Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

CSC-415.03

“Caffeinated Crew”

File System Project Report

Issac Moreno (921984788) - GitHub Master

Anisah Chowdhury (921677676)

Malieka Sutaria (922888691)

Katy Lam (921922518)

GitHub: Link to: Central GitHub

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

Assignment – File System Project

Description:

The file system project is to create a basic file system that provides the basic file management functions such as file creation, deletion, reading and writing. Our project contains a few basic components such as; file system structure, operations, and error handling.

Approach:

The file system is organized into blocks of fixed size, with each block containing a fixed number of bytes. The first block of the disk is reserved for the superblock, which contains metadata about the file system, such as the block size, total number of blocks, and the location of the root directory. To begin this project our group first designed the data structures to represent our files, directory, as well as the overall file system layout. Directory entry (dirEnt), file control blocks(FCB), and the file system itself (FileSystem). The use of dirEnt, was used to represent the directory entry, containing out metadata such as name, size, timestamps, and flags indicating whether it's a directory or a file. FCB represents a file control block, which contains information about open file, read/write position, and file status. Code was written to initialize the file system. This involved setting up the root directory, allocating space for the data blocks and other needed setup. As well as initialize the disk blocks and map them to the file system structure. Some of the file operations include: fs_create(const char *filename), fs_open(const char *filename), fs_read(int fd, void *buffer, size_t count), fs_write(int fd,

const void *buffer, size_t, count) and fs_close(int fd). The usage of fs_create(const char *filename) is to create a new file with the given name. The usage of fs_open(const char *filename) is to open an existing file for reading or writing. The usage of fs_read(int fd, void *buffer, size_t count) and fs_write(int fd, const void *buffer, size_t, count), was to read and write data from a buffer. File system operations allow users to create new files, open existing files for reading or writing, read data from files, write data to files, and close files. Some of our directory operations include: fs_mkdir(const char *pathname), fs_ls(const char *pathname), and fs_rmdir(const char *pathname). The usage of fs_mkdir(const char *pathname) is to create a new directory with the given name, similar to our fs_create(const char *filename) operation. The fs_ls(const char *pathname) directory operation is supposed to list all the contents of the directory. The fs_rmdir(const char *pathname) operation is used to remove a directory. The directory operations are implemented using functions that manipulate the file system's data structures and interact with the disk. The implementation includes error handling for basic scenarios, such as disk full, permission denied, and file not found. File system metadata is managed to ensure efficient use of disk space and proper tracking of the file and directory information. For our metadata management we implemented functions such as maintaining freespace, handling permissions and timestamps.

Our File System Design:

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

This includes some of our approaches to VCB, FB, and Directory Entry from when we first started the file system

- Volume Control Block:
  - For our volume control block, we have used the size 20 to define the size of the name array. Name[size] is used as an array of characters which represents the name of the volume which is 20 in our case. The blockSize integer will represent the max block size, and will be standard for all the blocks in the volume. Int signature is used to represent the index of the root directory in the volume, this will help us locate the root directory faster. To represent the total number of blocks in the volume we used totalBlock, this will also help us check if the volume is ready to be used. A freeBlock in our structure is used to represent the number of free blocks in our volume. This will also help us keep track of the available space in our VCB.

Free Space Structure Description:

- Tracking Free Space:
  - We want to use a bitmap to track free space.
    - Bitmaps are useful in tracking free space given that it is an array of bits, which each bit represents a data block.

- We will have a set bit and a clear bit, where the set bit represents which blocks are currently filled or being used and the clear bits are unoccupied or a free block.
- It also efficiently tracks the first free blocks, and the free blocks that are consecutive with each other.

➢ Drawback of bitmaps:

- Due to limited memory usage, it can be hard to keep track and use on large volumes of data. (This can be resolved or improved on by dynamically allocating space outside the VCB)
- This function utilizes the freeBlocks variable in VCB to keep track of disk space and help manage reading/writing requests.
- If freeBlocks is 0, totalBlocks will be incremented. If a block is freed, freeBlocks is incremented.

Directory Description:

- Directory Entry:

➢ Our directory entry will include the location, name, size, directory flag, and a date and time stamp. We define a size beforehand that it passed into the name. The directory flag is used to determine if we are in a file or directory. The date has a predetermined size because it never changes.

- **Directory Entry Contains:**

1. Name: The name of the file or directory, this is stored as a string with a max length of MAX_NAME_LENGTH.

2. Size: The size of the file or directory in bytes.
3. Date: The last modification of the file or directory, which is stored as a time_t value as well as when it was last accessed.
    a. The date is in readable format
        i. In example 04/07/2024 12:40:30
4. Location: The location of the file or directory within the file system itself .
5. isDir: A flag indicating whether the entry represents a directory(1) or file(0).

❖ Metadata in File System:

➢ File/Directory names

➢ File/Directory sizes

➢ File/Directory locations

➢ File/Directory creation/modification dates

Free Space Structure Definition:

createDirectory

- This function creates a new directory entry in the file system. It checks if there is enough free space to create a directory, then creates a new directory entry with the name, size, date, location, and type. It updates the parent directory entry to show the new changes of the new directory. It also updates the free block count in the Volume Control Block

- **Parameters for createDirectory**
    - Name: the name of the new directory
    - vcb: Pointer to the Volume Control Block
    - parentDirLocation: parent directory in the file system's location

VCB Structure Description:

Operating systems use the Volume Control Block (VCB) structure to handle important data related to a given volume. This includes information about the volume's name, unique identifier, size, file system type, mount status, mount point directory, availability of free space, details about allocation tables, access control information, and file system metadata such as the location of the root directory. As it serves an important part in managing and organizing volumes inside a system, this structure is essential for the effective management and access of data on storage devices.

1. **Structure Definition:**

Free Space Structure Description:

We used bitmap as a way to store our free space and use it to manage the storage and resources efficiently.

Steps taken:

- Allocate memory for bitmap by the total number of blocks
    - Check if current bit is in use or not

    - Initialize variable to index of free space
- Store information in the VCB (volume control block)
- Free memory after it is not in use

Information stored in this way is stored into each block where each block in the memory or in the disk will contain information of the block next to it in the LBA and the block after that in the file or if there is any free space system. Doing it this way also holds the data knowing where it begins and ends and the total amount of free blocks that are on the disk and in the memory.

The functions from the freespace system will be working together to make it easy for us to use the storage space without worrying how it is being organized at the lower levels. This way, it will hide the process and how data is being stored into each of the individual blocks, letting it do whatever it needs and letting us deal with the blocks of data afterwards and making it less complex for us to work with.

In addition, we went ahead and consolidated two functions (bitCheck & allocateSpace) to streamline the process. Originally we wanted to keep them separate, but bitCheck will only be used in allocateSpace. So, we added the arguments from bitCheck into allocateSpace.

Directory Description:

The directory structure in our file system is represented by the dirEnt structure. Each dirEnt entry contains information about a file or directory, including its name, size, date it has been modified, location, and if it is a directory or a file. We also have a field

that stores the date in a readable format. The directory entries are stored into an array, and then saved into the disk.

With each directory entry containing:

- Name: The name of the file or directory, this is stored as a string with a maximum length of MAX_NAME_LENGTH.
- Size: The size of the file or Directory in bytes.
- Date: The last modification date of the file or directory, which is stored as a time_t value and well as when it was last accessed.
- Location: The location of the file or directory within the file system itself.
- isDir: A flag indicating whether the entry represents a directory(1) or file (0).
- Dates: The date is in readable format
    - In example 04/07/2024 12:40:30

The root directory is initialized with at least two entries “.” and “..”. These entries represent the current directory and the parent directory, respectively. They are an important part of the file system structure navigation.

Group Work Table:

Names:	What we did:
Issac Moreno	- initVCB function

	- Free space initialization - Dump analysis - Write up - Planned meet up times - Code clean up/organization - GitHub management - Debugged code and worked on connecting functions made by other group members so it will properly work together - Was there to help others on on their functions - Worked on fs_stat, fs_delete, and fs_rmdir for milestone 2 - Participated in in person meet ups to plan and discuss what needed to be work on and who should do what - Open communication on what the progress is and what else needs to be done

Katy Lam	- Worked on the freespace and plan what should be inside and structuring of the freespace function - Worked on organizing the docs partially - Worked on the free space structure description - Worked on reviewing and double checking if the directory and the dump is correct and make sure we had everything - Finding the reasons behind some of the errors - Written the parent directory entry - Worked on the isFile and isDir for milestone 2 - Worked on the close function for milestone 2 - Worked with the parsepath - Did the writing of the approach for

	functions in the helper class and how some of the functions worked for the file system report. - Took screenshots and explained the problem and resolution of the bugs and errors the group encountered while working - Did the screenshots of the output and compiling - Planned time and dates and met up in the cs lab to work on file system project together - Layout milestone 2 functions and help figure out how many functions and which functions team members should be working on - Helped identify bugs - Available to communicate and answer questions if needed + double checking if what was needed are included

	- Provided sources and went out of way to ask for help and work with external to debug and problem solve - Work alongside with team members in class, in person, and online to figure out the possible causes of some bugs and errors

Anisah Chowdhury	- Free space
	  - Space allocation, checking if there's free space for a certain number of blocks.
	  - Header description
	  - Debugged with teammates on space allocation
	- Write up
	  - Teamwork efforts
	  - Added onto freespace description

Malieka Sutaria	- VCB structure - Created struct of VCB, with

	the appropriate descriptions - Created dirEnt structure - Initialization - Free space bitmap - VCB - Root Directory - Files worked on - b_io.c - directory.c - fsinit.c - fsVCB.c - mfs.c and mfs.h - Write up - VCB Structure description - Structure Definitions - Directory Description - Milestone 2 project updates and description of functions implemented. - Detailed description as well as its functions

	- fs_setcwd - fs_getcwd - fs_isDir - fs_isFile - fs_mkdir - fs_readdir - fs_closedir - fs_stat - fs_delete - fs_rmdir - initDirectory - isEntryUsed - Communication - Set up times to meet at the cs lab - Distribute the workload, and helped out where needed. - Took screenshots of the errors received as well as its resolution - Explained what the error was and what was done to

	fix the error and come up with the resolution - Fixed simple warning our compiled code gave due to simple issues.

Things we planned during milestone 1:

Function createDirectory

This function creates a new directory entry in the file system. It checks if there is enough free space to create a directory, then creates a new directory entry with the name, size, date, location, and type. It updates the parent directory entry to show the new changes of the new directory. It also updates the free block count in the Volume Control Block.

Parameters for createDirectory

- Name: the name of the new directory
- vcb: Pointer to the Volume Control Block
- parentDirLocation: parent directory in the file system's location

newDirEntry creates a new directory entry with a specified name, size, date and time, and isDir flag is set to 1, indicating it is a directory.

The write directory entry newDirEntry writes the new entry to the disk in the specified location.

Update Parent Directory , reads the parent directory entry from the disk at the parentDirLocation, writes the updated entry back to the disk.

Update Free Blocks, decrements the freeBlock count in the VCB to reflect the allocation of a new block for the directory and writes the updated VCB to disk.

Milestone 2:

For Milestone 2 of the file system project, our team focused on implementing and completing several key functions. The work was split up and distributed among the drop as such. Malieka worked on fs_setcwd, fs_getcwd, and fs_mkdir. fs_setcwd will set the current working directory to the specified path, fs_getcwd will get the current working directory, and fs_mkdir will create a new directory. Issac worked on fs_stat, fs_delete, and fs_rmdir. fs_stat will get the stats of a file, fs_delete will delete a file, and fs_rmdir will remove the directory. Katy worked on fs_isFile, fs_isDir, and parse path. fs_isFile will check if a given path is a file, fs_isDir will check if a given path is a directory, and parse path will parage a given path. Anisah worked on fs_opendir and fs_readdir. fs_opendir will open a directory, and fs_readdir will read a directory.

Functions (has to be functional):

- fs_setcwd(Malieka)
- fs_getcwd(Malieka)
- fs_isFile (Katy)
- fs_isDir (Katy)
- fs_mkdir(Malieka)(Issac)

//this function will probably need additional help

- fs_opendir (Anisah)(Malieka)
- fs_readdir (Anisah)(Malieka)
- fs_closedir (Katy)
- fs_stat (issac)

Optimal Functions (at least be complete) :

- fs_delete (issac)
- fs_rmdir (issac)

//this function will probably need additional help

Parse Path (Katy) (Issac)

Priority for milestone 2 was to finish all of the functions from fs_setcwd to fs_mkdir before we start working on the delete and rmdir. We also needed to prioritize the parse path because a lot of the functions needed to use the parse path for themselves to work.

Detailed Description of Functions and its Properties:

fs_setcwd

- Serves the purpose of adjusting the current working directory within our file system based on our pathname. It also allocates the memory to store information regarding the directory entry though returnInfo structure. This structure guides with parsing the given pathname to extract the details about directory entry. If parsing is done successfully, this function verifies the path to make sure that given directory exists. If the directory does not exist, an error is then printed and the function gives a return value of -1. This function also checks to see if the function is a directory, if it is not, then based on its contents we give a specific error message with a return value of -1. If all the checks pass, the function updates the global variable cwd to point to the directory entry corresponding to the provided path. Memory cleanup is done by freeing the allocated memory for returnInfo structure before returning 0. 0 is used to indicate successful completion of the operation.

fs_getcwd

- This function retrieves the current working directory within the file system by copying its name into a provided buffer. It utilized the global variable cwd to access the directory's name and checks to see that the buffer is large enough to accommodate the directory name. If this function is successful, it returns a

pointer to the buffer containing the directory name, giving it easy access to the current working directory's path.

fs_isFile

- This function determines whether a given filename corresponds to a file within the file system. Allocating memory stores information about the file by using the returnInfo structure. This structure is used to parse the given filename and extract essential details about the directory entry. Once the parsing has been completed, the function checks if the file exists. If the file does not exist, it prints an error message and returns 0. Then, it verifies if the identified entry represents a file rather than a directory. If it is a file, the function returns 1, otherwise it returns 0. The memory allocated for the returnInfo structure is deallocated before the function returns the result. This provides a clear indication if the given filename corresponds to a file in the file system.

fs_isDir

- This function serves to determine if a given pathname corresponds to a directory in the file system. First checks to see if the provided pathname corresponds to a file using the fs_isFile function. If an error occurs during this, it returns -1. If it returns 1 then the pathname corresponds to a directory, and 0 if it does not. This is determined on the results given by the fs_isFile function. This function helps with verifying the type of specific entry in the file system, which provides essential information for various file system operations.

fs_mkdir

- This function is responsible for creating a new directory within the file system. It first allocates memory for storing information about the directory using the returnInfo structure. This structure helps in parsing the provided pathname to extract essential details about the parent directory where the new directory is to be created. After parsing, the function checks if the specific path is valid and if the directory already exists. If the path is invalid or the directory already exists, error messages are printed, and the function returns -1. Otherwise, it initializes a new directory entry with the provided name, sets its attributes such as size, access time, modification time, and writes the new directory to the disk. Memory allocated for the returnInfo structure is then deallocated before the function returns. By doing so, this provided proper resource management.

fs_opendir

- Takes a pathname and allocates memory for the information returned by parsepath. Then it allocates memory again for the file directory. Both of these allocations are checked for errors. Parsepath is called and is checked if it is 0. If it is 0, then the operation was successful and we can populate the directory with appropriate information. Otherwise, an error is returned.

fs_readdir

- This function serves as a means of reading entries in the file system. It uses a directory stream, which is a file descriptor that is opened with O_RDONLY and refers to an open directory. It traverses its entries in order to read in the next valid directory item. It first reads the number of items in the directory, starting from

where the dirEntryPosition field of the open directory stream specifies. For each item, it checks if it is used, that is, a valid directory item. If it is, it reads in a fs_diriteminfo structure associated with the open directory stream, storing the name and file type: whether it is a directory or not. After reading in the structure, the dirEntryPosition is updated to point to the next directory entry for the next call. If no valid entry is found, it returns NULL. This function provides sequential access to directory entries. Among the operations it enables are listing directory contents, traversing directory structures within the file system, etc.

