Reporte técnico: Taller 02

Taller de Sistemas Operativos

Escuela de Ingeniería Informática

Felipe Castro Aguilar

felipe.castroa@alumnos.uv.cl

Resumen-*Hoy en día el multiprocesamiento es una característica fundamental de la programación de sistemas, aunque existan procesos alternativos a la creación de hilos, estos presentan un gran número de ventajas en especial en los procesadores multi núcleo. Como objetivo se tiene la creación de un diseño de solución adecuado a la problemática planteada, especificando y analizando cada punto de este. Con el objetivo de este taller en mente se generó un diseño, abarcando de manera general sin un detalle excesivo del código a implementar. Los hilos mencionados son parte fundamental para el desarrollo del taller ya que son los encargados de realizar las tareas presentadas en este, cumpliendo con lo establecido en los requisitos que debe cumplir el programa finalizado.*

1. Introducción

Para iniciar este taller debemos dominar y contextualizar las tecnologías que ocuparemos para el desarrollo de un diseño optimo que cumpla con la tarea de llenar un arreglo de números enteros y luego sumarlos. Ambas tareas se realizarán en forma paralela implementadas con threads POSIX.

Para dar paso al diseño de solución debemos entender el contexto en el cual está sumergido este reporte, para ello analizaremos los siguientes puntos.

1.1. Paralelismo

Antes de entender que es multiprocesamiento, multithreading y thread debemos entender que paralelismo es una función que realiza el procesador para ejecutar varias tareas al mismo tiempo, en otras palabras, puede realizar varios cálculos simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños. Al hablar sobre paralelismo surge un concepto fundamental llamado concurrencia, el cual se refiere a la existencia de varias actividades ejecutándose simultáneamente y necesita sincronizarse para ejecutarse de manera conjunta.

Dentro del paralelismo existen una gran cantidad de términos, pero para este informe solo nos enfocaremos en los cuales son necesarios para entender el diseño que se presentará.

1.1.1. Multiprocesamiento

Como su nombre lo indica es un procesamiento el cual permite la ejecución de uno o más hilo de ejecución a la vez. Esto es debido a la existencia de múltiples CPU las cuales pueden ser utilizadas para ejecutar múltiples procesos o múltiple thread dentro de un mismo proceso.

Muchas son las ventajas las cuales pueden traer un multiprocesamiento, algunas son:

- Mayor productividad: Esto es debido a completarán mayor cantidad de tareas en un menor periodo de tiempo.
- Mayor confiabilidad: En la ausencia de un procesador el sistema se tornará más lento, pero no se caerá, ya que posee más de un CPU.
- Ahorro de dinero: Esto se apega al punto de mayor productividad, ya que si baja el tiempo para realizar la misma tarea, se ve reflejado en los insumos necesarios para el funcionamiento de la máquina.

1.1.2. Multithreading

Multithreading o multihilo es la capacidad de ejecutar eficientemente múltiples hilos de ejecución. Esta técnica junto con Multiprocesamiento, descrito en el punto anterior, son los principales métodos para mejorar el rendimiento de una máquina. Un punto fundamental de esta técnica es que se utiliza para dar solución a las instrucciones ejecutadas secuencialmente, ya que al crear más de un hilo de ejecución este puede realizar trabajos de manera paralela.

1.1.2.1. Thread

Un Thread o hilo, es la unidad básica de ejecución de un Sistema Operativo. Además, puede ser ejecutado al mismo tiempo que otra tarea. Todos los hilos de ejecución comparten el espacio de direccionamiento del proceso y permiten simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Además, estos se diferencian de los procesos, ya que estos últimos son una instancia de ejecución del programa, en cambio los threads es la unidad de proceso más pequeña, otra diferencia es que los procesos se pueden dividir en varios hilos mientras que el hilo no se puede dividir.

Como ventaja los hilos tienen un tiempo de respuesta mejora notablemente, comparten recursos, por lo que se puede tener varios hilos de ejecución dentro del mismo espacio de direcciones. Otra ventaja es la utilización de múltiples CPU, esto permite que los hilos de un mismo proceso se ejecuten en diferentes CPU´s a la vez.

