

# Informe Taller 2

Taller de Sistemas Operativos  
Escuela de Ingeniería Informática

Fabián Rozas Alfaro

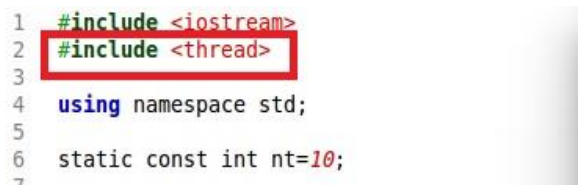
[fabian.rozas@alumnos.uv.cl](mailto:fabian.rozas@alumnos.uv.cl)

**Resumen.** Como objetivo principal del trabajo es realizar una implementación de un programa con threads POSIX que llene un arreglo de números enteros y luego los suma. Estas tareas se deben realizar en forma paralela.

## 1. Introducción

Un thread o hilo es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada al mismo tiempo que otra por un sistema operativo. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son conocidos en conjunto como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar estos recursos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente [1]. El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado.

En el caso de C++, este desconoce la existencia de hilos de ejecución y deben ser creados mediante llamadas a biblioteca especiales que dependen del sistema operativo en el que estos lenguajes están siendo utilizados. En esta ocasión, ocupamos Linux y el llamado a la biblioteca se muestra en la Figura 1.



```
1 #include <iostream>
2 #include <thread>
3
4 using namespace std;
5
6 static const int nt=10;
7
```

Figura 1

También pueden ser llamados mediante la biblioteca `#include <global.h>`. Esta es creada como `global.h` y abarca una serie de llamadas y/o funciones que son accesibles en todas partes.

POSIX significa Portable Operating System Interface (por Linux). Es un estándar orientado a facilitar la creación de aplicaciones confiables y portables. La mayoría de las versiones populares de UNIX cumplen este estándar en gran medida [2]. La biblioteca para el manejo de hilos en POSIX es `#include <pthread.h>` como se muestra en la Figura 2.

```

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

void *thread_func( void *ptr ){
    char *message = (char *) ptr;
    printf("%s \n", message);
}

```

Figura 2

La Figura 3 en la parte izquierda muestra un proceso en Unix, y en su parte derecha se muestran dos hilos en un proceso Unix.

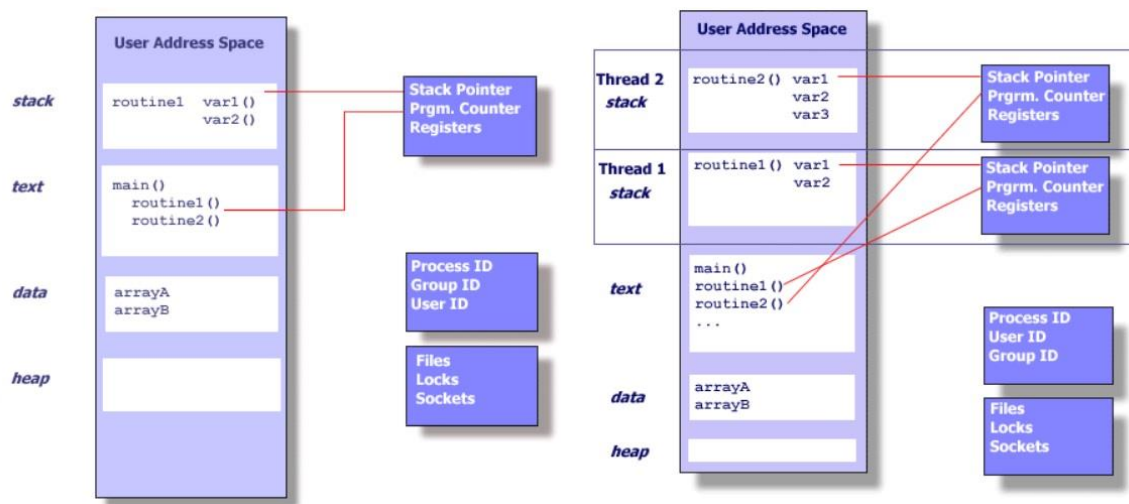


Figura 3.

## 2. Descripción del problema

El problema solicitado en el Taller 02, se basa en crear un programa que esté compuesto de dos módulos. Uno de ellos debe llenar un arreglo de números enteros aleatorios del tipo **uint32\_t** en forma paralela y otro que sume el contenido del arreglo en forma paralela. Junto con esto se deben realizar pruebas de desempeño que generen datos que permitan visualizar el comportamiento del tiempo de ejecución de ambas módulos dependiendo del tamaño del problema y de la cantidad de threads utilizados.

Se utilizó la biblioteca descrita en la Figura 2 (`#include <pthread.h>`) para poder trabajar con hilos en POSIX. Junto con esto se creó una biblioteca llamada **global.h**, la cual tendrá como objetivo llamar a las bibliotecas, secuencias, funciones u objetos que necesitaremos para trabajar con threads y demás funciones. También se crea una clase llamada **checkArgs**, la cual define los parámetros que se utilizarán para generar el arreglo y sus límites.

