CS3026 Assessment 02 – Virtual Disk

Run Code

In each folder there includes the same Makefile.

Make commands:

- Make clean – deleted compiled files in the Directory
- Make – compiles program inside Directory
- Make run – runs the program in the Directory

CGS D3-D1

Shell.c

```
int main()
{
    // test format
    format();
    // call print block
    printBlock(0);
    writedisk("virtualdiskD3_D1");
    return 0 ;
}
```

Calls format printBlock 0 then writes to disk

Format

- The write block preexisting function to write the created blocks to virtual disk
- The purpose of format was to create a structure for the virtual disk.
- It does this by Creating block 0, The FAT and the Root Dir block in the virtual disk.

```
diskblock_t block ; // block 0
    for(int i =0; i < BLOCKSIZE; ++i)
    {
        block.data[i] = '\0';
    }
    strcpy((char*)block.data, "CS3026 Operating Systems Assessment 
2023");
    writeblock(&block,0);
```

- FAT table consists of two blocks of size 512 each because each entry is 2 bytes of space
- The FAT table value for block 0 , The FAT table itself (block 1 and 2) and root block , block 3
- All FAT tables are set to UNUSED before adding default blocks in
- The FAT table blocks point to each other the

```
diskblock_t block_1;
    diskblock_t block_2;
    // all FAT entries are UNUSED
    for(int i = 0; i < BLOCKSIZE; ++i)
    {
       FAT[i] = UNUSED; // 1024 entries set to UNUSED
    }
    FAT[0] = ENDOFCHAIN; // block 0
    FAT[1] = 2; // fat block 1
    FAT[2] = ENDOFCHAIN; // fat block 2
    FAT[3] = ENDOFCHAIN; // root
    // 4-1023 entries == UNUSED
    for(int i=0;i<FATENTRYCOUNT; ++i)
    {
       block_1.fat[i] = FAT[i]; // fatblock 0 -> 512 stores FAT 0 -> 512 entries
    }
    for (int i = FATENTRYCOUNT; i < BLOCKSIZE; ++i)
    {
       block_2.fat[i-512] = FAT[i]; //fatblock 0 -> 512 stores FAT 512 -> 1023 entries
    }
    // write fat to disk
    writeblock(&block_1,1);
    writeblock(&block_2,2);
```

- Root block data initially set to '\0'
- Isdir to identify directory as such
- Nextentry set to 0 as default

```
  diskblock_t root_Block;
    // fill with \0
    for(int i=0;i<BLOCKSIZE;++i)
    {
       root_Block.data[i] = '\0';
    }
    //is a directory
    root_Block.dir.isdir = TRUE;
    root_Block.dir.nextEntry = 0; // starts at 0
    rootDirIndex = 3;
    // write root to block
    writeblock(&root_Block, 3);
```

hexdump -C virtualdiskD3_D1

CGS C3-C1

- Added helper functions to re use code easily

```
//added functions
int findUNUSEDfatentry () ;
```

```
void addfatentry ( int blokno) ;
void addtofatentry (int blokno , int newblokno) ;
int findfilebyname (dirblock_t * current, const char * filename);
```

- findUNUSEDfatentry gets a fat entry that is UNUSED in the virtualdisk and returns the index. If not found returns EOC.

```
// find an UNUSED fat entry in FAT
int findUNUSEDfatentry ()
{
   // 4 - 1023
   for (int i=4;i<BLOCKSIZE;++i)
   {
      // finds fat entry set to UNUSED
      if (FAT[i] == UNUSED){
         return i; //return index of fat block
      }
   }
   //return EOC if fat not found
   return ENDOFCHAIN;
}
```

- addfatentry adds the fat entry (blokno) to FAT table and equals it to ENDOFCHAIN

```
/*add fat entry to block_1 or block_2 of fat table
   */
void addfatentry (int blokno)
{

   //check if blokno size fits in block 1 or block 2
   if (blokno > 1024)
   {
      printf("block no is outside range of fat table\n"); // blokno outside
FAT table range
   }
   // from 4 to 511 add to block 1
   if ( blokno < 511)
   {
      FAT[blokno] = ENDOFCHAIN;
      virtualDisk[1].fat[blokno] = ENDOFCHAIN;

   }
   else
   {
      // or from 512 - 1024 add to block 2
      FAT[blokno] = ENDOFCHAIN;
      virtualDisk[2].fat[blokno] = ENDOFCHAIN;
   }
}
```

