File System Project

CSC 415-02/03 Operating Systems

Team Fantastic Four

Arslan Alimov | 916612104

Nakulan Karthikeyan | 920198861

Johnathan Huynh | 920375700

Hugo Moreno | 920467935

Primary GitHub Repository

https://github.com/CSC415-Fall2021/csc415-filesystem-knakulan2001

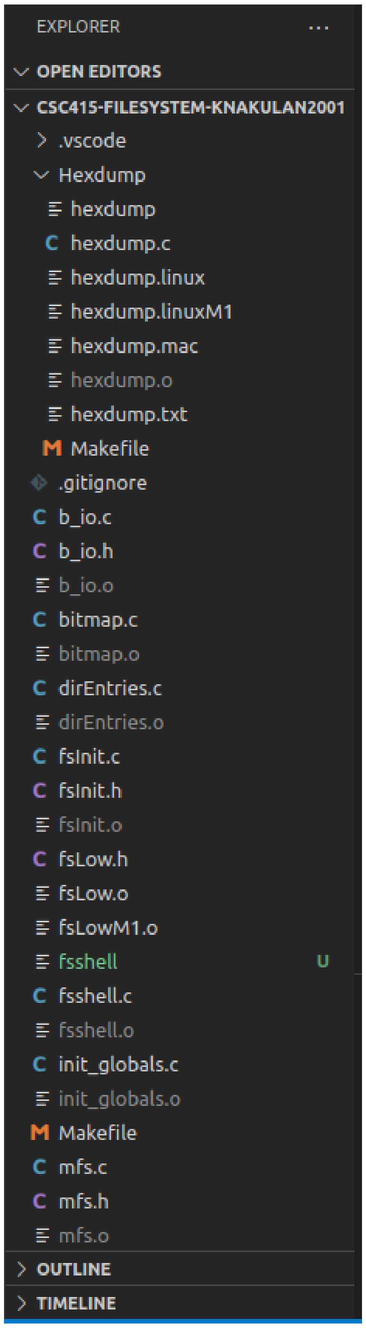
Table of Contents

Introduction

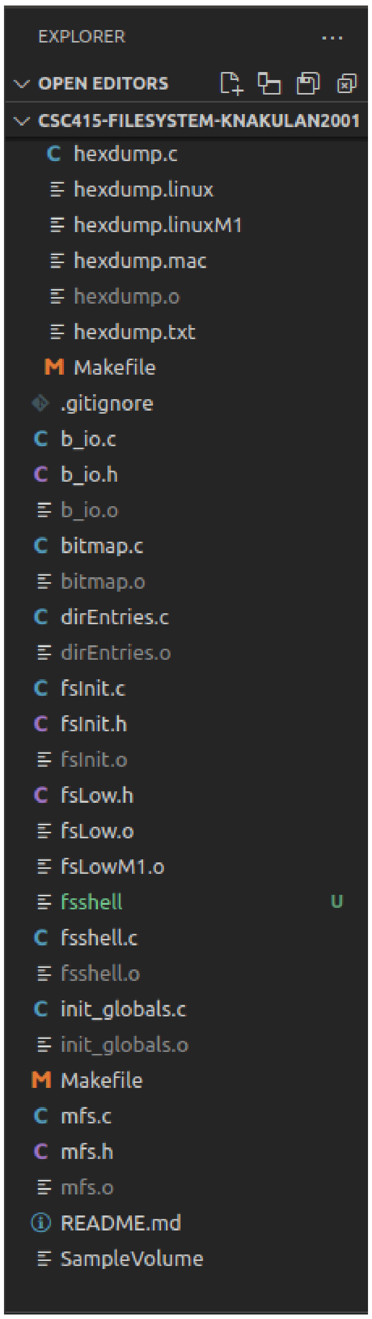
In this group project, we were tasked with creating a unique file system modelled after the linux shell. Our file system will need to be able to perform different commands such as *cd, md(mkdir), cat, clear, copy, ls,* and *pwd* among many others. This meant that our duties included formatting the provided volume, creating and maintaining a free space management system, initializing a root directory and maintaining directory information, creating, reading, writing, and deleting files, displaying info, among many other tasks. Hence, using the code and driver provided by the professor, our file system should ultimately be able to list directories, create directories, add and remove files, copy files, move files, copy from the local file system to our created filesystem, and copy from our created filesystem to the local filesystem.

NOTES: The root directory does not have any inherent name. Each directory has a '.' and a '..' directory entry for current and parent.

Project Layout



```
EXPLORER ...
v OPEN EDITORS
v CSC415-FILESYSTEM-KNAKULAN2001
  > .vscode
  v Hexdump
      hexdump
      hexdump.c
      hexdump.linux
      hexdump.linuxM1
      hexdump.mac
      hexdump.o
      hexdump.txt
      Makefile
    .gitignore
    b_io.c
    b_io.h
    b_io.o
    bitmap.c
    bitmap.o
    dirEntries.c
    dirEntries.o
    fsInit.c
    fsInit.h
    fsInit.o
    fsLow.h
    fsLow.o
    fsLowM1.o
    fsshell                      U
    fsshell.c
    fsshell.o
    init_globals.c
    init_globals.o
    Makefile
    mfs.c
    mfs.h
    mfs.o
> OUTLINE
> TIMELINE
```



```
EXPLORER ...
v OPEN EDITORS
v CSC415-FILESYSTEM-KNAKULAN2001
      hexdump.c
      hexdump.linux
      hexdump.linuxM1
      hexdump.mac
      hexdump.o
      hexdump.txt
      Makefile
    .gitignore
    b_io.c
    b_io.h
    b_io.o
    bitmap.c
    bitmap.o
    dirEntries.c
    dirEntries.o
    fsInit.c
    fsInit.h
    fsInit.o
    fsLow.h
    fsLow.o
    fsLowM1.o
    fsshell                      U
    fsshell.c
    fsshell.o
    init_globals.c
    init_globals.o
    Makefile
    mfs.c
    mfs.h
    mfs.o
    README.md
    SampleVolume
```

Issues we had

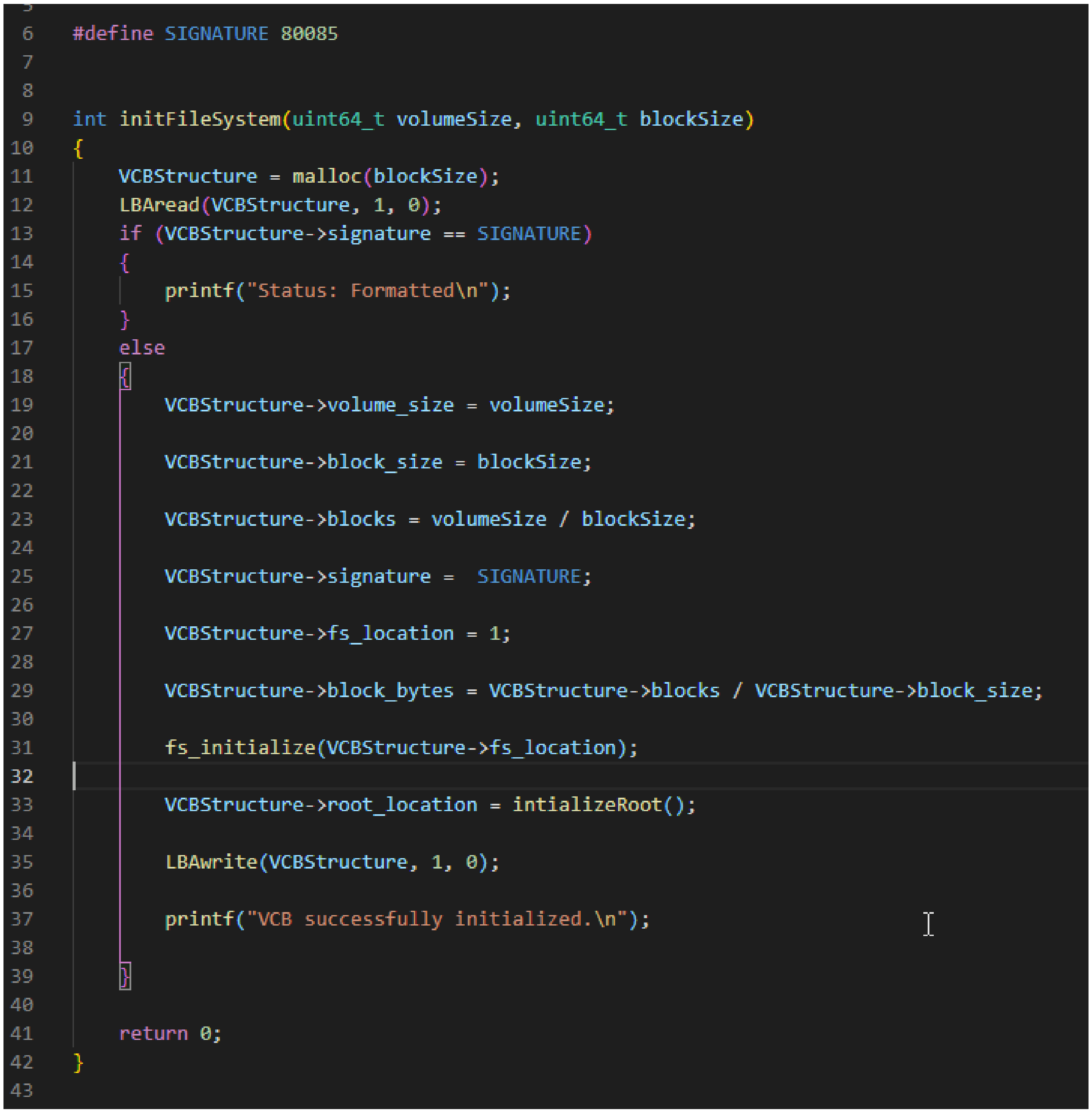
Our first issue came with the VCB. We were not completely sure as to what the function of the VCB was in a filesystem, but after some online research and discussion among ourselves, we found out the purpose of the VCB as well as how to implement it, as we explained below in the VCB structure section.

Another issue we had with this project is trying to figure out how to set up directories in our filesystem. We knew what directories were, but did not know how to code such a profound concept in our simple filesystem. After looking up how directories can be implemented and talking with other classmates, we found that directories were simply arrays that held other directory entries, which allowed us to implement our directories successfully.

Another issue we had came with fragmentation for our free space mapping (bitmap). Due to the way we marked free space, there ended up being unavoidable fragmentation, as after deallocating memory, the blocks were not grouped together based on if they were free or not free, resulting in files/directory contents being placed in discontiguous ranges of blocks. While we tried to solve this issue, we ended up realizing that with such a small filesystem and the ample memory we have assigned to it, defragmentation would not result in any major performance problems for our project (as there will, in most cases, always be enough contiguous blocks for our file system to write to). As such, we decided to forgo implementing a defragmentation method in our project as it was unnecessary and not a specified requirement.

Initializing Filesystem

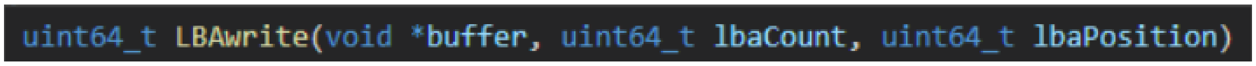
We are passing volume size and block size into our method called initFileSystem(), which is allocating space to our VCB structure. It reads the first block in our VCB structure and compares the signatures. If the signature does not match the signature in file system then we are filling our VCB structure volume size, block size, number of blocks , adding a signature , location ,block bytes, giving our filesystem a location in the vcb, initializing root directory and writing it inside our vcb struct. However, If our signature is being matched then this means that our file system was formatted and there is no other file system present.



```c
#define SIGNATURE 80085


int initFileSystem(uint64_t volumeSize, uint64_t blockSize)
{
    VCBStructure = malloc(blockSize);
    LBAread(VCBStructure, 1, 0);
    if (VCBStructure->signature == SIGNATURE)
    {
        printf("Status: Formatted\n");
    }
    else
    {
        VCBStructure->volume_size = volumeSize;

        VCBStructure->block_size = blockSize;

        VCBStructure->blocks = volumeSize / blockSize;

        VCBStructure->signature =  SIGNATURE;

        VCBStructure->fs_location = 1;

        VCBStructure->block_bytes = VCBStructure->blocks / VCBStructure->block_size;

        fs_initialize(VCBStructure->fs_location);

        VCBStructure->root_location = intializeRoot();

        LBAwrite(VCBStructure, 1, 0);

        printf("VCB successfully initialized.\n");

    }

    return 0;
}
```

LBAWrite

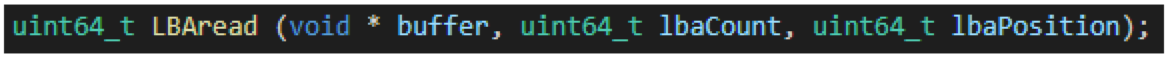
This function takes into parameters our struct, how long the struct/buffer, and position where to write. The object for this function is given to us by the professor in the fsLow.o file, so we merely needed to pass in our parameters for the function call. (how we use it)



```
uint64_t LBAwrite(void *buffer, uint64_t lbaCount, uint64_t lbaPosition)
```

LBARead

This function is almost the same as LBAWrite, except it reads from our buffer that we pass in, length and the position. LBARead is given by the professor and it essentially retrieves the relevant information from the LBA based on the passed in parameters.

