File System Project

Team: KAAT

November 20, 2020

Angela Garcia

Tolby Lam

Abishek Neralla

Kimberly Nivon

Table of Contents

Table of Contents

1 File System Planning
- 1.1 Project Information
- 1.2 Issues Encountered
- 1.3 Execution
- 1.4 Sample Output

2 Files
- 2.1 Description
- 2.2 Implementation
- 2.3 Tasks
  - 2.3.1 Creating a file
  - 2.3.2 Opening a file
  - 2.3.3 Writing a file
  - 2.3.4 Reading a file
  - 2.3.5 Repositioning within a file
  - 2.3.6 Deleting a file
  - 2.3.7 Truncating a file

3 Filenames
- 3.1 Description
- 3.2 How are we implementing it

4 Directories
- 4.1 Description
- 4.2 How are we implementing it
- 4.3 Tasks
  - 4.3.1 Search for a file
  - 4.3.2 Create a file
  - 4.3.3 Delete a file
  - 4.3.4 List a directory
  - 4.3.5 Rename a file
  - 4.3.6 Traverse the file system

5 Metadata
- 5.1 Description
- 5.2 How are we implementing it

6 Space Management

6.1 Description
6.2 Implementation
6.3 Tasks
7 Conclusion

1 File System Planning

1.1 Project Information

The KAAT File System uses a single-level directory structure and contiguous memory allocation.

Please read each section to see how each component of the file system was implemented. We were not able to implement any of the shell commands listed in the initial readme.

Link to repository: https://github.com/tolbs/csc415-file-system/tree/test-branch

We utilized a spreadsheet as a means to plan, track, and update each other on the project's progress over time.

Project status + timeline: Gantt Chart

1.2 Issues Encountered

1. The first obstacle that we had faced was figuring out the purpose of this assignment. As a group we found that we had a lot of research to do and that there were a lot of things that we did not understand about this project. We realized that the only way to actually understand the project we had to break it up piece by piece.
    a. We solved that issue by creating a document and compiling condensed notes pertaining to the file system. For each aspect of the file system, we provided a brief description of what it did, possible implementations as well as their pros/cons, and actionable tasks to implement it.
2. Meeting together outside of our class meetings. Not everyone in our group had easy access to reliable internet, and different work schedules so it made it super difficult to meet up. And due to social distancing we could not meet together as a group to work together. This taught us to be understanding and flexible with one another. Understanding that this project involved lots of communication and independent work as well.

3. A drawback of independent work that hindered development of this project was unfamiliarity with GitHub and its workflow in the context of a team setting. Prior to this project, GitHub was a place to host our project online and have different “save files” for the project. There were many times when group members were trying to implement the same thing independently and not having a way to see progress in real time. This resulted in a lot of redundant work being done. At one point, we used Google Docs to write code out in real time so we knew what each person was writing specifically. Our main approach to this project was each person taking their own stab at what’s needed to be done. In hindsight, we could have trusted each other more and relied on an individual person to be able to implement a particular feature and worked on a related feature in tandem.
4. One of our biggest conflicts that we did not realize at the beginning is that as a team we only focused on doing the “to do's” that the startup code had. This was a huge issue because it demonstrated that we did not really understand the assignment and that we underestimated the workload of the assignment . Realizing later that this project involved lots of our own implementation. Also demonstrating that we have a lot of freedom with this assignment.
5. One of the issues that we had for this assignment was trying to figure out how we wanted to allocate the memory and how we wanted to manage our freespace. There was lots of trial and error for this part of the assignment because we need a method that was easy to adapt and change.
6. Another issue was trying to figure out what the directory entries are.
7. In general b_io.c was one of the most difficult files to work on and had to restart multiple times. We were having the most difficulty trying to implement our space management. Understanding the logic and where freespace would be managed took us quite some time to develop.
8. Volume was difficult to work on the most. Each member tried to develop a volume file because we couldn't understand what we were doing wrong. Soon after we would bring our ideas together and notice that we all encountered similar issues and little by little started resolving them.