- addtofatentry adds fat entry(newblokno) to the end of the previous chain (blokno) in FAT table

```
/*adds a fat entry to existing FAT chain
   */
void addtofatentry (int blokno, int newblokno)
{

   // checks within range
   if (blokno > 1024)
   {
      printf("block no is outside range of fat table\n"); // blokno outside
FAT table range
   }
   //checks less than 512 add block 1
   if (blokno < 511)
   {
      FAT[blokno] = newblokno;
      virtualDisk[1].fat[blokno] = newblokno;
   }
   else
   {
      // if greater than 512 add to block 2
      FAT[blokno] = newblokno;
      virtualDisk[2].fat[blokno] = newblokno;
   }
}
```

- findfilebyname (finds the name of a file inside of the dirblock parameter current) and returns the index . if not found returns EOF

```
int findfilebyname (dirblock_t * current ,const char* filename)
{
   for (int i=0; i < DIRENTRYCOUNT; ++i)
   {
      //checks if filename exists then returns the index of the file
      if (strcmp(current->entrylist[i].name, filename) == 0)
      {
         //printf("file found at index %d\n", i);
         return i;
      }
   }
   // could not find name
   return EOF;
}
```

- findUNUSEDdirentry finds dir entry that has unused set to TRUE meaning it not in use currently and returns the index . if not found return EOF

```
•   /* find an unused in the specific dir
•      */
•   int findUNUSEDdirentry (dirblock_t *dir)
•   {
•      for (int i=0;i<DIRENTRYCOUNT;++i)
•      {
•         if (dir->entrylist[i].unused == TRUE && dir-
    >entrylist[i].entrylength == 0)
•         {
•            //printf("file found at index %d\n", i);
•            return i;
•         }
•      }
•      return EOF;
•   }
```

```
// check mode is in write or read
   if (strcmp("w", mode ) !=0  &&  strcmp("r", mode) != 0)
   {
      printf("file not opened in the appropriate mode\n");
      return NULL; // return nothing
   }
```

Function checks if mode is set to w or r meaning read or write mode

- If set to either mode the function checks if the file already exists and if it does returns the file

```
// if mode set to wtite
   if (strcmp("w", mode) == 0)
   {
      // get dir entry from root dir
      int dirIndex = findfilebyname(root, filename);

      //check file can be found in disk
      if (dirIndex == EOF)
      {
         // if entry name not found create start creating the file
         printf("Creating File...\n");
      }

      else // get exsiting file if name is found
      {

         file_ptr->blockno =
virtualDisk[rootDirIndex].dir.entrylist[dirIndex].firstblock;
         file_ptr->buffer = virtualDisk[file_ptr->blockno];
         //file_ptr->buffer.dir.entrylist[0] =
virtualDisk[rootDirIndex].dir.entrylist[dirIndex];
         return file_ptr;
      }
```

This not the direct structure of myfopen

```
// if opened in read mode
   if (strcmp(mode, "r") == 0)
   {
      // get the dir entry of the file
      int dirIndex = findfilebyname(root,filename);
      //check file name can be found in disk
      if (dirIndex == EOF)
      {
         printf("FileNotFoundError!\n");
         return NULL;
      }
      // get exsiting file
      file_ptr->blockno = 
virtualDisk[rootDirIndex].dir.entrylist[dirIndex].firstblock;
      file_ptr->buffer = virtualDisk[file_ptr->blockno];
      file_ptr->pos = 0;
      // mode message
      printf("File opened in '%c' mode\n",*file_ptr->mode);
      return file_ptr;
   }
```

- If set to read mode and file doesn't already existing then it will return a pointer allocated memory but equals nothing so (NULL) which can checking in shell.c

```
if (ptr_file == NULL)
    {
      printf("FILE NOT OPENED");
      return 0;
    }
```

