CS3026 Assessment
Bradley Scott
51661169
b.scott.16@aberdeen.ac.uk

CGS D3 – D1

In this task, I create a structure for the virtual disk by creating the FAT and root directory.

Firstly, I initialise block 0, this is used to store the name of the disk.

```
diskblock_t block ;
direntry_t  rootDir ;

int         pos             = 0 ;
int         fatentry        = 0 ;
int         fatblocksneeded =  (MAXBLOCKS / FATENTRYCOUNT ) ;

/* prepare block 0 : fill it with '\0',
 * use strcpy() to copy some text to it for test purposes
 * write block 0 to virtual disk
 */

 for (int i =0; i < BLOCKSIZE; i++) block.data[i] = '\0';

 //add the string to the block
 strcpy(block.data, "CS3026 Operating Systems Assessment");

 //write block to virtual disk
 writeblock(&block,0);
```

I then initialize the FAT and write it to the virtual disk.

```
/* prepare FAT table
 * write FAT blocks to virtual disk
 */

 //set all positions in fat to unused
 for (pos = 0; pos < BLOCKSIZE; pos++)
    FAT[pos] = UNUSED;

 //set known fat properties

 FAT[0] = ENDOFCHAIN;
 FAT[1] = 2;
 FAT[2] = ENDOFCHAIN;
 FAT[3] = ENDOFCHAIN;

 //write fat to virtual disk
 copyFAT();
```

In this Implementation, the FAT occupies 2 blocks as each fat entry is 2 bytes in size and a block can contain 1024 bytes, therefore the chain for the fat itself has a block chain size of 2.

The copyFAT() function copies the contents of the fat into 2 blocks and writes these blocks to the virtual disk.

```
void copyFAT()

{
   diskblock_t block;
   int x;
   int y;
   int fatblocksneeded = (MAXBLOCKS / FATENTRYCOUNT);

   for (x = 0; x < fatblocksneeded; x++)
   {
      for (y = 0; y < FATENTRYCOUNT; y++)
      {
         block.fat[y] = FAT[((x*FATENTRYCOUNT)+y)];
      }
      writeblock(&block, x + 1);
   }
}
```

Here the outer for loop iterates 2 times and the inner for loop iterates 512 times as each block holds 512 fat entries, given the size of a block is 2 bytes and the maximum number of blocks is 1024.

Finally, I create a directory block for the root directory.

```
    // prepare root directory & write root directory block to virtual disk

    //clearing block data for use
     for (int i =0; i < BLOCKSIZE; i++) block.data[i] = '\0';
     //set directory entries to empty
     for(int i = 0; i < DIRENTRYCOUNT; i++) block.dir.entrylist[i].unused = TRUE;

     //directory block
     block.dir.isdir = 1;
     //first element in entry list
     block.dir.nextEntry = 0;

     //writes to position 3 where the diretory should be
     writeblock(&block,fatblocksneeded+1);

     //block number of root directory
     rootDirIndex = fatblocksneeded+1;
```

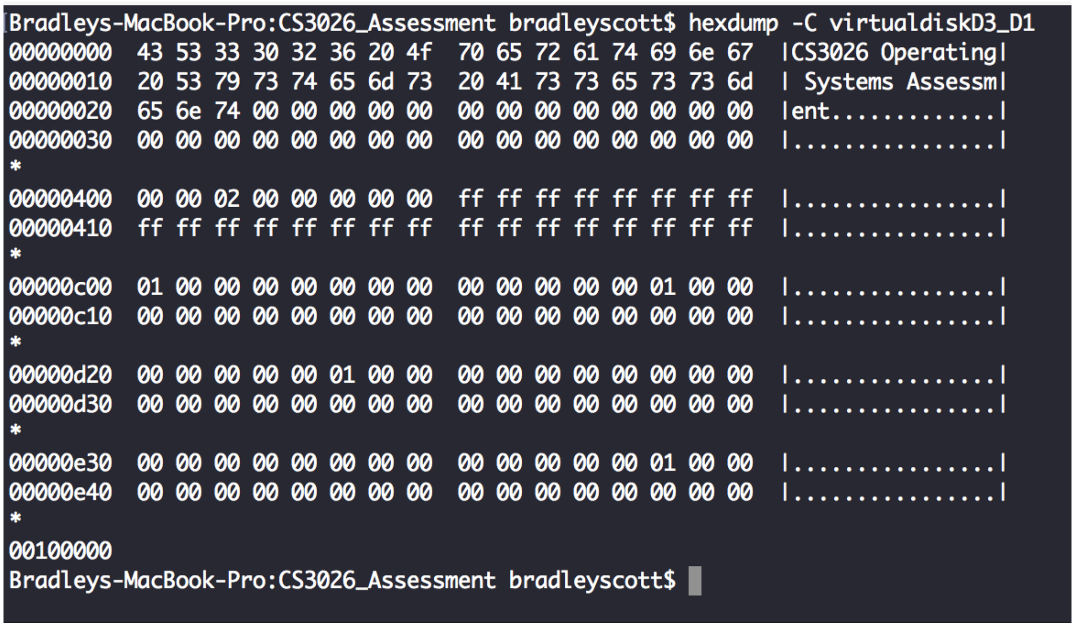
Instead of creating a new block instance, I clear the block used before. I then set all the directory entries in the entry list array of the blocks to be unused.

The directory block next entry index is set to 0 as the first element free for a directory entry as at the beginning.

The directory block is written to the position following the fat blocks, in this instance position 3. The root directory index is also set to this position.