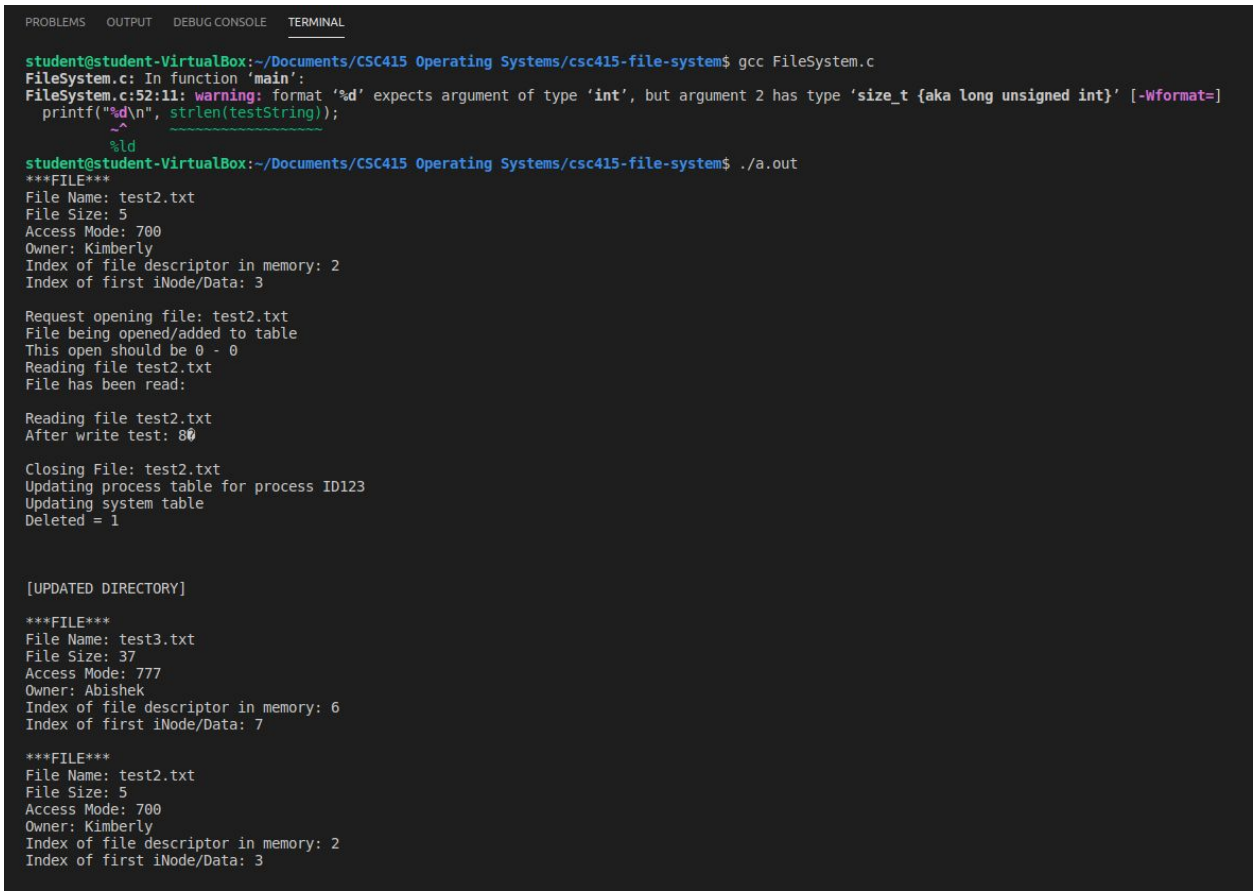
1.3 Execution

To run our project, type the following commands into the terminal:

```
gcc FileSystem.c
./a.out
```

Our driver program automatically creates a volume. A root directory will be made and all possible files will exist in this directory.

1.4 Sample Output



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

student@student-VirtualBox:~/Documents/CSC415 Operating Systems/csc415-file-system$ gcc FileSystem.c
FileSystem.c: In function ‘main’:
FileSystem.c:52:11: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘size_t {aka long unsigned int}’ [-Wformat=]
  printf("%d\n", strlen(testString));
          ~^     ~~~~~~~~~~~~~~~~~~
          %ld
student@student-VirtualBox:~/Documents/CSC415 Operating Systems/csc415-file-system$ ./a.out
***FILE***
File Name: test2.txt
File Size: 5
Access Mode: 700
Owner: Kimberly
Index of file descriptor in memory: 2
Index of first iNode/Data: 3

Request opening file: test2.txt
File being opened/added to table
This open should be 0 - 0
Reading file test2.txt
File has been read:

Reading file test2.txt
After write test: 8

Closing File: test2.txt
Updating process table for process ID123
Updating system table
Deleted = 1


[UPDATED DIRECTORY]

***FILE***
File Name: test3.txt
File Size: 37
Access Mode: 777
Owner: Abishek
Index of file descriptor in memory: 6
Index of first iNode/Data: 7

***FILE***
File Name: test2.txt
File Size: 5
Access Mode: 700
Owner: Kimberly
Index of file descriptor in memory: 2
Index of first iNode/Data: 3
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

File being opened/added to table
This open should be 0 - 0
Reading file test2.txt
File has been read:

Reading file test2.txt
After write test: 8

Closing File: test2.txt
Updating process table for process ID123
Updating system table
Deleted = 1



[UPDATED DIRECTORY]

***FILE***
File Name: test3.txt
File Size: 37
Access Mode: 777
Owner: Abishek
Index of file descriptor in memory: 6
Index of first iNode/Data: 7

***FILE***
File Name: test2.txt
File Size: 5
Access Mode: 700
Owner: Kimberly
Index of file descriptor in memory: 2
Index of first iNode/Data: 3


Deleting file: test2.txt
Removing memory block at index: 2
Removing file descriptor
Removing memory block at index: 3
Removing file data
Deleted = 0
23


[UPDATED DIRECTORY]