- However if the file does not already exist it will the be create afterward if mode set to write

```
// set the blockNo
      int UNUSED_fatentry = findUNUSEDfatentry(); // find a block number set 
to UNUSED

      //set blokno to founs index
      file_ptr->blockno = UNUSED_fatentry; // is still FAT[file_ptr->blockno] 
= UNUSED
      // set position
      file_ptr->pos = 0;


      //get unused directory in root
      dirIndex = findUNUSEDdirentry(root);
      if (dirIndex == EOF)
      {
         printf("All entry used up in directory");
         return NULL;
      }
      // add unused dir entry to root
      virtualDisk[rootDirIndex].dir.entrylist[dirIndex].entrylength = 0;//
entry length is 0 currently
      virtualDisk[rootDirIndex].dir.entrylist[dirIndex].filelength = 0;//
nothing in file so length 0
      virtualDisk[rootDirIndex].dir.entrylist[dirIndex].isdir = FALSE ;// is a
file
      virtualDisk[rootDirIndex].dir.entrylist[dirIndex].unused = FALSE;// set
to used
      virtualDisk[rootDirIndex].dir.entrylist[dirIndex].firstblock = file_ptr-
>blockno;// set firstblock
      //virtualDisk[rootDirIndex].dir.entrylist[dirIndex] = file_ptr-
>buffer.dir.entrylist[0]; // set root dir nextEntry to entry above
      strncpy(virtualDisk[rootDirIndex].dir.entrylist[dirIndex].name,
filename, MAXNAME); // set name

      virtualDisk[rootDirIndex].dir.nextEntry++;

      addfatentry(file_ptr->blockno);// add to fat block and FAT table
```

- Using helper functions to get unused dir entry inside root and fat entry, to add the FAT table and create entry in the root dir.
- Myfputc

```
/ check if file mode is set to write mode
   if ( strcmp(stream-> mode, "w") != 0)
   {
      //output error if not in "w" mode
      printf("MyFILE mode not set to 'w' mode \n");
   }
   else
```

- will not write to buffer data un less mode is set to write

```
// checks if the pos is >= to 1023
   if (stream->pos == BLOCKSIZE - 1 )
   {
      printf("buffer is full\n");
      // write buffer to if buffer is full to current block number location
      writeblock(current, stream->blockno);
```

- if buffer full write the buffer to the virtual disk then create new buffer to write on

```
// get UNUSED fat entry
      int newfatentry = findUNUSEDfatentry();

      // checks fat entry  does not equal EOC
```

```c
      if (newfatentry == ENDOFCHAIN)
      {
         // if returned EOC ouput error
         printf("There are no more fat entries left\n");
      }

      //add new entry to the fat block
      addfatentry(newfatentry);// new fat entry = EOC

      // add new entry to the end of firstblock
      addtofatentry(stream->blockno, newfatentry);
```

- then set block and file blockno so it write in correct position

```c
stream->blockno = newfatentry;// set new blokno to new fat entry found
      stream->pos = 0; //reset position to 0
      memset(stream->buffer.data, 0, BLOCKSIZE); // reset memory location to 0
```

- if not full write parameter (int b) to buffer data

```c
// add the data b to the buffer data at the current pos of stream(file)
   current->data[stream->pos] = (Byte) b;
   stream->pos++;//increase pos
```

- mfgetc

```c
// Check if file mode is set to read mode
   if (strcmp(stream->mode, "r") != 0) {
      // Output error if not in "r" mode
      printf("MyFILE mode not set to 'r' mode\n");
      return EOF;
   }
   else
```

- Will not run unless in read mode

```c
if (stream->pos == BLOCKSIZE - 1)
   {
      //print the current block to terminal
      printBlock(stream->blockno);
      // if eoc is reached then return eof
      if (FAT[stream->blockno] == ENDOFCHAIN)
      {
         return EOF;
      }
      // traverse block chain
      stream->blockno = FAT[stream->blockno];
      stream->pos = 0; // reset pos
```

- If the end of the buffer is reached print the block then traverse to the next block in the FAT chain and set the new blockno and pos

```c
int character;
   //stream->buffer = virtualDisk[stream->blockno]; // get buffer of current
block
```

```
   character = stream->buffer.data[stream->pos]; // get each character of the
buffer
   stream->pos++; // pos++
```

- While end not reach then set character equal to file pos in the buffer data the return character
- Myfclose

```
/*  myfclose function
 */
void myfclose ( MyFILE * stream )
{
   writeblock(&stream->buffer, stream->blockno);
   free(stream);
   printf("file is now closed\n");
}
```

- Write the filebuffer block to disk then frees the file pointer indicating it is now closed

```
/ test myfgetc
    int character = myfgetc(ptr_file);
    FILE * realfile = fopen("testfileC3_C1_copy.txt", "w"); // open file to
copy content
    while (character != EOF)
    {
      fprintf(realfile,"%c",character); // copy content
      character = myfgetc(ptr_file);
    }
    fclose(realfile);
    // close again
    myfclose(ptr_file);
```