1.2. Problema

Se me solicitó crear un programa en el lenguaje de programación c++ 2014 o superior, el cual llene un arreglo de números enteros aleatorios del tipo uint32_t y luego los sume. El programa de be estar separado por dos módulos, el primero será el encargado de llenar un arreglo con números aleatorios en forma paralela y otro encargado de sumar el contenido de aquel arreglo en forma paralela. Sumado a esto, se deben hacer pruebas de desempeño las cuales permitan visualizar el comportamiento del tiempo de ejecución de ambos módulos

dependiendo del tamaño del problema la cantidad thread a utilizar. Esto se verá reflejado en una próxima entrega del documento, ya que este solo se presenta el diseño de la solución

El programa debe responder a una forma de uso (ver figura 1), estos parámetros se detallan en la siguiente tabla (ver tabla 1)

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

Figura 1 Forma de uso.

Tabla 1 Parámetros

PARÁMETROS	DESCRIPCIÓN
-N	Tamaño del arreglo.
-t	Número de thread.
-l	Límite inferior del rango aleatorio.
-L	Límite superior del rango aleatorio.
[-h]	Muestra la ayuda de uso y termina.

Para lograr abarcar todo el problema primero se realizó un correcto estudio del lenguaje de programación mencionado y seguido a esto profundizar en el buen uso de hilos.

2. Diseño

Luego de dar a conocer el marco teórico de este reporte se procedió a generar el diseño de la solución a implementar, para ello se enfoco en la solicitud inicial la cual nos pide crear un programa que este dividido en dos módulos los cuales cumplan funciones especificadas. Todo esto en el lenguaje de C++. A continuación, se muestra el diseño de cada módulo para finalizar con una integración de cada uno de ellos en la última etapa del diseño.

2.1. Primer módulo

Como primer modulo se nos solicita que generemos un arreglo de números enteros aleatorios del tipo uint32_t en forma paralela. Como ya se menciono en el punto 1.2 el arreglo debe ser llenado a través de parámetros de entrada los cuales ya fueron especificados, esto establece que cada thread que creemos conoce el inicio y fin de donde debe almacenar el numero aleatorio.

Como primer paso para el desarrollo debemos crear un arreglo de tamaño dinámico el cual se encargará de almacenar los números aleatorios. El arreglo dinámico tendrá un largo dependiendo del parámetro de entrada. Ya

que se solicita que sea un llenado en paralelo se crearan una “n” cantidad de thread, la cual es entregada en la ejecución del programa (ver figura 1), los cuales trabajaran de manera paralela para llenar el arreglo a través de una función random(). Esta cantidad de thread serán creados en un bucle el cual almacene el valor entregado por consola y creará hilos hasta que el contador sea igual al valor entregado. Dependiendo la cantidad de hilos creados el arreglo mencionado será dividido en partes similares y en el caso de ser una cantidad impar de hilos se dividirá proporcionalmente, cada división será llenada con valores aleatorios con la función mencionada. Donde cada segmentación estará ligada a un solo thread. Una vez dividido el arreglo, cada thread aplicará la función random() y obtendrá los valores aleatorios que almacenaran en el arreglo de manera paralela junto con los demás hilos. Esto se ve reflejado en la siguiente figura.