The block chain has been initialised for the root directory FAT[3] = ENDOFCHAIN in the initialisation of the block chain itself.

A hexdump of my implementation of the d task produces the following output:



```
Bradleys-MacBook-Pro:CS3026_Assessment bradleyscott$ hexdump -C virtualdiskD3_D1
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 73 6d  | Systems Assessm|
00000020  65 6e 74 00 00 00 00 00  00 00 00 00 00 00 00 00  |ent.............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
00000410  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000d20  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00000d30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000e30  00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  |................|
00000e40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000
Bradleys-MacBook-Pro:CS3026_Assessment bradleyscott$ 
```

The first block represents block 0, numbers 43,53,33 etc represent the hex values for the text stored in that block ("CS3026 Operating Systems Assessment")

The next 02 00 is the index returned from FAT[1] which then points to 00 00 meaning ENDOFCHAIN.

The next 00 00 is the result of FAT[3] that being the root directory which is ENDOFCHAIN as block 3, the root directory is only 1 block in size.

0c00 is 1 indicating this is a directory, isdir = 1.

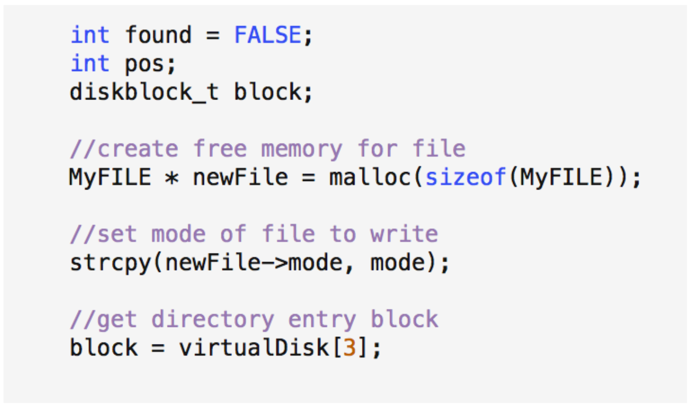
od20 is free entry in directory marked as true
0e30 is the same

each address has 16 bytes

CGS C3-C1

- create a file “testfile.txt” in your virtual disk: call myfopen (“testfile.txt”, “w”) to open this file

Firstly, I must initialise the directory block to insert a new directory entry for the file I am about to either create or open with myfopen.



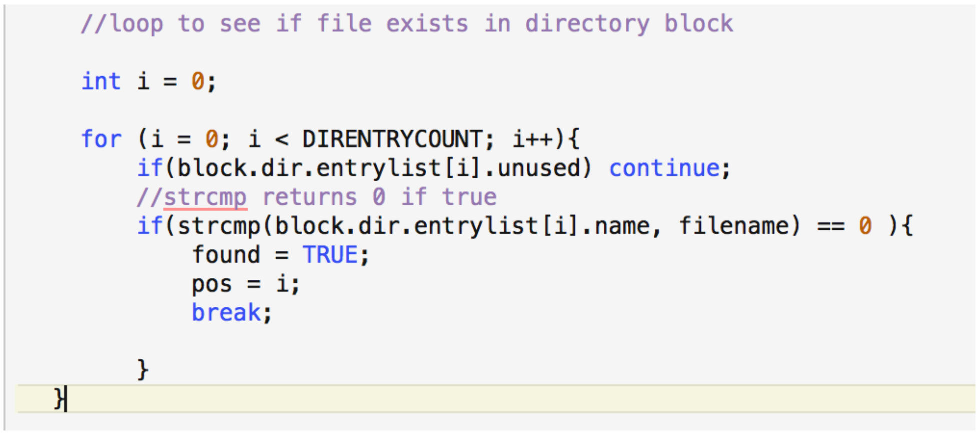
```
int found = FALSE;
int pos;
diskblock_t block;

//create free memory for file
MyFILE * newFile = malloc(sizeof(MyFILE));

//set mode of file to write
strcpy(newFile->mode, mode);

//get directory entry block
block = virtualDisk[3];
```

To do this I am creating a new file with adequate memory by using malloc.

I then copy the mode passed by the function to the file.

And set the block to be equal to the content of virtual disk position 3, which is the directory.

Next I must loop through the directory entrylist to see if the file already exists.

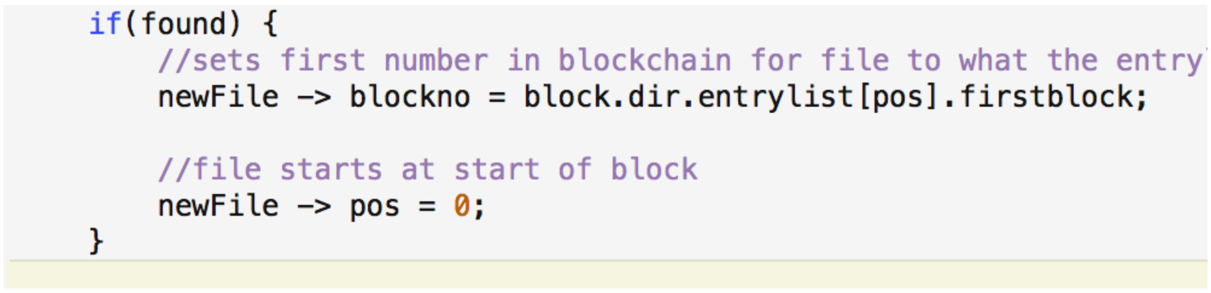


```
    //loop to see if file exists in directory block

    int i = 0;

    for (i = 0; i < DIRENTRYCOUNT; i++){
        if(block.dir.entrylist[i].unused) continue;
        //strcmp returns 0 if true
        if(strcmp(block.dir.entrylist[i].name, filename) == 0 ){
            found = TRUE;
            pos = i;
            break;

        }
}
```

This outer loop will iterate a maximum of 3 times as there can be a maximum of 3 directory listings.