- Content of the myfgetc is also copied to real file in the directory
- Using make run > tracefileC3_C1.txt the output in the terminal is redirect to this text file
- Hexdump -C virtualdiskC3_C1

<u>CGS B3-B1</u>

- For this section Add a directory hierarchy to your virtualdisk that allows the creation of subdirectories
- create a directory “/myfirstdir/myseconddir/mythirddir” in the virtual disk
  - call mylistdir(“/myfirstdir/myseconddir”): print out the list of strings returned by this function
  - write out virtual disk to “virtualdiskB3_B1_a”
  - create a file “/myfirstdir/myseconddir/testfile.txt” in the virtual disk
  - call mylistdir(“/myfirstdir/myseconddir”): print out the list of strings returned by this function
  - write out virtual disk to “virtualdiskB3_B1_b

```
// declare pointer to pointers
    char ** listdirs;
    //test mymkdir
    mymkdir("/myfirstdir/myseconddir/mythirddir");
    // test mylistdir
    listdirs = mylistdir("/myfirstdir/myseconddir");
    //prints all dir in path
    printf("Path contents ... \n");
    for (int i = 0; i < DIRENTRYCOUNT; i++)
    {
        if(strcmp(listdirs[i], "\0") != 0)
        {
            printf("%s\n", listdirs[i]);
        }
    }

    writedisk("virtualdiskB3_B1_a");

    // test myfopen in path
    MyFILE * ptr_file = myfopen("/myfirstdir/myseconddir/testfile1.txt", "w");
    //test mylistdir
    listdirs = mylistdir("/myfirstdir/myseconddir");
    printf("Path contents ... \n");
    //prints all dir in path
    for (int i = 0; i < DIRENTRYCOUNT; i++)
    {
        if(strcmp(listdirs[i], "\0") != 0)
        {
            printf("%s\n", listdirs[i]);
        }
    }
    myfclose(ptr_file);
    writedisk("virtualdiskB3_B1_b");
```

- For create a path using mymkdir

```
•   char *token, *rest; // tokenize path and save pointer
•       char *pathCopy = strdup(path); // copy path
•       diskblock_t *currentParent = &virtualDisk[rootDirIndex]; // root
    original parent direcotry
•       currentDirIndex = rootDirIndex;//get root block index
•       token = strtok_r(pathCopy, "/", &rest); // tokenize path
```

- This part token will contain tokenize strings of the path defined with strotk_r() split by '/' delimiter to search for the directory name individually
- pathCopy copies the path string
- rest is the save pointer used in the strtok_r() function
- currentParent diskblock is equal to the root block at the start of the function
- If token can not find a string after '/' it will equal NULL because strtok_r() will have returned NULL

```
•   while (token != NULL)
•       {
•           // find directory index in current directory
•           int dirIndex = findfilebyname(&currentParent->dir, token);
•         //printf("current directory is %s\n", token);
•         // if found
•         if (dirIndex != EOF) // return an index
•         {
•            // update the current parent to next dir
•            currentDirIndex = currentParent-
   >dir.entrylist[dirIndex].firstblock; // get fat index
•            currentParent = &virtualDisk[currentDirIndex]; // update
   currentparent
•         }
•         else // the index was not found
```

- While token doesn't equal NULL dirIndex will find the directory(token) by name using findfilebyname function I created earlier. If findfilebyname return EOF that means the name was not found in the currentParent entrylist. So if it dirIndex doesn't equal EOF then get the firstblock of that entry found firstblock = FAT block no .
- Equal global variable currentDirIndex to firstblock and the get the diskblock from disk with currentDirIndex and set currentParent to this block. This just a way traverse to the location of the last directory in the path

```
dirIndex = findUNUSEDdirentry(&currentParent->dir); // find unused dir in
current parent
         if (dirIndex  == EOF)
         {
            printf("All entries used up! \n");
         }
         else
         {
            int fatIndex = findUNUSEDfatentry(); // find an unused fat entry
            if (fatIndex == ENDOFCHAIN) // end not found
            {
               printf("FAT table is full!\n");
            }
            else
            {
```