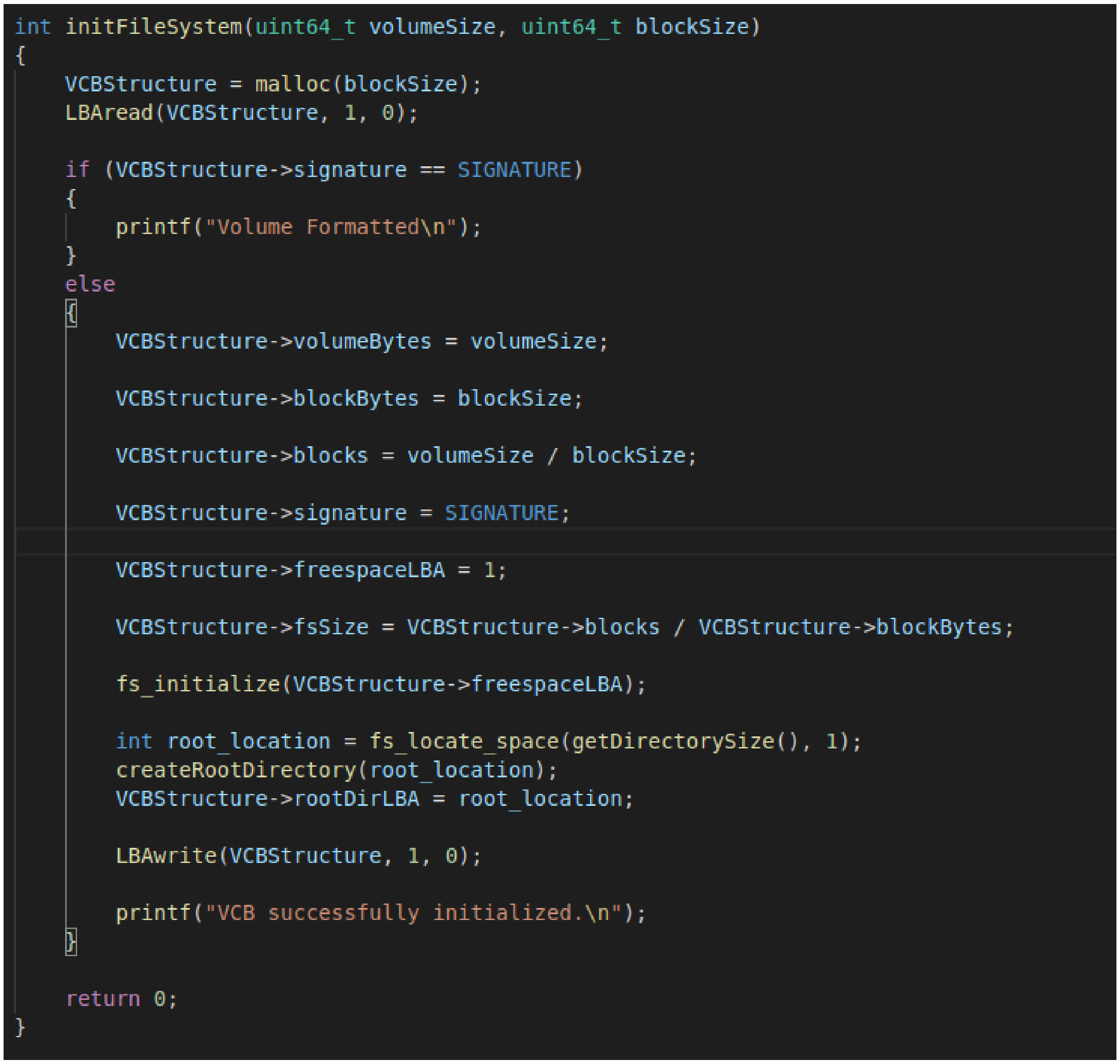


```
uint64_t LBAread (void * buffer, uint64_t lbaCount, uint64_t lbaPosition);
```

VCB Structure

The VCB is essentially the root of the filesystem as it is located in block 0 of the LBA. The VCB contains important information regarding the root of the filesystem as well as the amount of freespace present inside the filesystem. Furthermore, the VCB contains volume-specific information, meaning it also tells us how many blocks are present for the filesystem to make use of. It's important to note that the VCB structure does not inherently HOLD the root directory or freespace; it merely acts as a reference to where such information will be held in the LBA of the filesystem. Hence, if we had an empty VCB structure, or no VCB at all, then the filesystem itself cannot be initialized as important metadata information that the filesystem needs to operate will not be present.

In our fsInit.c file, we form the foundation of our file system by getting the VCB initialized when there is no sampleVolume. If there is already a sampleVolume, then the user will automatically skip the format process and enter the shell, after checking to make sure the directory has been initialized successfully (this is all done in the initFileSystem() function). We also ensure that our VCB structure is freed properly upon exiting the file system via an exit() command in our exitFileSystem() function. After the VCB initialization is completed successfully, we can now move on to free space management via the bitmap.c file!



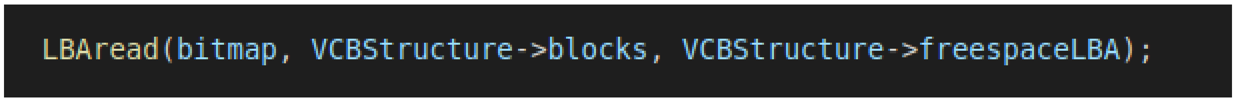
```
int initFileSystem(uint64_t volumeSize, uint64_t blockSize)
{
    VCBStructure = malloc(blockSize);
    LBAread(VCBStructure, 1, 0);

    if (VCBStructure->signature == SIGNATURE)
    {
        printf("Volume Formatted\n");
    }
    else
    {
        VCBStructure->volumeBytes = volumeSize;

        VCBStructure->blockBytes = blockSize;

        VCBStructure->blocks = volumeSize / blockSize;

        VCBStructure->signature = SIGNATURE;

        VCBStructure->freespaceLBA = 1;

        VCBStructure->fsSize = VCBStructure->blocks / VCBStructure->blockBytes;

        fs_initialize(VCBStructure->freespaceLBA);

        int root_location = fs_locate_space(getDirectorySize(), 1);
        createRootDirectory(root_location);
        VCBStructure->rootDirLBA = root_location;

        LBAwrite(VCBStructure, 1, 0);

        printf("VCB successfully initialized.\n");
    }

    return 0;
}
```

Bitmap

Free-Space Bitmaps are one method used to track allocated blocks by some file systems. In our filesystem, we created a boolean array to track the allocated/unallocated blocks that would be used by the OS as needed. The boolean array for the blocks would essentially tell the OS whether or not the corresponding block is free or allocated based on the value present in the array (1 if the block is allocated, 0 if it is free). Hence, when the OS needs to, for example, write a file, all it needs to do is scan the bitmap until it finds enough free locations (consecutive 0's in the array) to fit the file. While this method of implementation does pose certain concerns such as using up large amounts of space for large volumes and fragmentation risks, for our purposes, such a bitmap would work perfectly. The advantages of it in our small scale filesystem is that it is simple in implementation, has a fast random access allocation check, and there is a fixed cost as the bitmap does not need to grow or shrink meaning fewer lookups are needed to find the desired information.

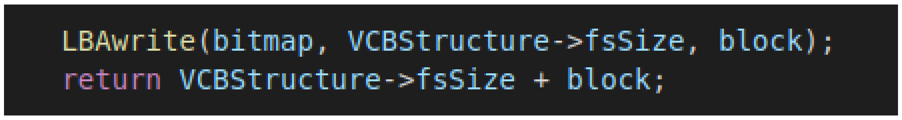
In the bitmap.c file (which helps with formatting/allocation of the freespace), To indicate a block(s) as used, we first have to access our bitmap, in this case we read from where it is located (done through the fs_initialize() function). What the free space map needs to be able to do is find the amount of free space from the Volume. This was accomplished by creating a bool* array called bitmap (where 1 would mean the relevant block is in use and 0 otherwise) and using calloc to allocate memory with its data all to zero. There are four important features that our bitmap will manage: the freespace initialization (fs_initialize()), free space locating (fs_locate_space()), indicating the block as used or otherwise (fs_locate_space()), and freeing memory (deallocate()).

The freespace init goes along with the VCB init, in which the freespace map location needs to be stored inside the VCB structure. This is so that in future bitmap reads, we can do something similar as to the read below (our VCB_struct stores the free space location as freespaceLBA):



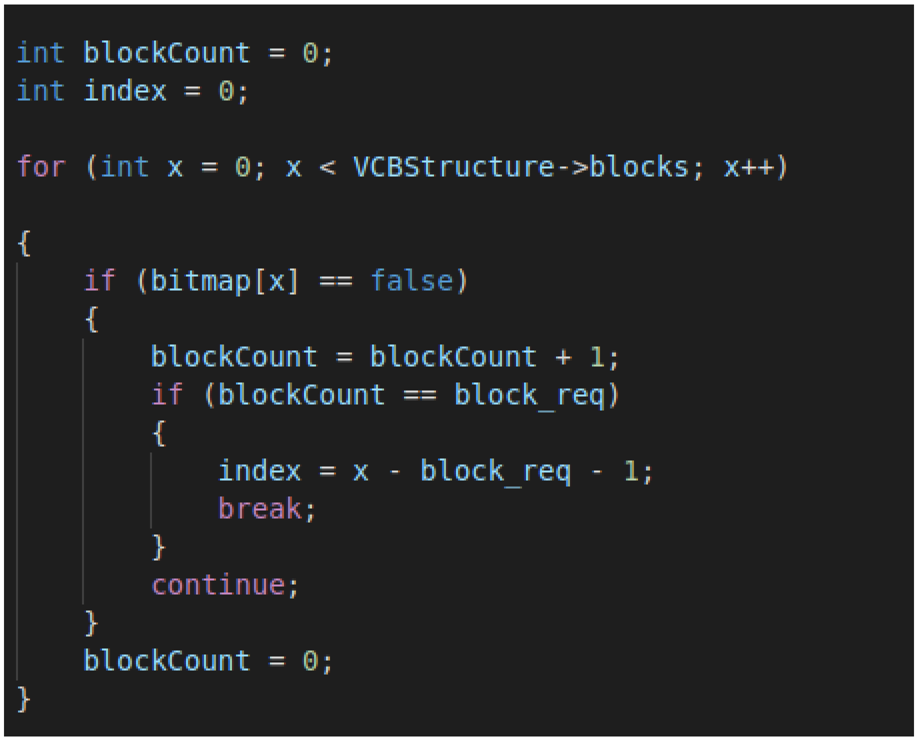
```
LBAread(bitmap, VCBStructure->blocks, VCBStructure->freespaceLBA);
```

To initialize the bitmap, we create a boolean array with all zeros then account for block 0, which is the VCB that is also marked with 1. Furthermore, we also include the size of the bitmap, which would be from the first block to the final one (no need for specification). We mark the blocks as used by changing the 0 to a 1, creating our boolean array which is saved inside the Volume using LBAwrite.



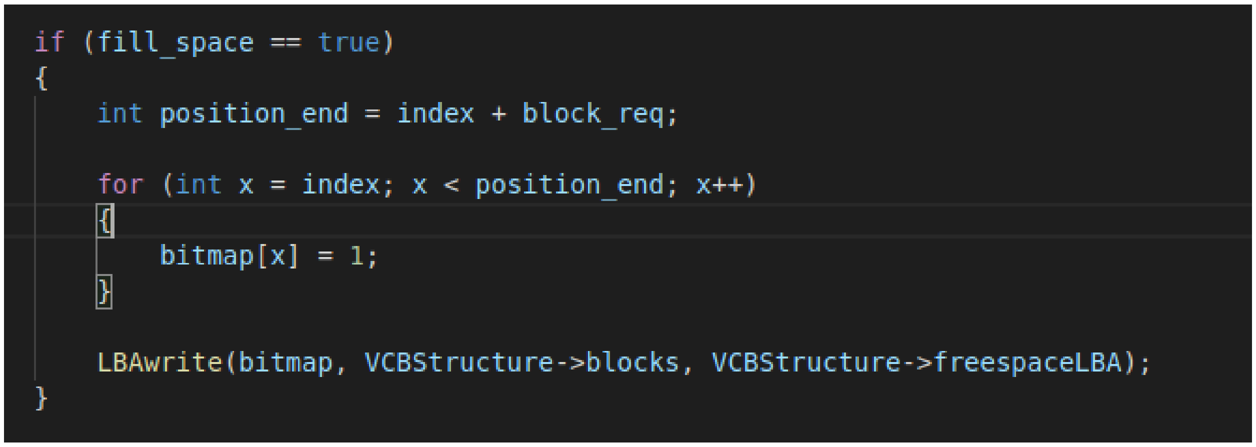
```
LBAwrite(bitmap, VCBStructure->fsSize, block);
return VCBStructure->fsSize + block;
```

In the fs_locate_space() function, we locate free space by accessing our bitmap via a LBAread function call to read from where the bitmap is located. Afterwards, using simple for-loops, to locate a consecutive amount of zeros (which represent free space as mentioned earlier). In the end, it would return the index as seen in the following image.