fs_closedir

- The function closes a directory stream associated with an open directory stream using the fs_opendir function. When it receives a pointer to a directory stream, it changes the access time of the associated directory entry to the current time. It then frees the memory used by the fs_diriteminfo structure associated with the directory stream and frees the memory used by the directory stream itself. The function then returns 0 to indicate that the directory stream was closed successfully. The function, therefore, ensures proper cleanup of resources associated with an open directory stream that has been opened using the fs_opendir function, contributing to efficient memory usage in the file system implementation.

fs_stat

- The function retrieves information that is statistical in nature about a file specified by the path within the file system. The function needs a file path from the system

as well as a pointer to a fs_stat structure, where the information of the file shall be stored. The function first allocates memory for a returnInfo structure, which shall help parse the file path to derive the details about the file's directory entry. It then checks if the directory entry exists. It prints an error message if the file does not exist and then returns -1. The function else fills the fs_stat structure with information like the file name, size, block size, number of blocks, access time, modification time, and creation time. The function then deallocates the memory allocated for the returnInfo structure and returns 0, which signifies successful retrieval of file statistics. The function provides information about the file required at various places in the file system and so correctly handles memory in all such places.

fs_delete

- The function is designed to delete a file from the file system based on its filename. The goal of this function is to remove a file entry from the file system, update the directory entry to a known free state, and update the parent directory accordingly. It probably initiates by parsing the provided filename to extract information about the file's directory entry. Then, it verifies if the file exists. If the file does not exist, it prints an error message and returns -1. It then estimates if the specified filename truly represents a file and not a directory using the fs_isFile function. If it indeed is a file, it removes the file by updating the directory entry that is associated with the specified file to a known free state. Finally, it updates the parent directory and frees the memory assigned for the returnInfo

structure before returning 0 to indicate successful deletion. This function is basic to the file system file management and enables a user to remove unwanted files and free up available storage space.

fs_rmdir

- The function is meant to remove a directory from the file system. The function first parses the provided directory pathname to get details about the directory's parent entry. Then it checks whether the directory exists. If the directory does not exist, it would print an error message and return -1. It then checks whether the provided pathname is a directory and not a file using the appropriate checks. If it is confirmed to be a directory, the function would then go ahead and then remove the directory by updating the directory entry to a known free state and updating the parent directory accordingly. It would then free any memory allocated and return 0 to indicate that the directory was successfully removed. The function is crucial in organizing directory structures within the file system, enabling users to remove the unnecessary directories and have their files well-organized.

parsePath

My approach to parsePath was similar to how I interpreted how strTok works if it was taken apart. It was also similar to what I did for assignment 4, so I used that as the base of what I wanted to do.

- I first check to see what a path of a directory or file contains and how I should be splitting it. I realize that each of the directory is being split by “/” so I went ahead and created a conditional statement looking for the “/” (delimiters)

- I later added the "," as a possible delimiter due to the rules of how parsing a path works
- After that, I was figuring how I will be accessing the whole path, so having a loop that will keep looping until everything was tokenized was my plan of approach
- I was also thinking of the possibility of needing a buffer and a pointer as well to keep track of what we are supposed to be doing. All information are then copied and stored into the buffer as well as the struct
- We then "cleaned up" the functions by freeing anything that needs to be freed or anything that needs to be set to NULL.

I wasn't sure what I was supposed to return until I looked more into which other functions will need to use parsePath. In the end my group and I decided it was best if it can check for a valid parse path and return 0 or -1 when we need to.

Changes that happened when working on this function:

During the time of creating the parsePath, it went through different design and use changes. The first version of the parsePath was a function that just modified the original path, meaning whichever path went in, it will come out as a new parsed char pointer rather than have it be copied then modified and placed into something else. During the second development of the function, I decided to redo the way the function works. Instead of having it directly modified, I had it so it took a copy of the path and then had it go through the conditions before storing it into the buffer as well as the struct. It then would return a buffer with the path

parsed and be ready to use for other functions. Then we talked together about the parse path as a group to figure out what we need to modify and what needs to be changed. My teammate actually needed it to return as an Int instead, to show if the part being parsed was valid or not and if it will be taking in the current pointer as a valid thing to be working with. I then copied what I had from the parse path and changed some conditions and what it will be returning but otherwise everything being stored and allocated are the same.

Helper functions in helpers file:

The helper functions were written based on how we plan to structure our work. Possibility that specific information might need to be called multiple times and are needed to be checked within other functions.

getVCB

- getVCB is one of the helper functions used in order to retrieve the VCB from the file system. It will find and take the VCB and return it.
- The approach to this was to allocate memory for the VCB
- Read the VCB from the disk, which we will redirect the pointer towards so we can mark the location of the VCB
- We will then return the pointer of the VCB

getRootDirectory

- getRootDirectory is one of the helper functions that is used in order to retrieve the rootDirectory from the file system
- Similar to how we get the VCB, we will also have this returning the pointer to the root directory. We will also have a “token” which will contain the name of the entry we are trying to search for.
- This function will also take in the VCB for its own uses.
- It will be allocating memory for the root directory
- Read from the disk to find the root directory and then we will have a pointer that will be keeping track of where that is located.

seekDirectory

- This functions will take the directory entries and illiterate through them
- We will then be comparing all the entries to the name of the token to see if it matches. This will return the pointer to the directory entry if a match was found, otherwise, it will be returning NULL.

readDEntry

This function will read an entry from the disk to to the memory

- It will allocate space and memory for the entry
- Will return the pointer of the Entry once it is being read

getIndexOfEntry

- This function will take the index of an entry in the directory

- The parameters will be the pointer to the directory and a pointer to the entry so we can look through both to get the information we need.
- This will return the int, meaning it will return the index of the entry and -1 if it fails to find a valid entry
- It will be iterating through all the directory entries and compare the entry pointer with the entry provided

initDirectory

- The initDirectory function is responsible for initializing a directory in the file system. It takes a parent directory as an argument and initializes the directory with two special entries: "." and "..", representing the current directory and the parent directory, respectively. It also initializes the rest of the directory entries with empty values.
    - It creates an array of dirEnt structure called directory to hold the directory entries. It loops through the directory array starting from index 2.
- For the implementation in mfs.c, initDirectory takes a pointer to a dirEnt structure) and sets its fields to initial values. This function is called before creating a new directory entry to ensure that all fields are properly initialized.

isEntryUsed

- In our code the isEntryUsed function is used to check if a directory entry is being used.
    - In our fs_readdir function

```
206  // Read directory
207  struct fs_diriteminfo *fs_readdir(fdDir *dirp)
208  {
209
210      int entries = dirp->directory->size / sizeof(dirEnt);
211      for (int i = dirp->dirEntryPosition; i < entries; i++)
212      {
213          // Only return a used entry
214          if (isEntryUsed(&(dirp->directory[i])) == 1)
215          {
216              strcpy(dirp->di->d_name, dirp->directory[i].name);
217              dirp->di->fileType = dirp->directory[i].isDir;
218              dirp->dirEntryPosition = i + 1;
219              return dirp->di;
220          }
221      }
222      return NULL;
223  }
224
```

- isEntryUsed(&(dirp->directory[i])) is used to check if the directory entry at index i in dirp->directory is being used.
    - If the entry is being used, it is then returned 1. Then it is copied to dirp->dir and the function then returns. If the function is not being used, the loop continues to the next entry

IsDir

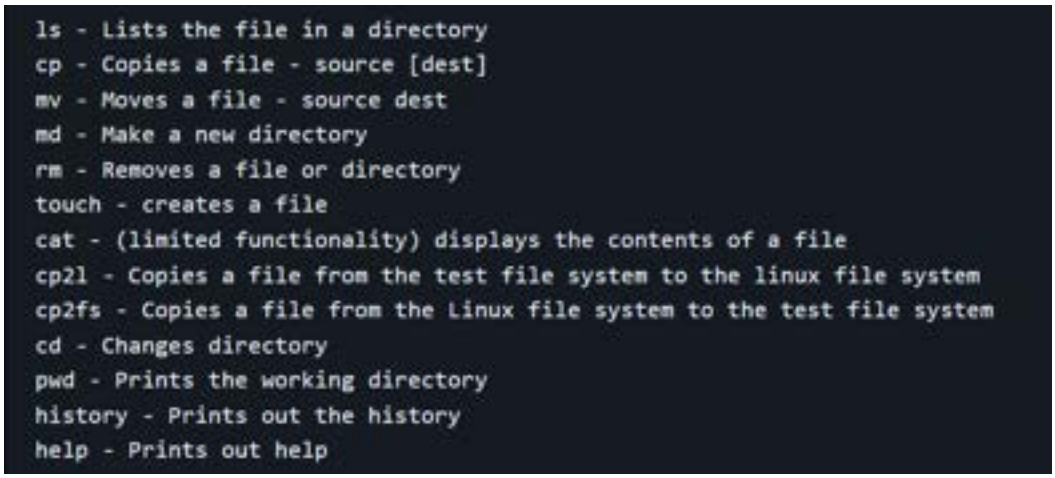
```
195  }
196  //Check if a directory entry is a directory
197  //Returns 1 if it is a directory, 0 if it is a file
198  int isDirectory(dirEnt* parent){
199      if(parent->isDir == 1){
200          return 1;
201      }
202      return 0;
203  }
```

- This function takes dirEnt pointer parents as an argument and checks if the isDir field of the dirEnt structure is set to 1.
    - If it is set to 1, that indicates that it is a directory.
        - If it is a directory, the function returns 1, otherwise it returns 0. This indicates that it is a file.
    - We used this function to determine

IsFile

- To check for a file, my first approach was to figure out what I am looking for when looking for a file.

Screenshots showing each of the commands listed:



fs_closeDir

My approach to this function was to check what the argument/parameters is taking in and using that info to figure out what I am having to free or what I need to check to be empty or not.

Research, I found out that the keyword “dirp” is often used as a variable to refer to the directory stream. This gave me an idea where I am looking to and where I am free.
I based the function off how I freed in assignment 5, which I check if dir was NULL and check if the directory is empty. If those conditions were to be true, we would have to return -1 as in there was an error closing the file. Otherwise, we are able to free and set pointers to NULL in wherever it is needed.

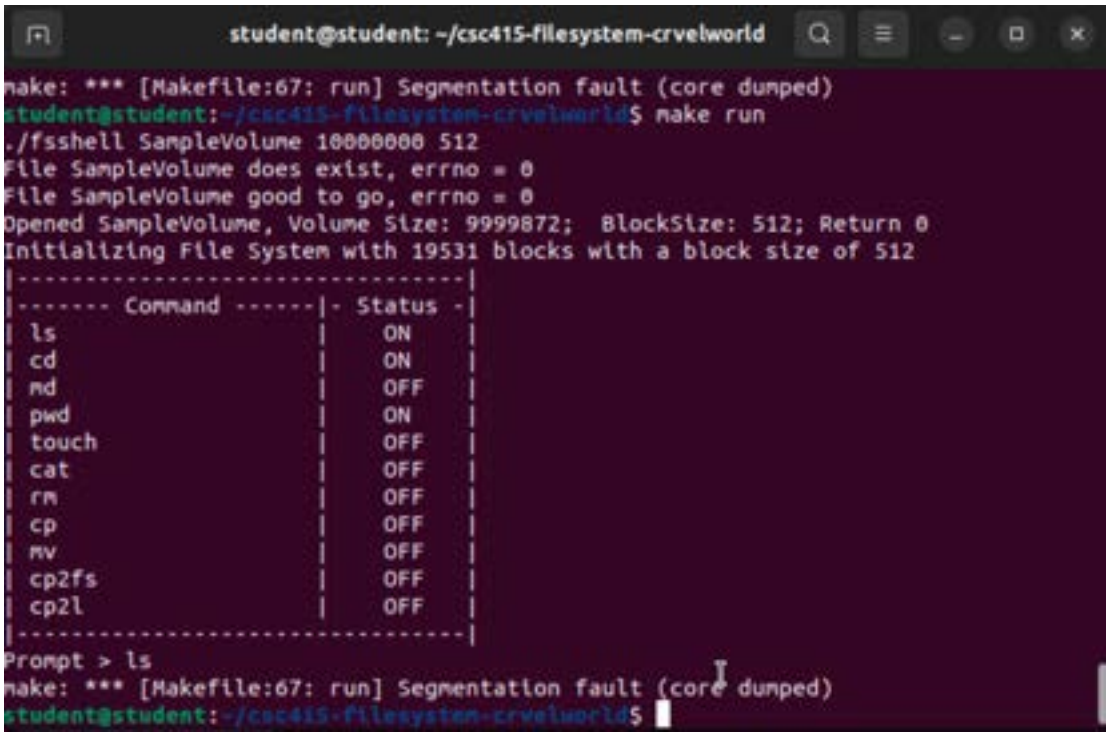
Issues and Resolutions:

- **Issues with Freespace:**
  - Free what needed to be freed, but still had some memory leaks.
    - As we went along with our code, we were closely checking on the memory management step by step to ensure that the memory was being managed correctly, and checked in the places where memory might need to be freed.
  - Had some trouble figuring out how to allocate properly, could not decide if it was best to allocate first based on the total amount of blocks or do it by the amount of free space available in the blocks to allocate the memory for.
- **Freespace Resolution:**
  - We figured out the best solution to the allocation was to check the block itself to see if it contained any bits before so that we can find the location of the freespace in the beginning.

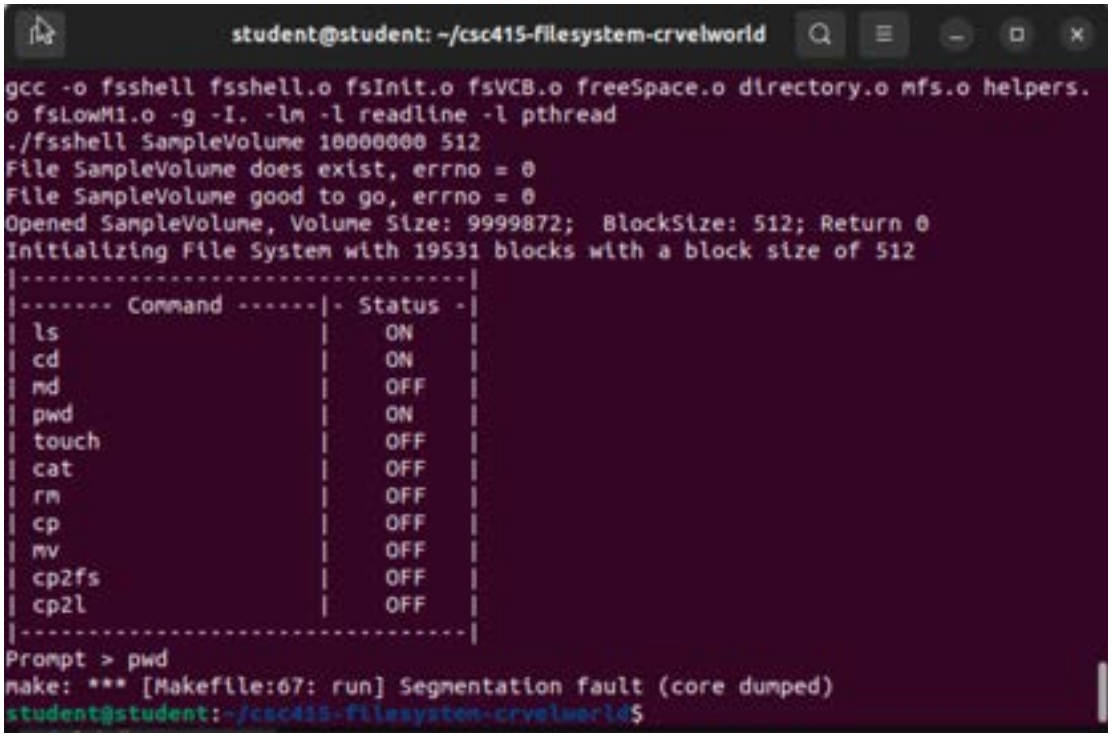
- **fs_opendir**
  - Issue: When making the function, we neglected to allocate space for the returnInfo. We made the mistake of assuming that returnInfo would be needed when we read the directory. This caused a segmentation fault because the information had nowhere to be placed once this function was called.
  - Resolution: To solve the segmentation fault, we allocated space for the returnInfo in the opendir. This was the first thing we did, even before dir allocation. Initially we placed it after parsePath, but quickly realized this is no use because returnInfo is one of the parameters we need in order for the function to return the correct value.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