If an entry is found the position in the entry list is recorded, and the look is exited.

If the file is found I must update the information I have about the file.

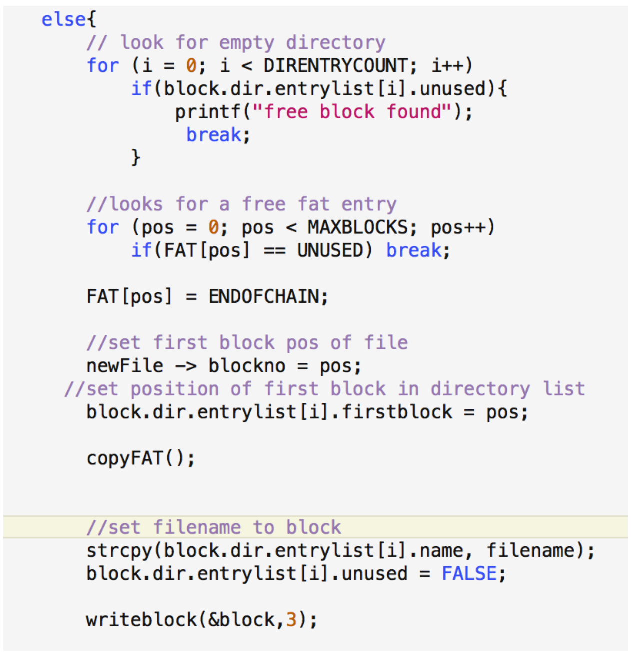


```
    if(found) {
        //sets first number in blockchain for file to what the entry
        newFile -> blockno = block.dir.entrylist[pos].firstblock;

        //file starts at start of block
        newFile -> pos = 0;
    }
```

The file attribute blockno represents the first block number of the file and is set to what the directory listing has stored as the first block of that file.

The file attribute position represents the position the file starts on the block which I set to 0 as it is the first file so should be at the start of the block.

If the file is not found I must create that file in my virtual disk.



```
   else{
       // look for empty directory
       for (i = 0; i < DIRENTRYCOUNT; i++)
           if(block.dir.entrylist[i].unused){
               printf("free block found");
                break;
           }

       //looks for a free fat entry
       for (pos = 0; pos < MAXBLOCKS; pos++)
           if(FAT[pos] == UNUSED) break;

       FAT[pos] = ENDOFCHAIN;

       //set first block pos of file
       newFile -> blockno = pos;
     //set position of first block in directory list
       block.dir.entrylist[i].firstblock = pos;

       copyFAT();


       //set filename to block
       strcpy(block.dir.entrylist[i].name, filename);
       block.dir.entrylist[i].unused = FALSE;

       writeblock(&block,3);
```

I must first look for an empty block and fat entry to insert my file.

I then set that free fat position to be ENDOFCHAIN as my file is no larger than one block.

I then set the file block number to be equal to the free fat entry block position.

I also set the position of the first block in the file in the directory entry record of that file.

CopyFAT then writes the FAT information to the disk.

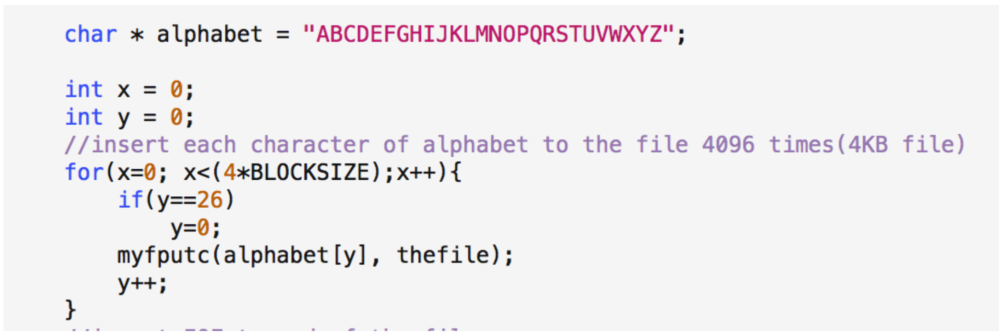
I set the name attribute on the directory entry to the name passed into the function.
And set the unused attribute to be true.

Finally, I write the directory block to the virtual disk.

The file is then returned to the myfopen call in shell.c.

- write a text of size 4kb (4096 bytes) to this file, using the function myfputc():

I must call the myfputc function 4096 times to insert my text character by character to fill a 4kb file.



