TEC | Tecnológico de Costa Rica

Tercer Proyecto Programado

Curso:
Principios de Sistemas Operativos

Sternap

Profesor:
Eddy Ramírez Jiménez

Estudiantes:
Alejandro Cerdas
Kener Castillo
Pablo Pérez

November 13, 2024

1 Introducción

Este proyecto tiene como objetivo construir un simulador de red P2P, donde varios clientes pueden compartir y solicitar archivos entre sí en una red. Para esto se maneja un servidor central con una tabla hash para indexar metadatos de archivos de acuerdo al identificador de este. Se distingue un archivo de otro de acuerdo a dos funciones de hash que se aplican sobre los archivos y el tamaño de este. Como metadatos se guardan los nombres de archivo e IP's de los peers que cuentan con el archivo.

De esta manera inicialmente un peer elige una carpeta con archivos que se registran en el servidor central y después puede realizar consultas a este para buscar archivos de acuerdo a un nombre y descargar archivos de acuerdo a su identificador. Cuando un peer descarga un archivo primero obtiene por parte del servidor la lista de IP's de los peers que cuentan con el archivo, seguidamente distribuye la descarga por medio de una solicitud a cada peer de una parte del archivo. Por lo que cada peer actua como cliente y servidor, con la ayuda de un servidor central.

Las dos operaciones mencionadas son:
- Find *****
- Request H1 H2 S

Donde Find recibe el substring a buscar y Request recibe el primer hash, el segundo y el tamaño del archivo.

Por último, se realizó una pequeña interfaz por consola para el uso de las operaciones y la observación de los datos.

2 Marco Teórico

2.1 Redes Peer-to-Peer

Las redes P2P permiten el intercambio directo de información entre nodos, llamados peers, sin la necesidad de un servidor central. Esta arquitectura se emplea en sistemas de distribución de archivos, donde cada cliente puede ser tanto proveedor como consumidor de recursos.

3 Hashing

El *hashing* es una técnica utilizada en informática para mapear datos de tamaño variable a valores de tamaño fijo, conocidos como *códigos hash*. Esta técnica se emplea en diversas aplicaciones, como la indexación de bases de datos, estructuras de datos como tablas hash, y la verificación de integridad de datos. El proceso

de hashing es crucial para obtener una representación rápida y eficiente de grandes volúmenes de datos, ya que permite realizar búsquedas, inserciones y eliminaciones en tiempo constante promedio.

La función de hash toma una entrada (o clave) y devuelve un valor entero, conocido como el valor hash, que corresponde a una posición en una estructura de almacenamiento, como una tabla hash. La principal propiedad de una función de hash eficiente es que minimiza las colisiones, que ocurren cuando dos entradas distintas generan el mismo valor hash. Sin embargo, las colisiones no se pueden evitar por completo, por lo que se deben emplear técnicas como la resolución de colisiones, que ayudan a manejar estas situaciones.

3.1 FNV-1A

Una de las funciones de hash más conocidas y simples es el algoritmo *FNV-1A* (Fowler–Noll–Vo), que fue desarrollado en 1991 por Glenn Fowler, Landon Noll y Phong Vo. El algoritmo FNV-1A es utilizado ampliamente debido a su simplicidad y eficiencia, especialmente en aplicaciones donde el rendimiento es crítico.

El algoritmo FNV-1A genera un valor hash de tamaño fijo a partir de una secuencia de caracteres, aplicando una serie de multiplicaciones y sumas para combinar los valores de los caracteres. A diferencia de otros algoritmos de hash, FNV-1A se caracteriza por su uso de una constante prime (un número primo) y la manera en que aplica una operación XOR en cada paso, lo que ayuda a dispersar los valores de manera uniforme.

El proceso de cálculo del hash con FNV-1A es el siguiente: comienza con un valor inicial predefinido (generalmente 0x811c9dc5) y, para cada byte del texto de entrada, se realiza la siguiente operación:

Una vez que todos los bytes han sido procesados, el valor resultante es el hash final.

El algoritmo FNV-1A es muy eficiente para cadenas cortas y es comúnmente utilizado en algoritmos de verificación de integridad y en aplicaciones de redes, como la detección de cambios en archivos o mensajes.