Project Errors:



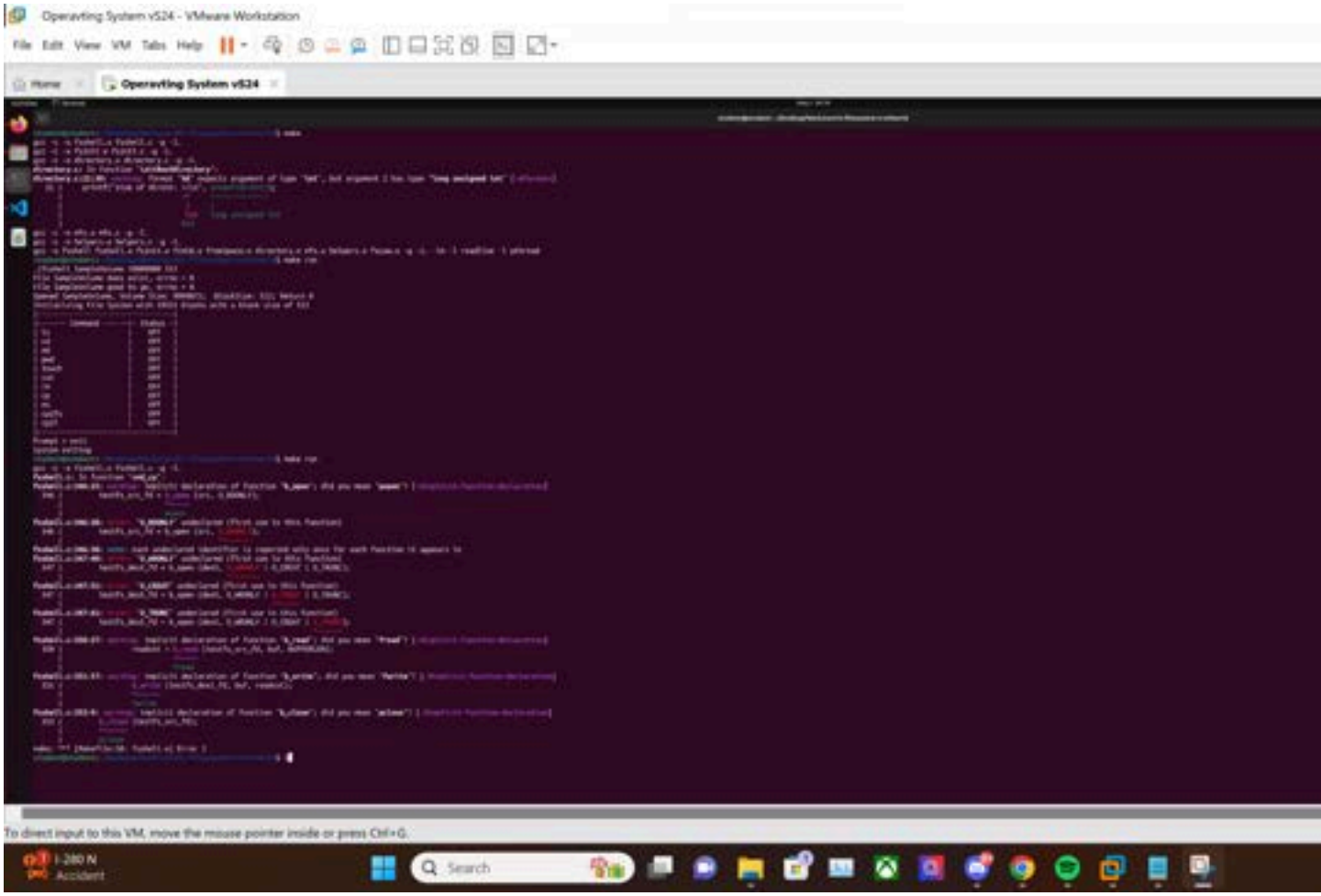
- Ls causes a segmentation fault



- PWD causes a segmentation fault

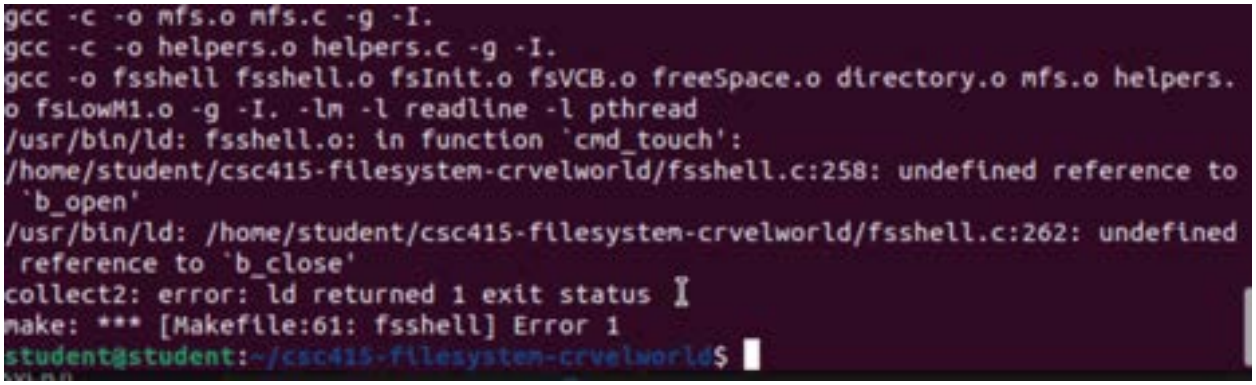
Error of cp command not recognizing some of the functions and conflicts with some other, causing it to crash when trying to do a new make. The errors also show that I am not calling a specific function that needs to be called, even though the function is properly written into the b_io file.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



This issue was resolved by properly linking the b_io.o into the makefile. Because the object was not accessible therefore the file was not able to read into the functions.

To fix this issue we needed to add our B_io.h file to the fsshell.c, but then we faced another error. Shown below

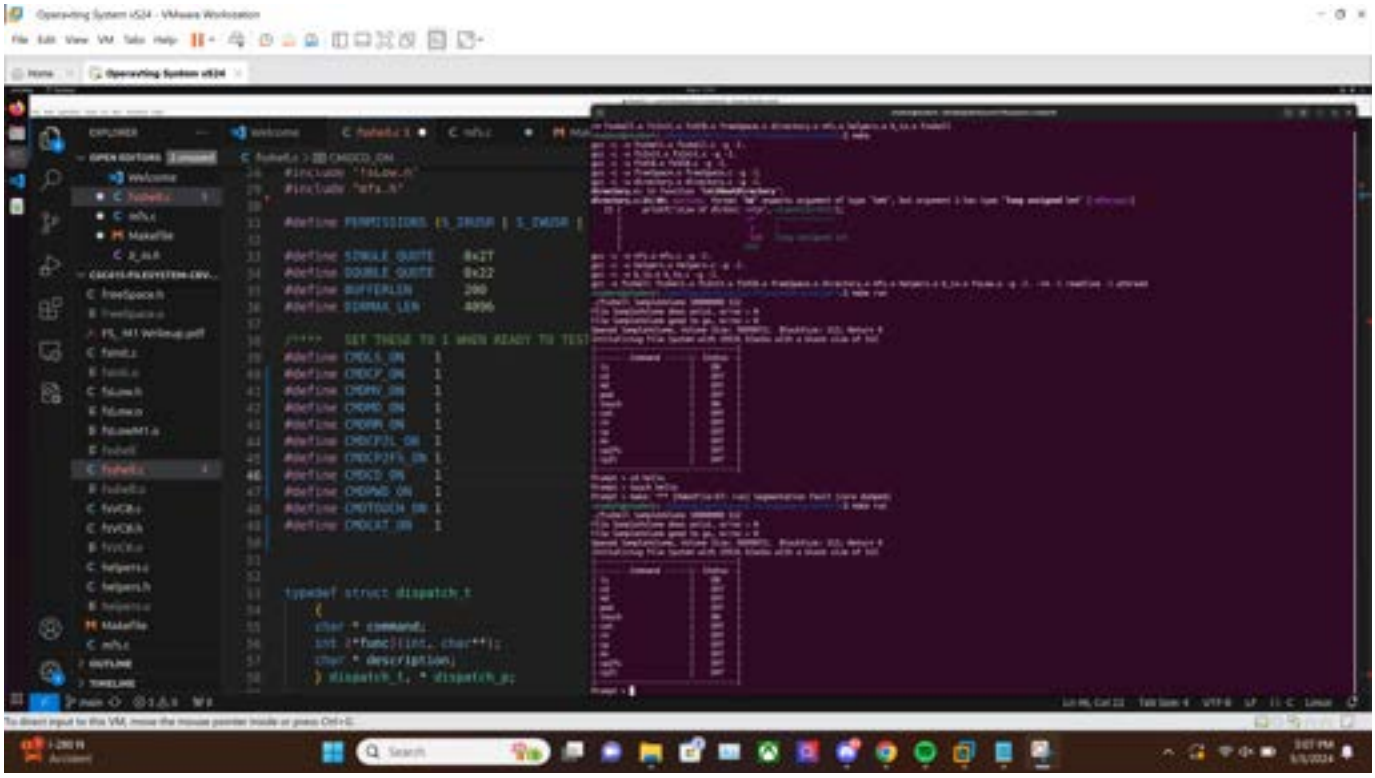


This error was due to the fact that the makefile did not have our b_io.h file. To fix this we touched out makefile and added more objects into the “ADDOBJ=” area. Once b_io was added we were able to run make run and have our project compile properly as before.

Another issue we are facing is the ON and OFF status is not being consistent. Meaning that the table does not update when we turn on and off some of the commands. Even though they are able to be called from the terminal.

Wasn’t able to fix it properly. Had to reclone the file from github and it worked. Wasn’t sure if the issue was caused by the changes not syncing from terminal and vscode, but on the backend it was being updated.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

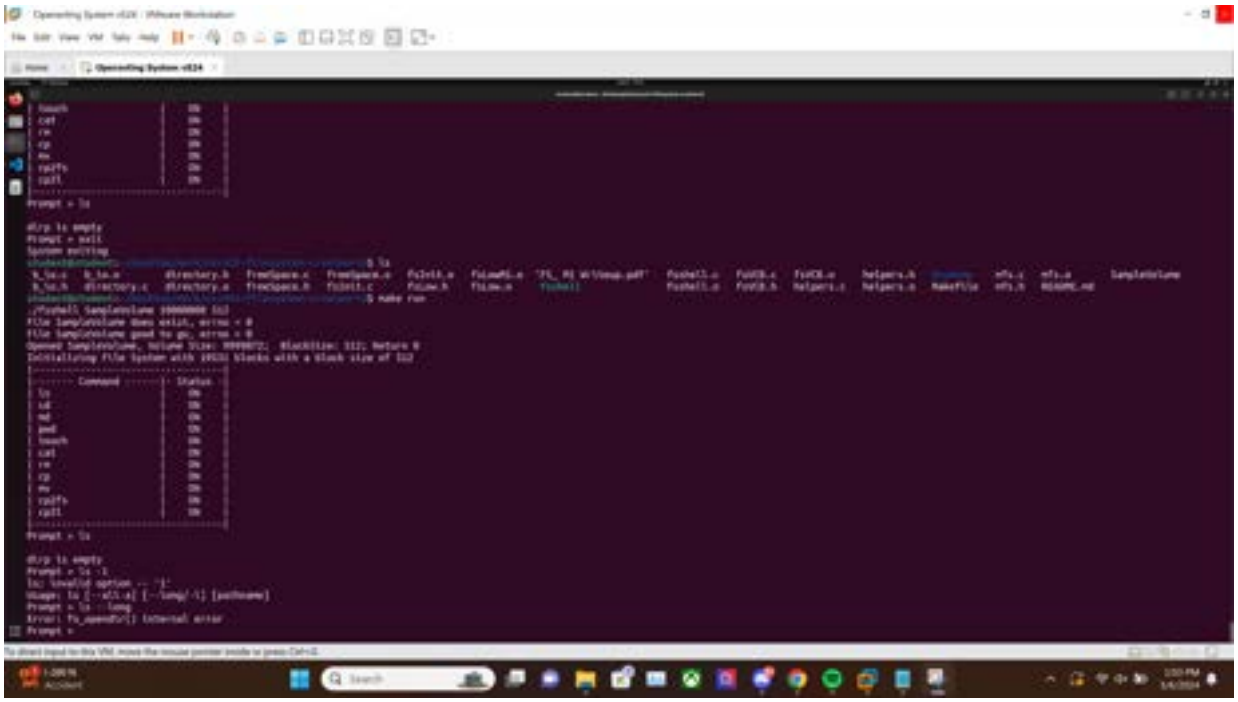


Another error we encountered was that some functions were written twice. I assume that overriding is not a thing in C, causing this error to occur in the process.

```
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsVCB.o freeSpace.o directory.o mfs.o helpers.
o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
/usr/bin/ld: fsLowM1.o: in function `LBAwrite':
/home/parallels/Documents/CSC415/AssignmentCreation/CSC415-FileSystem/fsLowM1.c:
216: multiple definition of `LBAwrite'; b_io.o:/home/student/csc415-filesystem-c
rvelworld/b_io.c:122: first defined here
/usr/bin/ld: fsLowM1.o: in function `LBAread':
/home/parallels/Documents/CSC415/AssignmentCreation/CSC415-FileSystem/fsLowM1.c:
261: multiple definition of `LBAread'; b_io.o:/home/student/csc415-filesystem-cr
velworld/b_io.c:163: first defined here
collect2: error: ld returned 1 exit status
make: *** [Makefile:61: fsshell] Error 1
```

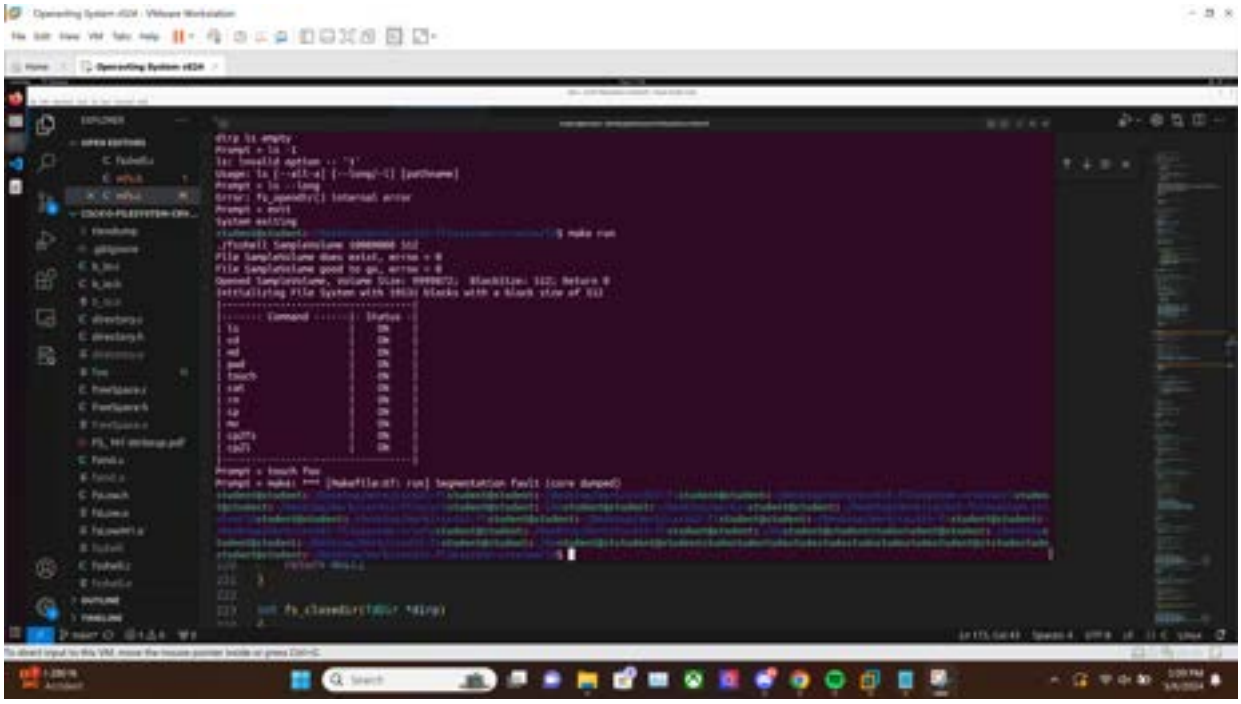
We fix this by removing what is not needed from the file.

