejecución del programa se ha dividido por la **mitad**. WOW