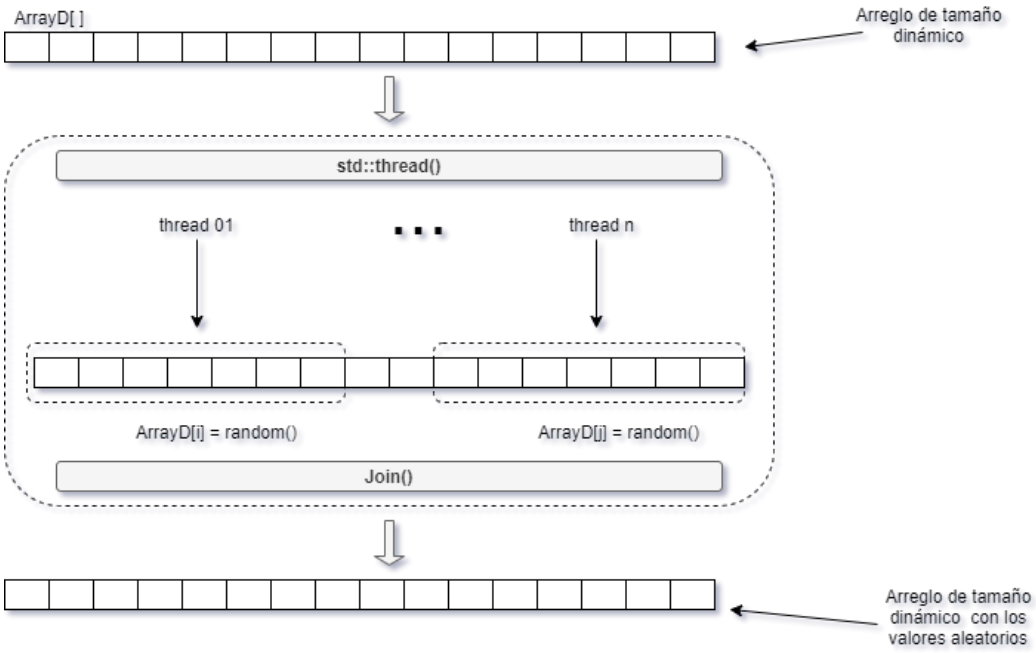

Figura 2 Diagrama general del módulo uno.

Como se aprecia en la figura 2 el nombre del arreglo a crear es ArrayD, donde por cada hilo se dividirá proporcionalmente a la cantidad de hilos y a cada sección se aplicará la función random() para almacenar los datos y finalmente con la función join() retornar los valores al arreglo.

Para el diseño se estima ocupar la función std::thread() para la creación de hilos y a su vez la función join() la cual retornará los valores aleatorios, generados en cada thread, los cuales llenaran el arreglo.

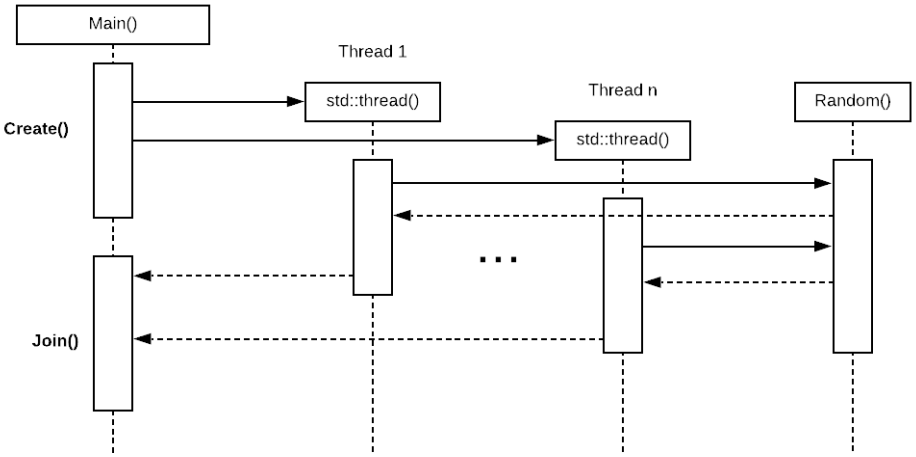

Figura 3 Diagrama de secuencia del proceso de llenado.

En la figura (ver figura 3) se muestra el proceso de como se llena el arreglo de manera aleatoria donde el Main() es el código a generar, el cual contiene el arreglo dinámico. Como se ve en la figura 3 en el Main() se crearán cada uno de los hilos y cada uno ellos instanciaran la función random() y los valores serán almacenados en cada hilo hasta el momento de hacer join() a cada hilo y solo en ese momento se le agregaran los números al arreglo.

2.2. Segundo módulo

Para este módulo se solicitó sumar los valores del arreglo obtenidos anteriormente. Para esto los hilos creados en la parte inicial se deben volver crear, ya que una vez utilizada la función join() en ellos quedan inutilizables. Estos nuevos hilos se encargará de volver a dividir el arreglo en las mismas secciones que en el módulo uno para luego, sumar los datos creados en la función random() para cada sección. Es importante destacar que la división del arreglo en el módulo uno es y debe ser la misma. Este proceso se realizará solo cuando exista una valor en el arreglo, esto gatillará la ejecución de la función join() donde cada hilo nos dará un valor el cual es la suma de los valores obtenidos por cada hilo(hilo creado en el módulo uno). Luego de obtener la suma de cada hilo de manera paralela se procederá a sumar cada una de las sumas ya mencionadas.