Another error we faced was when we were working with the ls. We have trouble having the ls function working with our fs_opendir(). Possibility is that we are not reading into anything or possible that we don't have the correct information.

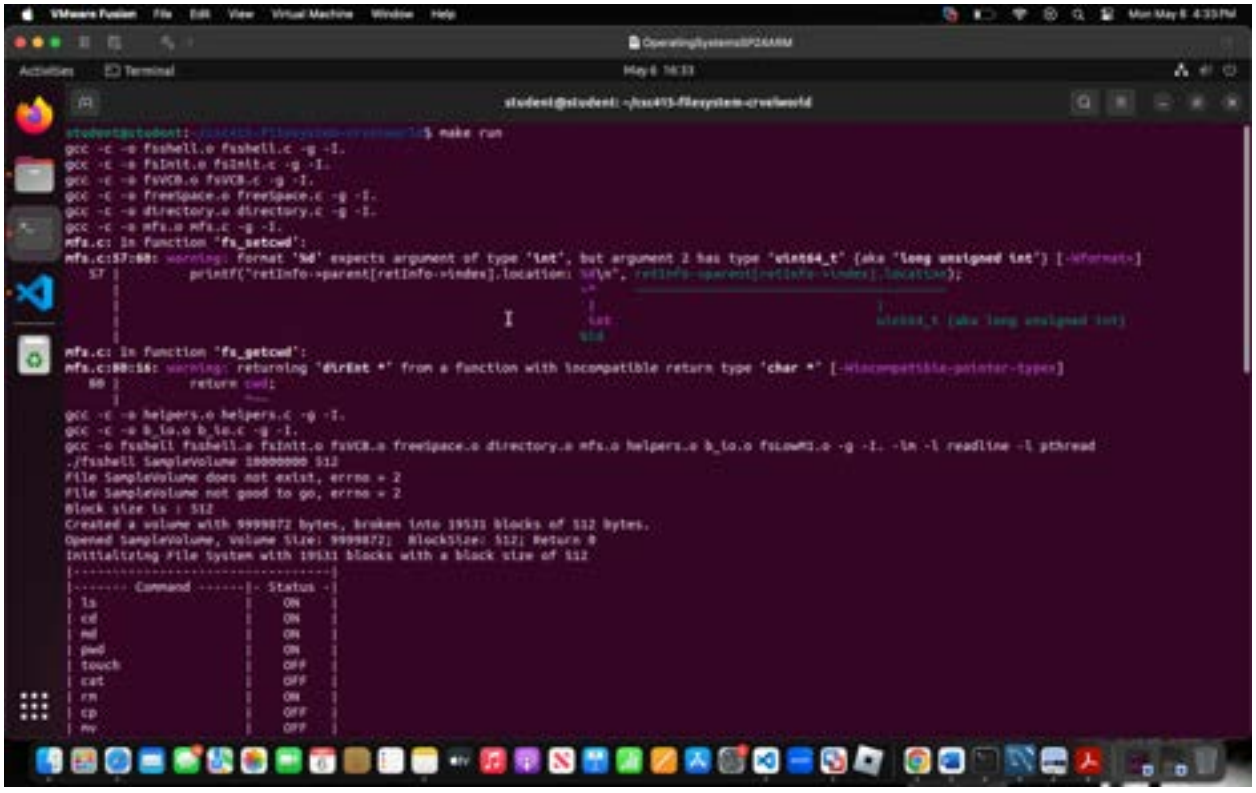


Another issue is with the touch command. While we are able to create a file, we are not able to finish the process without having a seg fault and having the system crash. The possibility of this is it is not being properly stored and written.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



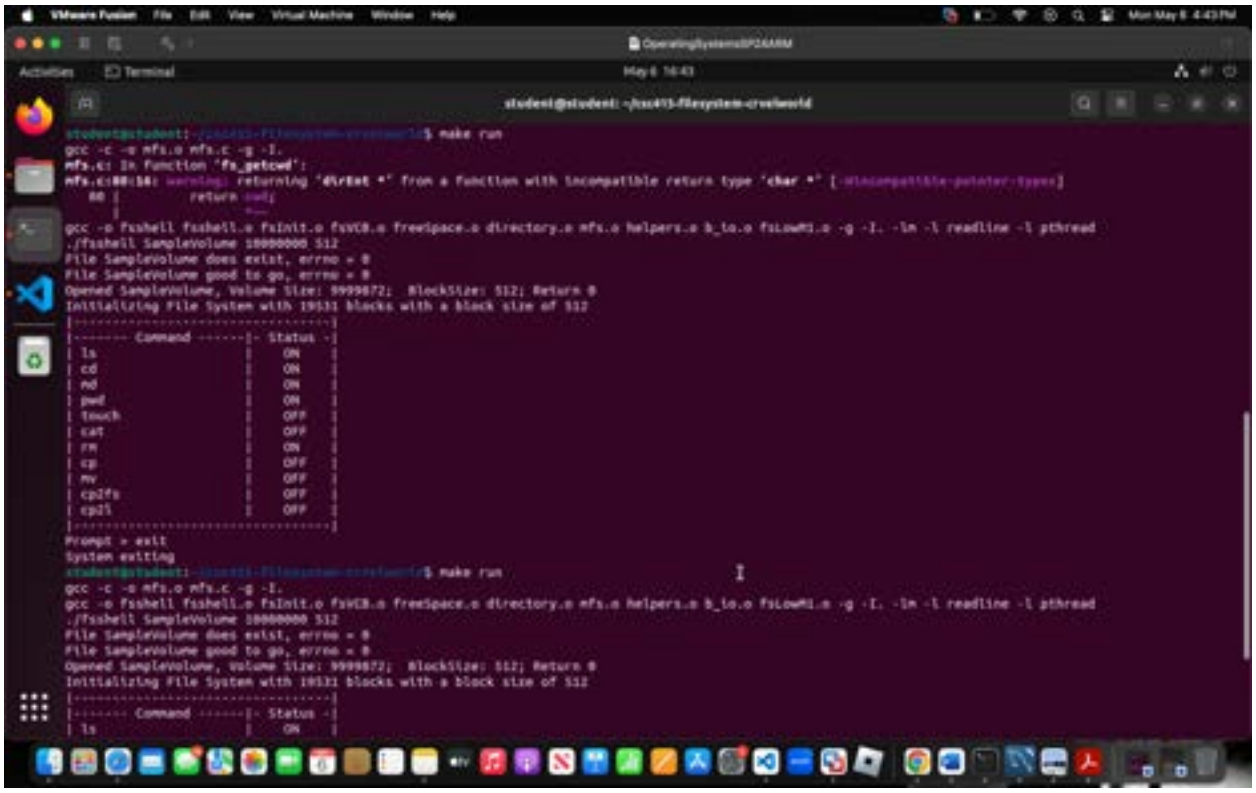
While running the command make run we received these warnings.



The first warning was due to type mismatches in our code. We used %d instead of %lu in our printf statement which caused it to print incorrectly.

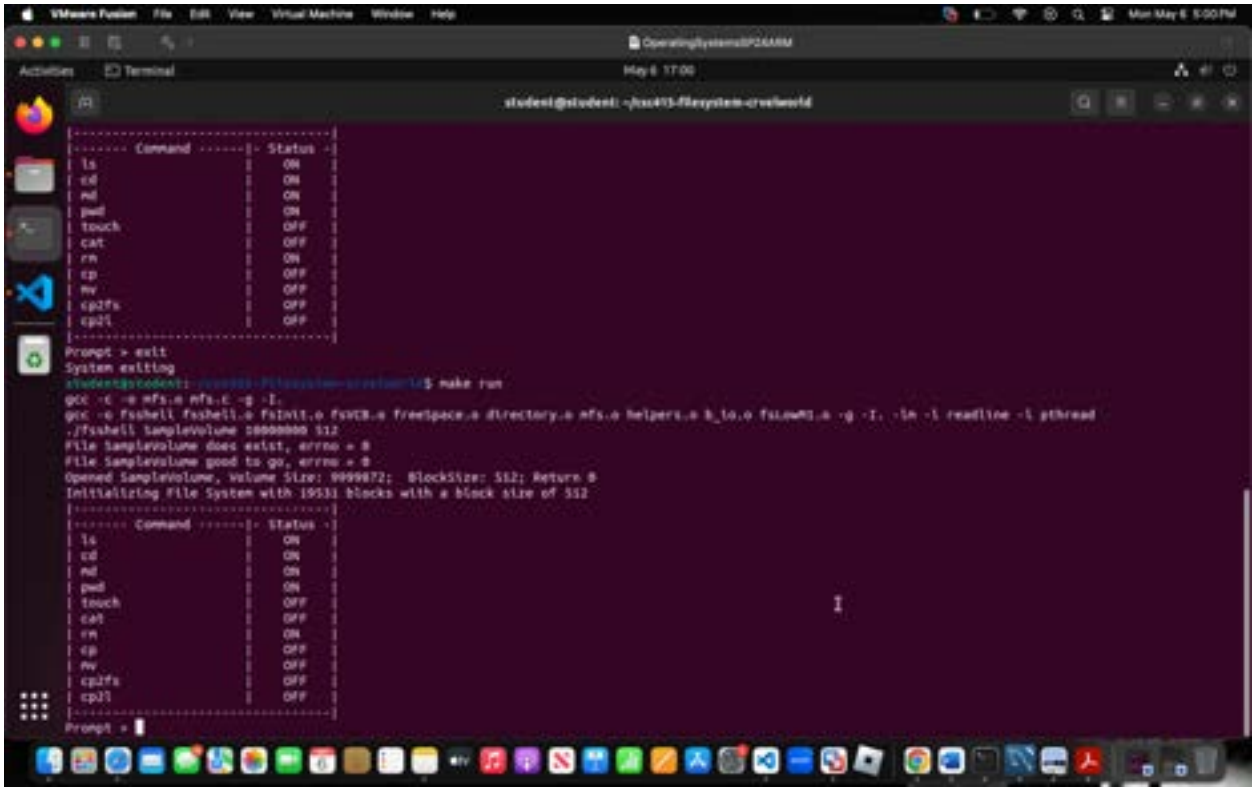
printf("retInfo->parent[retInfo->index].location: %lu\n",

retInfo->parent[retInfo->index].location); by changing this code we were able to solve the warning that we were receiving.



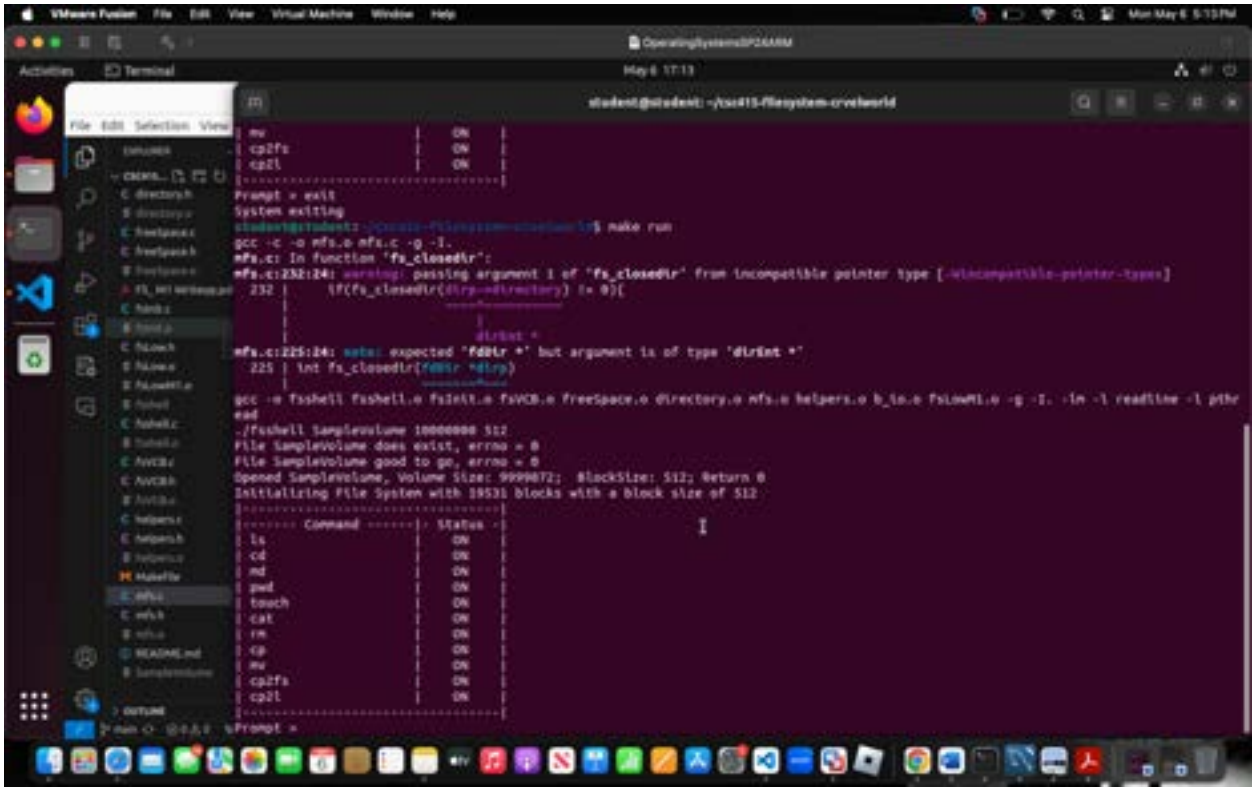
The second warning came from is encountered due to the fs_getcwd function returning a dirEnt* type when it's declared to return a char *. To resolve this warning, I modified the fs_getcwd function to return a char *. Once this issue was resolved I was able to compile with no warnings.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



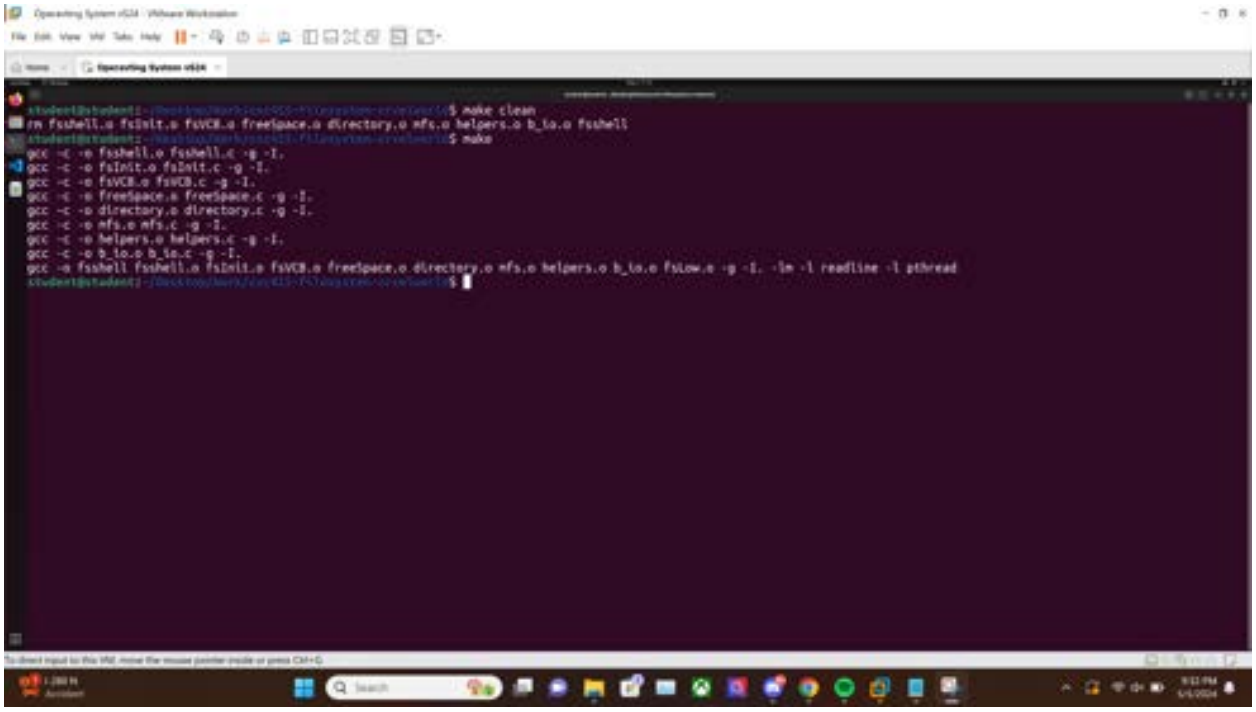
Another error was found when running our file system project. Shown in the screenshot below.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

