Informe Taller 2

Taller de Sistemas Operativos

Escuela de Ingeniería Informática

Abel Núñez Cataldo

Abel.nunez@alumnos.uv.cl

Resumen. *Este informe documenta los procedimientos que se llevaron a cabo sobre el desarrollo de un programa utilizando threads, y el diseño de una solución que permita el llenado de un arreglo y la suma de sus contenidos en forma paralela, usando las herramientas proveídas en clase.*

1. Introducción

Bash es una herramienta de desarrollo de software que trabaja directamente sobre el sistema operativo, agilizando la utilización de memoria y procesamiento de código. Para llevar a cabo este ejercicio se utilizó, el software de virtualización distribuido por Oracle llamado VirtualBox, con el cuál se instalo una máquina virtual de Ubuntu Server versión 18.04, que se basa principalmente en el úso de comandos Linux. En esta entrega se prioriza la implementación de threads (hilos), para lo cual se utilizara principalmente el lenguaje de programación C++ estándar 2014 o superior.

El objetivo de este taller, consiste en realizar el diseño e implementación de un programa que llene un arreglo de números enteros y luego los sume. Estas acciones deben ser ambas realizadas de forma paralela, utilizando threads POSIX. Esta tecnología de threads, permite la realización de tareas en paralelo, desencadenando un mejor rendimiento del programa.

Este documento cuenta con una introducción que abarca la tecnología utilizada, los objetivos a alcanzar. Seguida de la descripción del problema planteado en el taller, que incluye la descripción de los datos utilizados, el significado de las variables y algunos ejemplos en que se instancian. Finalizando con el diseño de la solución, en la que se muestran diagramas de alto nivel y los métodos utilizados para la solución del problema.

2. Descripción del problema

El prolema planteado es crear un programa que esté compuesto de dós módulos. Uno que llene un arreglode números enteros aleatorios en forma paralela y otro que sume el contenido del arreglo en forma paralela. Se deben realizar prubas de desempeño, que generen datos que permitan visualizar el tiempo de ejecución de ambos módulos, dependiendo del tamaño del problema y de la cantidad de threads utilizados. Se establece que la forma de ejecutar este programa debe ser la siguiente:

```
./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.
```

2.1 Descripcion de los datos

Para esta situación a diferencia del taller anterior, no se hara uso de datos almacenados previamente en archivos de texto, sino que se utilizarán datos de tipo entero generados aleatoriamente para llenar un arreglo. Los otros datos que se apreciarán, son los del desempeño que tengan los procesos del programa a la hora de su ejecución.

3. Diseño de la solución

3.1 Metodología

La metodología consiste en analizar los requerimientos, diseñar un modelo del funcionamiento del programa, donde se pueda ver un orden de los procesos a realizar para lograr las acciones de llenado del arreglo y la suma de su contenido. Una vez estudiado el funcionamiento general de la solución , se debe establecer un modelo de diseño para los módulos de la solución, es decir, para cada módulo es necesario realizar un diseño de comportamiento, ya que si bien el procedimiento de bifurcación para los hilos es similiar, estos realizan distintas funciones.

3.2 Diseño

Para el diseño de la solución primero se procede a analizar el problema, estableciendo los siguientes requerimientos:

- Debe recibir como parametros de entrada, el tamaño del arreglo, npumero de threads y rango de los números en el arreglo
- Generar un arreglo de tamaño dinámico vacío.
- Llenar el arreglo con números enteros aleatorios en forma paralela.
- Sumar el contenido del arreglo y almacenarlo.

Seguido por el diseño de un diagrama de alto nivel, que ayude a explicar el funcionamiento general de la posible solución para el problema. Para esto se creo un diseño basado en el ejemplo de llenado paralelo de arreglo proveído por el profesor, en el que se asúme que el tamaño del arreglo es dinámico, configurable por parámetros

de entrada. La solución se estructura en las etapas denominadas "*Etapa de llenado*" y "*Etapa de sumado*" (Ver Figura 1), esta última etapa se comporta de forma similar a la lógica de la primera.

