Informe técnico taller 02

Taller de Sistemas Operativos

Profesor: Gabriel Astudillo Muñoz

Alumno: Sebastián González Morales

Escuela de Ingeniería Civil Informática,
Universidad de Valparaíso
18 de junio de 2020

1 Introducción

En este informe se explicará el contexto y el diseño de un problema que se resolverá a través de programación paralela con el fin de dividir tareas en partes independientes para obtener resultados en menos tiempo.

El objetivo general es implementar un programa que llene un arreglo de números enteros y luego los sume. Ambas tareas se harán en forma paralela, implementadas con threads POSIX.

POSIX Threads, generalmente conocido como pthreads, es un modelo de ejecución que existe independientemente de un lenguaje, así como un modelo de ejecución paralela. Permite que un programa controle múltiples flujos de trabajo diferentes que se superponen en el tiempo. Cada flujo de trabajo se conoce como un subproceso, y la creación y el control sobre estos flujos se logra haciendo llamadas a la API POSIX Threads. POSIX Threads es una API definida por POSIX.1c estándar, extensiones de Threads (IEEE Std 1003.1c-1995) [1].

2 Problema

El problema está compuesto en dos módulos, el primero consiste en llenar un arreglo unidimensional de números aleatorios del tipo uint32_t y el segundo debe sumar el contenido del arreglo. Posteriormente se implementará y luego se realizarán pruebas de desempeño para ver el comportamiento del tiempo de ejecución de ambos módulos según el tamaño del arreglo, la cantidad de threads utilizados y el rango de números aleatorios. La implementación será en C++ versión 2017.

Tanto el tamaño del arreglo, la cantidad de threads utilizados y el rango de números aleatorios serán en forma dinámica, configurable a través de parámetros de entrada del programa.

2.1 Forma de uso:

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

Parámetros:

```
-N   : tamaño del arreglo.
-t   : número de threads.
-l   : límite inferior rango aleatorio.
-L   : límite superior rango aleatorio.
[-h] : muestra la ayuda de uso y termina.
```

2.2 Ejemplo de uso:

1) Crea un arreglo de 2000000 posiciones, con 5 threads. Los números enteros aleatorios están en el rango [20,60]

```
./sumArray -N 2000000 -t 5 -l 20 -L 60
```

2) Muestra la ayuda y termina

```
./sumArray -h
./sumArray
```

3 Diseño

3.1 Descripción general

Como se dijo anteriormente, el tamaño del arreglo, la cantidad de threads utilizados y el rango de números aleatorios serán en forma dinámica, configurable a través de parámetros de entrada del programa y es global al proceso. Esto quiere decir que es visible para el número de threas creados dentro de él. La solución de la primera parte del módulo se denomina "Etapa de llenado" (ver Figura 1). Cada thread conoce los índices de inicio y fin donde debe almacenar un número aleatorio.

A medida que se va llenando el arreglo global, la segunda parte del modulo ira sumando los números, esta parte tiene el nombre de "Etapa de suma"(ver Figura 2). En esa figura se utilizaron 2 threads y el arreglo estaba compuestos por números entre 0 y 50.

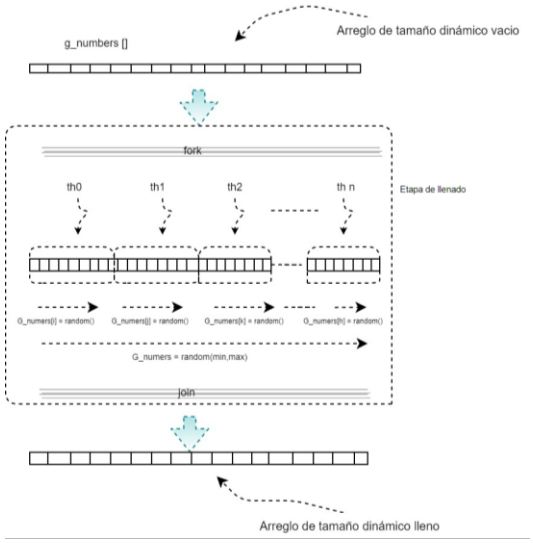

Figura 1

3.2 Etapa de llenado

El llenado del arreglo global **`g_numbers[]`**, se realizará a través de una función denominada **`fillArray`**, que tiene dos parámetros: uno que indica el índice de inicio (**`beginIdx`**) y otro para el índice de fin (**`endIdx`**). Esto permite que la función sea llamada en forma secuencial o parcial por los threads. Esta función tendrá un generador de número aleatorios (RNE, Random Number Engine) con dos parámetros de entrada (min,max) las cuales indicaran el rango mínimo y máximo de los números que almacenara él arreglo. Para las pruebas de funcionamiento, se probará con la función **`std::uniform_int_distribution<>`** disponible en la biblioteca **`<random>`**.

3.3 Estapa de suma

Esta etapa se realizará a través de una función llamada sumas parciales, la cual tendrá cuatro parámetros de entrada, el vector con los números a sumar, una variable llamada suma que ira almacenando la suma y los índices de inicio y fin de los números que irán sumando el thread.

Luego en la función principal se iran sumando las sumas parciales para obtener la suma total.

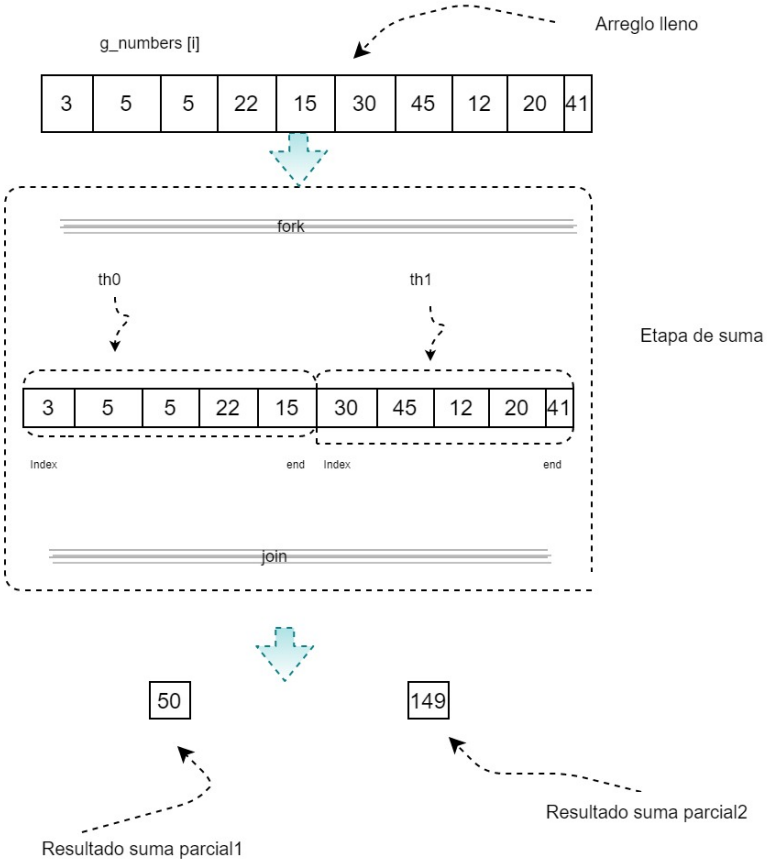

Figura 2

4 Referencias

[1] https://en.wikipedia.org/wiki/POSIX_Threads