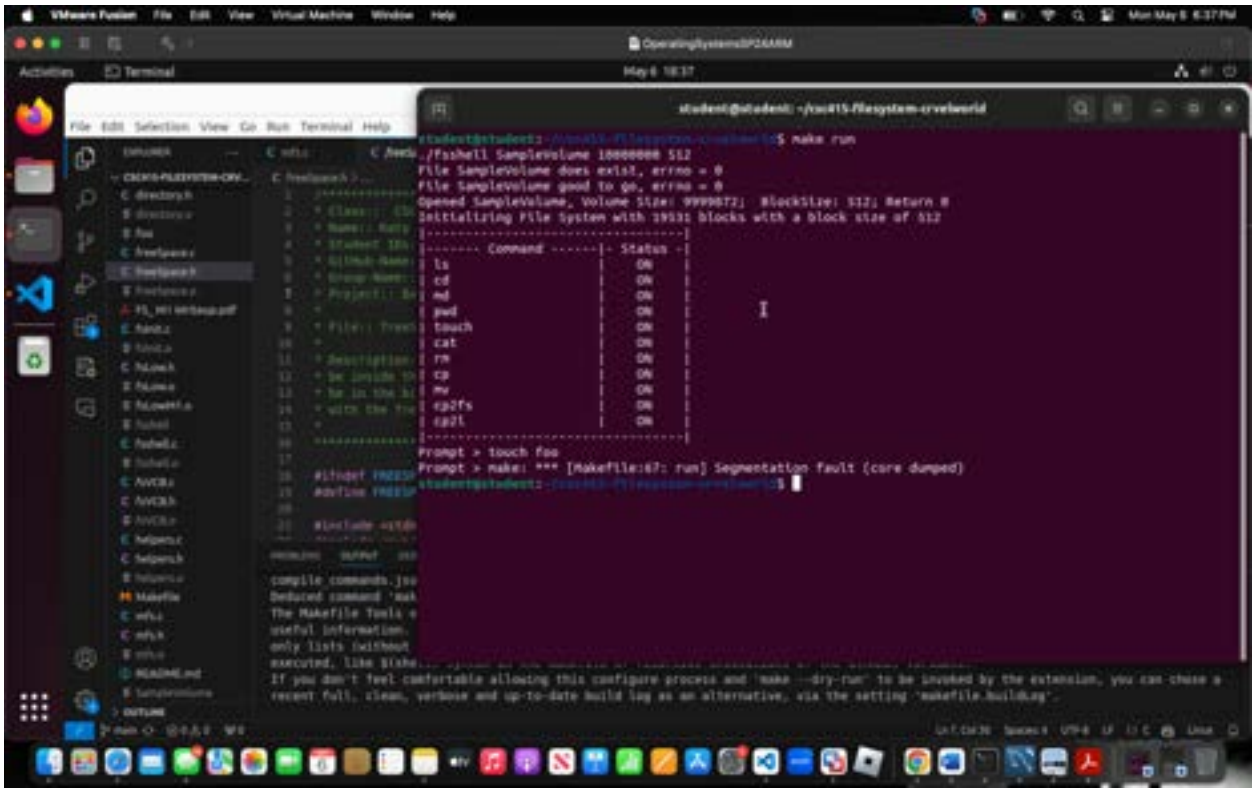


This warning indicates that we are passing dirEnt * type to the fs_closedir function, but fdDir * type was expected. To fix this we need to make sure dirp is of type fdDir* when calling fs_closedir.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

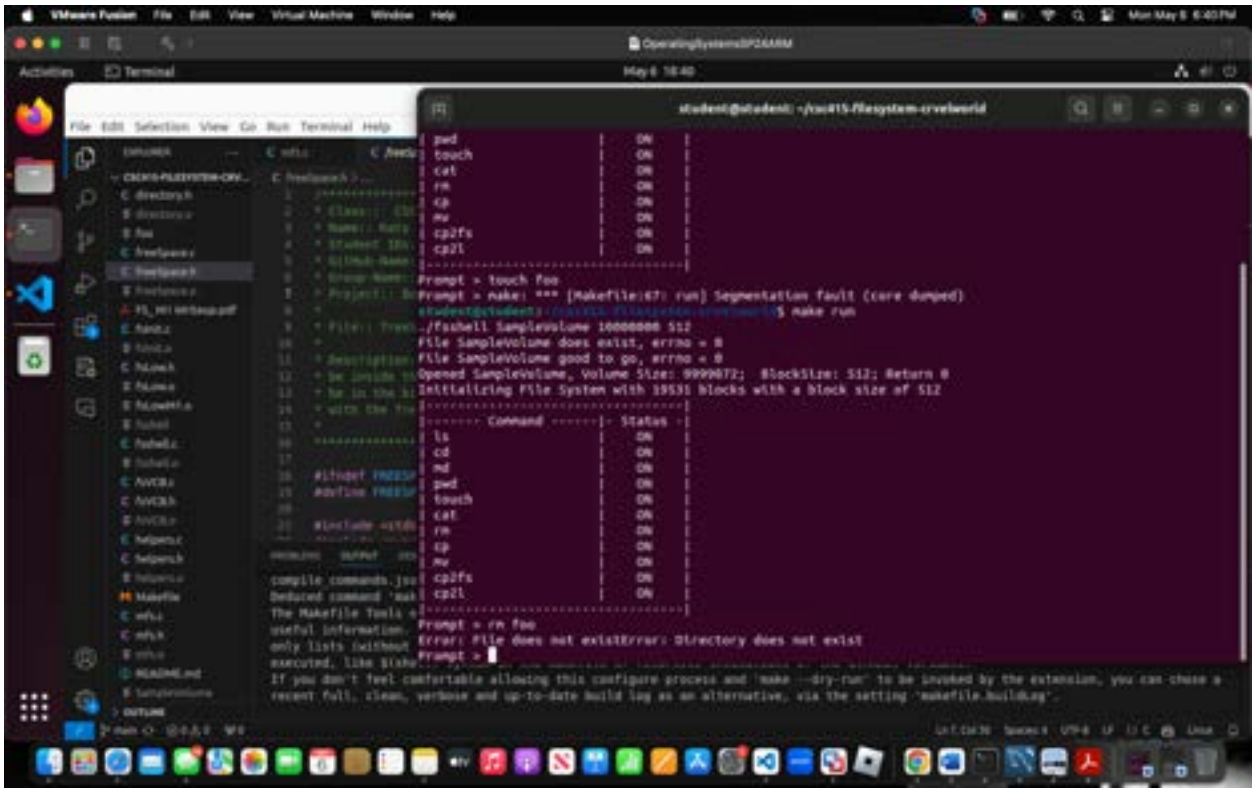


Another issue is the touch command, as you can see in the screenshot below that the command touch foo works but then a segmentation fault occurs.



There is another issue with the rm function, as you can see above that we were able to create foo but then when we run the command rm foo, we get an error saying “directory

does not exist”.

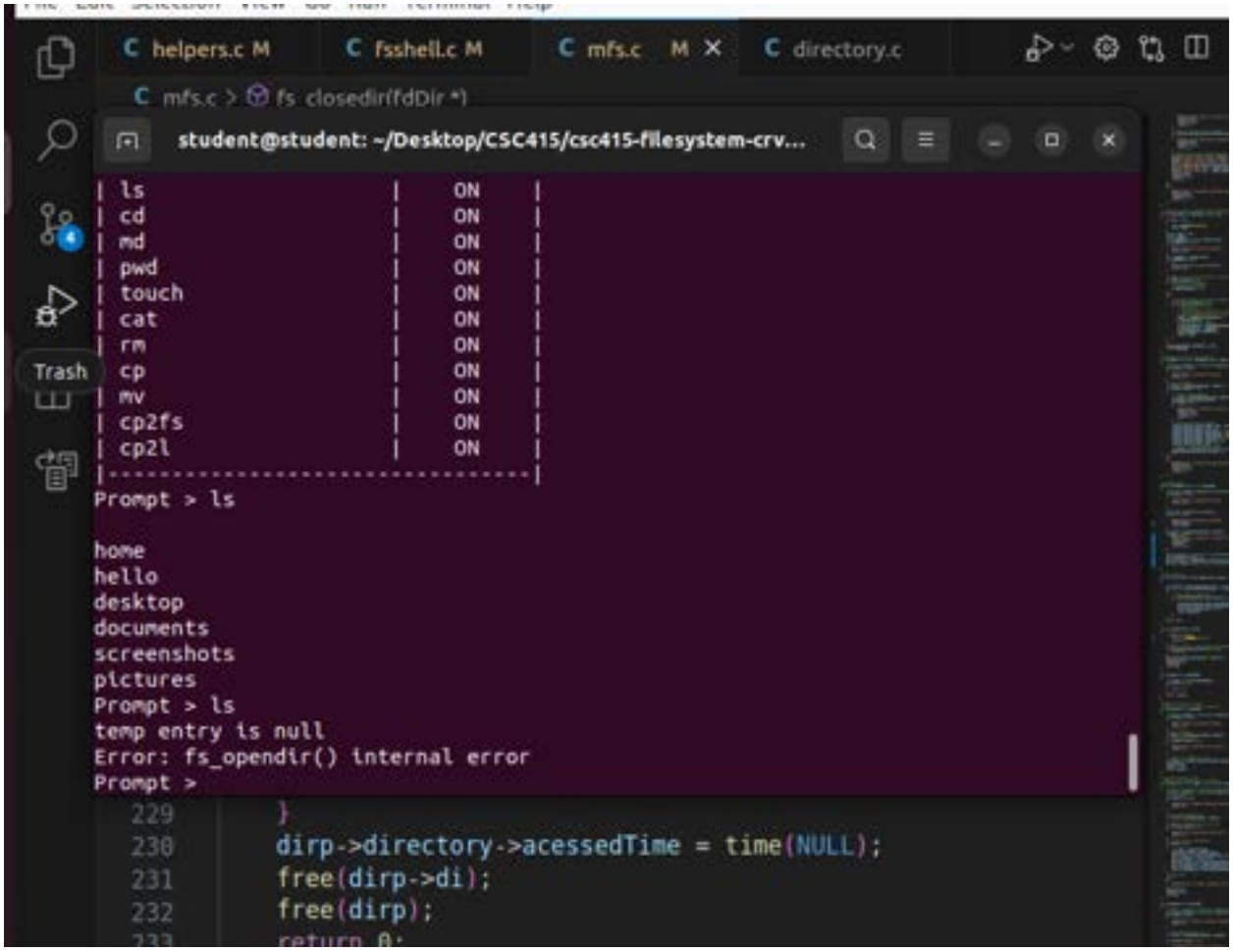


Another issue that was encountered was there was a possible bug or glitch in the terminal due to possible corruption of data in the blocks, causing it to return some strange characters and symbols.

The problem was fixed after clearing out the sample volume.

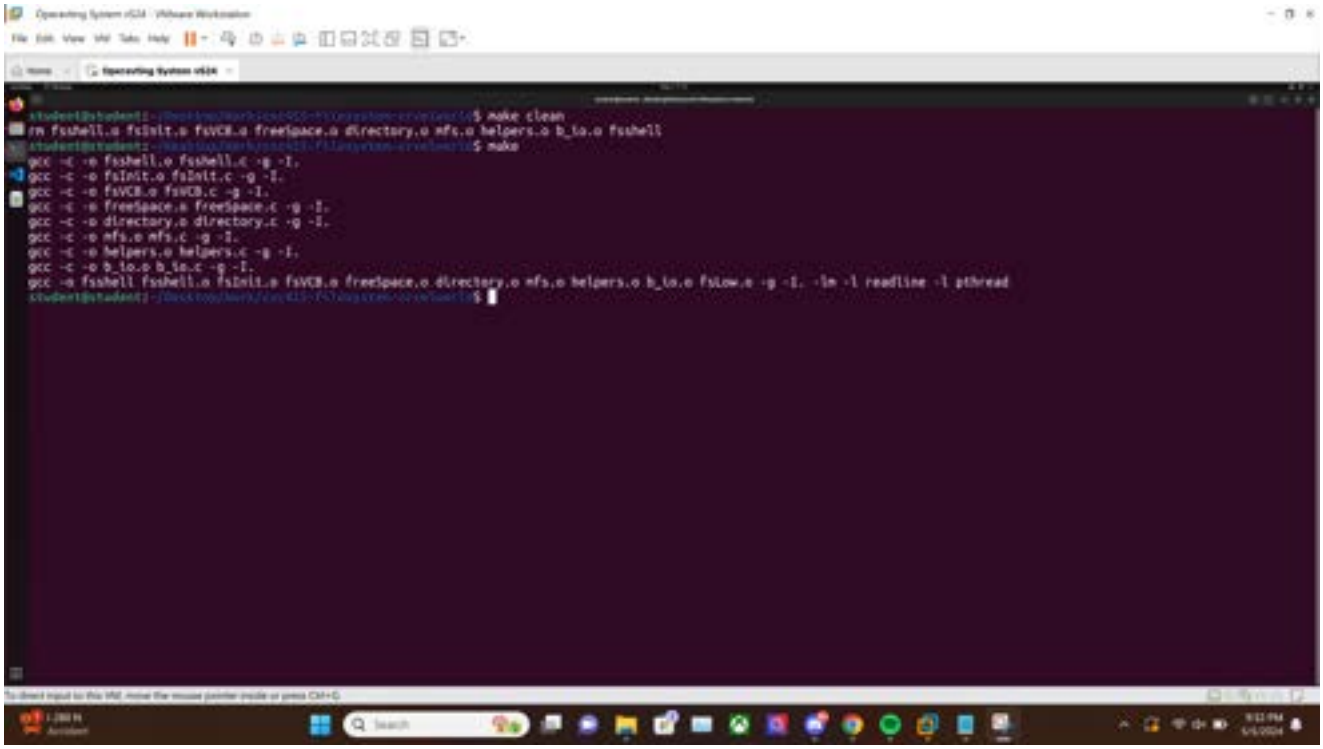
subsequent calls of this command results in an error because the token that's being parsed in parsePath becomes NULL. Because of this, it doesn't enter the for loop, which leaves the tempPath variable to be NULL as well.