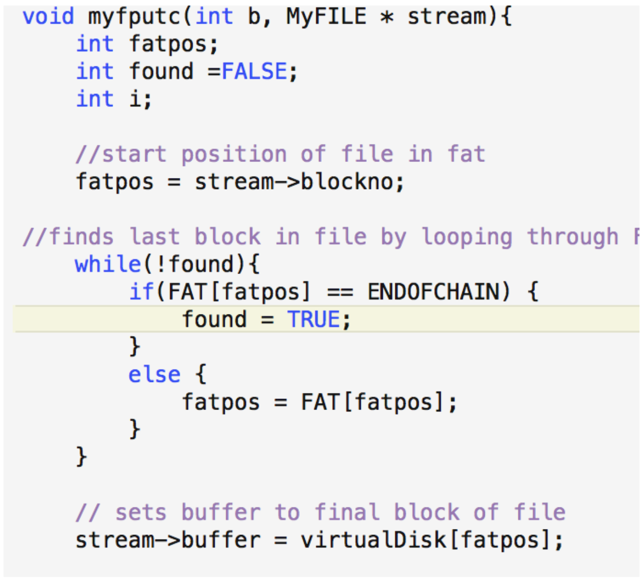
```
char * alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int x = 0;
int y = 0;
//insert each character of alphabet to the file 4096 times(4KB file)
for(x=0; x<(4*BLOCKSIZE);x++){
    if(y==26)
        y=0;
    myfputc(alphabet[y], thefile);
    y++;
}
```

My loop will iterate 4 * BLOCKSIZE, as each block is 1kb in size.

The if statement checks when to reset the position of the y variable which iterates through the alphabet.

Myfputc is called to insert a single character at a time to the file.

The myfputc function must find the correct block to insert next character into.



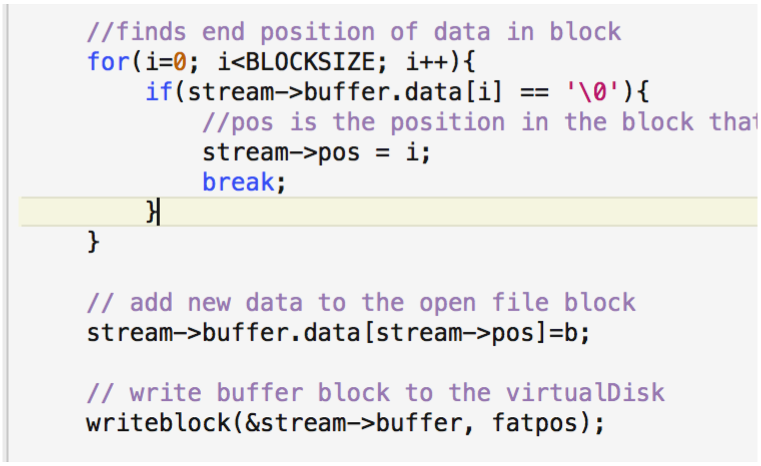
```
void myfputc(int b, MyFILE * stream){
    int fatpos;
    int found =FALSE;
    int i;

    //start position of file in fat
    fatpos = stream->blockno;

//finds last block in file by looping through F
    while(!found){
        if(FAT[fatpos] == ENDOFCHAIN) {
            found = TRUE;
        }
        else {
            fatpos = FAT[fatpos];
        }
    }

    // sets buffer to final block of file
    stream->buffer = virtualDisk[fatpos];
```

fatpos is set to the first block position of the file in the fat.

I then loop to find the last fat entry in the chain, which is marked with ENDOFCHAIN.

Finally, I set the buffer to equal the last block of the file as this is where the character will be inserted.

Myfputc must then find the first free position in the block to insert the character.



```
    //finds end position of data in block
    for(i=0; i<BLOCKSIZE; i++){
        if(stream->buffer.data[i] == '\0'){
            //pos is the position in the block that
            stream->pos = i;
            break;
        }
    }

    // add new data to the open file block
    stream->buffer.data[stream->pos]=b;

    // write buffer block to the virtualDisk
    writeblock(&stream->buffer, fatpos);
```

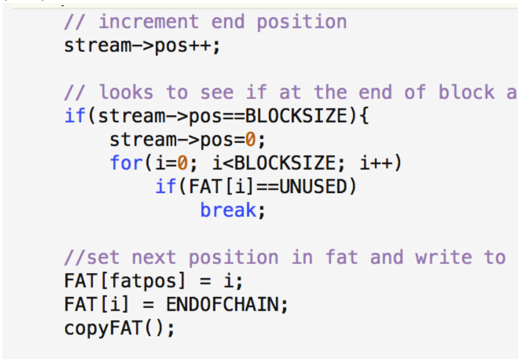
I look for a maximum amount of 1024 times(BLOCKSIZE) to find the first free position in the block, which is marked with '\0'.

I then set the position attribute to the file to equal that position.

I then add the character passed to the function to that position in buffer block of the file.

Finally, I write the buffer block to the virtual disk.

Myfputc must finally look to see if there is any more space in the block and write the FAT to the virtual disk to prepare for more inserts into the file.



```
    // increment end position
    stream->pos++;

    // looks to see if at the end of block a
    if(stream->pos==BLOCKSIZE){
        stream->pos=0;
        for(i=0; i<BLOCKSIZE; i++)
            if(FAT[i]==UNUSED)
                break;

    //set next position in fat and write to
    FAT[fatpos] = i;
    FAT[i] = ENDOFCHAIN;
    copyFAT();
```

The end position of the file is incremented by 1.

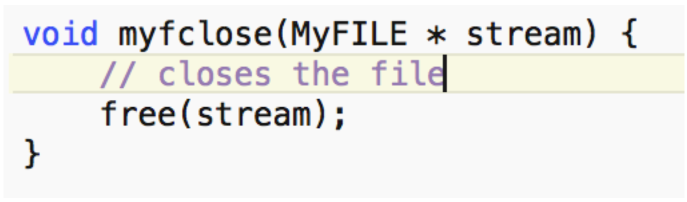
I then check to see if the position is equal to the block size. If it is that means that we are at the end of the file. If the position is equal to the blocksize then I look to find the next available fat entry to insert a new block into as we will have run out of space in the current block.