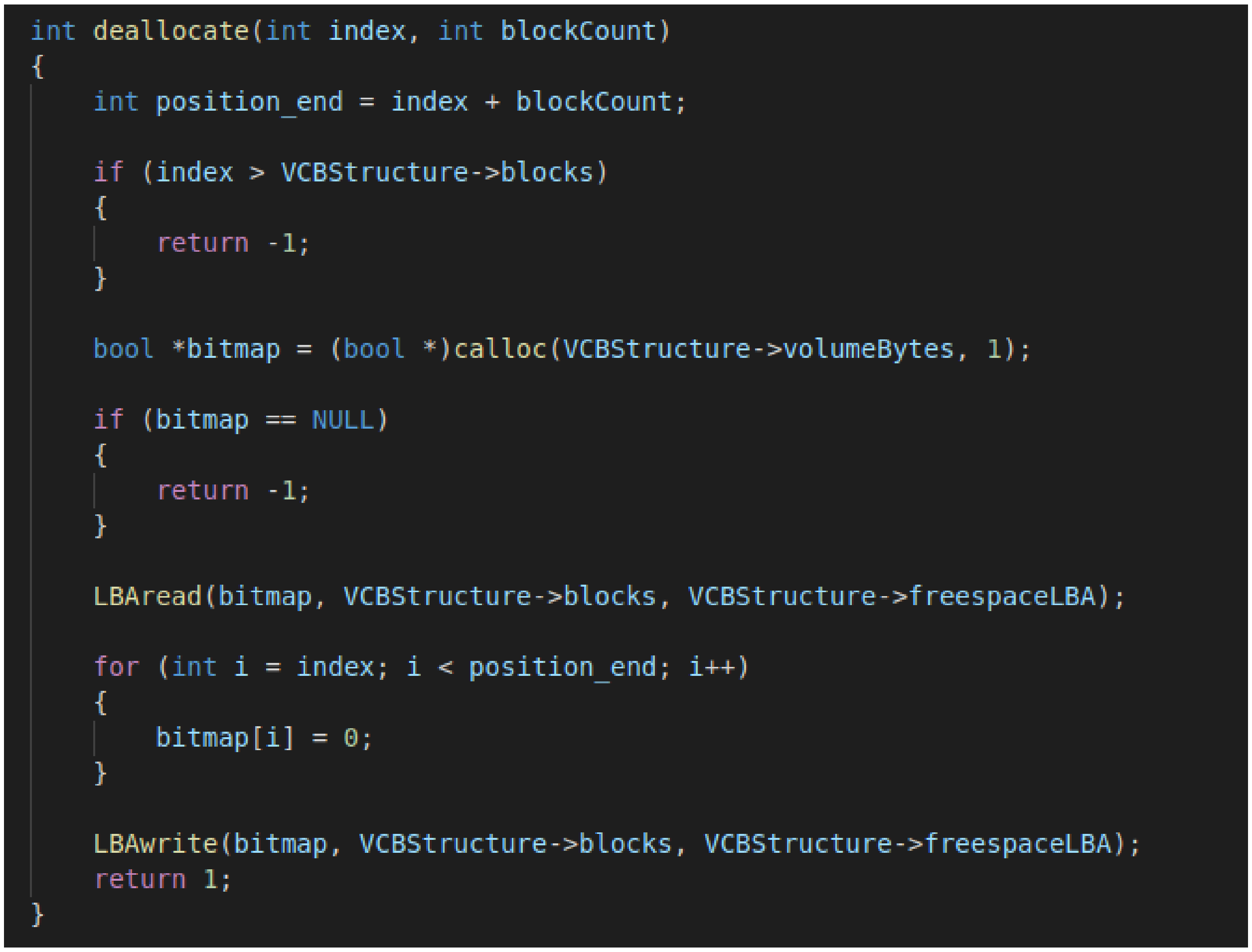


```c
int blockCount = 0;
int index = 0;

for (int x = 0; x < VCBStructure->blocks; x++)

{
    if (bitmap[x] == false)
    {
        blockCount = blockCount + 1;
        if (blockCount == block_req)
        {
            index = x - block_req - 1;
            break;
        }
        continue;
    }
    blockCount = 0;
}
```

For marking blocks as used/in use, we start by validating the data passed in to our bitmap; if the block that needs to be marked as used is out of scope, then it would obviously be impossible to access said index, meaning it would return an error value. After such validation is completed, we index to the starting block which needs to be marked as used and proceed to mark all further blocks that are used after gathering which ones are already in use.


```
if (fill_space == true)
{
    int position_end = index + block_req;

    for (int x = index; x < position_end; x++)
    {
        bitmap[x] = 1;
    }

    LBAwrite(bitmap, VCBStructure->blocks, VCBStructure->freespaceLBA);
}
```

Finally, to indicate a block as free (done in the deallocate() function), we first have to read from where it is located and validate the data that is passed in. Provided the block that needs to be marked is in scope, we simply index to the starting block that needs to be marked as free (with a boolean value of 0) and repeat this process with any further blocks that needs to be marked as free. While this process of freeing up memory will end up causing fragmentation (as we do not proceed to sort the free and not free blocks), the professor did not specify a need for defragmentation. While we attempted to do so, it ended up being much too difficult, and we ended up deciding to not go ahead and implement a defragmentation feature for our file system.



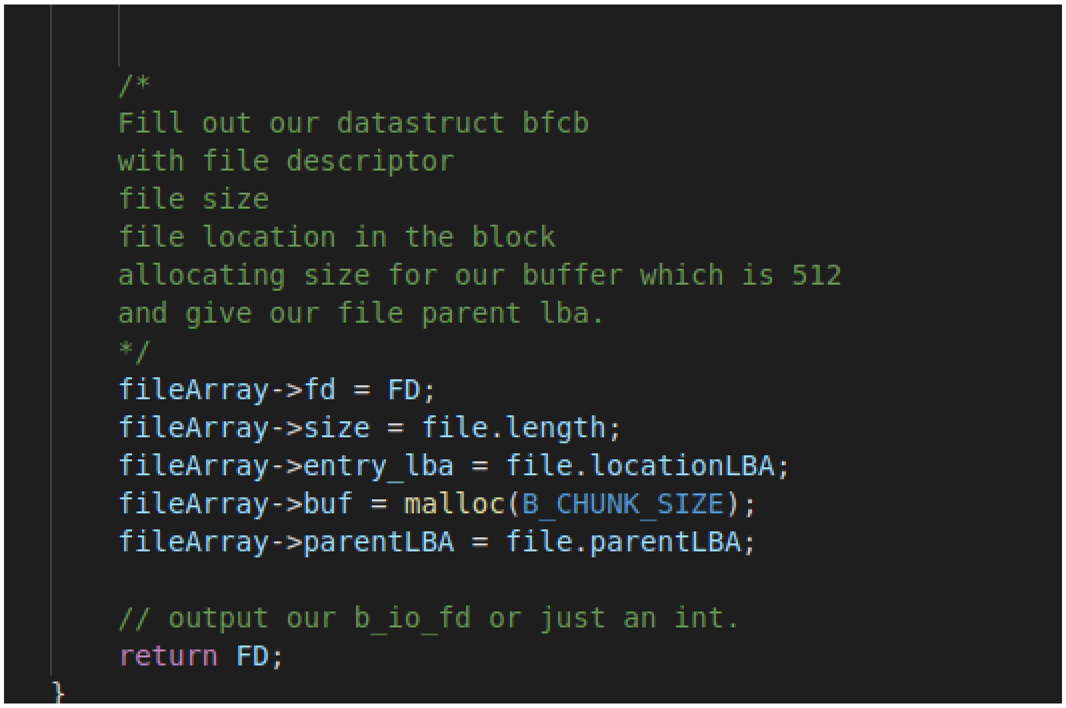
```
int deallocate(int index, int blockCount)
{
    int position_end = index + blockCount;

    if (index > VCBStructure->blocks)
    {
        return -1;
    }

    bool *bitmap = (bool *)calloc(VCBStructure->volumeBytes, 1);

    if (bitmap == NULL)
    {
        return -1;
    }

    LBAread(bitmap, VCBStructure->blocks, VCBStructure->freespaceLBA);

    for (int i = index; i < position_end; i++)
    {
        bitmap[i] = 0;
    }

    LBAwrite(bitmap, VCBStructure->blocks, VCBStructure->freespaceLBA);
    return 1;
}
```

Directory Entries

A directory entry is an entry inside of a directory. They can either be a file or a folder (another directory). If another directory is created it needs to be another array of directory entries since more can be added to the new directory. Each directory in our file system has a position in the LBA, and a length. Whenever we want to access a directory we just need to use the LBAread function to access the starting block and the size of the directory. Once the shell is initialized, we load our root directory into memory to keep track of where the user is in our system. When we want to create a new directory entry we would simply just need to add an array of fdDir objects into our current working directory but when we want to create a file we would only need to add a single fdDir object.

Read, Write and Close

Our b_io.c file consist of b_fcb struct which holds values such as file descriptor (gives us a number or -1 if all file descriptors are in use), it has a buffer that will store the data that is being passed to us, it has a file size, entry_lba which is our first "block" of the file. Max FCBS value of 20, startup value which is boolean int variable 0 is initialized and any other value is initialized. b_getFCB() gives us an empty file descriptor value starting from 0 till 20 (same as in our assignment 2b). If the file descriptors are busy it will output -1 ,meaning that we are not able to use file descriptors (since they are all busy). Open function gets a file descriptor forms a struct for our file which we are opening, creates an fddir for our file in order to get file information like size ,filename. In open we are checking for location of the file if it is at 0 in order to see that it is in our first block and then we fill up the struct for our fcbArray, which stores file descriptor, file size, file location in the block, buffer with allocated space and parent LBA. It returns a file descriptor for the file that is open.



```
    /*
    Fill out our datastruct bfcb
    with file descriptor
    file size
    file location in the block
    allocating size for our buffer which is 512
    and give our file parent lba.
    */
    fileArray->fd = FD;
    fileArray->size = file.length;
    fileArray->entry_lba = file.locationLBA;
    fileArray->buf = malloc(B_CHUNK_SIZE);
    fileArray->parentLBA = file.parentLBA;

    // output our b_io_fd or just an int.
    return FD;
}
```

Write function is taking three parameters file descriptor (the file array number) , the buffer (the one which we are writing into) and count size of the bytes to be written into a buffer, it handles cases such as if the length passed is more than what we have in buffer in this case we write whatever space we have in the buffer and we can not overfill the buffer due to the segmentation faults that will be given to us. The read function is similar to our write function except instead of writing to a buffer we are reading into a buffer, having three parameters that are being passed such as filedescriptor , buffer and the size of the data. It checks for different cases such as if the size of the file passed into the method is larger than bytes we have left out then we just read whatever is left in the buffer at the end we output the size of the data that needs to be read. At the end of the b_io.c we are closing the file in order to free up memory to avoid any memory leakages and set our buffer back to the size of our b_fcb structure.

MFS: User Command Interface

This is the user command interface provided to us by the professor. We use these methods in our specific command implementations.

fs_mkdir:

The fs_mkdir method is used to create a directory.

fs_rmdir:

The fs_rmdir method is used to remove a directory using a recursive solution.

fs_opendir:

The fs_opendir method will search for the name of the directory and fetch a directory.

fs_readdir:

The fs_readdir method is used to open a directory and sets an index to prepare for the next read.

fs_closedir:

The fs_closedir method is used to reset the count and set the opened pointer to null.

fs_getcwd:

The fs_getcwd method is to get the current working directory.

fs_setcwd:

The fs_setcwd method is to change the current working directory.

fs_isFile:

The fs_isFile method is to check if it is a file.

fs_isDir:

The fs_isDir method is to check if it's a directory.

fs_delete:

The fs_delete method is to delete a directory.