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



Screenshot of compilation:

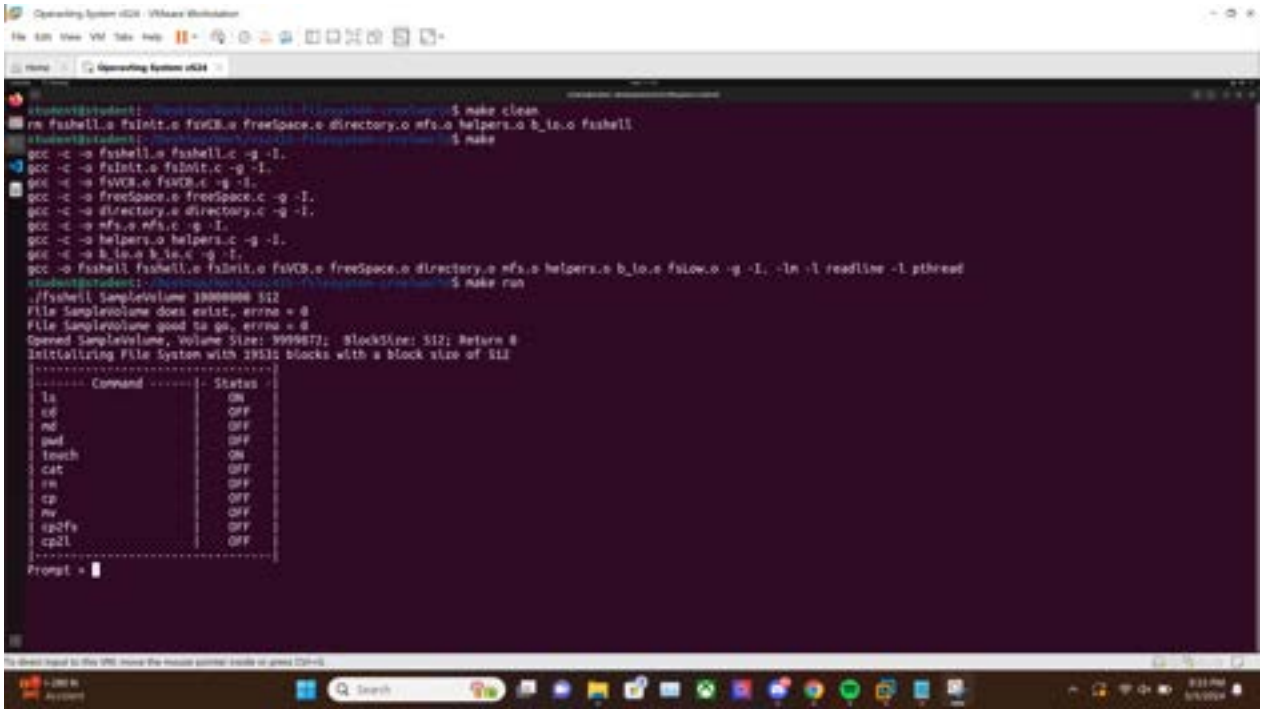
Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



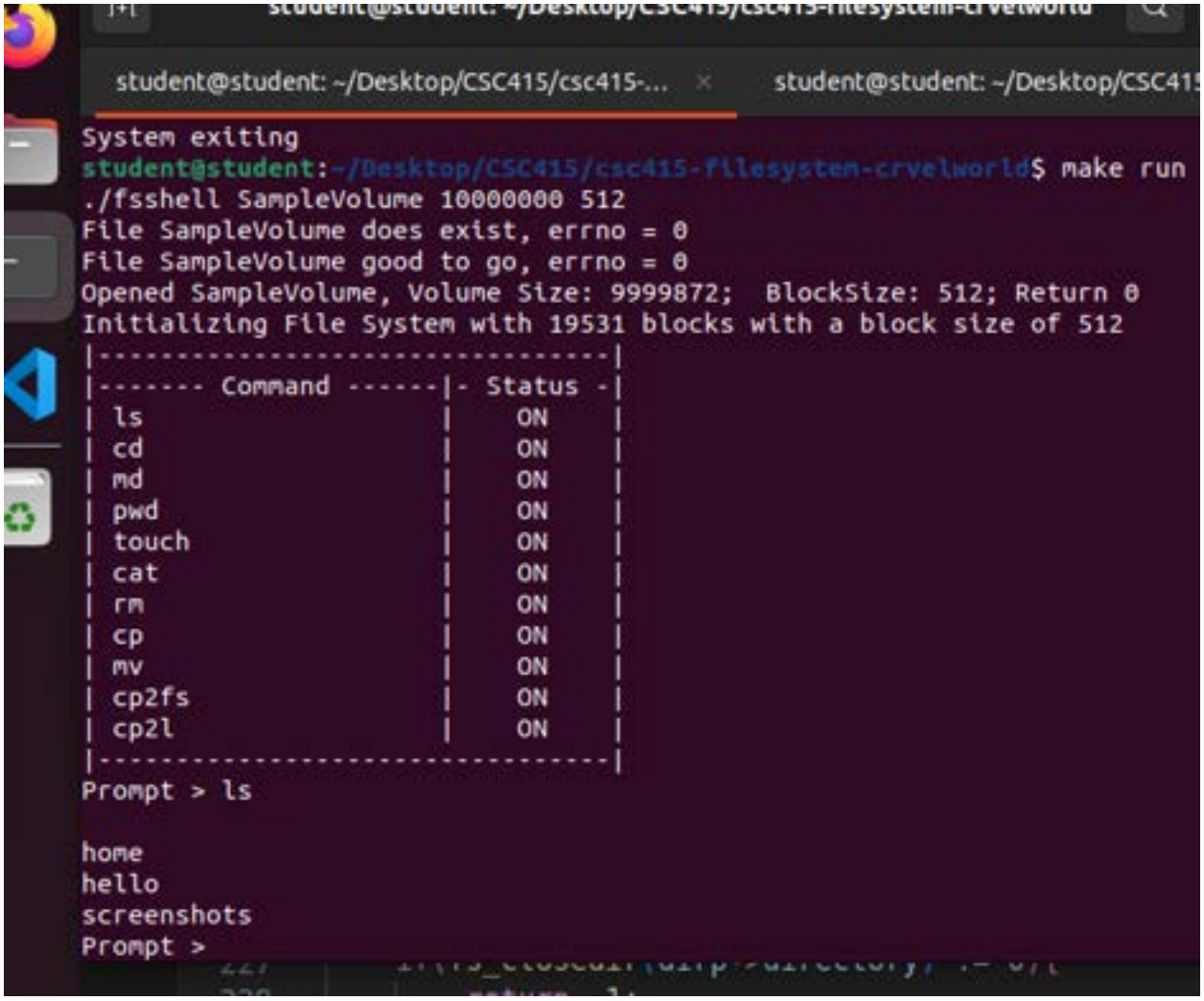
Another issue is the touch command, as you can see in the screenshot below that the command touch foo works but then a segmentation fault occurs.

Screenshot(s) of the execution of the program:

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



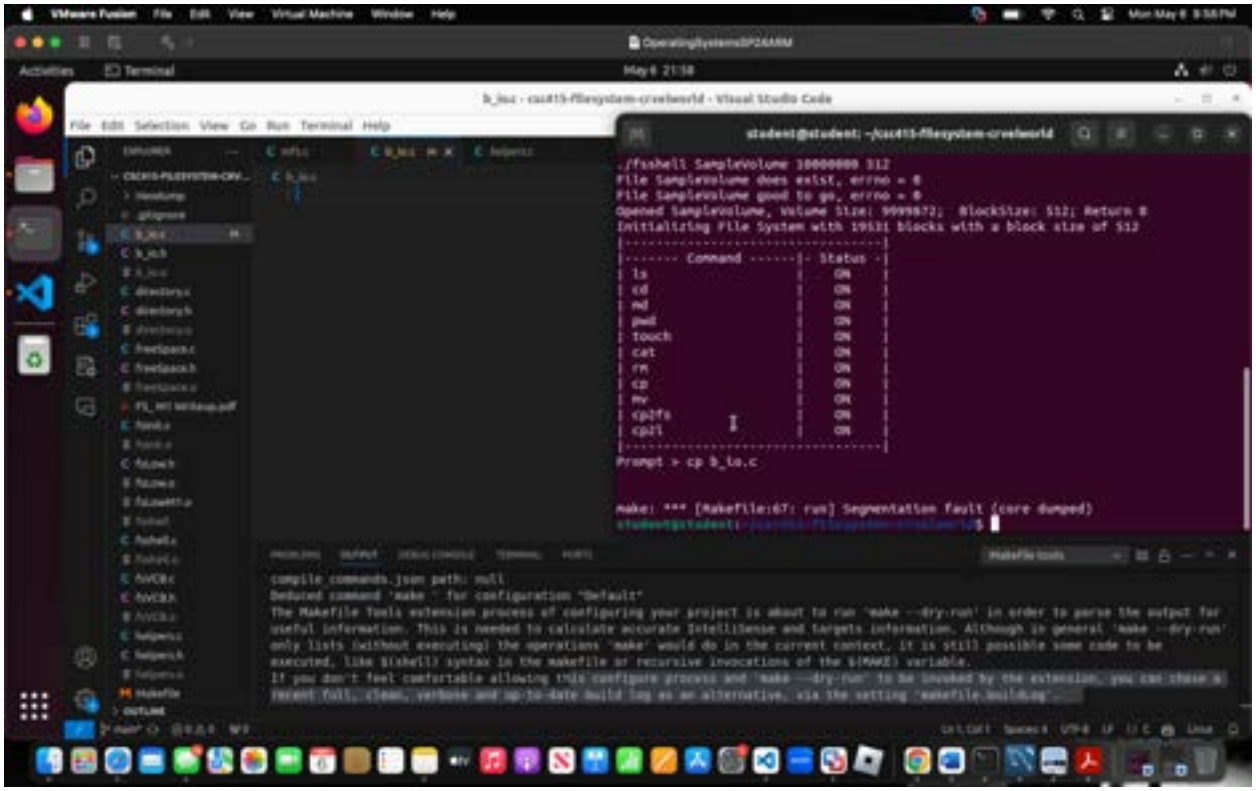
ls - Lists the file in a directory



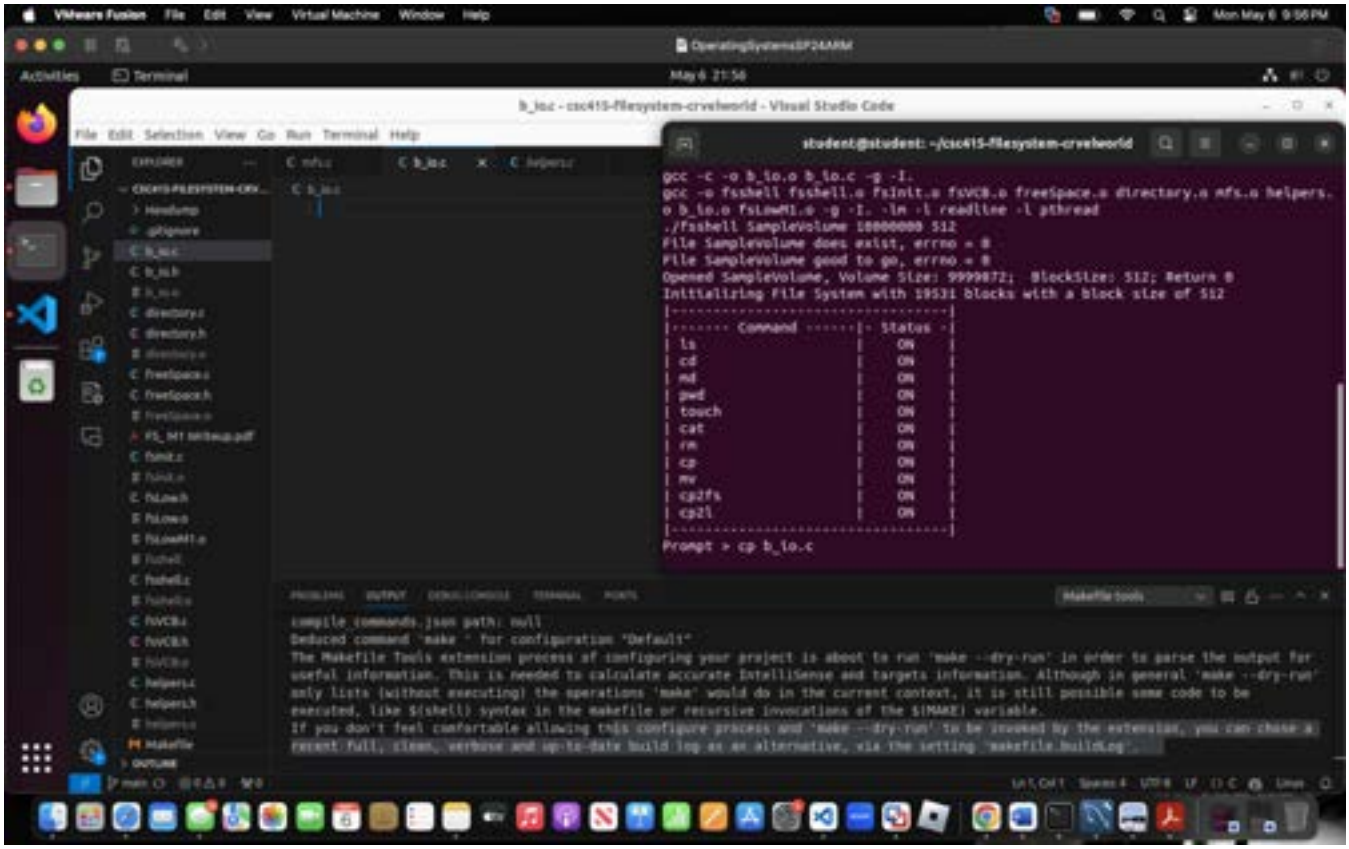
cp - Copies a file - source [dest]

Works partly, ends up causing a segmentation fault and crashing.

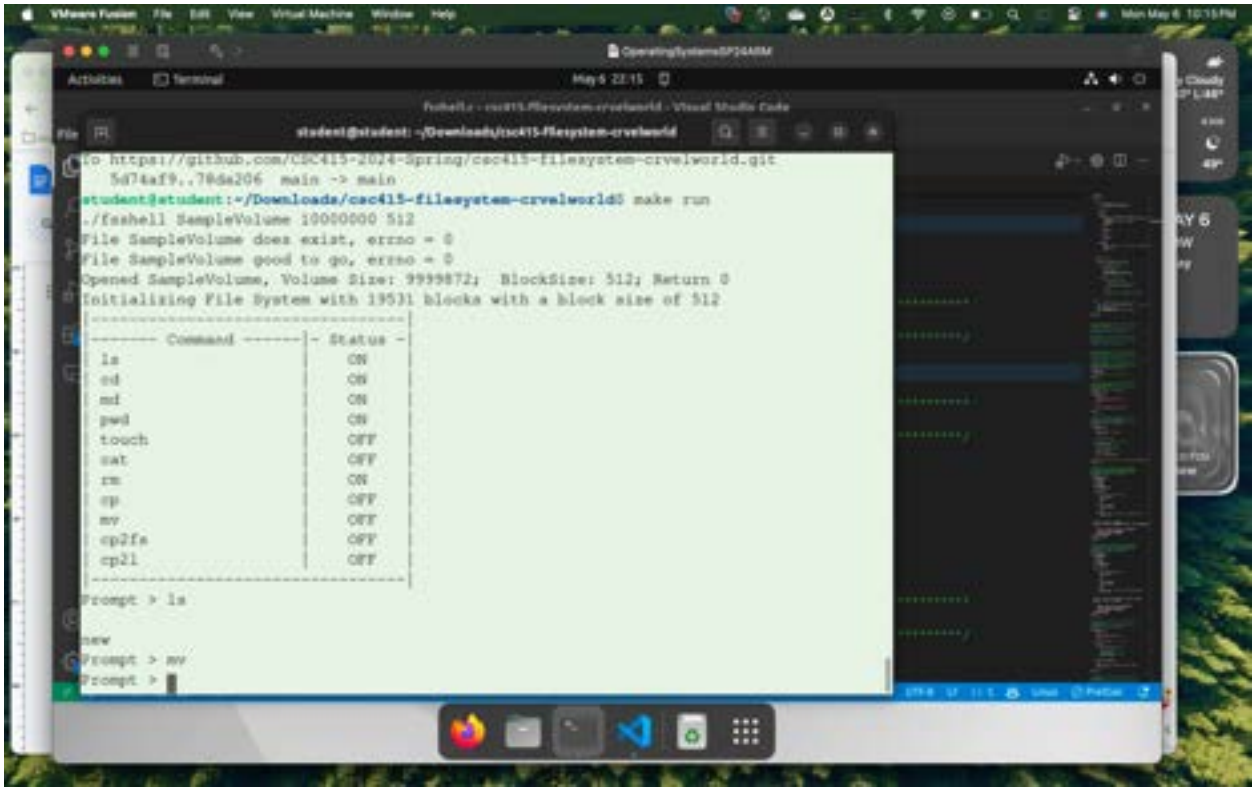
Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