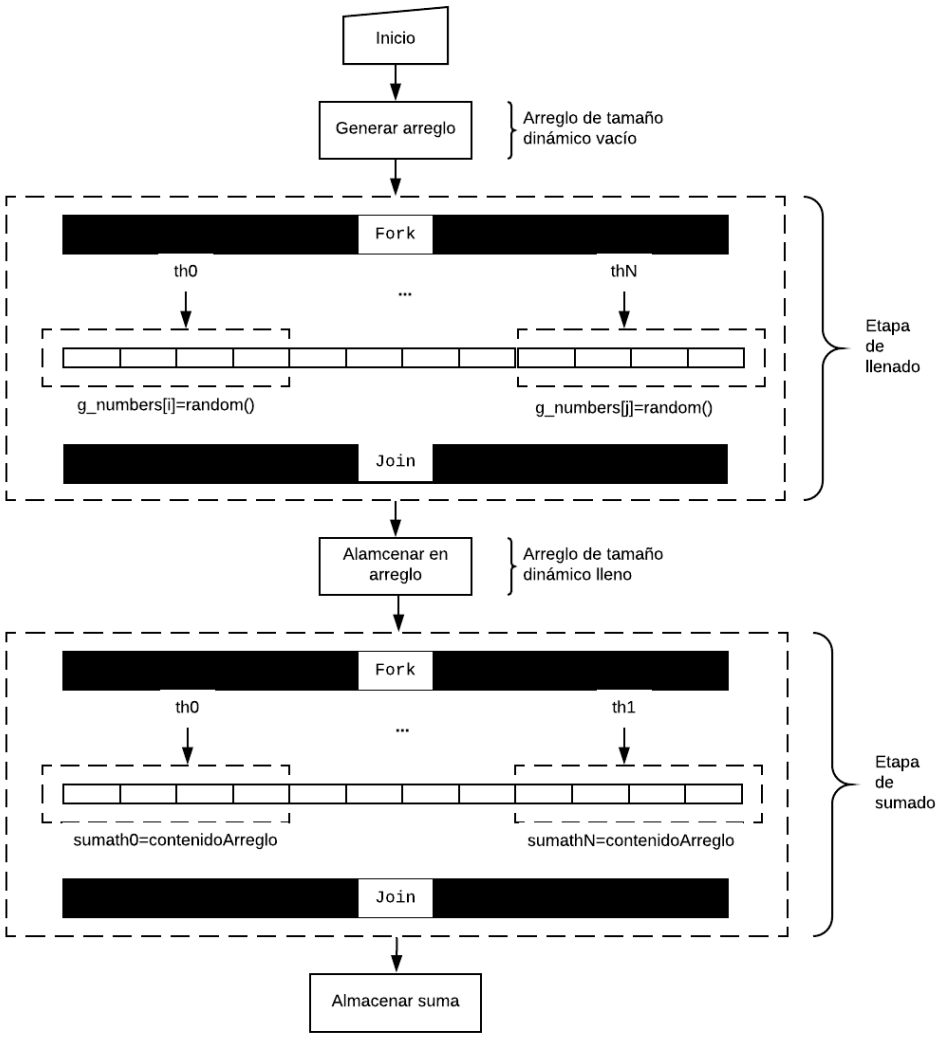

Figura 1 Diseño general de solución.

En la figura anterior se puede apreciar el funcionamiento general de la solución, todo este procedimiento se debe completar a través del uso de hilos de forma paralela. Previamente se solicito que a la hora de ejecutar el programa, este reciba como parametros de entrada, el tamaño del arreglo, número de hilos y el ragno de los número aleatorios. Una vez ejecutado, el programa debe crear de manera secuencial el arreglo del tamaño entregado, para continuar con la creación de los hilos respectivos, que serán utilizados en los módulos. Una vez hecho esto se procede al proceso de llenado del arreglo en el modulo uno, dividiendo el hilo principal en la cantidad de hilos establecida. En esta estapa, se decidio utilizar una de las funciones del "RNE" (Random Number Engine), proveída por el profesor en el ejemplo de llenado de arreglo, solo se uso la función uniform_distribution, reconocida por ser *thread-safe*. Tras terminar el llenado del arreglo, se procede a la etapa

de suma con un procedimiento similar, pero con distinta finalidad que es la entrega de la suma total de los datos. Ambas etapas se realizaron de forma secuencial y paralela, para llevar a cabo un analisis de rendiemiento del cada etapa.

4. Implementación

4.1 *Llenado del arreglo*

La implementación de la función **fillArray** se muestra en la Tabla 1.

Tabla 1

```
void fillArray(size_t beginIdx, size_t endIdx, size_t limInferior,
size_t limSuperior){
	std::random_device device;
	std::mt19937 rng(device());
	std::uniform_int_distribution<> unif(limInferior,limSuperior);
	for(size_t i = beginIdx; i < endIdx; i++){
		arreglo[i] = unif(rng);
	}
}
```

4.2 *Función de suma*

La implementación de la función **sumaParcial** se muestra en la Tabla 3.

