Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3641 - Lenguajes de Programación I
Enero-Marzo 2018

Proyecto 2:
Indizador y Buscador de Archivos

Erick Flejan 12-11555
Rafael Cisneros 13-11156

21 de marzo de 2018

1. Introducción y Estructura del Informe.

La búsqueda de un archivo en un sistema de archivos es una tarea común realizada por los usuarios. El sistema operativo es el encargado de realizar esta búsqueda de la manera más eficiente posible, de modo que el usuario pueda ver los resultados en un corto periodo de tiempo. Para realizar la búsqueda rápidamente, los archivos no son buscados directamente en sistema de archivos, sino en una tabla de índices. Sin embargo este índice no se actualiza muy frecuentemente.

Buscando mejorar la eficiencia de esta tarea, se propone la implementación de una nueva estrategia en el buscador. La estrategia consiste en convertir la tabla de índices en una tabla de hash, y actualizar dicha tabla cada vez que se realiza una búsqueda.

En este informe se explicara una posible implementación de esta estrategia, haciendo uso de hilos y otros recursos proveídos por el sistema para la obtención de los meta-datos de los directorios y archivos.

El contexto de este informe tendrá como base los archivos “hash.c” y “main.c”. A continuación, se detallará la estructura del informe:

- En el punto 2 se explicará cómo compilar y correr el programa suministrado.
- En el punto 3 se explicará la estrategia de creación de hilos usada para manejar las distintas partes de nuestro programa.
- En el punto 4 se explicará la estrategia de exclusión mutua implementada para evitar conflictos entre los hilos usados
- En el punto 5 se explicará todo lo referente a la tabla de hash
- En el punto 6 se explicara el proceso de ingreso de un path a la tabla de hash
- En el punto 7 se explicará el formato del archivo Índice que se recibe como entrada para cargar un índice de archivos y que se retorna al usuario al finalizar el programa.
- En el punto 8 se detallarán las conclusiones del trabajo realizado.

2. Compilación y Corrida.

Compilación:

El programa puede ser compilado mediante 2 métodos. Ambos métodos necesitan el uso de un sistema operativo derivado de Unix y ubicarse con la terminal en el directorio en donde se encuentra ubicado el programa.

El método preparado es el del archivo “Makefile”, este permite la compilación automática del programa, convirtiendo el código fuente en un archivo objeto (“*.o”). Ubicado en el directorio del programa, desde la terminal se puede usar el comando “make”, este creará el archivo ejecutable “main.out” el cual podrá ser ejecutado desde la terminal para correr el programa.

El otro método es la compilación manual desde la terminal, mediante el comando del compilador “gcc” siguiente:

```
>: gcc main.c -pthreads -o main.out
```

Esto igualmente creará el archivo ejecutable “main.out”

Corrida:

El programa puede ser ejecutado mientras se encuentre ubicado en el mismo directorio del archivo ejecutable usando el comando “main” seguido de un parámetro obligatorio que representa el término a ser buscado en la tabla de índices. Esto imprimirá en pantalla los archivos de la tabla de índices cuyos nombres contengan el término de búsqueda. Además se actualiza el archivo de Índice con los nuevos archivos encontrados que no se encontraban en la tabla de Índices. Las nuevas entradas no serán encontradas en la búsqueda actual, pero en las siguientes sí. Existen un número de opciones que pueden ser agregadas al programa por medio del uso de “flags”.

Mediante el flag “-d <directorio>”, o su versión a larga “–dir<directorio>”, se especificará la carpeta desde la cual se inicia el recorrido del árbol de directorios en búsqueda de archivos para agregar a la tabla de Índices.

Mediante el flag “-m <altura>”, o su versión larga “–max<altura>”, se especificará la profundidad máxima de exploración hasta la cual se llegará.

Mediante el flag “-i”, o su versión larga “–i<archivo>”, se especificará el archivo de Índice desde el cual se cargara la tabla de hash y sobre la cual se realizara la búsqueda del término.

Mediante el flag “-u”, o su versión larga “–noupdate”, se indica que no se deben entrar en directorios que ya estén en el índice, buscando que agregar al índice; solo en los directorios que se encuentren nuevos.

Mediante el flag “-a”, o su versión larga, “–noadd”, se indica que no se debe entrar en directorios que no estén en el índice, buscando que agregar al índice, solo en directorios que ya estén en él.

Si se reciben los flags –a y –u al mismo tiempo, solo se realiza la búsqueda en el índice, sin realizar actualización de la tabla de hash.

3. Creación y Manejo de Hilos.

El programa inicia con la carga de la tabla de hash inicial proveniente de un archivo. Una vez creada la tabla, se procede a la creación de dos hilos: el primero realiza la búsqueda del término deseado en la tabla de hash; el segundo realiza el recorrido en el árbol de directorios buscando archivos y directorios a ser agregados a la tabla.