4 Algoritmo Knuth-Morris-Pratt (KMP)

El algoritmo Knuth-Morris-Pratt (KMP) es un algoritmo eficiente para la búsqueda de patrones dentro de un texto, propuesto por Donald Knuth, Vaughan Pratt y James H. Morris en 1977 [5]. La principal ventaja del algoritmo KMP sobre otros métodos de búsqueda es que evita la repetición de comparaciones innecesarias, lo que lo hace más eficiente, especialmente en textos largos.

4.1 Preprocesamiento del Patrón

El primer paso del algoritmo KMP es la construcción de una tabla de fallos para el patrón P. Dada una cadena de longitud m, la tabla de fallos es un arreglo de

tamaño m tal que para cada posición i del patrón, la tabla indica la longitud del mayor sufijo que también es prefijo de la subcadena $P[0..i]$.

Este preprocesamiento tiene una complejidad de tiempo $O(m)$ y permite al algoritmo evitar comparaciones redundantes durante la búsqueda en el texto.

4.2 Búsqueda del Patrón

Una vez construida la tabla de fallos, el algoritmo recorre el texto T de longitud n, comparando el patrón con las subcadenas de T. Cuando ocurre una discrepancia, el algoritmo utiliza la tabla de fallos para determinar cuánto debe desplazarse el patrón sin perder ninguna coincidencia posible. Esto reduce significativamente el número de comparaciones en comparación con otros algoritmos como el de búsqueda por fuerza bruta.

El tiempo total de ejecución del algoritmo es $O(n + m)$, lo que lo hace particularmente eficiente cuando se busca el mismo patrón en textos grandes[6].

4.3 Lenguaje de programación

El lenguaje de programación usado en todo el proyecto es C++. C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C y añadir mecanismos que permiten la manipulación de objetos [4]. Gracias a estas características es posible implementa programación orientada a objetos y programación funcional, convirtiéndolo en un lenguaje multiparadigma. El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre 'C con clases'. En C++, la expresión C++ significa incremento de C y se refiere a que C++ es una extensión de C [4].

4.4 IDE

El IDE utilizado para desarrollar este proyecto en su totalidad, fue Visual Studio Code. VS Code es un editor de código desarrollado por Microsoft y lanzado un 29 de abril de 2015. Este IDE está disponible para Linux, macOS, Windows y tiene una versión web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Además, presenta un resaltado de sintaxis para alrededor de 52 lenguajes de programación. [1]

4.5 Bibliotecas

Finalmente, en este proyecto se dio uso a las siguientes bibliotecas:

- bits/stdc++.h: la biblioteca estándar es una colección de clases y funciones, escritas en el núcleo del lenguaje. La biblioteca estándar proporciona

varios contenedores genéricos, funciones para utilizar y manipular esos contenedores, funciones objeto, cadenas y flujos genéricos (incluyendo E/S interactiva y de archivos) y soporte para la mayoría de las características del lenguaje. La biblioteca estándar de C++ también incorpora la biblioteca estándar de C. Las características de la biblioteca estándar están declaradas en el espacio de nombres std.[2]

- httplib.h: es una biblioteca HTTP/HTTPS multiplataforma con encabezado de archivo único de C++11 [3]. Permite implementar métodos HTTP mediante puertos y emplear una comunicación con otras instancias.

5 Descripción de la Solución

El sistema de Sternap fue implementado mediante una comunicación peer to peer haciendo uso de la biblioteca de httplib para permitir la comunicación entre los pares y el servidor. La estructura de la solución es

5.1 Estructura General

El sistema se compone de dos elementos principales:

- **Servidor Central (MainServer):** Responsable de registrar los archivos compartidos por los clientes, realizar las busquedas de los nombres de los archivos y proporcionar la lista de IPs y puertos de los peers que tienen el archivo solicitado.
- **Cliente (Peer):** Realiza un doble hash a cada archivo y se conecta al servidor central enviando los datos antes procesados, puede buscar archivos en la red y maneja solicitudes de descarga de otros peers.