Tabla 2

```
void sumaParcial(uint64_t &sumaArreglo, uint32_t beginIdx, uint32_t endIdx){
	sumaArreglo=0;
	mux.lock();
	for(uint32_t i = beginIdx; i < endIdx; i++){
		sumaArreglo += arreglo[i];
	}
	mux.unlock();
}
```

5. Resultados

Para la sección de resultados se realizaron una seria de pruebas del programa dandole distintos valores de entrada como se mostrará a continuación:

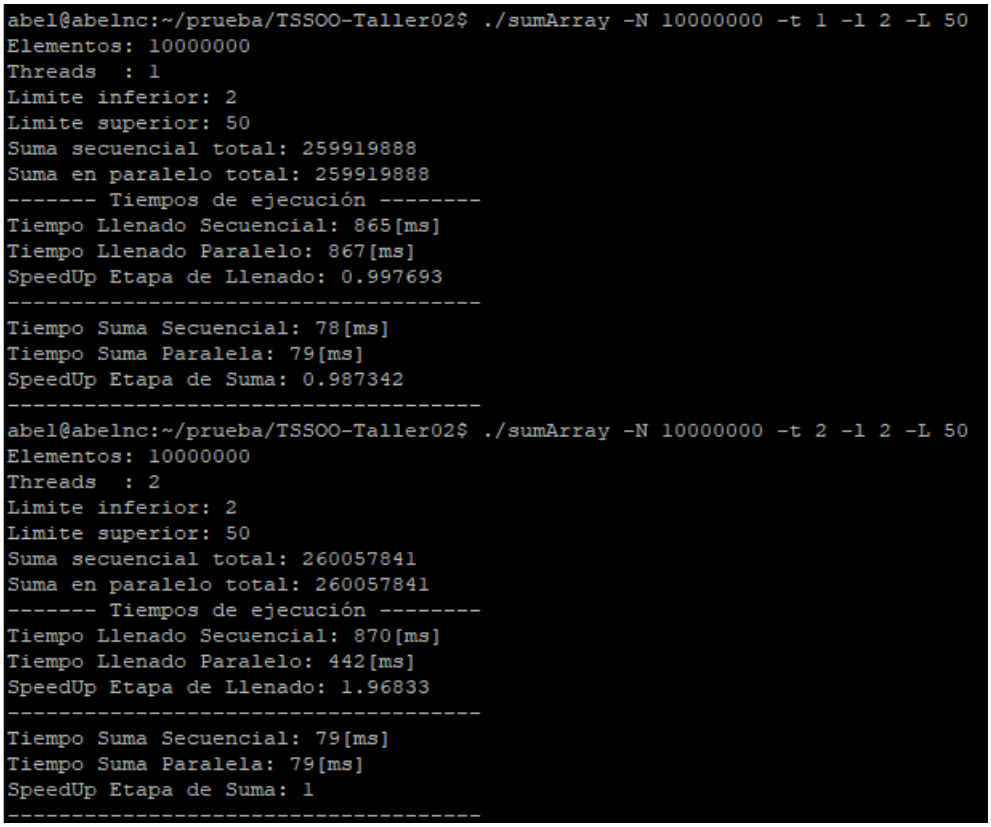

Figura 2

La Figura 2 muestra una prueba realizada con los valores de entrada, diez millones para el tamaño del arreglo, con uno y dos hilos respectivamente, como muestra el orden de ejcución, también para estas pruebas se estableció un rango de números entre 2 y 50.

Como se puede apreciar, tanto la suma secuencial y paralela coinciden entre ellas, para ambas pruebas. También notamos que existe una diferencia entre los resultados del Tiempo de llenado paralelo entre las pruebas, en la que este disminuye en la segunda ejecución con un hilo más. Además como se mencionó anteriormente se utilizó una función thread-safe por lo tanto no exite una gran diferencia en el costo cuando utilizamos un solo thread. Prueba de esto, es que el SpeedUp de 1.02 y 1.93, en la primera y segunda prueba respectivamente, lo que da a entender que en la segunda prueba, la implementación con threads es un 98% más rápida que la secuencial.

6. Conclusiones

Este documento mostró el desarrollo del taller 2 de la asignatura Taller de Sistemas Operativos, que consistía en el diseño e implementación de una solución de un problema, utilizando threads en forma paralela, con el lenguaje de programación C++. El problema a resolver, es llenar un arreglo de tamaño, cantidad de threads y rangos dinámicos definidos por el usuario. Se realizaron pruebas para el algoritmo implementado, solamente con funciones de generación de números aleatorios thread-safe.