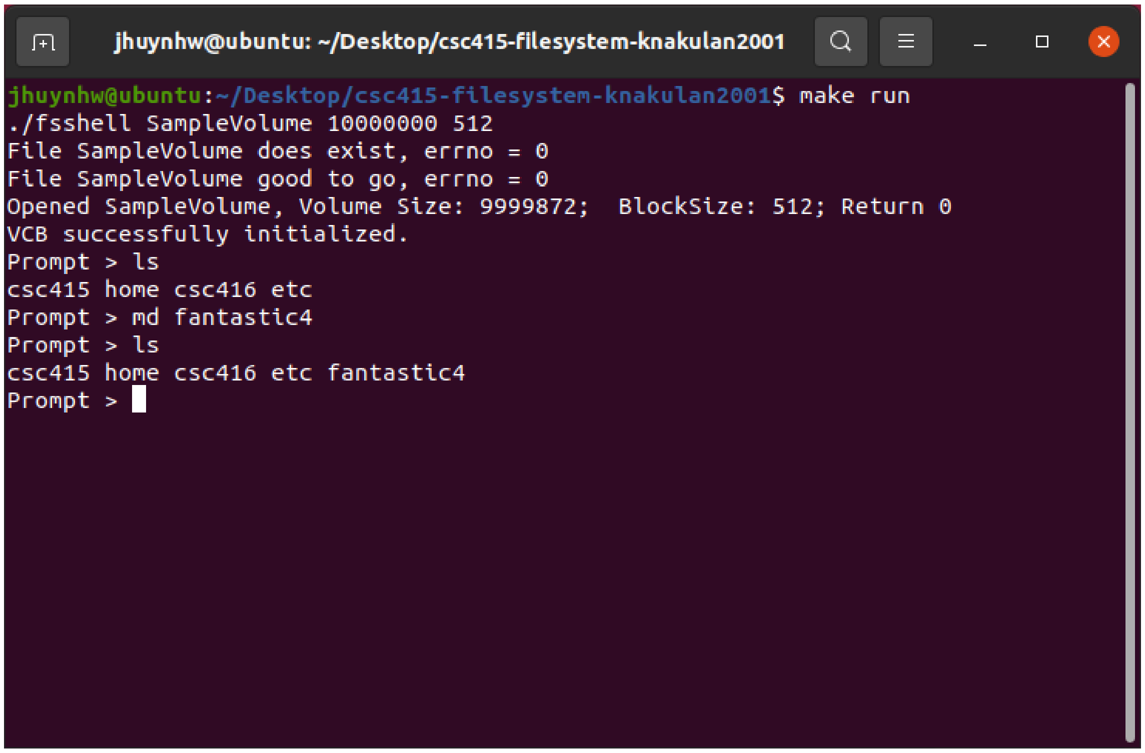
fs_stat:

The fs_stat method is to get the information of a directory or file.

Shell Commands:

Make Directory (md)

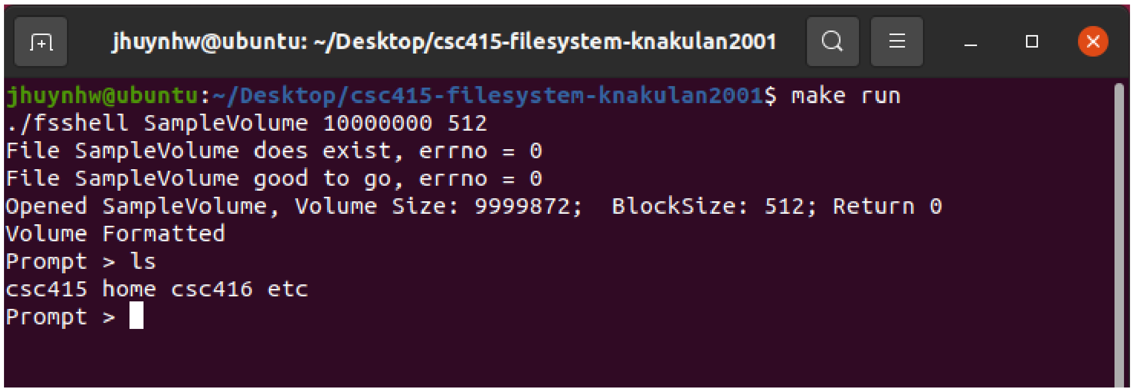
The md command will create a directory in the current directory or if a path is given, mkdir will create a directory to the given path.



```
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
VCB successfully initialized.
Prompt > ls
csc415 home csc416 etc
Prompt > md fantastic4
Prompt > ls
csc415 home csc416 etc fantastic4
Prompt > 
```

List Directory (ls)

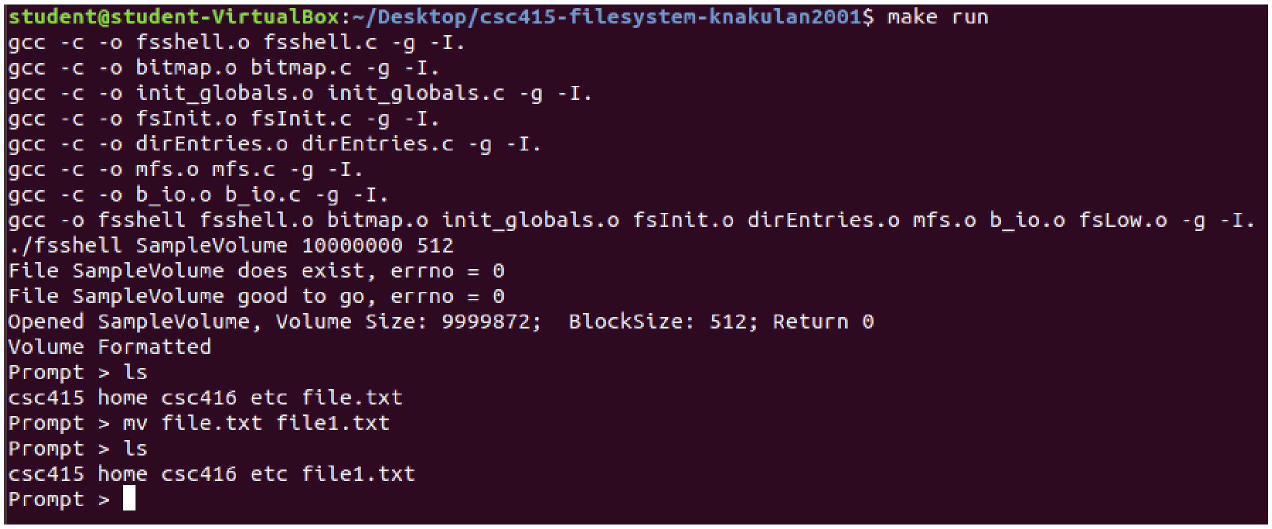
The ls command will display all the files in the currmd ent working directory.



```
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc
Prompt > 
```

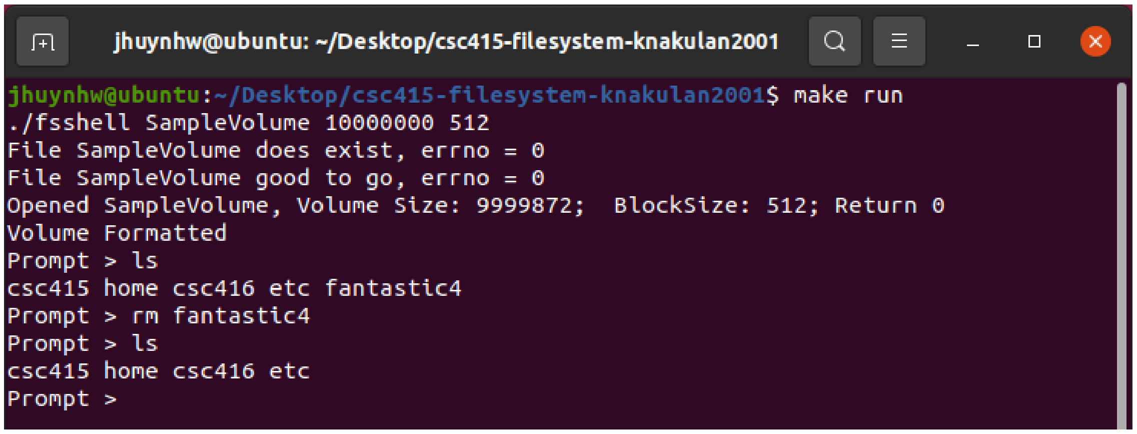
Move (mv)

The mv command will move a file to a specified relative or absolute path.


```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make run
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o bitmap.o bitmap.c -g -I.
gcc -c -o init_globals.o init_globals.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o dirEntries.o dirEntries.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o bitmap.o init_globals.o fsInit.o dirEntries.o mfs.o b_io.o fsLow.o -g -I.
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc file.txt
Prompt > mv file.txt file1.txt
Prompt > ls
csc415 home csc416 etc file1.txt
Prompt > 
```

Remove (rm)

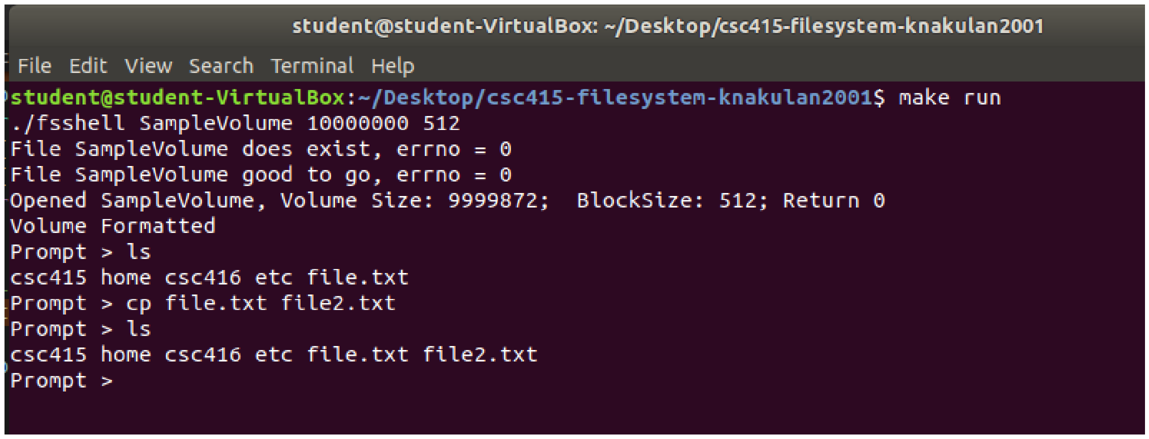
The rm command will delete any given directory or directory entry recursively (this was one of the hardest commands in terms of implementation).



```
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc fantastic4
Prompt > rm fantastic4
Prompt > ls
csc415 home csc416 etc
Prompt > 
```

Copy (cp)

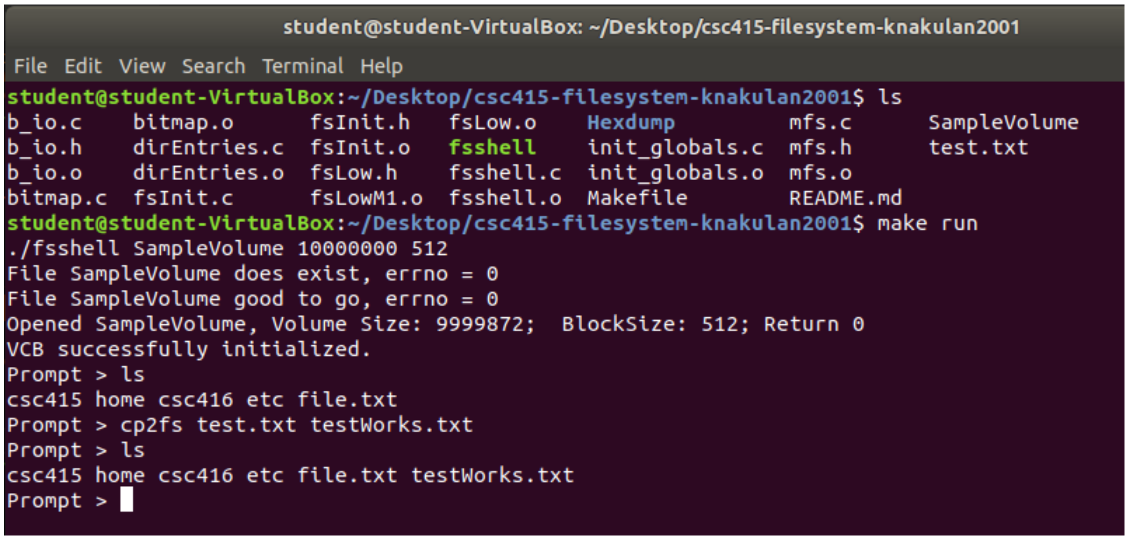
The copy command is used to copy any given file from one directory or relative/absolute path to another directory or relative/absolute path given a path or current working directory.


student@student-VirtualBox: ~/Desktop/csc415-filesystem-knakulan2001
File Edit View Search Terminal Help