La forma que se deben generar números enteros aleatorios se muestran en la Figura 4.

```
../sumArray -N <nro> -t <nro> -l <nro> -L <nro>
```

-N : tamaño del arreglo  
-t : número de threads  
-l : límite inferior  
-L : límite superior

Figura 4.

### 3. Diseño de la solución

Se comenzó analizando el problema descrito anteriormente para diseñar un buen esquema de la solución y con el fin de llevar a cabo un orden en la creación de cada módulo.

En las Figuras 6 y 7 se muestran los diseños de solución que utilicé para implementar el problema.

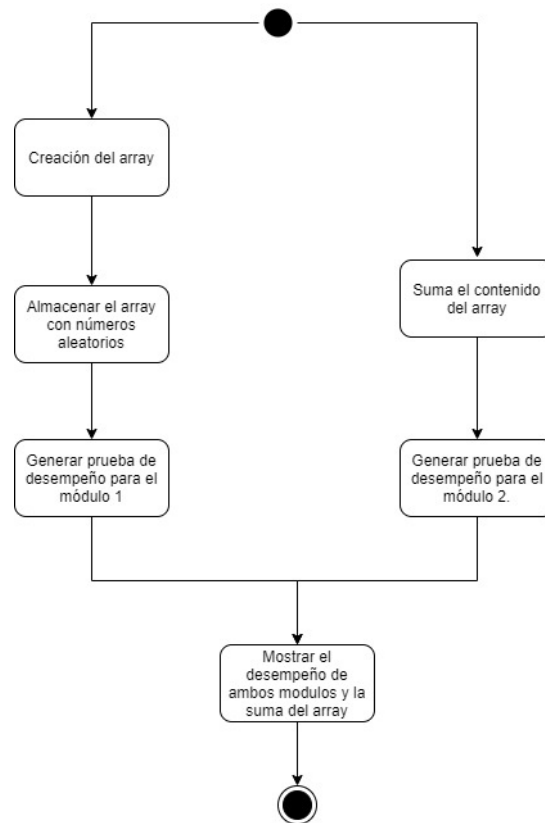


Figura 6. Diagrama de actividades que muestra la ejecución de dos hilos en forma paralela.

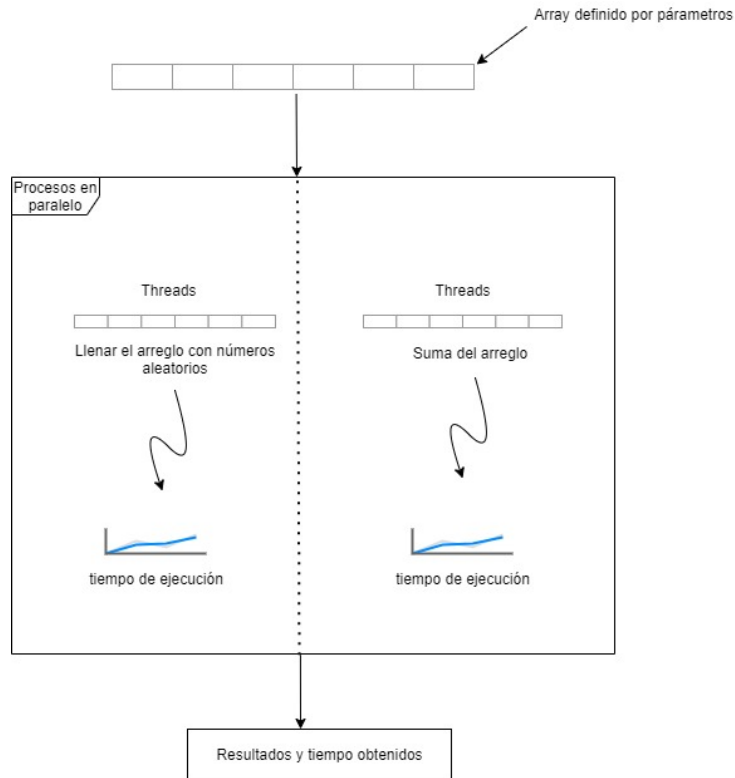


Figura 7.

A través de los diseños descritos, se pueden extraer algunos puntos fundamentales:

- Se debe de crear un arreglo y ser llenado con números aleatorios definidos por los parámetros de entrada.
- Calcular el tiempo de ejecución para el módulo que llena el arreglo.
- De forma paralela se deben de sumar los números que van siendo almacenados en el arreglo.
- Calcular el tiempo de ejecución del módulo de sumas del arreglo.
- Mostrar resultados de la suma, el tiempo de ejecución y rendimiento.