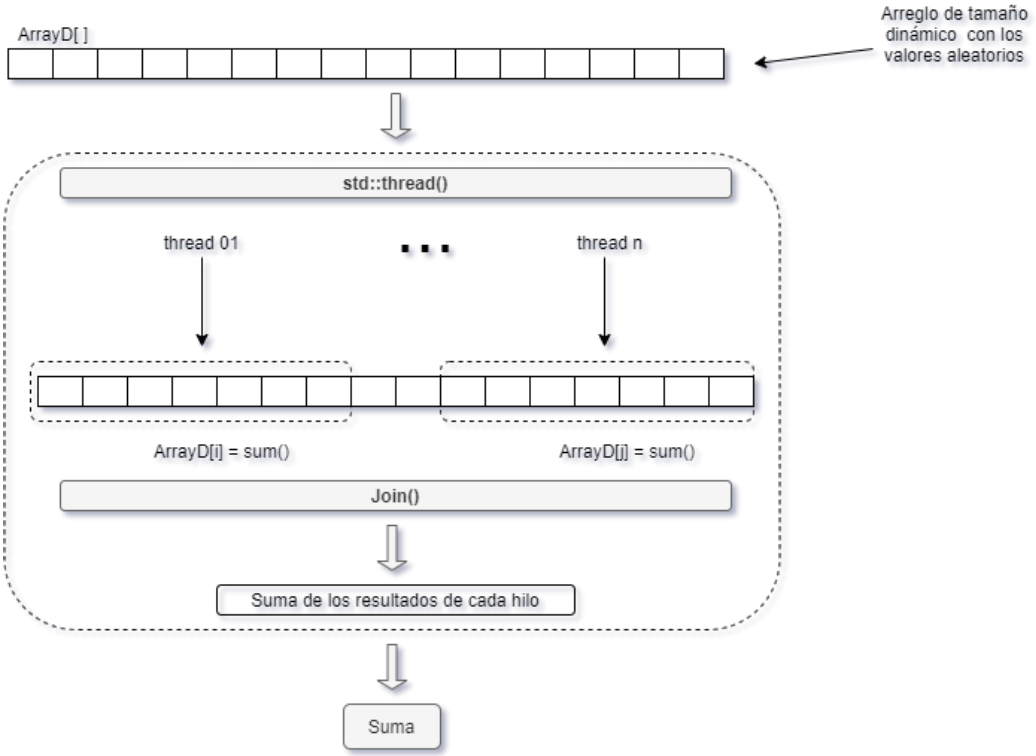

Figura 4 Diagrama de secuencia del proceso de suma.

Como se muestra en la figura los hilos se encargarán de sumar cada sección del arreglo para luego ser sumadas y finalmente entregar el valor esperado. El resultado del proceso de diseño se muestra en el módulo dos.

Para finalizar con el proceso de diseño se muestra el siguiente diagrama el cual muestra la integración de ambos módulos.

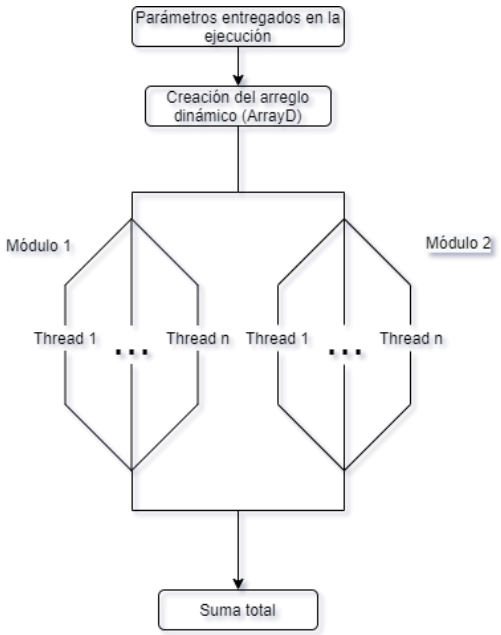

Figura 5 Diagrama general de los módulos.

Es importante destacar que ambos módulos contienen hilos que trabajan de manera paralela y a su ves los módulos trabajan de manera paralela, pero con la limitación de que los hilos del módulo dos solo se ejecutarán si el hilo del módulo uno, correspondiente a su sección del arreglo ya haya llenado dicha sección con valores aleatorios.

