CS3026 Operating Systems – Assignment 2  
File Allocation Table  
Julia Wiktoria Zięba (ID: 51768280)

REPORT

Contents

[CSG\_D3\_D1 3](#_Toc25320460)

[Function format() 3](#_Toc25320461)

[Function copyFAT() 3](#_Toc25320462)

[Helper functions 4](#_Toc25320463)

[Program shell 4](#_Toc25320464)

[Hexdump output 5](#_Toc25320465)

[CSG\_C3\_C1 5](#_Toc25320466)

[Function myfopen() 5](#_Toc25320467)

[Function myfclose() 5](#_Toc25320468)

[Function myfputc() 5](#_Toc25320469)

[Function myfgetc() 5](#_Toc25320470)

[Helper functions 5](#_Toc25320471)

[Hexdump output 6](#_Toc25320472)

[CSG\_B3\_B1 6](#_Toc25320473)

[Function mymkdir() 6](#_Toc25320474)

[Function mylistdir() 6](#_Toc25320475)

[Helper functions 6](#_Toc25320476)

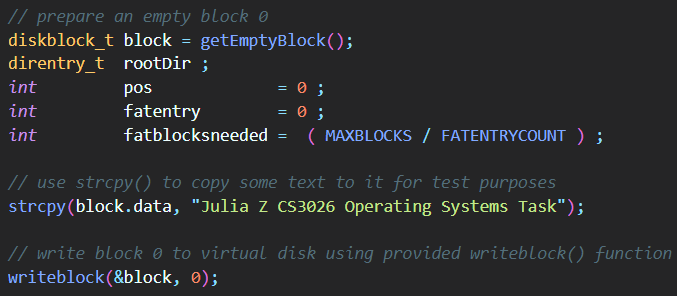
[Hexdump output 6](#_Toc25320477)

[CSG\_A5\_A1 6](#_Toc25320478)

# CSG\_D3\_D1

## Function format()

#### Initialising the first block



* An empty diskblock\_t is initialised using a helper [getEmptyBlock()](#_getEmptyBlock()) function
* A sample text is copied to the block using a built-in strcpy() function
* Block is saved to the virtual disk on the 0th position

#### Preparing the fat

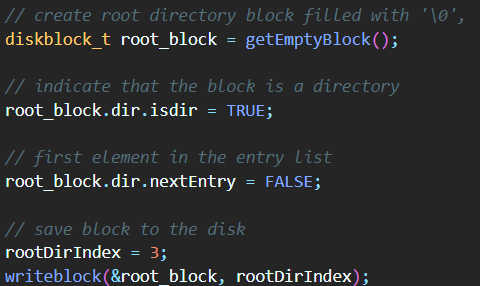
* The FAT table is initialised – all positions are marked as UNUSED
* The FAT has 1024 2-byte entries, so it needs 2 blocks on the disk; thus, there is a chain in FAT from position 1 to 2 for the FAT itself
* FAT[0] = ENDOFCHAIN, since 0th disk block is reserved
* FAT[3] = ENDOFCHAIN, since that indicated the root directory
* FAT is written to the virtual disk

A screenshot of a video game

Description automatically generated

#### Initialising the root directory block

* An empty block is initialised for the root directory using a helper [getEmptyBlock()](#_getEmptyBlock()) function
* The block is marked as a directory block
* The index of the next entry is marked as FALSE (0), since it is an empty directory
* The FAT index of the root directory is set to 3 according the assessment instructions
* The root directory block is saved to the virtual disk on the 3rd position



## Function copyFAT()

#### copyFAT() - copies the content of FAT and saves it to the virtual disk (on the reserved blocks)

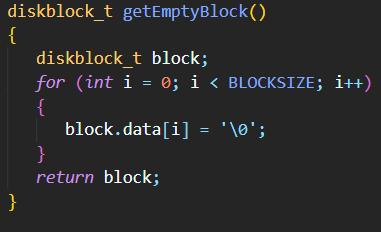
* The outer loop iterates 2 times, since FAT occupies 2 blocks on the virtual disk
* A new block is created for the copying purposes
* The inner loop iterates 512 times – there are 512 FAT entries in a block
* Value of each FAT entry is copied to the block
* The block is written to the virtual disk

Since FAT has 1024 entries, the index of the saved entry must be adjusted using the outer loop iterator. During the first iteration entries 0 - 511 are copied (entry + 512\*0); during the second iteration entries 512 - 1023 are copied (entry + 512\*1).