mv - Moves a file - source dest

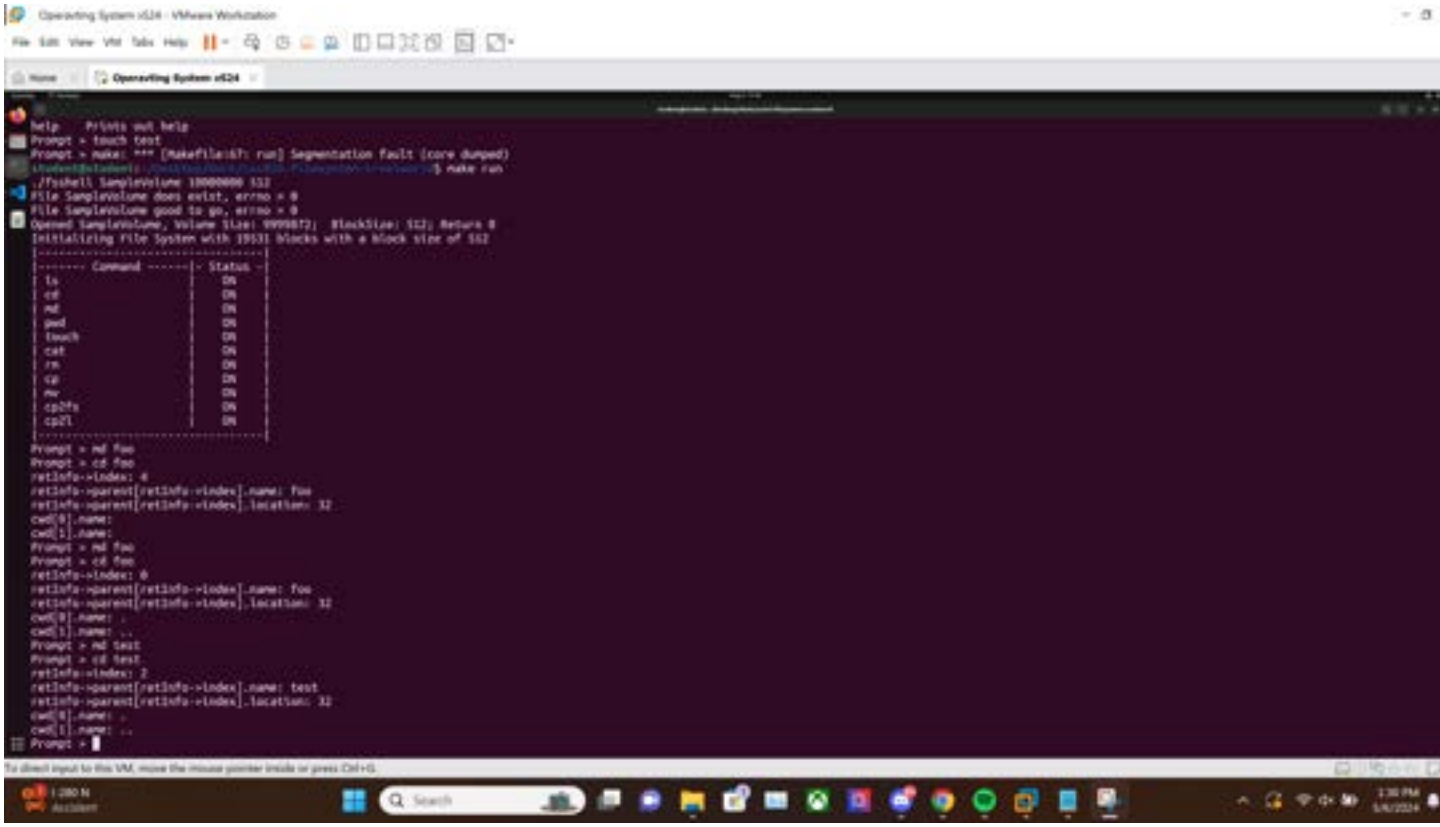
- 

Result: mv command not implemented.

Expected behavior: The mv command would move a file from one place to another. To approach this command, we would have first started by checking that both files exist. To do this, we can call is_File() to ensure that the files exist in our file system. Then, we would need to check that both files are not identical. After all of the necessary checks, we then get file descriptors from our b_open() functions using the relevant flags. Upon success, we would then continuously streamline the data from the src file to the dest file. Finally, we delete the src file and we are done!

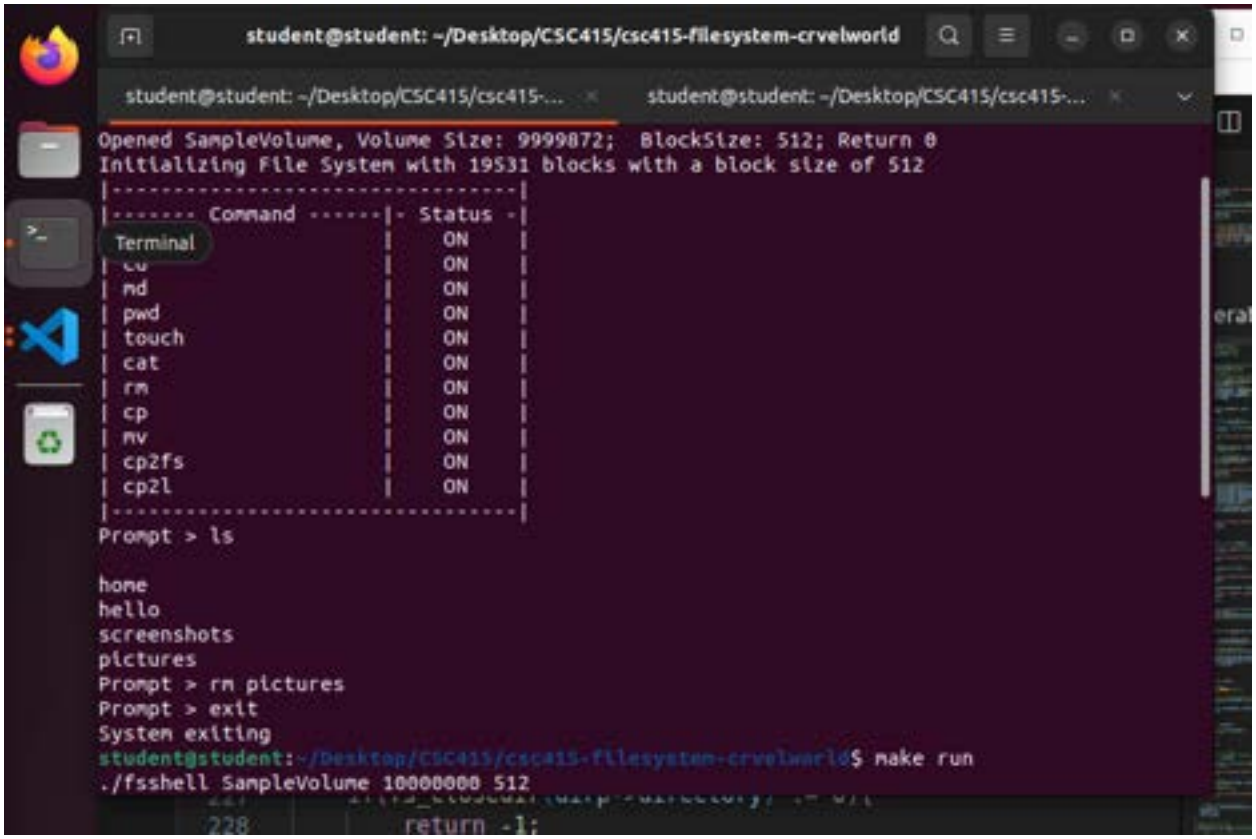
md - Make a new directory

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



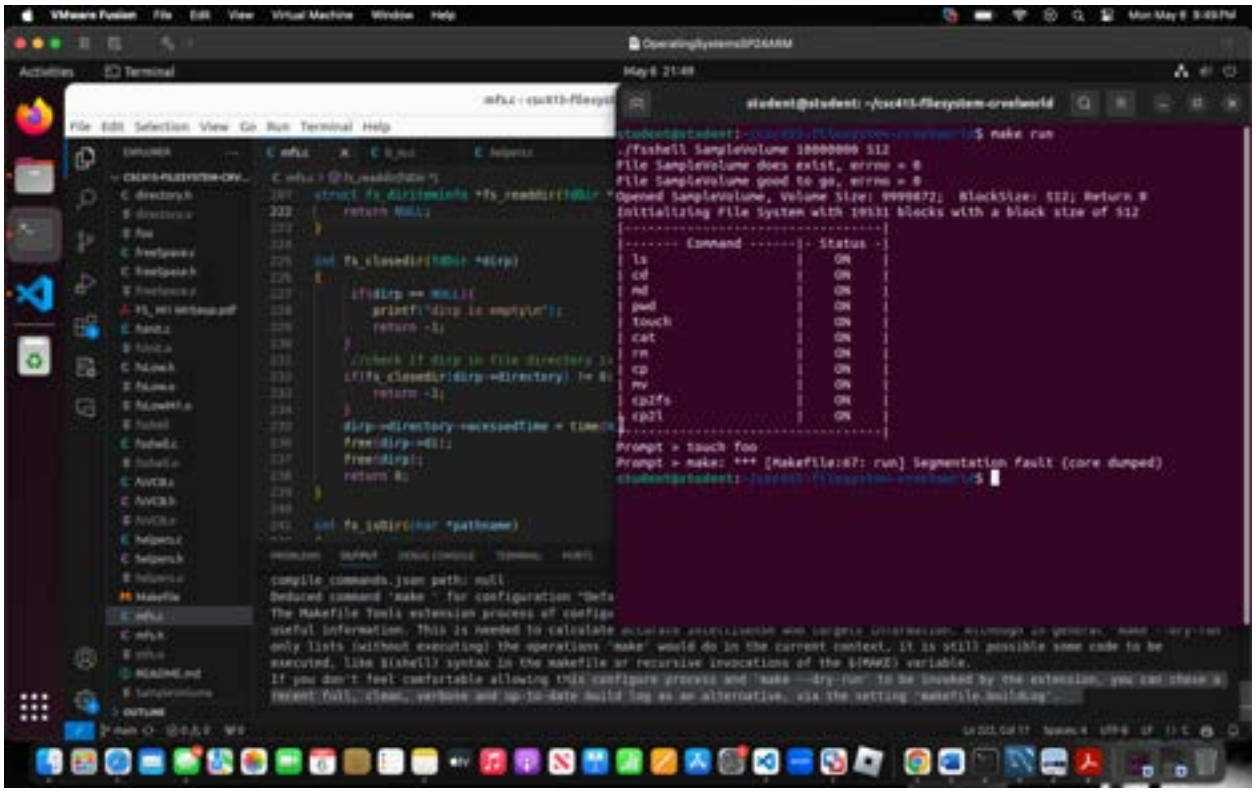
rm - Removes a file or directory

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

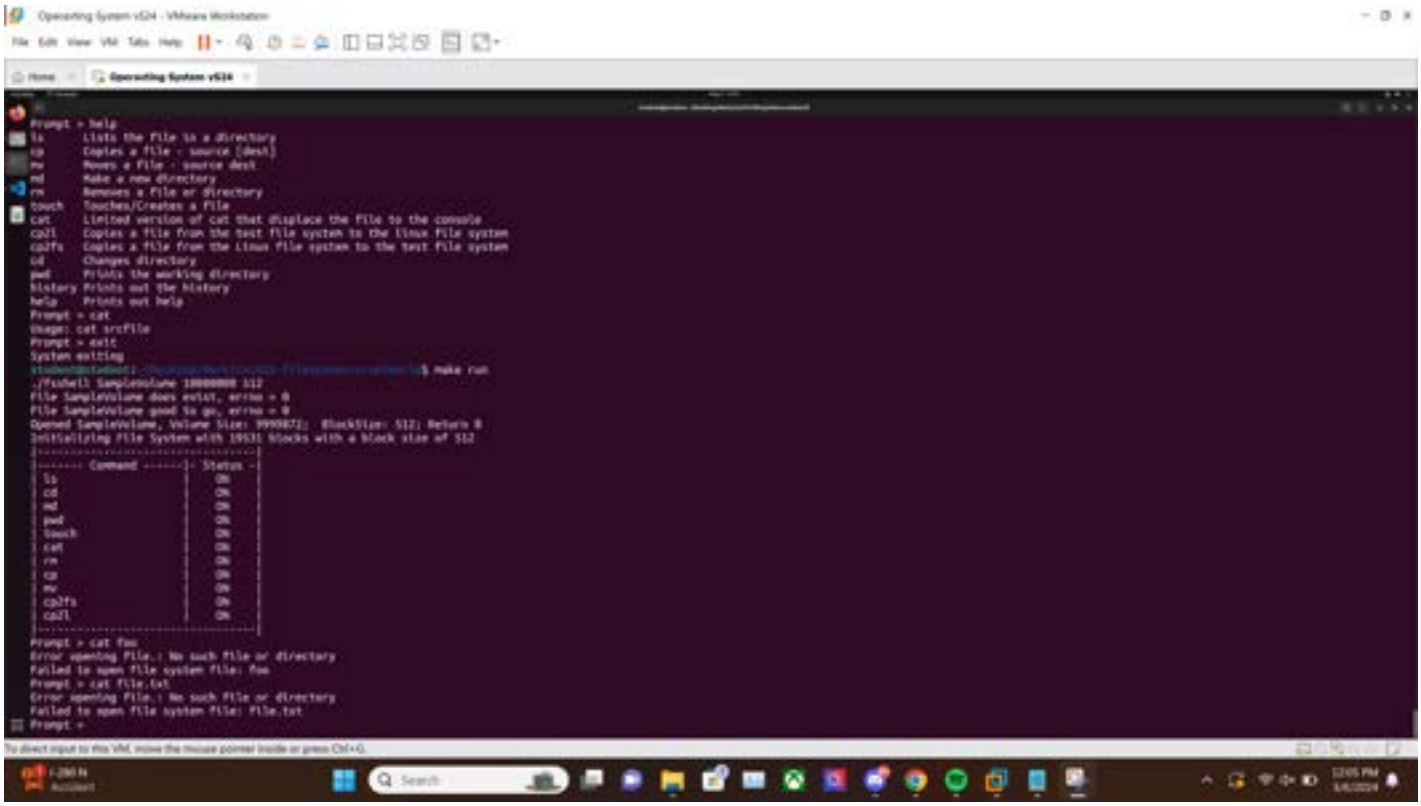
touch - creates a file but then will crash and create a segmentation fault



cat - (limited functionality) displays the contents of a file

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

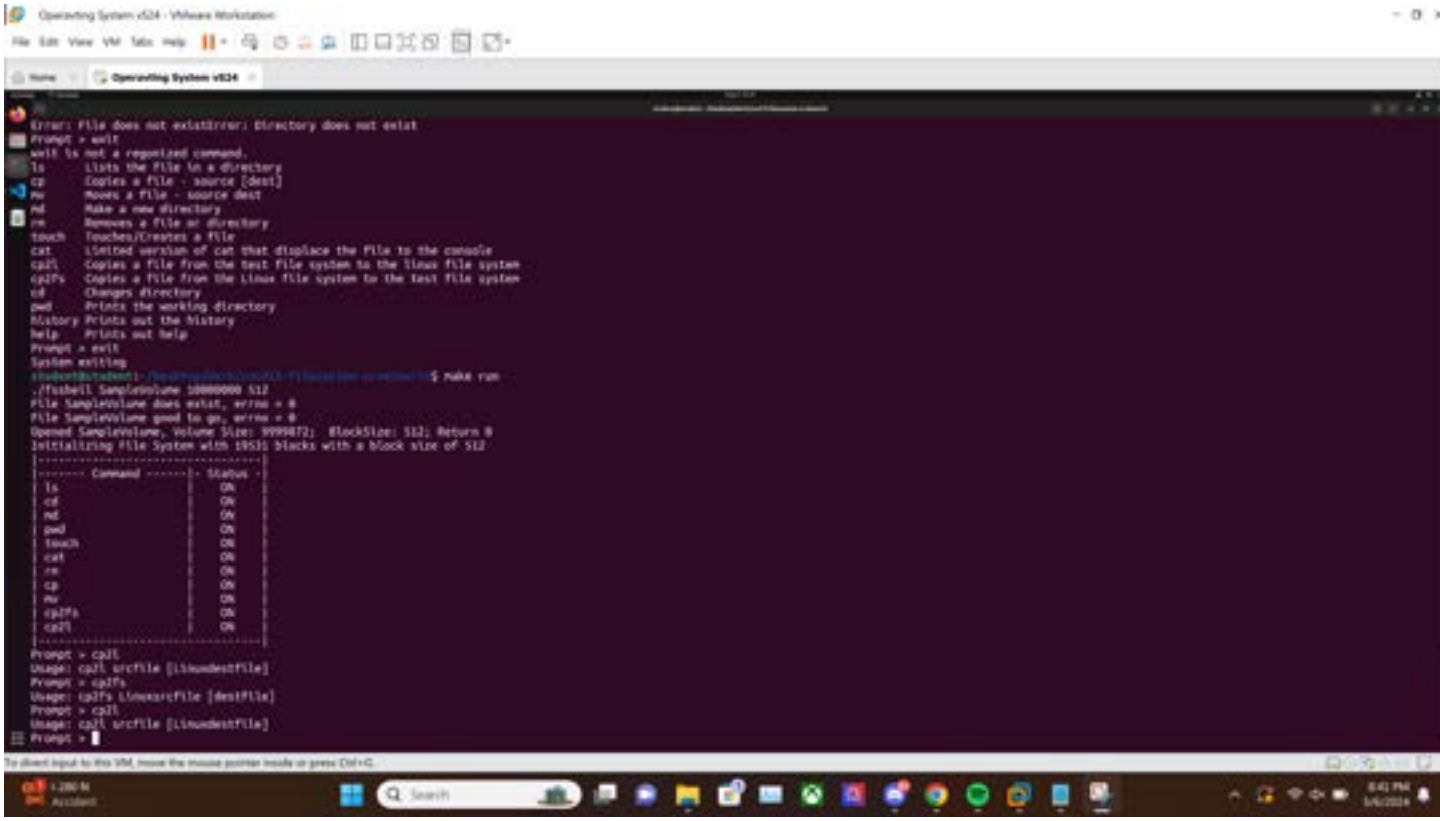
Screenshot of Cat being able to recognize if a file does not exist