3. Resultados

Una vez analizado, comprendido, trabajado y desarrollado el código solicitado en este taller y a través de múltiples pruebas tanto en mi equipo y en la máquina virtual instalada generé los siguientes resultados:

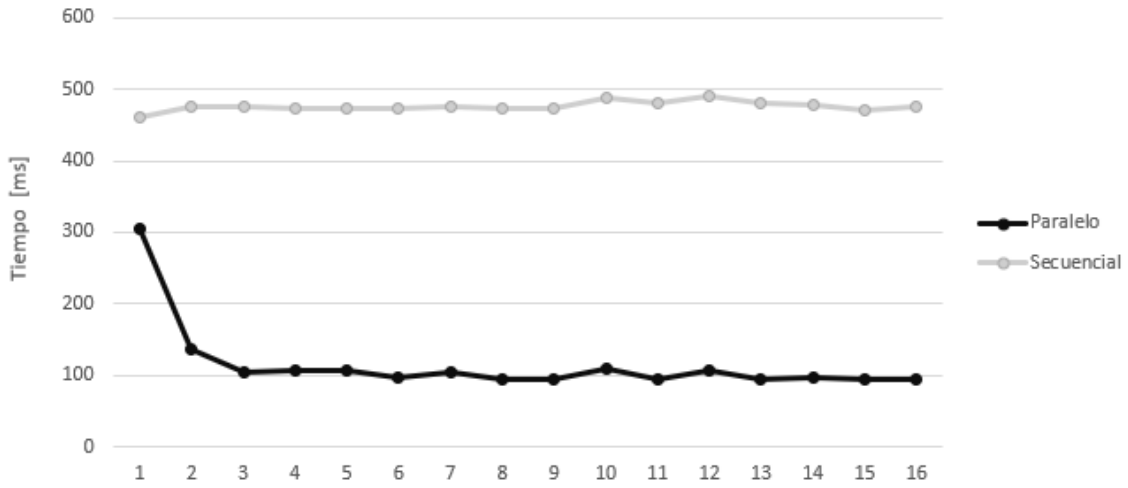

Figura 6 Grafico Tiempo de llenado vs cantidad de threards.

La Figura 6 se muestra los datos obtenidos de las pruebas realizadas en la máquina virtual donde claramente la ejecución con threads presente un mejor rendimiento en un inicio a mayor cantidad de thread a utilizar. Esto se explica, ya que al ocupar hilos estamos dividiendo el trabajo en partes “similares” y esto mejora los tiempos de ejecución.

La gráfica muestra una comparativa de los tiempos de llenado mediante hilos vs de manera secuencial. Para cada valor se hicieron 7 pruebas donde se promedió para obtener los datos presentados en la tabla. Es evidente que el método secuencial se mantenga “constante”, ya que la cantidad de datos y el rango de ellos, para todas las pruebas fueron la misma.

También es importante entender que la curva de threads se aplana debido a la capacidad física del computador donde se implementa dicho código y la calidad del código realizado.

Como comentario para esta etapa puedo decir que el tiempo al ocupar un solo hilo y el tiempo secuencial deben ser similares, pero en la gráfica se muestra con una diferencia de 150 [ms], esto es debido, luego de un análisis, al como se llenaron los arreglos correspondientes, ya que se ocuparon vectores y arreglos para este módulo.