- After if a directory is not found then we create on inside the currentParent dir block
- We do this be finding a unused dir entry and unused fat entry used findUNUSEDdirentry and findUNUSEDfatentry if both return a valid index then continue

```
               if ( path[0] == '/')
               {
                  printf("Creating directory...\n");
                  // initialise the next level directory
                  currentParent->dir.entrylist[dirIndex].firstblock =
fatIndex; // set firstblock to found entry
                  currentParent->dir.entrylist[dirIndex].isdir = TRUE; // is
dir
                  currentParent->dir.entrylist[dirIndex].unused = FALSE;//
used
```

```
                  strncpy(currentParent->dir.entrylist[dirIndex].name, token,
MAXNAME); // set name
                  writeblock(currentParent, currentDirIndex); // write the
current parent block

                  addfatentry(fatIndex); // add entry to fat table
                  currentDirIndex = fatIndex; // update current Index
                  currentParent = &virtualDisk[currentDirIndex]; // give it a
block
                  currentParent->dir.isdir = TRUE; // new block is dir
                  currentParent->dir.nextEntry = 0; // set to 0

                  // set all entry in current to unused
                  for (int i = 0; i< DIRENTRYCOUNT;++i)
                  {
                     currentParent->dir.entrylist[i].unused = TRUE;
                  }
                  writeblock(currentParent, currentDirIndex);
               }
```

- If the first string in the path was equal to "/" then that means its absolute and we should create the path we initialize the new directory in the currentparent then we add it FAT then we equal currentDirIndex to FAT block no and set currentParent to that block no in virtualDisk. The dir block of the block with be set to isdir=TRUE to indicate it's a dir. All entries is equal to unused.

MYLIST

- Uses a list of pointer Charlist to store the names of directories found in the path

```
// find directory index in current directory
      int dirIndex = findfilebyname(&currentParent->dir, token);
      // if found
      if (dirIndex != EOF)
      {
         // update the current parent to next dir
         currentDirIndex = currentParent->dir.entrylist[dirIndex].firstblock;
// get fat index
         currentParent = &virtualDisk[currentDirIndex]; // update
currentparent
         for (int i = 0; i < DIRENTRYCOUNT; i++)
         {
            // allocate memory to each pointer in the list
            CharList[i] = malloc(sizeof(char)*MAXNAME);
            // set the name of each entry (i) in the current parent
(entrylist) into the list at index i
            strcpy(CharList[i], currentParent->dir.entrylist[i].name);
         }
```

- This is done by copying the name every entry from the currentParent entrylist to the same an index in the Charlist each index is simultaneously allocated memory large enough to hold this entry name

- After this loop terminates the CharList will contain the names of all the directories in the path

MyFopen

- Some code in myfopen was changed to support calling the path as a parameter

```
/* seperate the directory from the file name*/
   char * lastSlash = strrchr(filename, '/');
   if (lastSlash != NULL)
   {
      size_t newlength = lastSlash - filename; // the size of dir path

      char newpath[newlength]; // create string

      strncpy (newpath, filename, newlength); // copy dir path into string

      newpath[newlength] = '\0';

      mymkdir(newpath); // call mymkdir to make the directories or get the
currentDirIndex

      filename = lastSlash + 1;  // this will be the name of the file only
   }
```

- This was done using the strrchr function the returns the index of the string where the last time the '/' was found in the string
- I used this because I separate the /firstdir/secondir / from the actual file name testfile.txt
- Strrchr will return NULL if delimiter '/' etc was not found so why lastSlash does not equal NULL. Newlength will equal the exact length of the path from the 0 to the index – 1 the lasr '/' was found the an array newPath is recreate with the the newlength allowing me to copy the from the path the exact length of the path that doesn't hold the actual file name
- Then mymkdir is called to create this dir path if it doesn't not exist of cd this path to currentDirIndex. Filename is then cut from the lastSlash index to the end of the path ("testfile.txt"). filename can be used all through the function so it was the easier way to change like this
- If a path isn't specified in when calling myfopen eg.

```
myfopen("testfile2.txt", "w");
```

- Then file will be create inside the currentDirIndex block

```
diskblock_t * currentParent = &virtualDisk[currentDirIndex];
```

- Further parts of the code wasn't necessary to be changed

<u>CGS A5-A1</u>

- mychdir(char * path), using the global variable "currentDir" as specified in filesys.c: a change into a
  directory will change the variable "currentDir"
  myremove(char * path) removes a file; the path can be absolute or relative

myrmdir(char * path) removes a directory, if it is empty; the path can be absolute or relative

mylistDir

```
// ------------- Code Changed ----------------- ||
   // first check that path is self referenced
   if (path[0]== '.')
   {

      // the current block using currentDirIndex
      currentParent = &virtualDisk[currentDirIndex];
      for (int i = 0; i < DIRENTRYCOUNT; i++)
      {
         // allocate memory to each pointer in the list
         CharList[i] = malloc(sizeof(char)*MAXNAME);
         // set the name of each entry (i) in the current parent (entrylist)
into the list at index i
         strcpy(CharList[i], currentParent->dir.entrylist[i].name);
      }
      return CharList; // return List
   }
```