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc file.txt
Prompt > cp file.txt file2.txt
Prompt > ls
csc415 home csc416 etc file.txt file2.txt
Prompt >
```

CP2FS (cp2fs) [From Test FS to Linux FS]

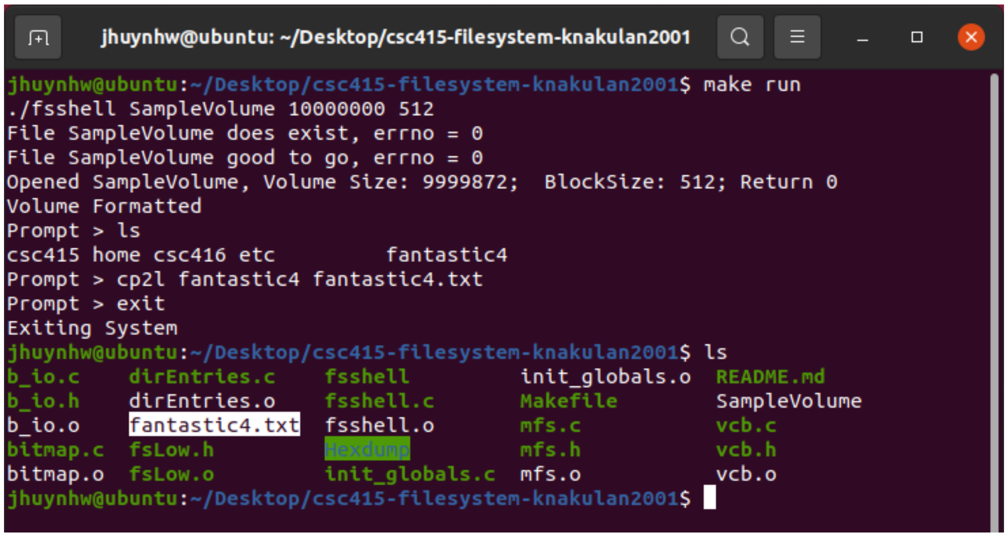
The cp2fs command is used to copy a file from the local linux file system to our test file system. However, note that this command will not work if you do not have the permissions to access the file in the first place.


student@student-VirtualBox: ~/Desktop/csc415-filesystem-knakulan2001
File Edit View Search Terminal Help

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ ls
b_io.c     bitmap.o       fsInit.h    fsLow.o    Hexdump        mfs.c      SampleVolume
b_io.h     dirEntries.c   fsInit.o    fsshell    init_globals.c mfs.h      test.txt
b_io.o     dirEntries.o   fsLow.h     fsshell.c  init_globals.o mfs.o
bitmap.c   fsInit.c       fsLowM1.o   fsshell.o  Makefile       README.md
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
VCB successfully initialized.
Prompt > ls
csc415 home csc416 etc file.txt
Prompt > cp2fs test.txt testWorks.txt
Prompt > ls
csc415 home csc416 etc file.txt testWorks.txt
Prompt >
```

CP2Linux (cp2l) [From Linux FS to Test FS]

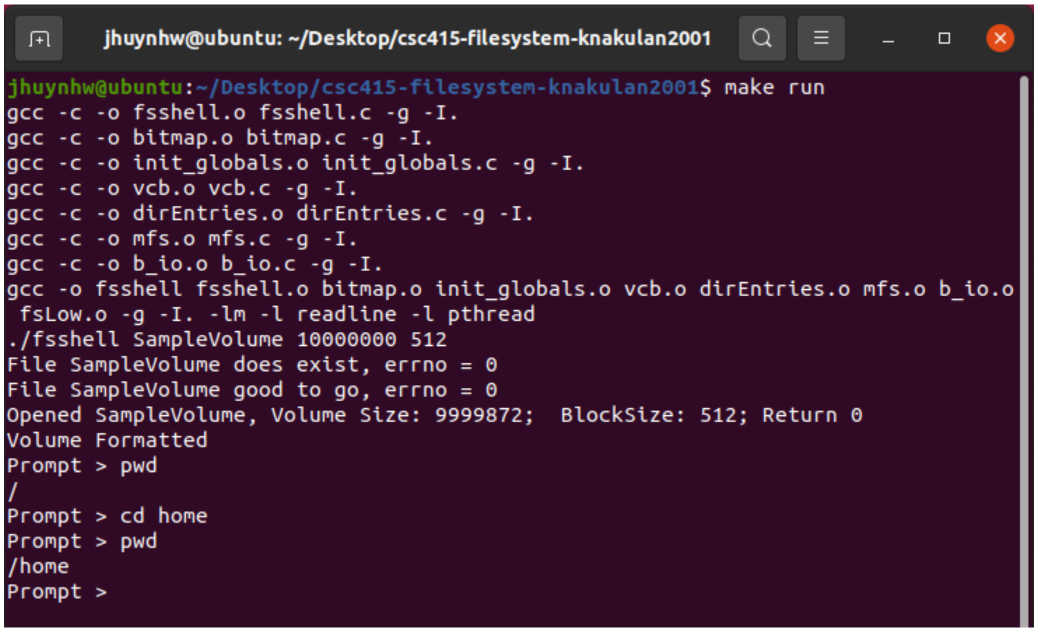
The cp2l command is used to copy a file from our file system to the local linux file system.



```
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc          fantastic4
Prompt > cp2l fantastic4 fantastic4.txt
Prompt > exit
Exiting System
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ ls
b_io.c     dirEntries.c    fsshell         init_globals.o  README.md
b_io.h     dirEntries.o    fsshell.c       Makefile        SampleVolume
b_io.o     fantastic4.txt  fsshell.o       mfs.c           vcb.c
bitmap.c   fsLow.h         Hexdump         mfs.h           vcb.h
bitmap.o   fsLow.o         init_globals.c  mfs.o           vcb.o
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$
```

Print Working Directory (pwd)

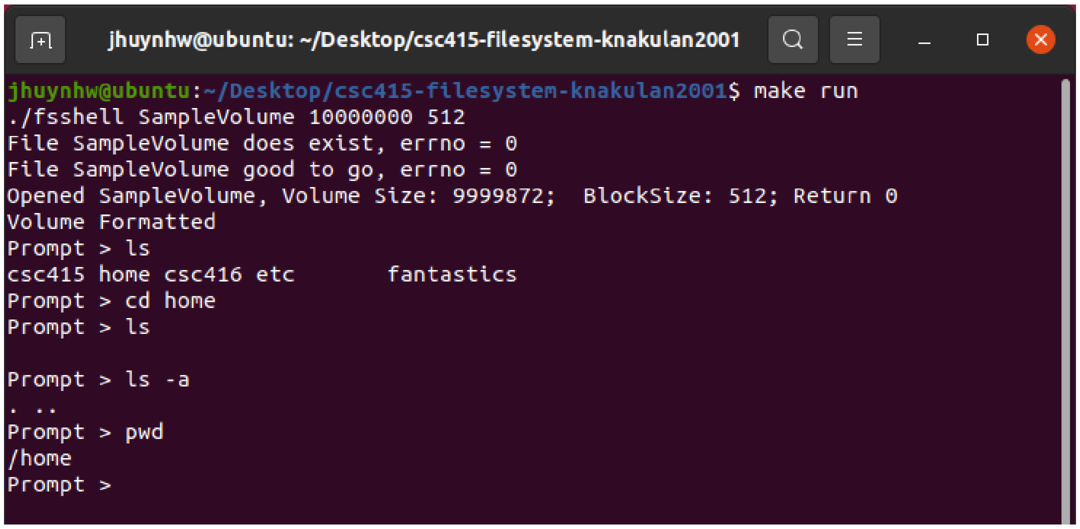
The pwd command will display the current absolute path (where you are in the file system).



```
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o bitmap.o bitmap.c -g -I.
gcc -c -o init_globals.o init_globals.c -g -I.
gcc -c -o vcb.o vcb.c -g -I.
gcc -c -o dirEntries.o dirEntries.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o bitmap.o init_globals.o vcb.o dirEntries.o mfs.o b_io.o
 fsLow.o -g -I. -lm -l readline -l pthread
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > pwd
/
Prompt > cd home
Prompt > pwd
/home
Prompt >
```

Change Directory (cd)

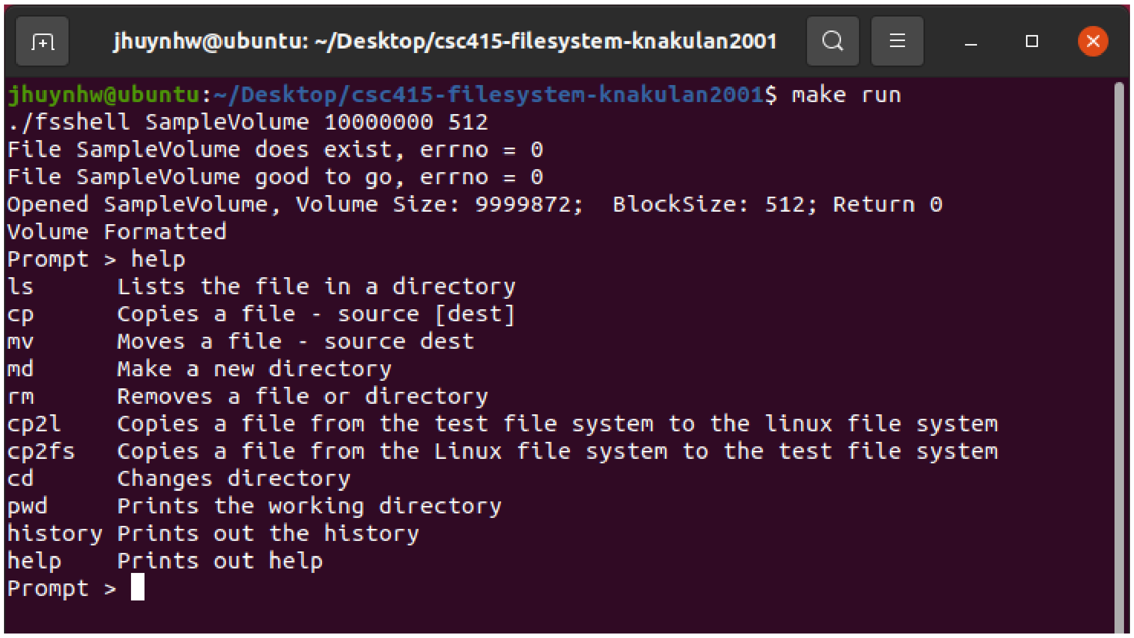
The cd command is used to traverse through the Linux file system.



```
jhuynhw@ubuntu: ~/Desktop/csc415-filesystem-knakulan2001
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc       fantastics
Prompt > cd home
Prompt > ls

Prompt > ls -a
. ..
Prompt > pwd
/home
Prompt >
```

Help (help)

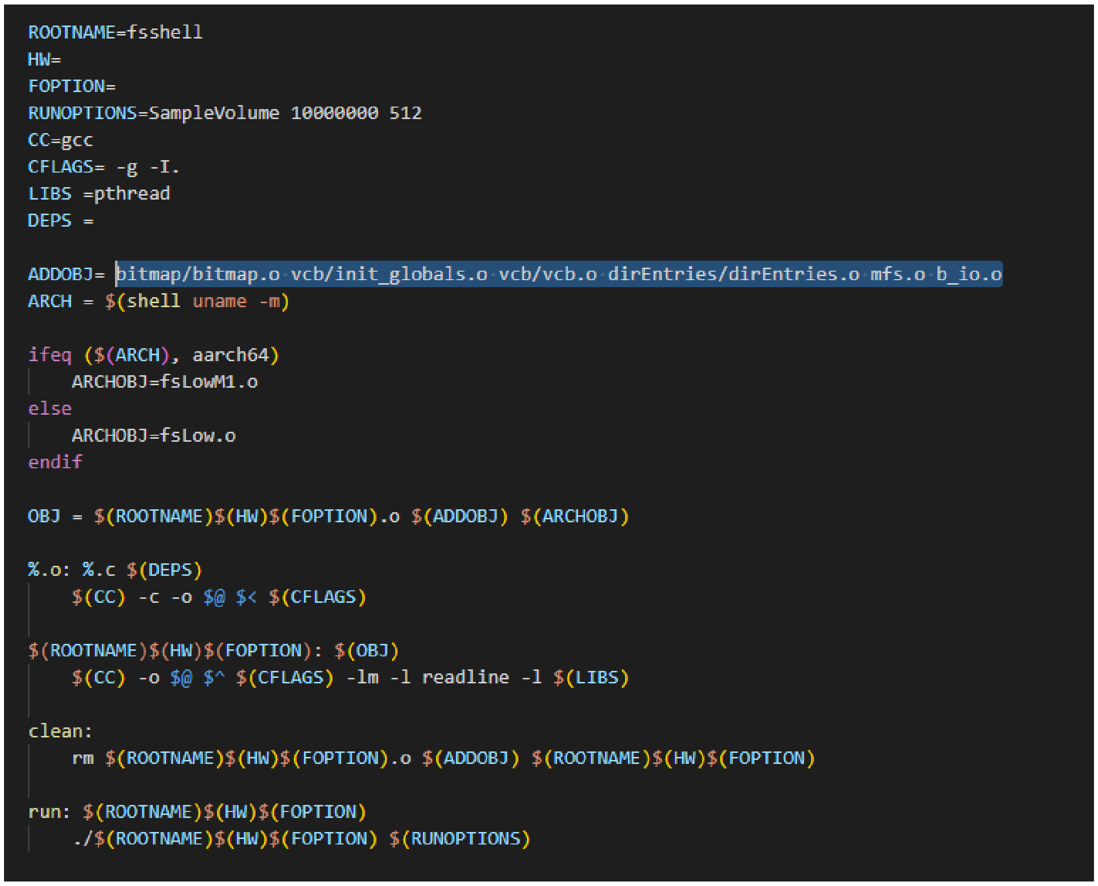
The help command will display the list of all the working commands in our file system.