The fat position is set to the next free block in the chain.

The fat is then written to the virtual disk.

- close the file with myfclose()

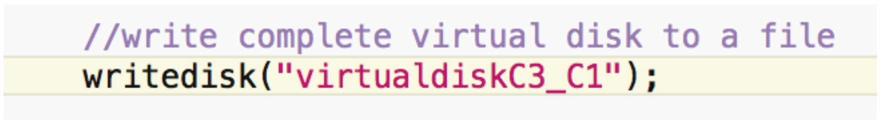
Since I am implementing my myfputc to write character by character all I had to do for this step was free the memory used by the file. This can be seen below:



```
void myfclose(MyFILE * stream) {
    // closes the file
    free(stream);
}
```

- write the complete virtual disk to a file "virtualdiskC3_C1"

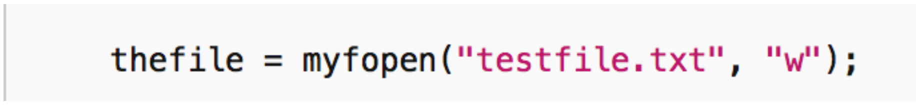
This step was performed with the included writedisk function called in my shell.c. As seen below:



```
    //write complete virtual disk to a file
    writedisk("virtualdiskC3_C1");
```

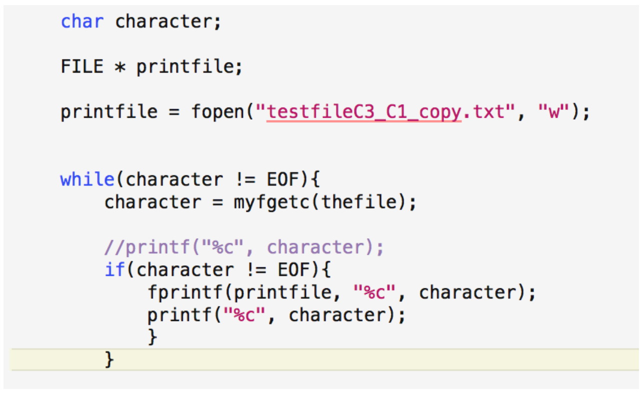
- test myfgetc():

Firstly, I open the file again in my shell.c.



```
    thefile = myfopen("testfile.txt", "w");
```

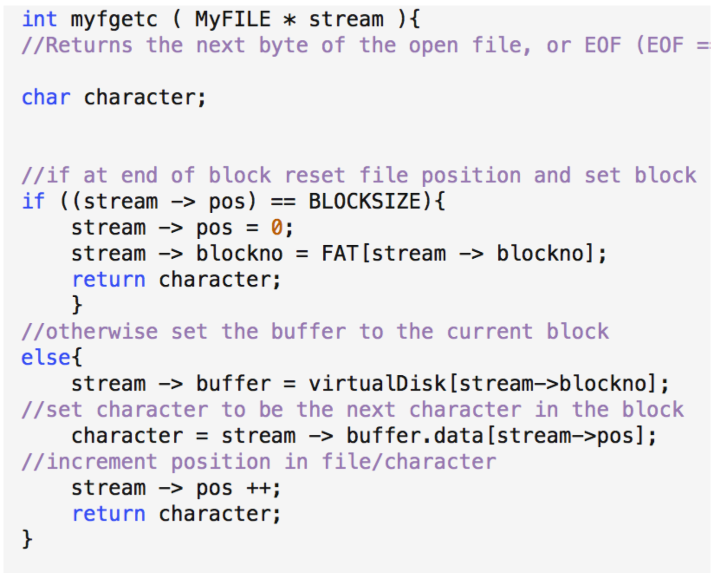
I then read out the contents of the file character by character using myfgetc and write that contents to a real file on my real hard disk called "testfileC3_C1_copy.txt".



```
    char character;

    FILE * printfile;

    printfile = fopen("testfileC3_C1_copy.txt", "w");


    while(character != EOF){
        character = myfgetc(thefile);

        //printf("%c", character);
        if(character != EOF){
            fprintf(printfile, "%c", character);
            printf("%c", character);
            }
        }
```

Here I check to see if the character returned is equal to EOF(end of file) if it isn't I print the character to the screen and write that character to the file .

Mygetc returns the next byte of the open file.