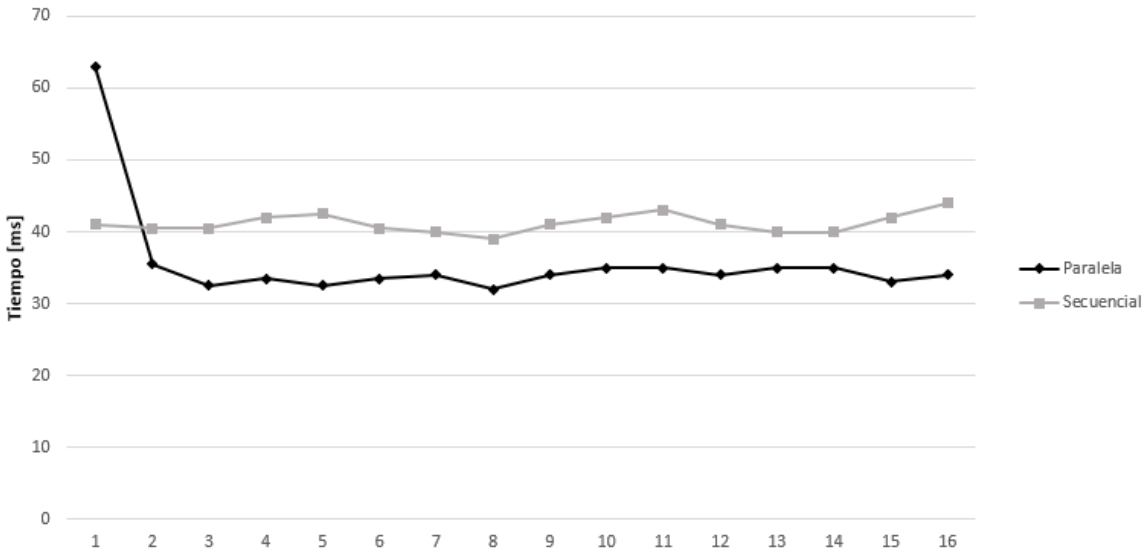

Figura 7 grafico Tiempo de Suma vs cantidad de Threads

En la Figura 7 se muestra una comparativa de los tiempos obtenidos al momento de la suma requerida en el módulo dos. Como se aprecia en la gráfica, cuando se utiliza solo un hilo para la ejecución paralela el tiempo es mayor que para la ejecución de manera secuencial, esto es debido a que el proceso para obtener la suma total tiene mayor trabajo que de manera secuencial. La suma total de la ejecución paralela se obtuvo al "dividir" el arreglo base y por cada thread creado sumar lo seleccionado, para luego sumar todas las sumas parciales. Por este motivo en la ejecución paralelo con un solo hilo tiene un tiempo mayor al del secuencial.

Luego al ocupar dos threads el tiempo disminuye notoriamente para luego mantenerse en una línea "constante" ya que de igual manera que el módulo anterior dividimos el trabajo para ganar en tiempo.

Es destacable que las pruebas fueron con los mismos datos y rango de ellos.

Los datos de mi equipo junto, el de la máquina virtual y los parámetros utilizados para realizar las pruebas se muestran en las siguientes tablas:

Tabla 2 Parámetros utilizados

PARÁMETROS	TAMAÑO
-N	100000000
-t	Variable
-l	1
-L	1000000

Tabla 3 Datos máquina virtual

TITULO	DESCRIPCIÓN
Memoria base	**2048 MB**
Procesadores	**3**
Aceleración	**VT-x/AMD-V, Paginación anidada, Paravirtualización KVM**

Tabla 4 Datos de mi equipo

TITULO	DESCRIPCIÓN
Procesador	**intel® Core ™ i5 -7400 CPU @ 3.00GHZ 3.00 GHZ**
RAM	**16,0 GB**
Sistema operativo	**64 bits**

Las tablas 2, 3 y 4 fueron mencionadas debido a que la ejecución puede variar dependiendo de la capacidad física de cada equipo. Este punto será explicado en mayor detalle en la conclusión.

4. Conclusión

De esta manera y mediante el desarrollo del código realizado gracias a un correcto diseño previamente realizado se corroboro la utilidad de los thread en tiempo de ejecución para una misma tarea realizada de manera secuencial. Las gráficas presentadas respaldan el correcto uso de ellos ya que los tiempos van disminuyendo en un inicio, para luego mantenerse. Como detalle agrego que para una gran cantidad de datos (800.000.000) puede que no sea factible el uso de estos, debido que el proceso de sumar los datos (sumas de sumas) obtenidos por cada thread puede extender el tiempo de ejecución de este. Este caso me ocurrió al hacer las pruebas en mi equipo y luego en la máquina virtual disminuye esa cantidad máxima debido a que la maquina virtual no posee la misma capacidad física que el equipo hospedador.

Finalmente, gracias a las pruebas realizadas y el estudio de los conceptos asociados se logró generar un código que cumple con los requerimientos iniciales, pero esto no significa que nos pueda ser optimizado en algún futuro.

5. Referencias

[1] The cplusplus website. [Online]. Avalaible: http://www.cplusplus.com