5.2 Servidor (MainServer)

La clase `MainServer` es implementada mediante un servidor de la biblioteca httplib.h que le permite recibir Post de los clientes con un string que contiene los datos de los archivos (hash 1 y 2, el tamaño y la ruta) y la ip y el puerto unidos. Todos estos datos son almacenados en un multimap que contiene tres llaves: hash1, hash2 y el tamaño.
Para recibir solicitudes find, se implementa un post en el server de httplib para procesar este tipo de solicitudes. Este post redirige a una función encargada de aplicar KMP sobre el multimap para realizar la busqueda de las rutas que contiene el string deseado por el cliente y retorna un string con los hashes, tamaños y los diferentes nombres que coinciden con lo buscado.
Posteriormente cuando un peer realiza un request de los darchivo, nuevamente se implementa un post en el servidor httplib, el cual redirige a una funcion que obtiene del multimap los datos requeridos y retorna la ip y la ruta para cada archivo en las diferentes instancias.

5.3 Hashing

Para identificar archivos de forma única, se implementaron dos funciones hash, basadas en algoritmos de hashing, uno por operaciones con bits y números primos y el otro una variante del FNV-1A. Estos permiten diferenciar archivos aunque tengan nombres diferentes. Esto facilita la comparación y búsqueda eficiente en la red.

5.4 Cliente (Peer)

La clase `Peer` antes de su conexión con el servidor central, se encarga de recibir una ip, un directorio y un puerto. Para los archivos del directorio, se genera dos hashes como se describió anteriormente y posteriormente se genera un request al servidor central con los datos generados.
Para procesar solitudes de find, el peer recibe un string y se lo envia al servidor central. La respuesta se presenta presenta al usuario por consola para que puede obtener el identificador de algun archivo que le interese.

Para procesar solitudes de obtener un archivo, se solicita ambos hashes y el tamaño del archivo que se desea solicitar. Una vez con estos datos se realiza una solitud al servidor central para obtener los datos de los dueños del archivo. Cuando se obtienen los datos se procede a generar varios hilos para que ejecuten una función que realiza un post a los otros peers. Adicionalmente, se utiliza las propiedades de la biblioteca future para poder obtener mensajes de los hilos, en caso de que haya fallado la comunicación con el peer. Si esto último ocurre se realizan solicitudes a los peers que si respondieron, para obtener las partes faltantes del archivo. Para concluir con el proceso, se procede a unir los archivos obtenidos en uno solo y guardarlo.

Finalmente, para procesar los request de otros peers, se genera un hilo que se mantiene ejecutando un servidor de httplib con un método post para recibir estas solicitudes. Para enviar una parte de una archivo se utiliza la técnica de streaming de la biblioteca httplib para no cargarlo completamente en disco.

6 Conclusiones

El proyecto demuestra la viabilidad de implementar un sistema P2P básico en C++, usando estructuras de datos y algoritmos de búsqueda para gestionar y localizar archivos en la red. El uso de dos funciones de hash asegura la identificación precisa de archivos y una distribución eficiente de las solicitudes en la red. Por otra parte el uso de buffer para el momento de compartir trozos de archivos, aseguran que se comparta de forma correcta un archivo sin sobrecargar ni la red ni a cada uno de los peers. La reconstrucción del archivo fue gracias al manejo con futures, para asegurar la consistencia de cada una de las partes. De esta forma el proyecto en sí cumple sus objetivos, demostrando la gran utilidad y viabilidad de estos sistemas P2P.

7 Descripción de la Experiencia

La implementación del proyecto de intercambio de archivos P2P fue una experiencia desafiante y enriquecedora, que implicó una serie de complejidades técnicas a lo largo de su desarrollo. El sistema, diseñado para permitir a los peers compartir y descargar archivos de otros usuarios sin necesidad de un servidor centralizado, requería una gestión precisa de la sincronización y comunicación entre los nodos. Uno de los mayores retos fue el manejo de la sincronización de los datos entre los diferentes peers. Dado que los archivos debían dividirse en partes y distribuirse entre múltiples peers, era fundamental asegurar que la descarga de las partes del archivo ocurriera de manera eficiente y en el orden correcto. Implementar un mecanismo de verificación y control de errores para garantizar que todas las partes fueran recibidas sin corrupción también presentó dificultades. El protocolo de comunicación HTTP fue una elección acertada, ya que permitió simplificar la implementación de la transferencia de archivos al usar una estructura familiar para la transmisión de datos, aunque en algunos casos se necesitaba gestionar la persistencia de las conexiones para mejorar la eficiencia. Otro desafío importante fue el manejo adecuado de los sockets para la comunicación entre los peers. Debido a que cada peer actuaba tanto como servidor como cliente, era necesario implementar un sistema eficiente para aceptar y manejar múltiples conexiones simultáneas sin que los recursos compartidos se vieran comprometidos. Esto implicó el uso de hilos (threads) para gestionar las conexiones entrantes y salientes de manera concurrente. El diseño de los hilos tuvo que ser cuidadosamente planeado para evitar condiciones de carrera y asegurar que las conexiones de descarga y subida no se bloquearan mutuamente. En resumen, el proyecto de P2P para la descarga de archivos desde otros peers proporcionó una valiosa experiencia en la implementación de redes descentralizadas, manejo de concurrencia y optimización de recursos. A pesar de los retos encontrados, especialmente en cuanto a la sincronización de los datos y la eficiencia de la comunicación, el sistema resultó ser exitoso y funcional, ofreciendo una solución robusta para el intercambio de archivos de manera eficiente y confiable entre los usuarios.