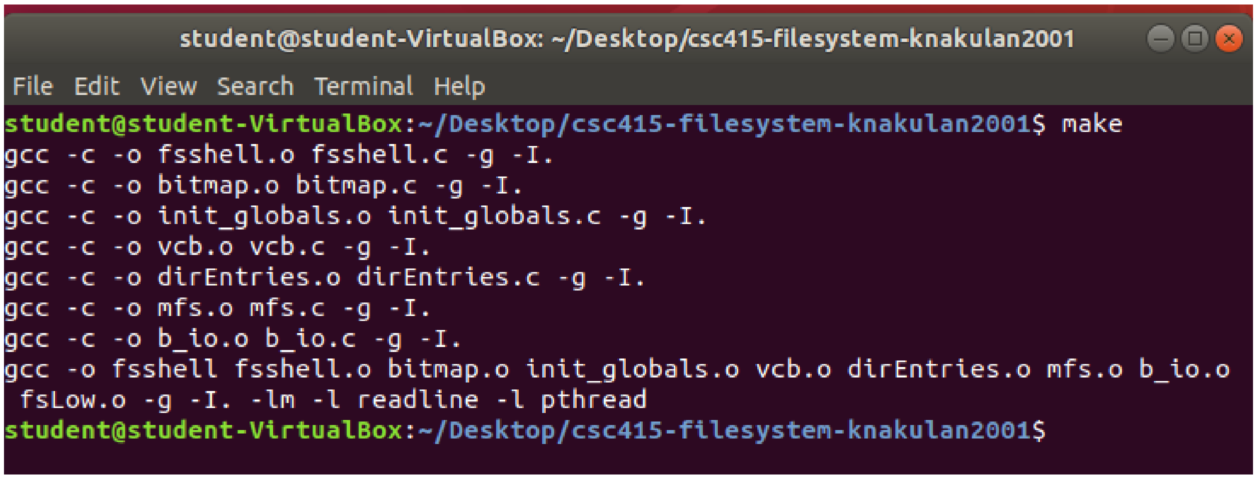
```
jhuynhw@ubuntu: ~/Desktop/csc415-filesystem-knakulan2001
jhuynhw@ubuntu:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt >
```

Screenshot of Compilation:

To link and compile our source code, we added other object files such as the bitmap, VCB, and directory entries to our makefile.



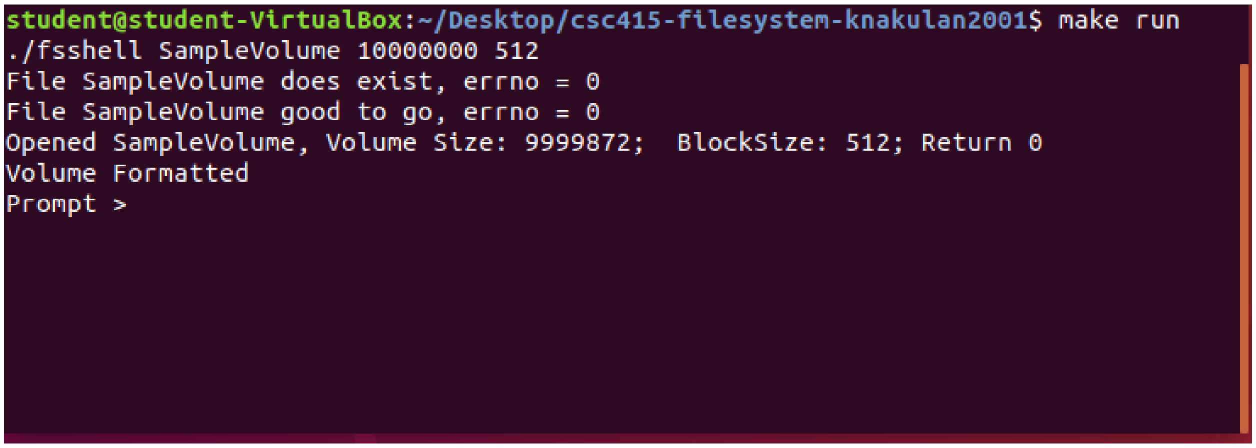
```
ROOTNAME=fsshell
HW=
FOPTION=
RUNOPTIONS=SampleVolume 10000000 512
CC=gcc
CFLAGS= -g -I.
LIBS =pthread
DEPS =

ADDOBJ= bitmap/bitmap.o vcb/init_globals.o vcb/vcb.o dirEntries/dirEntries.o mfs.o b_io.o
ARCH = $(shell uname -m)

ifeq ($(ARCH), aarch64)
    ARCHOBJ=fsLowM1.o
else
    ARCHOBJ=fsLow.o
endif

OBJ = $(ROOTNAME)$(HW)$(FOPTION).o $(ADDOBJ) $(ARCHOBJ)

%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

$(ROOTNAME)$(HW)$(FOPTION): $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS) -lm -l readline -l $(LIBS)

clean:
    rm $(ROOTNAME)$(HW)$(FOPTION).o $(ADDOBJ) $(ROOTNAME)$(HW)$(FOPTION)

run: $(ROOTNAME)$(HW)$(FOPTION)
    ./$(ROOTNAME)$(HW)$(FOPTION) $(RUNOPTIONS)
```



```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o bitmap.o bitmap.c -g -I.
gcc -c -o init_globals.o init_globals.c -g -I.
gcc -c -o vcb.o vcb.c -g -I.
gcc -c -o dirEntries.o dirEntries.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o bitmap.o init_globals.o vcb.o dirEntries.o mfs.o b_io.o
 fsLow.o -g -I. -lm -l readline -l pthread
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$
```

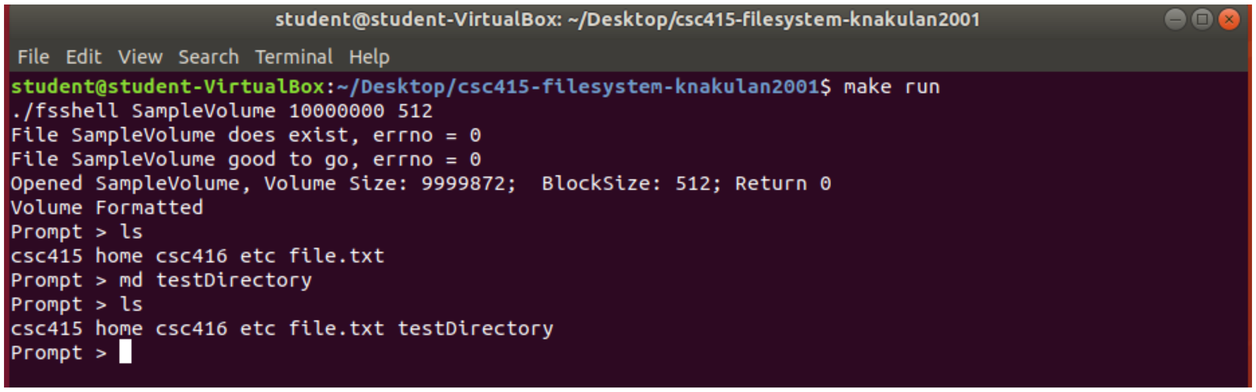
Screen Shot(s) of the Execution of the Program:

<u>Make Run (Starts FileSystem):</u>



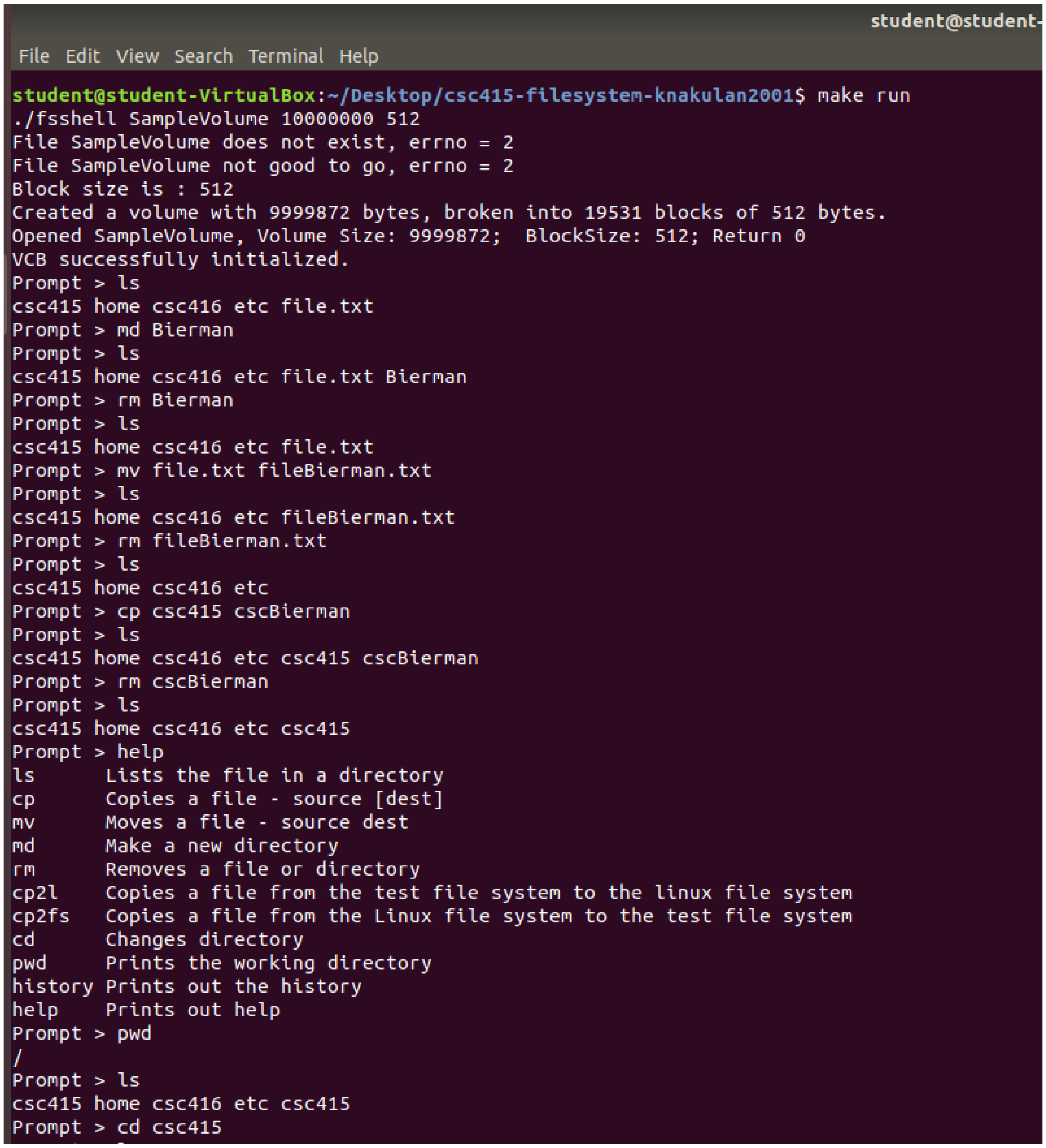
```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt >
```

Make Directory (md):