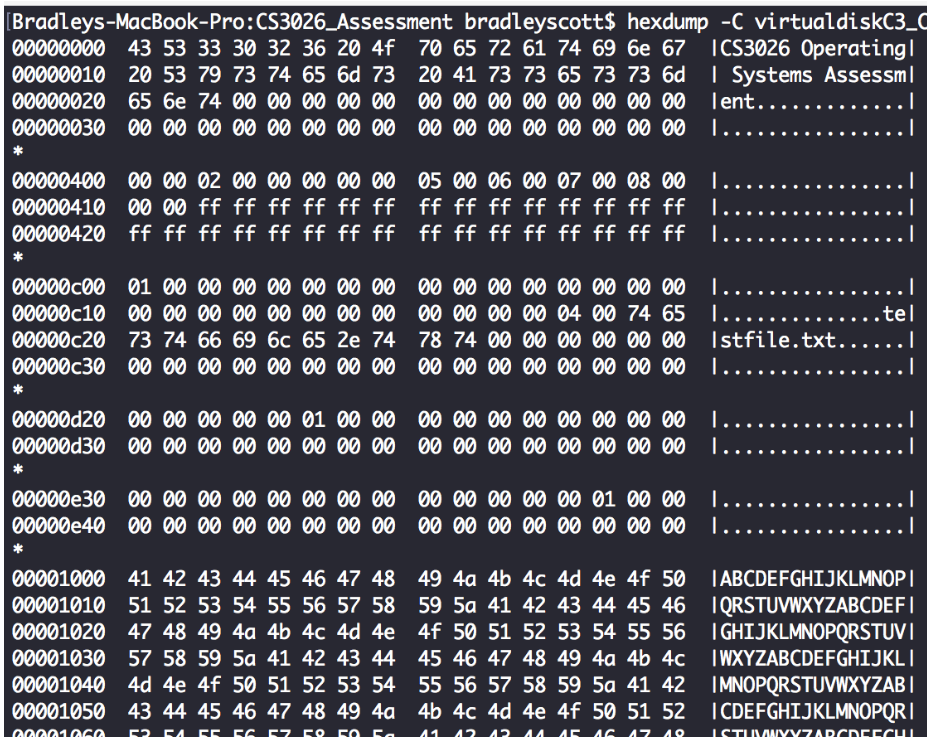


```
int myfgetc ( MyFILE * stream ){
//Returns the next byte of the open file, or EOF (EOF =

char character;


//if at end of block reset file position and set block
if ((stream -> pos) == BLOCKSIZE){
    stream -> pos = 0;
    stream -> blockno = FAT[stream -> blockno];
    return character;
    }
//otherwise set the buffer to the current block
else{
    stream -> buffer = virtualDisk[stream->blockno];
//set character to be the next character in the block
    character = stream -> buffer.data[stream->pos];
//increment position in file/character
    stream -> pos ++;
    return character;
}
```

If the position attribute of the file is equal to blocksize, meaning we are at the end the current block.
I set the position to 0 and the blocknumber to be equal to the next blockchain entry in the fat, before returning the final character of the block.

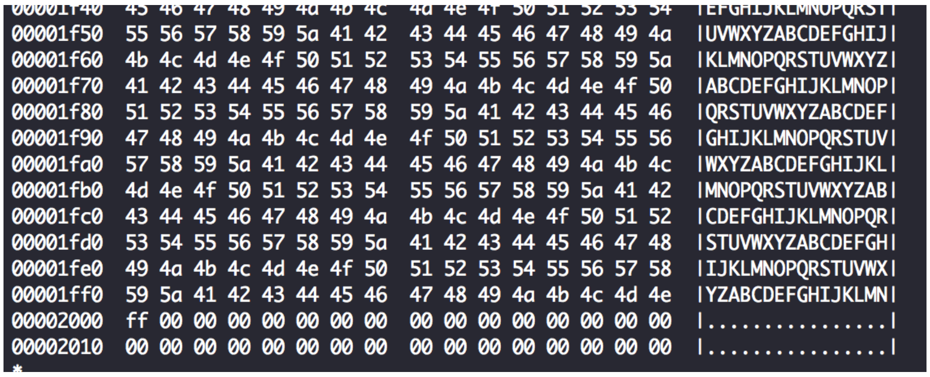
Otherwise I set the buffer to be equal to the current block. I also set the character that must be returned to equal the next character in the block, next being the next one that has not yet been returned. Finally, I increment the position in the block by 1 so that the next time this function Is called the next character will be returned and return the character.

A hexump of my implementation of the C task produces the following output:



```
Bradleys-MacBook-Pro:CS3026_Assessment bradleyscott$ hexdump -C virtualdiskC3_C
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 73 6d  | Systems Assessm|
00000020  65 6e 74 00 00 00 00 00  00 00 00 00 00 00 00 00  |ent.............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  05 00 06 00 07 00 08 00  |................|
00000410  00 00 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
00000420  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 04 00 74 65  |..............te|
00000c20  73 74 66 69 6c 65 2e 74  78 74 00 00 00 00 00 00  |stfile.txt......|
00000c30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000d20  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00000d30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000e30  00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  |................|
00000e40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
00001010  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
00001020  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001030  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001040  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001050  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001060  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
```

