Obtención de datos de un archivo de tamaño masivo en entornos de hardware limitados

Emmanuel Alvarado
Estudiante
Universidad Rafael Landívar
Carnet: 1109117
josue0ghost@gmail.com

Pablo Alvarado
Estudiante
Universidad Rafael Landívar
Carnet: 1104017
pablo.andre$_{a}$*lvarado@hotmail.com*

Erick Contreras
Estudiante
Universidad Rafael Landívar
Carnet: 1009017
ericon99sdb@gmail.com

Javier Mollinedo
Estudiante
Universidad Rafael Landívar
Carnet: 1062217
jmollinedoalonzo@gmail.com

María José Porres
Estudiante
Universidad Rafael Landívar
Carnet: 1080417
mariajoporres@hotmail.com

La implementación de procesos distribuidos cobra prescencia en tareas pesadas, que demandan gran cantidad de recursos. En esta ocasión tomamos en cuenta el caso de ficheros de gran tamaño, hablando de GB en un solo archivo que, inclusive, no pueden ser abiertos en editores de texto debido a su peso en la memoria principal por lo que se nos hace imposible ver su contenido. Se desarrolló un programa en C el cual, si bien no nos permite ver la estructura del contenido, sí nos permite obtener los datos del mismo. Se utiliza un archivo con formato:

c1,c2,c3,c4,...,cn
a,r,g,a,...,t

Una serie de letras aleatoreas organizadas en columnas en formato .csv

1. Introducción

Para leer y escribir archivos en C se pueden utilizar muchas funciones tales como fopen(), fwrite(), fread() y fclose(). Estas se encuentran en todos los compiladores de este lenguaje. A diferencia de Linux que utiliza las funciones open(), write(), read() y close() tanto para la lectura como para la escritura de ficheros. Estas son de bajo nivel.

2. Apertura de un fichero

Para modificar un fichero en C primero hay que abrirlo. Un fichero puede abrirse en modo texto o modo binario. La diferencia es que en un fichero en modo texto existen finales de línea y si se abre en modo binario, la información podría ser de cualquier tipo. En modo texto las funciones podrían buscar finales de línea y en modo binario no lo hacen.

La función que permite abrir un fichero es fopen() cuyos parámetros son:

- Nombre del fichero a abrir (path absoluto o relativo)
- Modo de apertura (si se pone "b" se abre en binario, sino, lo pone en modo texto)
  - r o rb: abre el fichero para lectura. Si no existe, da error.
  - w o wb: abre el fichero para escritura. El fichero puede no existir.
  - a o ab: abre el fichero para añadir datos al final.
  - r+, rb+ o r+b: abre el fichero para lectura y escritura. El fichero debe existir.
  - w+, wb+ o w+b: abre el fichero para lectura y escritura. Si el fichero no existe, lo crea.
  - a+, ab+, a+b: abre el fichero para lectura y escritura. La escritura comienza al final del fichero.

Esta función retorna un FILE*, un puntero a una estructura que contiene información sobre el fichero abierto. Se pueden hacer dos cosas con este:

- Comprobar si es o no es NULL. Si es NULL, ocurrió un problema y el fichero no se pudo abrir.
- Guardarlo. Debido a que este puntero es requerido por las demás funciones, debemos guardarlo ya que apunta al fichero que queremos modificar.

3. Lectura de un fichero binario

Para leer los bytes de un fichero podemos usar la función fread() que contiene los siguientes parámetros:

- void *ptr: un puntero a un espacio de memoria donde se guardarán los datos leídos del fichero.
- size_t size: el tamaño en bytes de los datos que vamos a leer.
- size_t nitems: Número de datos que queremos leer.
- File *stream: puntero al fichero que queremos leer.

Devolviendo el número de elementos leídos (nitems).

- Si hay más elementos que los mandados a leer (nitems), fread() devuelve solamente la cantidad de elementos que mandamos.
- Si hay menos elementos que los mandados a leer (nitems), fread() devuelve solamente los alcanzados a leer, que será menor que nitems.
- Devuelve 0 si no ha leído ningún elemento porque se llegó al final del fichero.
- Devuelve -1 si ocurrió un error durante la lectura.

4. Escribir en un fichero binario

Para escribir en un fichero podemos utilizar la función fprintf(), que envía un formato de salida a un stream. Tiene los siguientes parámetros:

- FILE *stream: El puntero al fichero en el que queremos escribir
- const char *format: La cadena que contiene el texto que se desea escribir en el stream. Opcionalmente, puede tener formatos embebidos de etiquetas que son reemplazadas por los valores especificados en argumentos adicionales subsecuentes y con el formato requerido.