8 Aprendizajes

8.1 Pablo Perez

Durante este proyecto, aprendí un poco sobre manejo de redes P2P y comunicación mediante red. También repasé en algoritmos de hashing y estructuras de datos para optimizar la búsquedas y gestionar archivos en un entorno distribuido.

8.2 Kener Castillo

Para este proyecto, me adentré en el mundo de las redes P2P, donde los archivos se distribuyen entre varios nodos llamados peers. Esto me permitió comprender tanto las ventajas, como la mayor autonomía y rapidez en la transferencia de

archivos, como los obstáculos, como los problemas de fiabilidad y la posible desincronización entre los participantes. Fue una experiencia que me mostró cómo estos sistemas pueden ser muy eficaces, pero también desafiantes de gestionar de manera consistente.

8.3 Alejandro Cerdas

En este proyecto, aprendí mucho sobre las redes P2P y cómo los nodos se comunican directamente entre sí para compartir archivos. Experimenté tanto los beneficios, como la descentralización y la escalabilidad, como los retos, como la gestión de conexiones inestables o la sincronización de archivos entre varios nodos. Esta experiencia me permitió comprender mejor las ventajas y limitaciones de los sistemas distribuidos para intercambio de archivos.

8.4 Bibliografía

[1] Colaboradores de los proyectos Wikimedia. "Visual Studio Code - Wikipedia, la enciclopedia libre". Wikipedia, la enciclopedia libre. Accedido el 6 de septiembre de 2024. [En línea]. Disponible: https://es.wikipedia.org/wiki/Visual_Studio_Code

[2] Colaboradores de los proyectos Wikimedia. "Biblioteca estándar de C++ - Wikipedia, la enciclopedia libre". Wikipedia, la enciclopedia libre. Accedido el 14 de noviembre de 2024. [En línea]. Disponible: https://es.wikipedia.org/wiki/Biblioteca_estándar_de_C++

[3] "GitHub - yhirose/cpp-httplib: A C++ header-only HTTP/HTTPS server and client library". GitHub. Accedido el 14 de noviembre de 2024. [En línea]. Disponible: https://github.com/yhirose/cpp-httplib

[4] Colaboradores de los proyectos Wikimedia. "C++ - Wikipedia, la enciclopedia libre". Wikipedia, la enciclopedia libre. Accedido el 14 de noviembre de 2024. [En línea]. Disponible: https://es.wikipedia.org/wiki/C++

[5] D. Knuth, V. Pratt, J. H. Morris. "Fast Pattern Matching in Strings", *SIAM Journal on Computing*, vol. 6, no. 2, 1977, pp. 323-350. [En línea]. Disponible: https://epubs.siam.org/doi/abs/10.1137/S0097539701193652

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*, 3ra edición, MIT Press, 2009. [En línea]. Disponible: https://mitpress.mit.edu/books/introduction-algorithms-third-edition

[7] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 1973. [En línea]. Disponible: https://www.amazon.com/Art-Computer-Programming-Vol-Sorting/dp/0321751043

[8] G. Fowler, L. Noll, P. Vo, "FNV-1 and FNV-1a: A Good Hash Function", 1991. [En línea]. Disponible: https://web.archive.org/web/20170325024930/http://www.isthe.com/chongo/tec