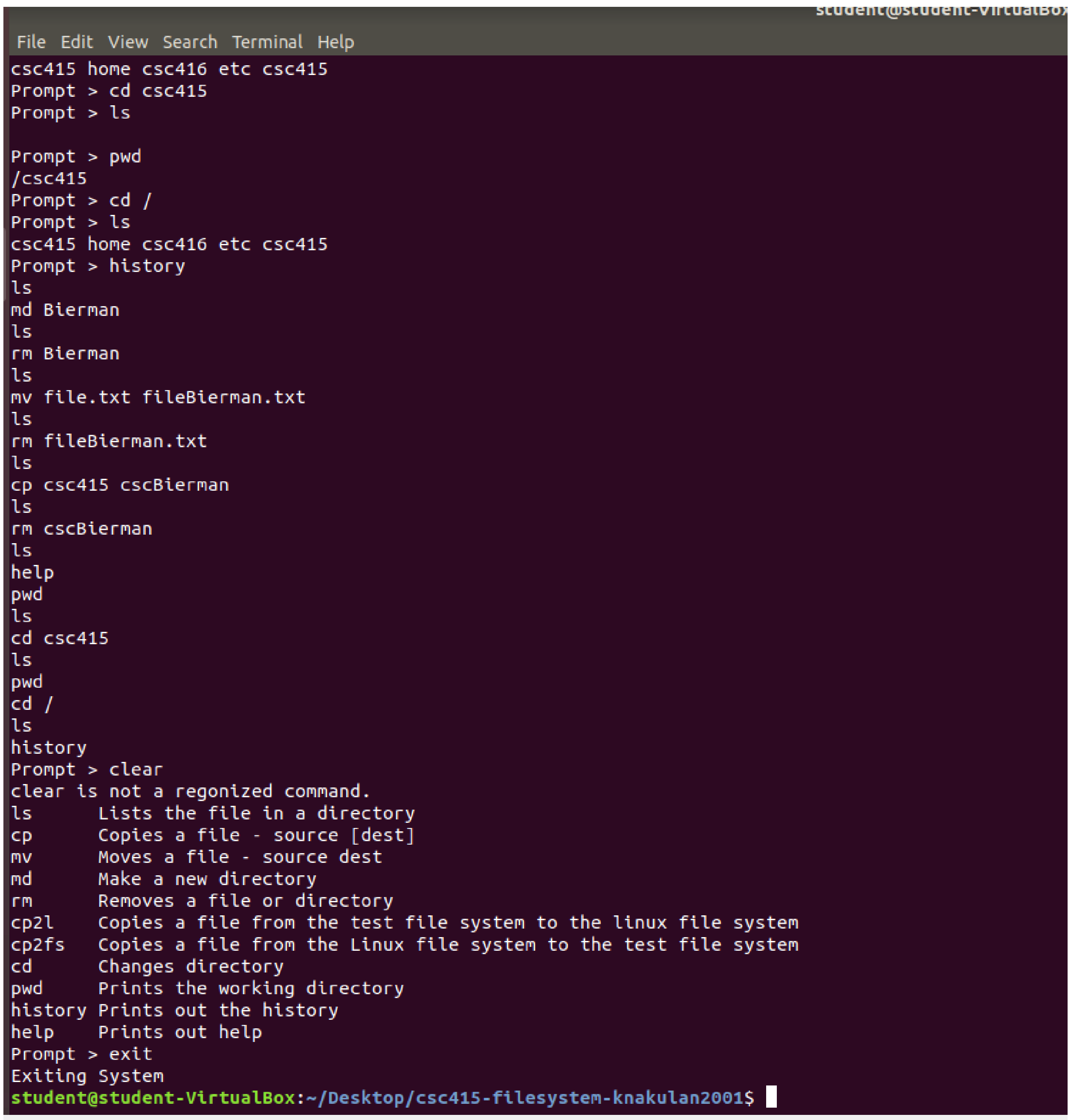

student@student-VirtualBox: ~/Desktop/csc415-filesystem-knakulan2001
File Edit View Search Terminal Help

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Volume Formatted
Prompt > ls
csc415 home csc416 etc file.txt
Prompt > md testDirectory
Prompt > ls
csc415 home csc416 etc file.txt testDirectory
Prompt > 
```

Test Run of File System:



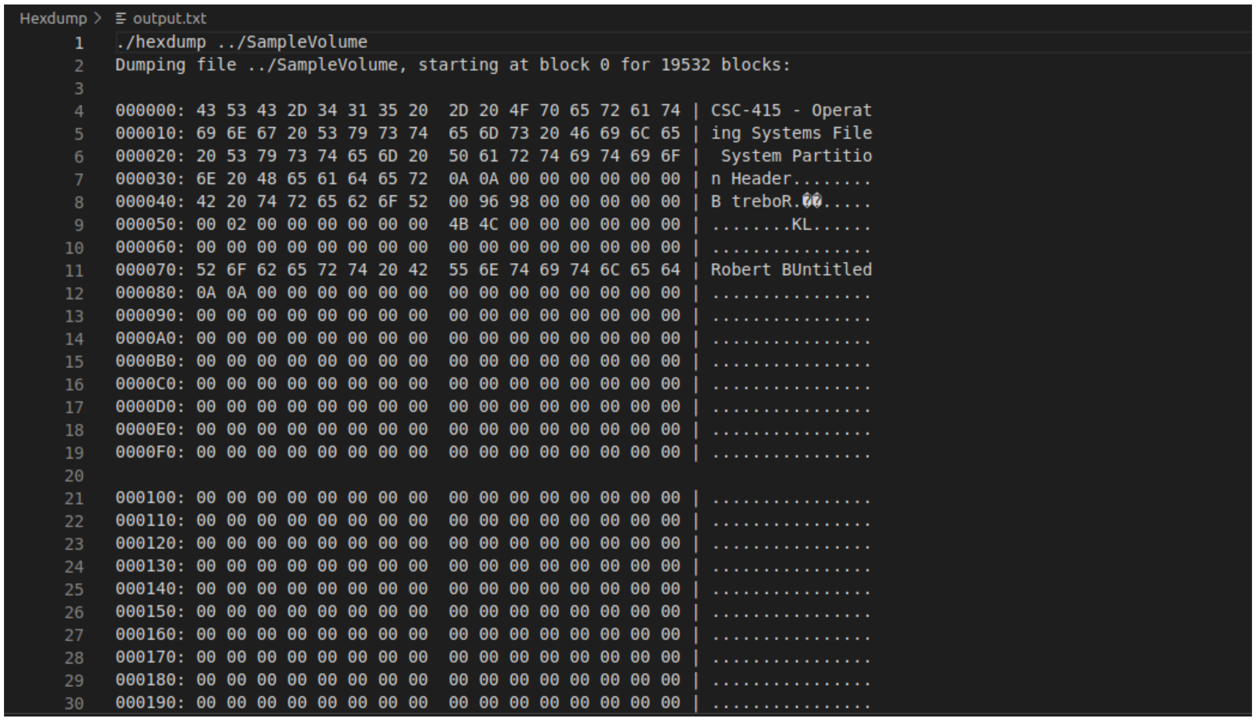
```
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
VCB successfully initialized.
Prompt > ls
csc415 home csc416 etc file.txt
Prompt > md Bierman
Prompt > ls
csc415 home csc416 etc file.txt Bierman
Prompt > rm Bierman
Prompt > ls
csc415 home csc416 etc file.txt
Prompt > mv file.txt fileBierman.txt
Prompt > ls
csc415 home csc416 etc fileBierman.txt
Prompt > rm fileBierman.txt
Prompt > ls
csc415 home csc416 etc
Prompt > cp csc415 cscBierman
Prompt > ls
csc415 home csc416 etc csc415 cscBierman
Prompt > rm cscBierman
Prompt > ls
csc415 home csc416 etc csc415
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt > pwd
/
Prompt > ls
csc415 home csc416 etc csc415
Prompt > cd csc415
```


student@student-VirtualBox
File Edit View Search Terminal Help

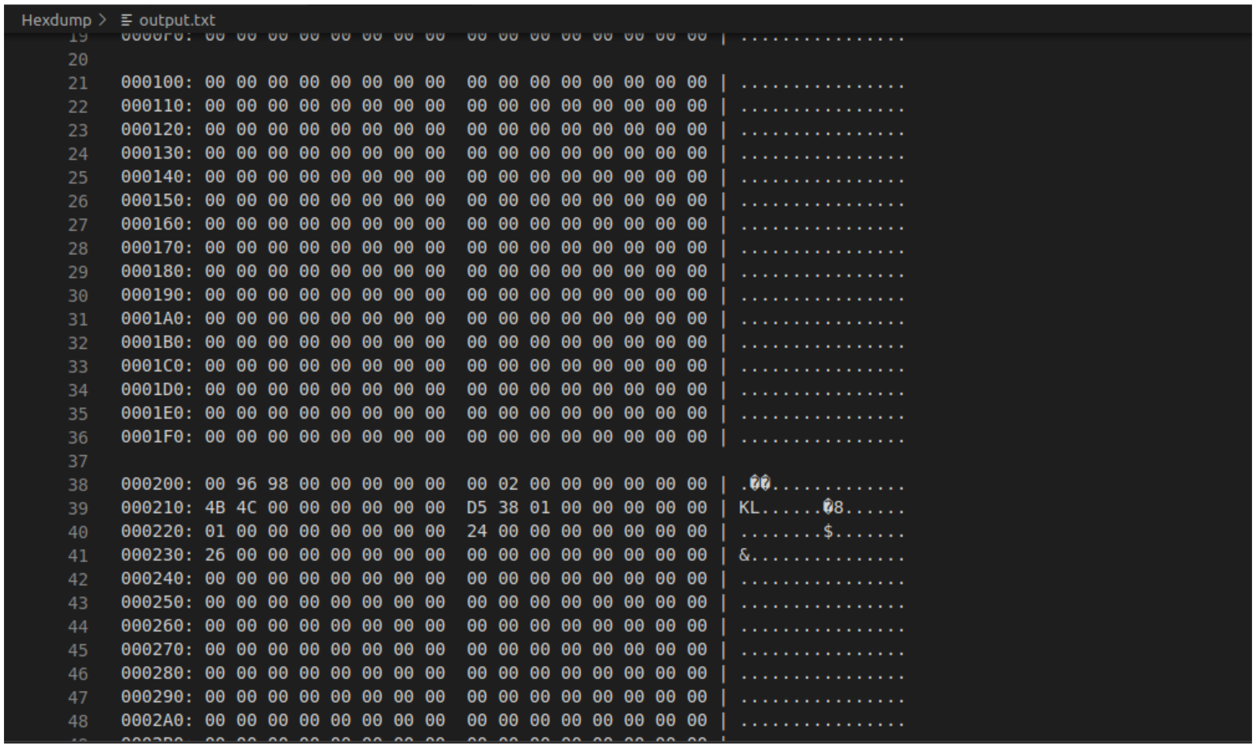
```
csc415 home csc416 etc csc415
Prompt > cd csc415
Prompt > ls