Devolviendo el número total de caracteres escritos

- Devuelve -1 si ocurrió un error durante la escritura

5. Cerrar un fichero

Cuando terminamos de utilizar un fichero debemos cerrarlo con fclose(FILE *) Debido a que cuando se termina un programa el sistema operativo cierra todos los ficheros abiertos, podríamos omitir esta función, sin embargo, pueden existir problemas por no utilizarlo en programas más complejos:

- Hay un máximo de ficheros que se pueden abrir a la vez. Esto depende del sistema operativo por lo que no existe un límite estándar definido, pero si no se tiene el debido cuidado se puede llegar a este límite "fácilmente".
- Al cerrar el fichero nos aseguramos de que esté escrito en disco. Si no cerramos el fichero es posible que no se hayan escrito los datos en disco y los hayamos perdido para siempre.

6. Funcionamiento del programa

6.1. Especificaciones utilizadas

Se utilizaron las siguientes especificaciones para simular un peor escenario para el procesamiento del programa.

- Sistema Operativo: Linux Lite
- Espacio en Disco: 20 GB (Preferible)
- RAM: 1 GB
- CPU con 2 cores

6.2. Funcionamiento

Para lograr el objetivo del proyecto, se implementó el programa de la siguiente forma:

En el encabezado se declararon las variables globales:

- constante bufsize con valor 8388608 (8 MB)*
- char bytes[bufsize] que es buffer de lectura
- long res[128] para la respuesta del análisis (128 son los caracteres ASCII)
- char readdone bandera que el segundo thread utiliza para indicarle al thread principal que terminó de leer el archivo
- FILE *fp apuntador para manejo del archivo

*Se realizaron pruebas con diferentes tamaños y se llegó a que, con 8MB de buffer de lectura, se logra el menor tiempo de procesamiento. Incrementos o decrementos del tamaño de este buffer hacen variar los tiempos en ejecución de 3 a 4 segundos. La razón por la cual 8 MB fue nuestro tamaño de buffer óptimo fue la siguiente: En la RAM, para mover una sección de memoria, se requiere de un procesador de instrucciones y un bus de operaciones para realizar el traslado de un espacio de memoria a otro; y esta sección de memoria a manejar es procesada por un ciclo dentro del código del sistema operativo que, entre menor tamaño tenga esta sección, más veces debe ejecutarse y, a su vez, no puede tener un tamaño demasiado grande por las limitaciones de la cantidad de RAM en el equipo.

La función ReadFile() tiene como objetivo leer un archivo (en nuestros casos de prueba, de 2.5 GB o más). Esta tiene dos ciclos While, el primero lee la fila del fichero para descartarla ya que esta solo representaría el encabezado de cada columna, los cuales no nos interesan en estas pruebas. Este proceso se realiza leyendo 1 KB del archivo hasta encontrar el primer salto de línea mediante la comparación del caracter con su código ASCII (se compara con 10 o 13, los cuales representan fin de línea y salto de línea respectivamente). A partir de ahora el programa puede empezar a contabilizar la cantidad de veces que aparece cada caracter.

El segundo ciclo While se mantiene leyendo hasta la finalización del archivo (comparando cuando el buffer de 8 MB no es llenado por completo) asignando el valor *1* a la variable *readdone*.

En la función main() se ingresa la ruta del archivo el cual se va a leer y la ruta del archivo donde se escribirán los resultados, además, muestra en pantalla las cifras contabilizadas de cada letra. Se realiza un fopen() del archivo a leer y se crea un Thread para ReadFile(). Muestra el tiempo en ejecución y la contabilización de caracteres (guardadas en el arreglo *res* mientras que la bandera *readdone* sea igual a *0*. Al terminar, se realiza el fclose() del archivo leído y ejecuta fopen() al archivo de resultados para escribir con fprintf() los resultados obtenidos junto con el tiempo acumulado para la finalización del proceso cerrándolo con fclose() al finalizar de guardar

estos datos en disco.

7. Anexos

Enlace de descarga del código: https://github.com/josue0ghost/GetData-from-large-size-file

Referencias

[1] Lectura y escritura de un fichero binario http://www.chuidiang.org/clinux/ficheros/fichero-binario.php

[2] C library function - fprintf() https://www.tutorialspoint.com/c_standard_library/c_function_fprintf.htm

[3] Is using a bigger buffer useful? https://softwareengineering.stackexchange.com/questions/163931/is-using-a-bigger-buffer-useful