- With this change mylistdir(".") can be used to return the name of directories and files inside the currentDirIndex entrylist.

```
if (path[0] == '/')
   {
      currentParent = &virtualDisk[rootDirIndex];
      currentDirIndex = rootDirIndex;
   }
```

- Also if it start with "/" then currentDirIndex is set to rootDirIndex which is enabled globally and currentParent set to root dir block.

```
for (int i = 0; i < DIRENTRYCOUNT; i++)
   {
      // allocate memory to each pointer in the list
      CharList[i] = (char*) malloc(sizeof(char)*MAXNAME);
      // set the name of each entry (i) in the current parent (entrylist) into
the list at index i
      strcpy(CharList[i], currentParent->dir.entrylist[i].name);
   }
```

- Also this for loop was taken out of while loop to make the code run after and because it was necessary to be in the loop as it was only needing to copy the contents inside the end directory as was previously doing it for all of them
- Added helper function deletefat() to remove fat entry from the FAT table the it just the opposite of addfatentry().

```
void deletefat (int blokno)
{
   //check if blokno size fits in block 1 or block 2
   if (blokno > 1024)
   {
      printf("block no is outside range of fat table\n"); // blokno outside
FAT table range
   }
   // from 4 to 511 remove from  block 1
   if ( blokno < 512)
   {
      FAT[blokno] = UNUSED;
      virtualDisk[1].fat[blokno] = UNUSED;

   }
   else
   {
      // or from 512 - 1024 remove from block 2
      FAT[blokno] = UNUSED;
      virtualDisk[2].fat[blokno] = UNUSED;
   }
}
```

Mychdir

```

    if (path == '/')
    {
       currentDirIndex = rootDirIndex;
       currentParent = &virtualDisk[currentDirIndex];
    }
```

- If the path parameter as a whole equal only this '/' string then the currentDirIndex is equal to rootDirIndex

```
if (path == "..")
    {
       // update current
       currentDirIndex = parentDirIndex;
    }
```

- If the path == ".." then the currentDirIndex is equal to parentDirIndex which is th prev currentDirIndex
- `int parentDirIndex = rootDirIndex; // parenntDirIndex = prev level Dir index`

```
if (dirIndex != EOF)
       {
          parentDirIndex = currentDirIndex;
          currentDirIndex = currentParent->dir.entrylist[dirIndex].firstblock;
          currentParent = &virtualDisk[currentDirIndex];
       }
```

- Inside the loop the parentDirIndex is equal to the previous value of currentDirIndex

Myremove

```
char * lastSlash = strrchr(path, '/');

   if (lastSlash != NULL)
   {
      size_t newlength = lastSlash - path + 1; // the size of dir path

      char newpath[newlength]; // create string

      strncpy (newpath, path, newlength); // copy dir path into string

      newpath[newlength] = '\0';

      pathCopy = strdup(newpath); // dir path copied

      path = lastSlash + 1;  // this will be the name of the file only
```

- Like the method used to separate the file name from the dir path
- pathCopy copies the newpath which is the dir path
- and path parameter changed to file's name
- The loop only executes if a path or "/" is used but since no name who be found after "/"

```
mychdir(pathCopy); // change to directory
      currentParent = &virtualDisk[currentDirIndex];
      int dirIndex = findfilebyname(&currentParent->dir, path);
      if (dirIndex != EOF)
      {
```

- This Condition inside this loop wont execute

```
if (currentParent->dir.entrylist[dirIndex].isdir == FALSE)
         {
```

- Followed by a check if what is deleting is file or directory; will only delete file

```
// un-initialise the file dir
            currentParent->dir.entrylist[dirIndex].unused = TRUE;
            currentParent->dir.entrylist[dirIndex].filelength = 0;
            currentParent->dir.entrylist[dirIndex].entrylength = 0;
            for (int i=0;i<MAXNAME;++i)
            {
               currentParent->dir.entrylist[dirIndex].name[i] = "\0"; //This
can be used also
            }
```