Prompt > pwd
/csc415
Prompt > cd /
Prompt > ls
csc415 home csc416 etc csc415
Prompt > history
ls
md Bierman
ls
rm Bierman
ls
mv file.txt fileBierman.txt
ls
rm fileBierman.txt
ls
cp csc415 cscBierman
ls
rm cscBierman
ls
help
pwd
ls
cd csc415
ls
pwd
cd /
ls
history
Prompt > clear
clear is not a regonized command.
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt > exit
Exiting System
student@student-VirtualBox:~/Desktop/csc415-filesystem-knakulan2001$
```

Hexdump Blocks

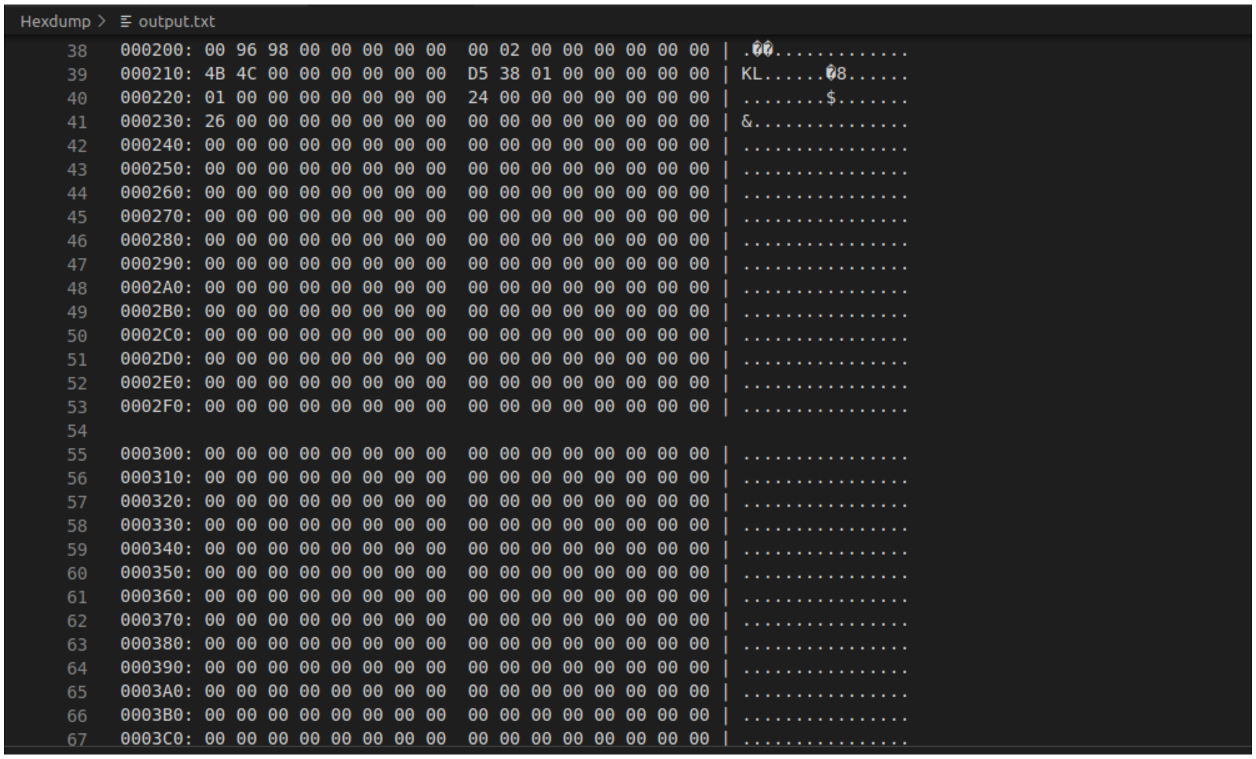
The hexdump displays every block from the SampleVolume. The screenshots below show the first few blocks which represent the VCB (block 0) and freespace map (bitmap).



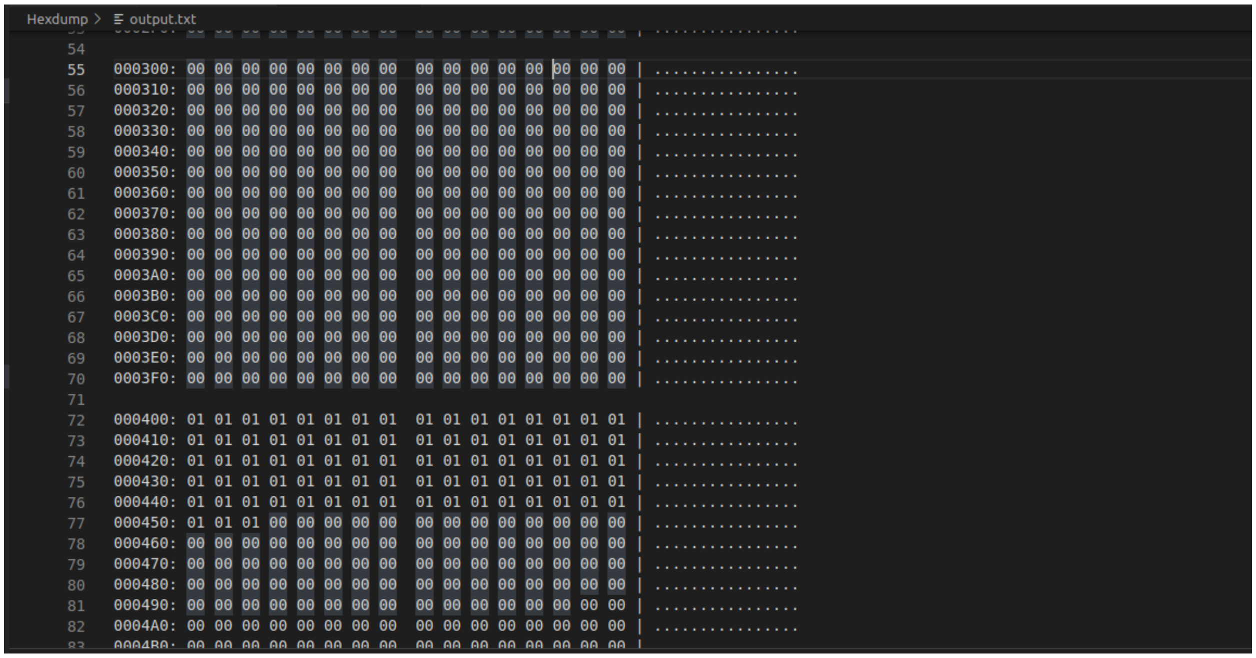
```
./hexdump ../SampleVolume
Dumping file ../SampleVolume, starting at block 0 for 19532 blocks:

000000: 43 53 43 2D 34 31 35 20  2D 20 4F 70 65 72 61 74 | CSC-415 - Operat
000010: 69 6E 67 20 53 79 73 74  65 6D 73 20 46 69 6C 65 | ing Systems File
000020: 20 53 79 73 74 65 6D 20  50 61 72 74 69 74 69 6F |  System Partitio
000030: 6E 20 48 65 61 64 65 72  0A 0A 00 00 00 00 00 00 | n Header........
000040: 42 20 74 72 65 62 6F 52  00 96 98 00 00 00 00 00 | B treboR.��.....
000050: 00 02 00 00 00 00 00 00  4B 4C 00 00 00 00 00 00 | ........KL......
000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000070: 52 6F 62 65 72 74 20 42  55 6E 74 69 74 6C 65 64 | Robert BUntitled
000080: 0A 0A 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000100: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000110: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000120: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000130: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000140: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000150: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000160: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000170: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000180: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000190: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```



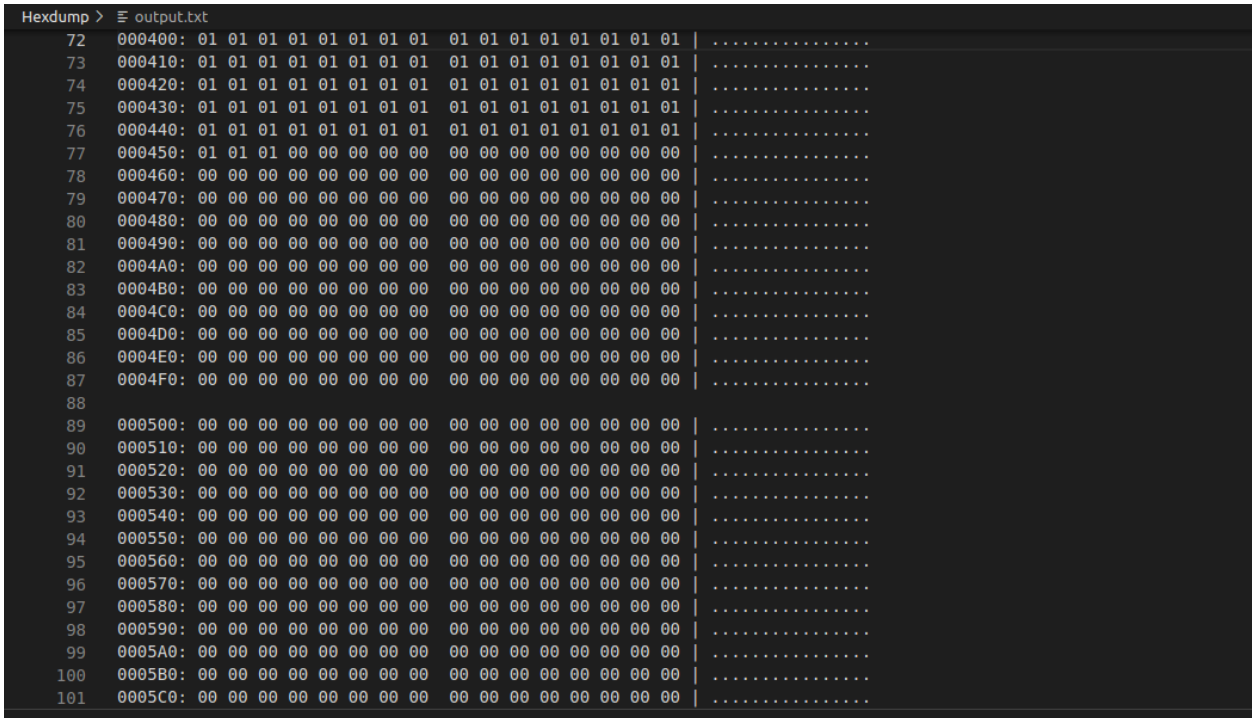
```

000100: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000110: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000120: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000130: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000140: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000150: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000160: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000170: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000180: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000190: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000200: 00 96 98 00 00 00 00 00  00 02 00 00 00 00 00 00 | .��.............
000210: 4B 4C 00 00 00 00 00 00  D5 38 01 00 00 00 00 00 | KL......�8......
000220: 01 00 00 00 00 00 00 00  24 00 00 00 00 00 00 00 | ........$.......
000230: 26 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | &...............
000240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```



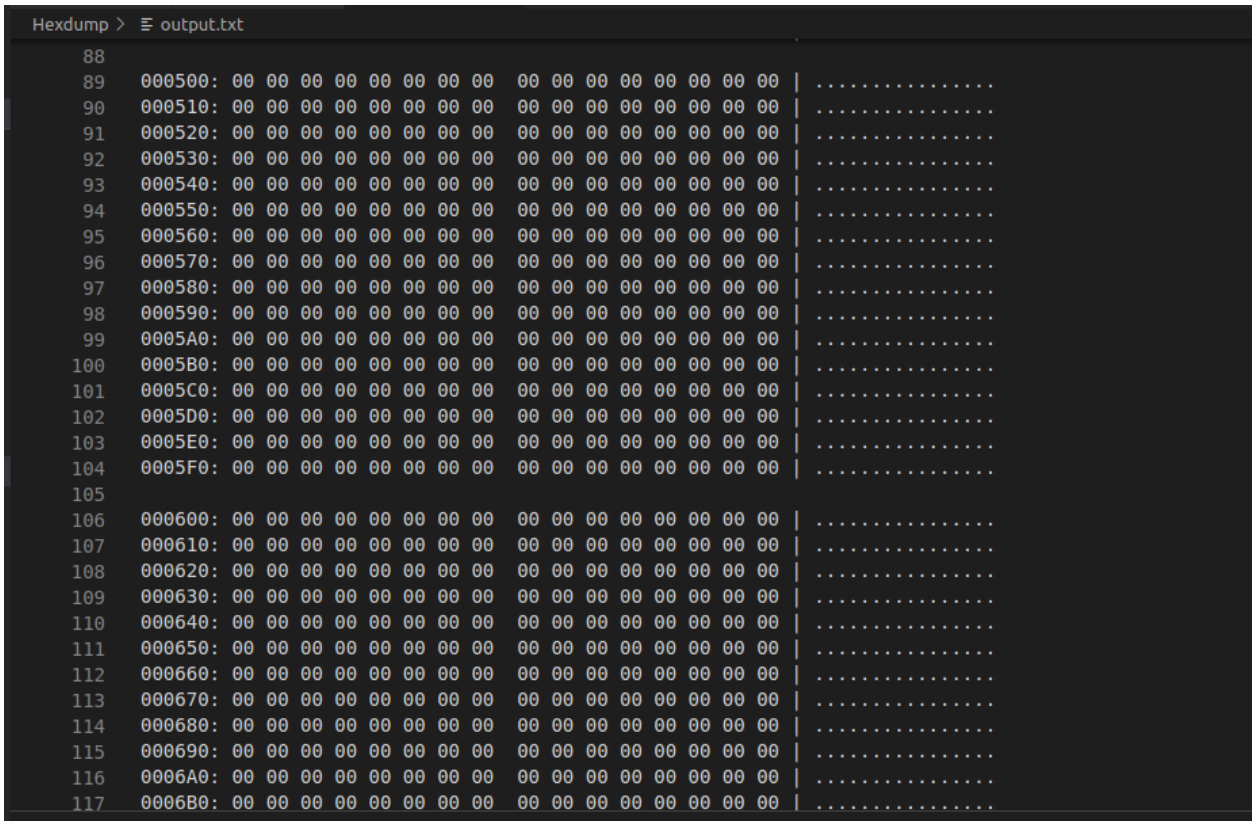
```
Hexdump > output.txt
000200: 00 96 98 00 00 00 00 00  00 02 00 00 00 00 00 00 | .��.............
000210: 4B 4C 00 00 00 00 00 00  D5 38 01 00 00 00 00 00 | KL......�8......
000220: 01 00 00 00 00 00 00 00  24 00 00 00 00 00 00 00 | ........$.......
000230: 26 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | &...............
000240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000300: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000330: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000340: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000370: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000380: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000390: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```



```
Hexdump > output.txt

000300: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000330: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000340: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000370: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000380: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000390: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000400: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000410: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000420: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000430: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000440: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000450: 01 01 01 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |
```



```
000400: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000410: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000420: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000430: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000440: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01 | ................
000450: 01 01 01 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```



```

000500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000600: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Conclusion

Overall this assignment taught us the importance of team work, organizational skills, and the importance of communicating within each other. There were a couple of parts that we struggled with throughout this assignment and we learned it by working together as a team and asking, explaining everything to one another. Our team communicated with one another via Discord and often screen shared to show each other what we implemented, how we did so, and what its purpose was in the larger picture. We each completed our own individual portions and often came together when stuck on any one portion of the larger assignment and sent each other our work for review or help. One of the problems that we faced was understanding the VCB structure, as it was the “main” part of this assignment; without the VCB structure, it is almost impossible to start on the assignment. While learning the structure implementation was a long and arduous process, since each member of a team inputted their individual knowledge/work and shared it with everyone else, we were able to successfully implement a proper VCB. Another challenging portion was working with a bitmap and understanding how the free space map works. Some of us were visual learners and had to have a clear, physical representation of how free space was formatted before even starting to code its implementation. The overall project was definitely a challenge that took lots of time, caused lots of stress, and even left some of us in total disarray as to what parts of our work should be kept and what parts should just be tossed out/completely reworked. However, through the input and hard work of each team member through either coding or explaining theoretical concepts (as well as ample communication), we were able to resolve our doubts and successfully implement a working file system!