Continuing to the final block, block 8.



```
00001f40  45 46 47 48 49 4a 4b 4c  4d 4e 4f 50 51 52 53 54  |EFGHIJKLMNOPQRST|
00001f50  55 56 57 58 59 5a 41 42  43 44 45 46 47 48 49 4a  |UVWXYZABCDEFGHIJ|
00001f60  4b 4c 4d 4e 4f 50 51 52  53 54 55 56 57 58 59 5a  |KLMNOPQRSTUVWXYZ|
00001f70  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
00001f80  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
00001f90  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001fa0  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001fb0  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001fc0  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001fd0  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001fe0  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
00001ff0  59 5a 41 42 43 44 45 46  47 48 49 4a 4b 4c 4d 4e  |YZABCDEFGHIJKLMN|
00002000  ff 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Here much of the same structure is visible that was included in my D task implementation.
There are differences that reflect the changes made in my C task.

The fat entries have been updated to reflect the 4 inserted blocks that hold the text information. Block 7 points to block 8 which is the end of file.

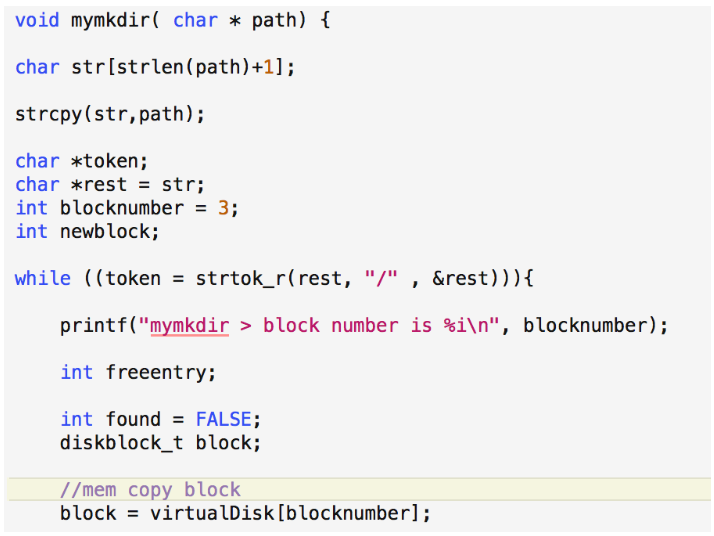
The data shown reflects the hexadecimal values of the letters stored in the memory addresses of the blocks.

CGS B3-B1

- mymkdir(char * path) that creates a new directory

I have successfully implemented the function my mkdir that creates a new directory given the path.

Firstly, I must tokenize the path to search the current directory for that directory name.

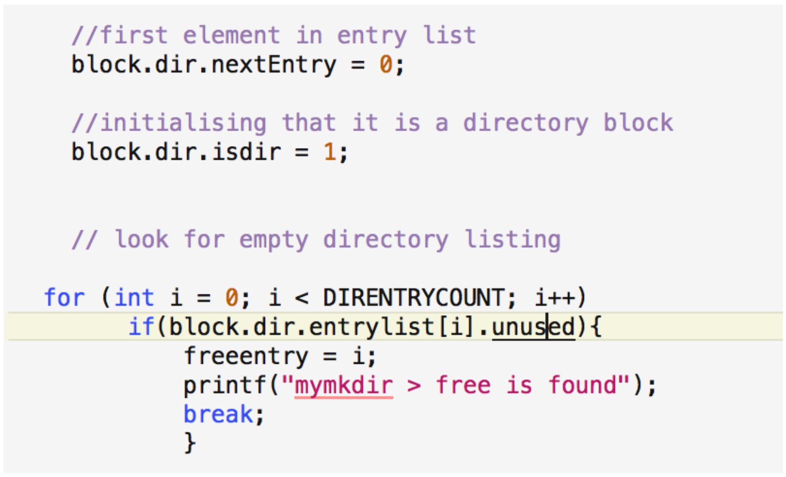


```
void mymkdir( char * path) {

char str[strlen(path)+1];

strcpy(str,path);

char *token;
char *rest = str;
int blocknumber = 3;
int newblock;

while ((token = strtok_r(rest, "/" , &rest))){

    printf("mymkdir > block number is %i\n", blocknumber);

    int freeentry;

    int found = FALSE;
    diskblock_t block;

    //mem copy block
    block = virtualDisk[blocknumber];
```

I tokenize the path using a function called strtok_r which is part of the c standard library. It splits the path given at the "/" character.

I also set block number to be 3 outside the loop at that is the block number of the root directory, where the creation of the directory will begin.

The block will set equal to the current working directory as the tokenize loop iterates.



```
    //first element in entry list
    block.dir.nextEntry = 0;

    //initialising that it is a directory block
    block.dir.isdir = 1;


    // look for empty directory listing

  for (int i = 0; i < DIRENTRYCOUNT; i++)
        if(block.dir.entrylist[i].unused){
            freeentry = i;
            printf("mymkdir > free is found");
            break;
            }
```

Next I set the block nextDirectory property to be equal to 0. And the isdir property to be true so that it is visible in the hexdump.

I then loop through the directory listing until I find a free space in that directory and set the freeentry variable to that positon.

```
printf(" \n mymkdir > free dir entry found at pos %i \n", freeentry);
    //set filename to block
strcpy(block.dir.entrylist[freeentry].name, token);
    //set block to be used
block.dir.entrylist[freeentry].unused = FALSE;

    //looks for a free fat entry
for (int pos = 0; pos < MAXBLOCKS; pos++)
    if(FAT[pos] == UNUSED) {
        newblock = pos;
        break;
        }
```

Once I find a free directory space I then insert the name of the directory I wish to create into that space and set its position to be used.

Once a space in the current directory to list my new directory I must find a free block to insert the directory into. To do this I loop over the fat until a position marked as UNUSED is found.

```
printf(" \n mymkdir > %s is going to fat pos %i \n",token, newblock);

FAT[newblock] = ENDOFCHAIN;


     //set position of first block in directory list
block.dir.entrylist[freeentry].firstblock = newblock;

copyFAT();

writeblock(&block,blocknumber);