Screenshot of cat being able to recognize that the file exists

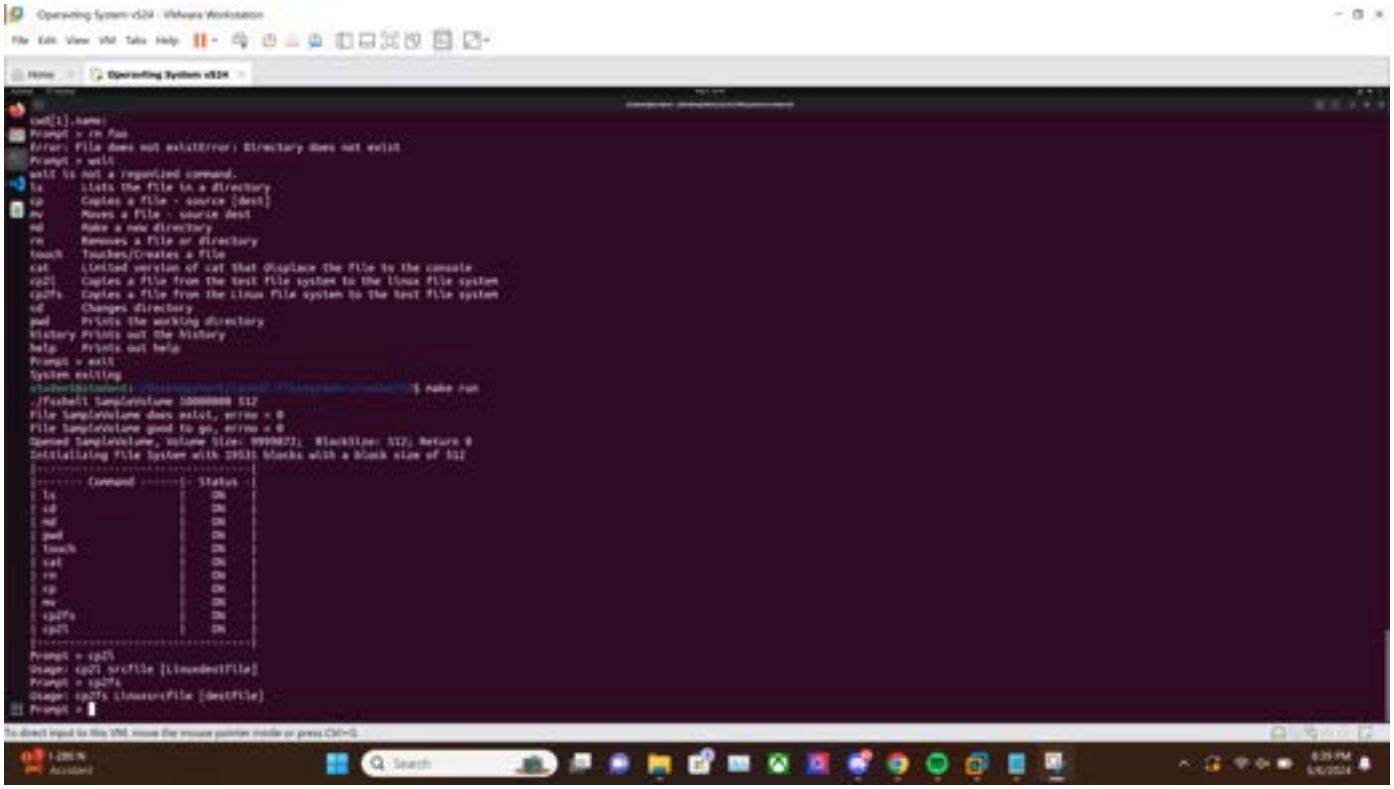
cp2l - Copies a file from the test file system to the linux file system

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



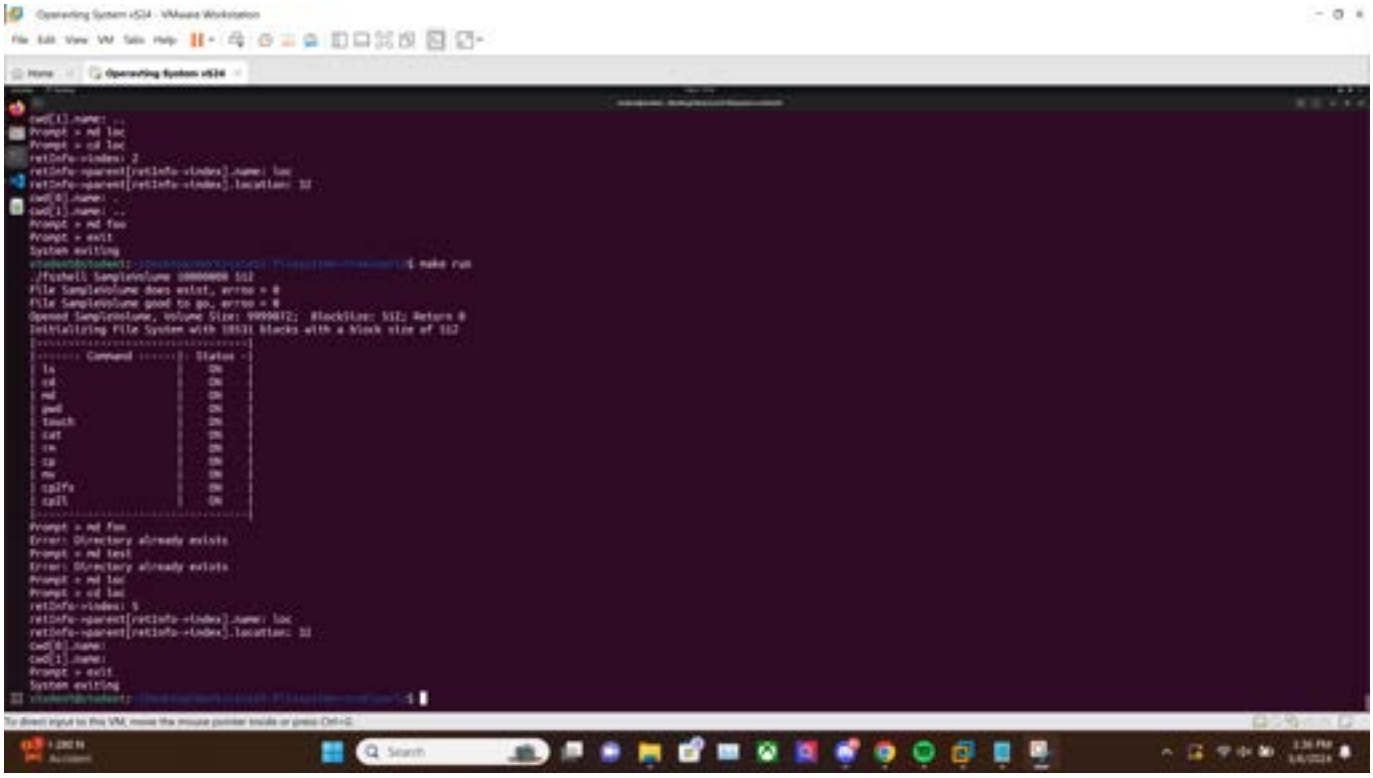
cp2fs - Copies a file from the Linux file system to the test file system

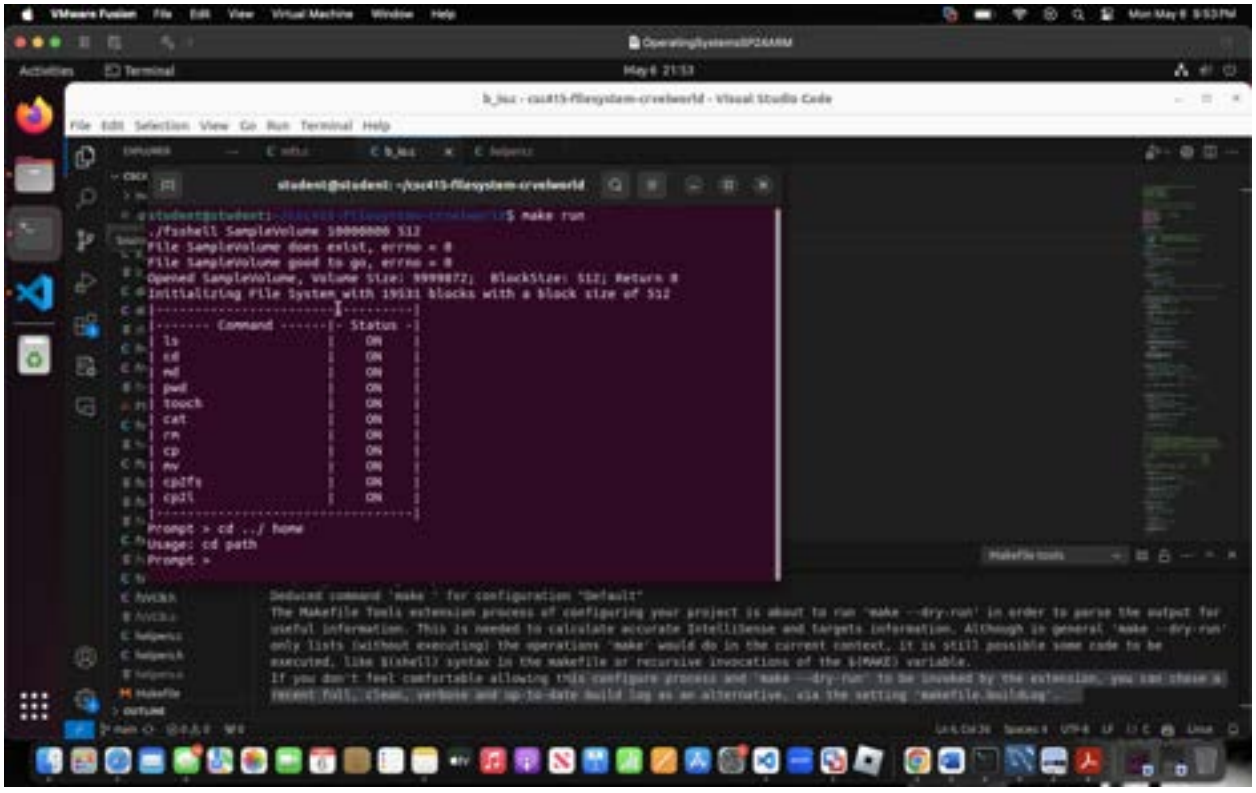
Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



cd - Changes directory

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld



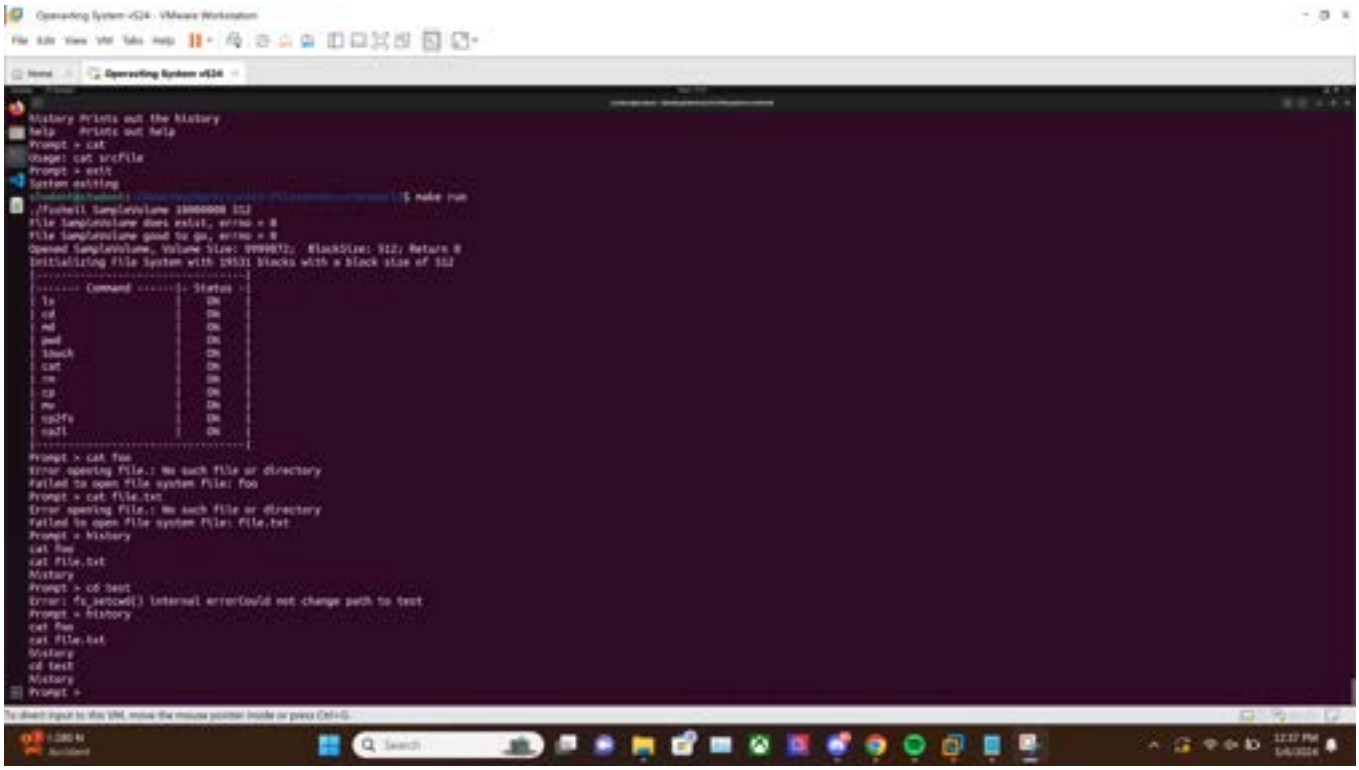


Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

pwd - Prints the working directory

Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

history - Prints out the history



Group Name: Caffeinated Crew
Members: Issac Moreno, Katy Lam, Anisah Chowdhury, Malieka Sutaria
ID: 921984788, 921922518, 921677676, 922888691
GitHub: crvelworld

help - Prints out help