## 4. Implementación y Resultados

### • Llenado del arreglo

Se utilizó la función **fillArray** para poder llenar de forma aleatoria como se muestra en la Figura 8.

```
uint64_t* array = nullptr;

void fillArray(size_t beginIdx, size_t endIdx, size_t limInfer, size_t limSuper)
{
    std::random_device device;
    std::mt19937 rng(device());
    std::uniform_int_distribution<> unif(limInfer, limSuper);

    for(size_t i = beginIdx; i < endIdx; i++){
        array[i] = unif(rng);
    }
}
```

Figura 8.

- **Suma paralela**

La función para realizar las sumas almacenadas en el arreglo se muestra en la Figura 9.

```
uint32_t sumaArray = 0;

void sumaArrayP(uint32_t &sumaArray, uint32_t beginIdx, uint32_t endIdx){
    sumaArray=0;
    for(uint32_t i = beginIdx; i < endIdx; i++){
        sumaArray += array[i];
    }
}
```

Figura 9.

- **Calculo del tiempo de ejecución**

Para realizar las pruebas de desempeño y obtener el tiempo de cada proceso en la Figura 5 se muestra un ejemplo.

```
auto end = std::chrono::high_resolution_clock::now();
auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);
auto tiempo_secuencial = elapsed.count();
```

Figura 10. Tiempo de ejecución de un módulo

Ahora para observar los resultados y obtener el cálculo de la suma junto con el rendimiento de los módulos, se ejecutó el programa con algunos parámetros distintos con la finalidad de comparar sus resultados. En la Figura 11 se muestran los resultados obtenidos.

```
fabian@LAPTOP-OL75E7B2:/mnt/c/users/fabia/desktop/src$ make clean
fabian@LAPTOP-OL75E7B2:/mnt/c/users/fabia/desktop/src$ make
g++ -c -o objs/main.o main.cc -std=c++17 -Wall -O1 -I../include -I../include
g++ -o ../sumArray objs/main.o -std=c++17 -Wall -O1 -lpthread
fabian@LAPTOP-OL75E7B2:/mnt/c/users/fabia/desktop/src$ ./sumArray -N 1000000 -t 2 -l 10 -L 50
Tamaño del array: 1000000
Threads : 2
Limite inferior array: 10
Limite superior array: 50
Suma total en paralelo: 17781946
==== Tiempos de las sumas ====
Tiempo de la suma secuencial: 1[ms]
Tiempo de la suma en paralelo: 1[ms]
SpeedUp : 1
==== Tiempos llenado del array ====
Tiempo secuencial: 31[ms]
Tiempo paralelo: 17[ms]
SpeedUp : 1.82353
fabian@LAPTOP-OL75E7B2:/mnt/c/users/fabia/desktop/src$ ./sumArray -N 1000000 -t 5 -l 1 -L 500
Tamaño del array: 1000000
Threads : 5
Limite inferior array: 1
Limite superior array: 500
Suma total en paralelo: 114085547
==== Tiempos de las sumas ====
Tiempo de la suma secuencial: 1[ms]
Tiempo de la suma en paralelo: 2[ms]
SpeedUp : 0.5
==== Tiempos llenado del array ====
Tiempo secuencial: 30[ms]
Tiempo paralelo: 16[ms]
SpeedUp : 1.875
fabian@LAPTOP-OL75E7B2:/mnt/c/users/fabia/desktop/src$ ./sumArray -N 1000000 -t 8 -l 1 -L 1900
Tamaño del array: 1000000
Threads : 8
Limite inferior array: 1
Limite superior array: 1900
Suma total en paralelo: 177263066
==== Tiempos de las sumas ====
Tiempo de la suma secuencial: 1[ms]
Tiempo de la suma en paralelo: 2[ms]
SpeedUp : 0.5
==== Tiempos llenado del array ====
Tiempo secuencial: 31[ms]
Tiempo paralelo: 14[ms]
SpeedUp : 2.21429
fabian@LAPTOP-OL75E7B2:/mnt/c/users/fabia/desktop/src$
```

Figura 11. Resultados obtenidos.

## **5. Conclusión**

Se puede observar que al utilizar más threads en los procesos, el tiempo del módulo que realizó la suma en paralelo es un poco más rápido que al realizarlo de forma secuencial. Cabe mencionar que se realizó la implementación bajo la ayuda de los ejemplos vistos en clases, por lo que surgieron varios errores que pudieron ser resueltos. Finalmente se pudo cumplir el objetivo del taller 02, pudiendo así mejorar la eficiencia en los procesos con la utilización de los pthreads.

## **6. Referencias**

- [1] [https://es.wikipedia.org/wiki/Hilo\\_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Hilo_(inform%C3%A1tica))
- [2] [http://profesores.elo.utfsm.cl/~agv/elo330/2s08/lectures/POSIX\\_Threads.html](http://profesores.elo.utfsm.cl/~agv/elo330/2s08/lectures/POSIX_Threads.html)