blocknumber = newblock;
```

This unused position is then inserted into the fat.

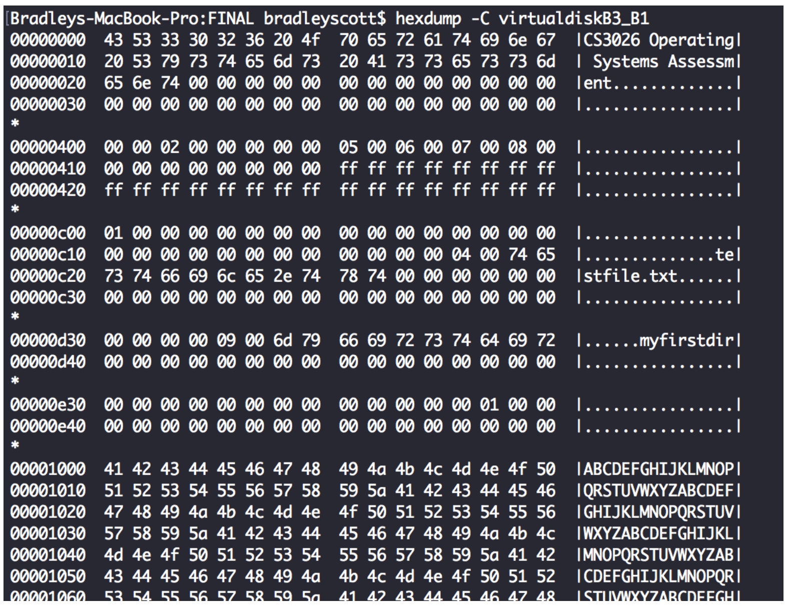
The entry list entry for the new sub directory is updated to point to the newly found space for the sub directory.

The Fat and blocks are written to the disk.

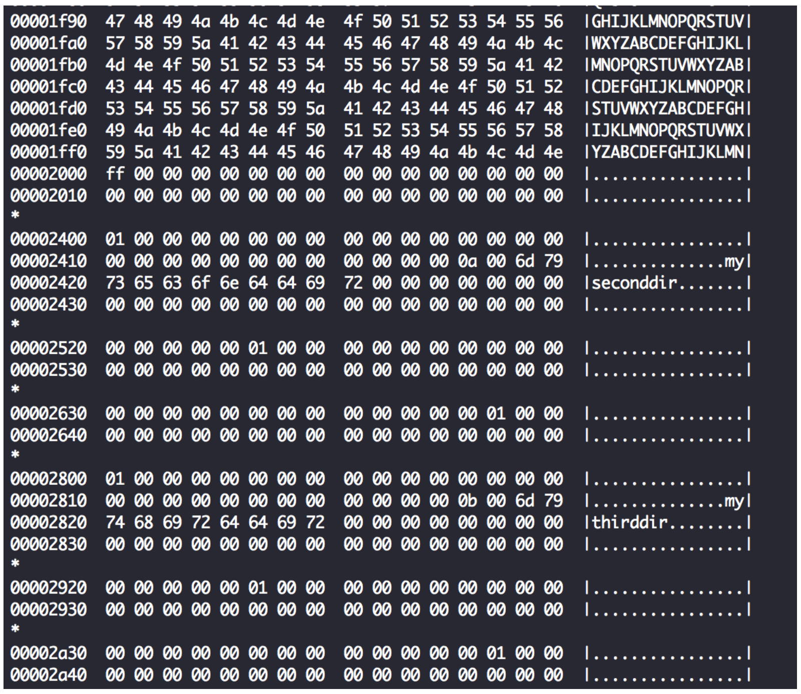
Finally, the blocknumber is set to the new-found block. This is the new current working directory.

The above will iterate until all tokens/sub directories have been processed. When the token = NULL.

A hexdump demonstrating my implementation of the mymkdir function can be seen below:



```
Bradleys-MacBook-Pro:FINAL bradleyscott$ hexdump -C virtualdiskB3_B1
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 73 6d  | Systems Assessm|
00000020  65 6e 74 00 00 00 00 00  00 00 00 00 00 00 00 00  |ent.............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  05 00 06 00 07 00 08 00  |................|
00000410  00 00 00 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
00000420  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 04 00 74 65  |..............te|
00000c20  73 74 66 69 6c 65 2e 74  78 74 00 00 00 00 00 00  |stfile.txt......|
00000c30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000d30  00 00 00 00 09 00 6d 79  66 69 72 73 74 64 69 72  |......myfirstdir|
00000d40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000e30  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00000e40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
00001010  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
00001020  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001030  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001040  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001050  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001060  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
```



```
00001f90  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001fa0  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001fb0  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001fc0  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001fd0  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001fe0  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
00001ff0  59 5a 41 42 43 44 45 46  47 48 49 4a 4b 4c 4d 4e  |YZABCDEFGHIJKLMN|
00002000  ff 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002400  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002410  00 00 00 00 00 00 00 00  00 00 00 00 0a 00 6d 79  |..............my|
00002420  73 65 63 6f 6e 64 64 69  72 00 00 00 00 00 00 00  |seconddir.......|
00002430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002520  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00002530  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002630  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00002640  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002800  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002810  00 00 00 00 00 00 00 00  00 00 00 00 0b 00 6d 79  |..............my|
00002820  74 68 69 72 64 64 69 72  00 00 00 00 00 00 00 00  |thirddir........|
00002830  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002920  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00002930  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002a30  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00002a40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

The new directories were correctly inserted into the appropriate free blocks.
First directory was inserted into the root directory at entrylist positon one and its content stored in block 9.
Second directory was inserted into block 10 and third directory block 10.
The is.dir flags being true are also clearly visible.

Conclusion

Given more time I would refactor my code to remove any repetitions in statements such as looking for a free block, which is used throughout the program. I would also make the system more flexible overall by allowing for directories to index more than three items.

Running Instructions

I have created a makefile using the c99 standard.

To run my assessment, take the following steps:

1. Open a new terminal window
2. Change into the directory of my assessment
3. Type make (to complie)
4. Type ./shell (to run the program)

For hexdumps use the following commands:

For D task:

```
hexdump -C virtualdiskD3_D1
```

For C task:

```
hexdump -C virtualdiskC3_C1
```

For B task:

```
hexdump -C virtualdiskB3_B1
```