- Removes file from the directory it is in

```
int nextblock = currentParent->dir.entrylist[dirIndex].firstblock;
            while (nextblock != ENDOFCHAIN)
            {
               // pointer instead of re writing to block
               diskblock_t * buffer = &virtualDisk[nextblock];
               for (int i = 0;i<BLOCKSIZE;++i)
```

```
                {
                   buffer->data[i] = '\0';
                }
                // save the number ---> go to next block ---> delete previous
block number
                int saveblkno = nextblock; nextblock = FAT[nextblock];
deletefat(saveblkno);
             }
             currentParent->dir.entrylist[dirIndex].firstblock = UNUSED;
```

- Nextblock = firstblock in chain
- Traverse through FAT chain equal all the data in the buffer to '\0'
- Save the block no , traverse through to the nextblock and delete the prev block no from the chain
- Set the firstblock in the chain to UNUSED in the currentParent directory

```
// if just the testfile name
   currentParent = &virtualDisk[currentDirIndex];
   int dirIndex = findfilebyname(&currentParent->dir, path);
   if (dirIndex != EOF)
   {
         if (currentParent->dir.entrylist[dirIndex].isdir == FALSE)
         {
            // un-initialise the file dir
            currentParent->dir.entrylist[dirIndex].unused = TRUE;
            currentParent->dir.entrylist[dirIndex].filelength = 0;
            currentParent->dir.entrylist[dirIndex].entrylength = 0;
            for (int i=0;i<MAXNAME;++i)
            {
               currentParent->dir.entrylist[dirIndex].name[i] = '\0';
            }
            currentParent->dir.nextEntry = 0;
            // remove the file buffer(s) from the FAT table
            int nextblock = currentParent->dir.entrylist[dirIndex].firstblock;
            while (nextblock != ENDOFCHAIN)
            {
               // pointer instead of re writing to block
               diskblock_t* buffer = &virtualDisk[nextblock];
               for (int i = 0;i<BLOCKSIZE;++i)
               {
                  buffer->data[i] = '\0';
               }
               // save the number ---> go to next block ---> delete previous
block number
               int saveblkno = nextblock; nextblock = FAT[nextblock];
deletefat(saveblkno);
            }
            currentParent->dir.entrylist[dirIndex].firstblock = UNUSED;
            printf("File is now deleted!\n");
         }
         else
         {
            printf("Can not delete a directory\n");
         }
   }
      else
      {
         printf("FileNotFoundError!\n");
      }
```

- Outside lastSlash if statement the code is repeated so that if a file is removed with specifying a path with it then it would remove the file from the current directory if it is found

Myrmdir

```
char *token, *rest; // tokenize path and save pointer
   char *pathCopy = strdup(path); // copy path
   diskblock_t *currentParent = &virtualDisk[currentDirIndex];
   fatentry_t prevDirIndex = 0;
   int dirIndex;
   token = strtok_r(pathCopy, "/", &rest);
   while (token != NULL)
   {
      dirIndex = findfilebyname(&currentParent->dir, token); // find directory
by name
      //currentDir = &currentParent->dir.entrylist[dirIndex]; // change into
path directory !!
      if (dirIndex != EOF)
      {
         prevDirIndex = currentDirIndex;
         currentDirIndex = currentParent->dir.entrylist[dirIndex].firstblock;
         currentParent = &virtualDisk[currentDirIndex];
      }
      else
      {
         printf("Not found path \n");
      }
      token = strtok_r(NULL, "/",&rest);
   }
```

- prevDirIndex stored directory fat indexes
- dirIndex declared outside while loop so it can be used outside loop
- prevDirIndex equal to previous value of currentDirIndex each iteration

```
diskblock_t * prevParent = &virtualDisk[prevDirIndex];
   direntry_t p;
   if (dirIndex != EOF)
   {
      strncpy(p.name, "\0", MAXNAME);
      prevParent->dir.entrylist[dirIndex] = p;
      prevParent->dir.entrylist[dirIndex].unused = TRUE;
      deletefat(currentDirIndex);
```

```
        printf("deleted\n");
    }
```

- creates a null dir entry ‘p’ and block that equal the previous block of currentParent
- if dirIndex is not EOF then set name equal to “\0”
- clear dir entry from the prevParent using empty dir entry p
- set the entry to unused so it maybe used again
- delete currentDirIndex from FAT using delete fat function

A3