4. Exclusión Mutua.

Para evitar la corrupción de los datos, es necesario implementar alguna estrategia de exclusión mutua entre los hilos. Siguiendo el modelo del problema de los Lectores/Escritores, mientras un escritor se encuentre escribiendo datos en un archivo, ningún otro escritor o lector puede hacer uso del archivo; por otro lado, si un lector se encuentra leyendo el archivo, otros lectores pueden hacer uso, del archivo para leer, pero los escritores deben esperar a que todos los lectores terminen antes de poder hacer uso del archivo.

En nuestro problema solo tenemos un lector (el buscador) y un escritor (el indizador). Por lo tanto debemos evitar que si uno de los hilos está usando la tabla de hash, el otro acceda a ella. Para esto usamos un Mutex, cuando el buscador está leyendo en la tabla, bloqueamos el mutex para que el indizador no pueda modificarlo; de igual forma, cuando el indizador logra acceder a la tabla, bloqueamos la misma para que el buscador no pueda acceder.

5. Tabla de Hash.

La tabla de hash fue implementada como un struct Índice. Este struct tiene dos elementos: un apuntador a un arreglo de Listas de Llaves (explicadas a continuación) y un entero con el tamaño del arreglo. Las Listas de llaves son otro struct que contiene el número de llaves que luego de pasar una función de hash, caen en el índice del arreglo en el cual está contenida la Lista de llaves; además la Lista contiene un apuntador al primer elemento de una lista enlazada de estructuras Nodo_Hash. Los Nodo_Hash son elementos de una lista enlazada, contiene un apuntador al próximo elemento de la lista, y un apuntador a al primer elemento de una nueva lista enlazada de Nodo_Paths. Los Nodo_Paths contienen los paths absolutos de los archivos que contienen en su nombre la llave del Nodo_Hash, además de esto posee un indicador que determina si este path conduce a un archivo o a un directorio, este indicador es un 1 si es un archivo o un 0 si es un directorio. El método para indicar si el path conduce a un directorio o a un archivo consiste en utilizar el inode del para conseguir que características tiene este objeto. Explicado de forma más sencilla, nuestra tabla es un arreglo de listas enlazadas de llaves, donde cada elemento de la lista es una lista enlazada de paths absolutos.

La tabla de hash viene implementada con una serie de funciones para insertar elementos en ella, o imprimir su contenido. Además de una función de rehash que es usada cuando un elemento del arreglo contiene demasiadas colisiones, se usan funciones auxiliares para realizar el rehash de forma más rápida y sencilla. La función de hash usada para calcular el índice correspondiente a una llave es la suma de la representación en enteros de cada carácter de la llave, al resultado de esto lo dividimos entre el tamaño de la tabla y el resto de la división es el índice en donde se va a introducir el elemento.

6. Ingreso de paths a la tabla de Hash.

Cuando se recibe un path para ser agregado a la tabla de hash, primero se obtiene el path absoluto correspondiente al path recibido, seguidamente se parsea el path absoluto, saltando los nombres de los directorios, y posicionándose en el comienzo del nombre

del archivo. Posteriormente se parsea el nombre del archivo, separando cada palabra del mismo usando como separadores los caracteres de espacio (“ “), slash (“/”), punto (“.”), guion (“-“) y guion bajo (“_”). Por cada palabra obtenida, se procede a buscar en la tabla de hash el lugar correspondiente a dicha palabra, y se introduce el path absoluto en la lista correspondiente. No se agrega el path a la lista únicamente cuando el path absoluto ya se encuentra en la tabla; Dos archivos con diferentes paths absolutos son considerados distintos.

7. Archivo de Índice.

El archivo comienza con el tamaño del arreglo a ser usado seguido de un slash (“/) en la siguiente línea (Linea 2). A partir de la línea 3, se encuentran las llaves de la tabla y los paths que contienen dicha llave, con el indicador 0 si es un directorio o 1 si es un archivo. Al encontrarse una cadena de caracteres en una línea, se sabe que es una nueva llave, las líneas siguientes contienen los paths absolutos que contienen la llave en el nombre del archivo o del último directorio en el caso de que el path no corresponda a un archivo. Una línea que contiene únicamente un slash (“/”) indica el fin de los elementos correspondientes a la llave.

8. Conclusiones.

Con la realización de este trabajo hemos aprendido un poco más como es el funcionamiento de un sistema operativo, en este caso de un buscador de archivos. Además aprendimos sobre la implementación y funcionamientos de hilos paralelos de un proceso, en contraste al uso de procesos padres e hijos que consumen más recursos. Aprendimos que los mismos recursos de sincronización usados para procesos pueden ser usados también para hilos.