***FILE***
File Name: test3.txt
File Size: 37
Access Mode: 777
Owner: Abishek
Index of file descriptor in memory: 6
Index of first iNode/Data: 7

student@student-VirtualBox:~/Documents/CSC415 Operating Systems/csc415-file-system$
```

2 Files

2.1 Description

2.2 Implementation

We used a file control block that points to the file's location in the volume. It also contains relevant information pertaining e.g. file size. Maybe a generic struct File with a pointer to the file control block.

2.3 Tasks

We learned that seven basic operations compromise the minimal set of required file operations. This section will provide an overview of what our group has or hasn't implemented.
Green = implemented, yellow = partial, red = unattempted.

2.3.1 Creating a file

2.3.2 Opening a file

2.3.3 Writing a file

2.3.4 Reading a file

2.3.5 Repositioning within a file

2.3.6 Deleting a file

2.3.7 Truncating a file

3 Filenames

3.1 Description

Identifies a storage location in a file system

3.2 How are we implementing it

Case sensitive or insensitive?
Restrictions on characters? If so, which ones?

Filenames are stored as strings (case sensitive). A map associates that string to the relevant file control block containing a pointer to the file's location in the volume.

4 Directories

4.1 Description

The directory is a symbol table that translates file names into their *file control blocks.*

Directory Structures

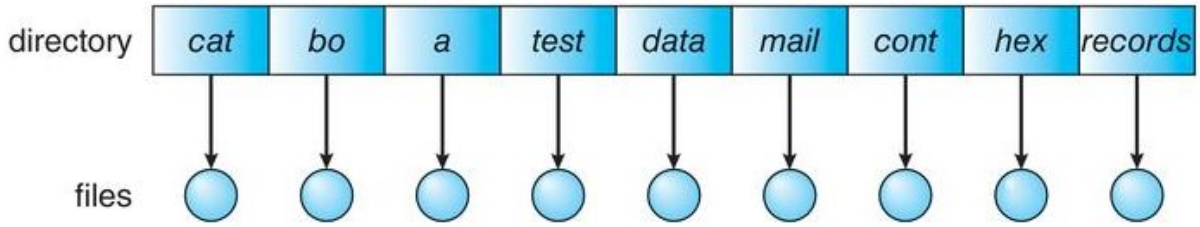


Figure 13.7 Single-level directory.

4.2 How are we implementing it

We implemented a single level directory.

There are two ways to implement a directory: linear list and hash table

Linear List

- Pro
  - Simple to program
- Con
  - Time-consuming to execute (linear search)

Hash Table

Linear list stores directory entries. Hash table takes a value computed from file name and returns a pointer to the file name in the linear list.

- Pro
  - Increased directory search time
  - insertion/deletion is straightforward
- Con
  - Provisions must be made for collisions

    - Generally fixed size and dependence of hash function on that size
        - A workaround could be a chained-overflow hash table where each hash entry can be a linked list instead of an individual value, resolving collisions by adding the new entry to the linked list

4.3 Tasks

4.3.1 Search for a file

We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.

4.3.2 Create a file

New files need to be created and added to the directory.

4.3.3 Delete a file

When a file is no longer needed, we want to be able to remove it from the directory. Note a delete leaves a hole in the directory structure and the file system may have a method to defragment the directory structure.

4.3.4 List a directory

We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.

4.3.5 Rename a file

Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.

4.3.6 Traverse the file system

We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals. Often, we do this by copying all files to magnetic tape, other secondary storage, or across a network to another system or the cloud. This technique provides a backup copy in case of system failure. In addition, if a file is no longer in use, the file can be copied to the backup target and the disk space of that file released for reuse by another file.

5 Metadata

5.1 Description

Contains information pertaining to each file. May include:

- Length of data or file size (in bytes or whatever)
- Owner
- Permissions (read/write/execute,etc)
- time/date created
- time/date last modified

5.2 How are we implementing it

We’re using a struct that contains the relevant information.

6 Space Management

6.1 Description

We need to be able to organize files and directories.

- Keeping track of which areas of the disk/block belong to which file
    - Which areas are used vs which areas are free

6.2 Implementation

We used a continuous implementation. In order to manage free space, we used an array with number of elements equal to the number of blocks in the volume. Array element will contain 1 for used space and 0 for free space (unused).

6.3 Tasks

- Allocate/Reallocate space if a file is created/deleted
    - Figure out which spaces the file takes up
        - Helper function 1 (counts bytes)
            - Might be dependent on metadata for file
        - Helper function 2
    - Allocate space if file is created
    - Deallocate space if file is deleted
- Manage free space
    - Something something

7 Conclusion

The file system project was the most rigorous project we've done. While it was extremely challenging and our group was not able to implement a working file system, the process of research and development reinforced what we learned conceptually. Tying back all the previous coding assignments to the lecture, we saw how it all comes together in the end.

While learning and understanding concepts taught in class is important, we also learned other important skills while doing this project- teamwork and communication. Over 14,000 individuals and over 1,300 companies contribute to the Linux kernel, making it one of the largest collaborative projects ever. Even though our project is not at the scale, we experienced how difficult it is to work on something together. Recognizing our own shortcomings is the first step to growth, and we can continue to work towards overcoming that hurdle and onto the next one. Even though we aren't fully successful in our implementation, we are proud that we were communicative, accommodating, and understanding of each others' circumstances and schedules.