## Helper functions

#### getEmptyBlock()



This function has been implemented, since a creation of a new, empty block was a common operation

* New block of type diskblock\_t is initialised
* Block data is set to ‘\0’
* An empty block is returned

## Program shell

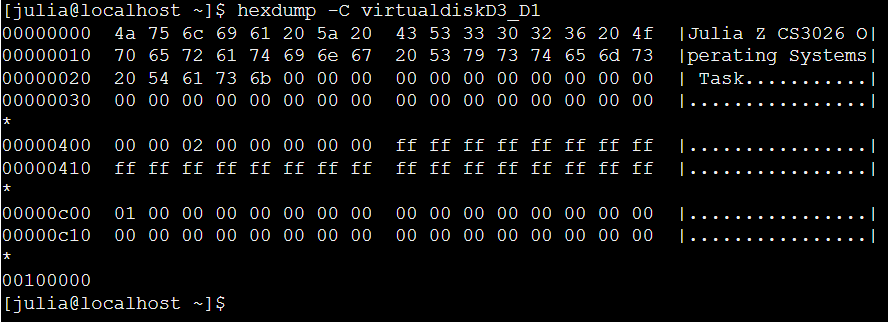
#### Shell.c

A screenshot of a cell phone

Description automatically generated

* Function [format()](#_Function_format()) is called
* The state of the disk is saved and written into “virtualdiskD3\_D1” file

## Hexdump output



Block 0 (0x0 – 0x3ff) – reserved block; the numbers represent hexadecimal values of the characters stored in that block (“Julia Z CS3026 Operating Systems Task”)

The FAT starts at 0x400:

* Value ’00 00’ is the value in FAT[0] set to ENDOFCHAIN
* Value ’02 00’ is the value in FAT[1], which is the index value of the next entry in the chain – FAT[2]
* Value ’00 00’ is the value in FAT[2] set to ENDOFCHAIN
* Value ’00 00’ is the value in FAT[3] set to ENDOFCHAIN (indicating the root directory)
* All other FAT entries are marked as UNUSED – ‘-1’ or 0xff

Value ’01 00’ at 0xc00 represents the root directory (root\_block.dir.isdir = TRUE)

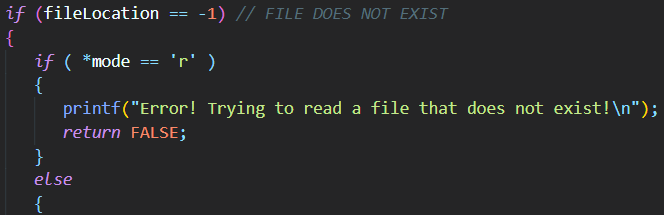
# CSG\_C3\_C1

## Function myfopen()



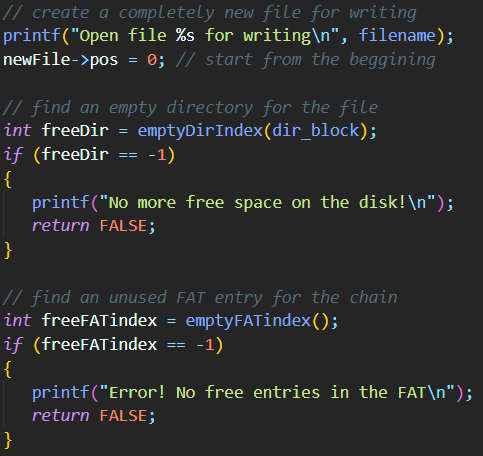
* Initialise a diskblock\_t and assign to it contents of the root directory (occupying 3rd block of the virtual disk)
* Check if the passed file mode is correct – either ‘w’ for writing, or ‘r’ for reading; display an error message and exit the function if not correct
* Allocate memory for the file descriptor using built-in malloc() function
* Assign passed file mode to the newly created file descriptor using built-in strcpy() function
* Check if file to be opened already exist on the in the directory using a helper [lookForFile()](#_lookForFile()_–_returns) function

Case when file does not exist in the root directory



Creating a new file in a writing mode:

* The ‘pos’ attribute is set to 0 – points at the current buffer position
* The next free entry in the directory is found and its index is returned using helper [emptyDirIndex()](#_emptyDirIndex()_–_returns) function
* The next free entry in FAT is found and its index is returned using helper [emptyFATindex()](#_emptyFATindex()_–_returns) function
* If the mode is ‘r’ – reading – an error message is returned
* The function is exited – a file cannot be read if it does not exist

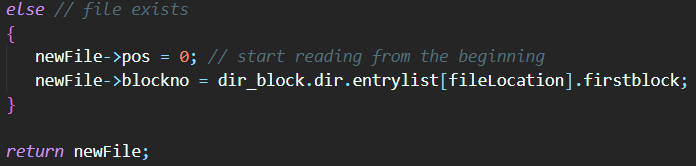


* The next free FAT entry is set to ENDOFCHAIN, creating a FAT chain for the new file (there are no contents in this file, so it will not occupy more than one block for now)
* The ‘blockno’ attribute of the file descriptor is set to file’s index in the FAT
* The position in the root directory of the new file is set
* FAT is written to the virtual disk
* Name of the file is saved in the directory using a built-in strcpy() function
* The entry where the new file is now stored is marked as USED (.unused = FALSE)
* The new state of the directory is saved to the virtual disk



Case when file exist in the root directory

* The ‘pos’ attribute resets to point at the beginning of the buffer
* The ‘blockno’ attribute of the file descriptor is set to the corresponding value in the directory



A pointer to a file descriptor is returned after all operations have been performed.

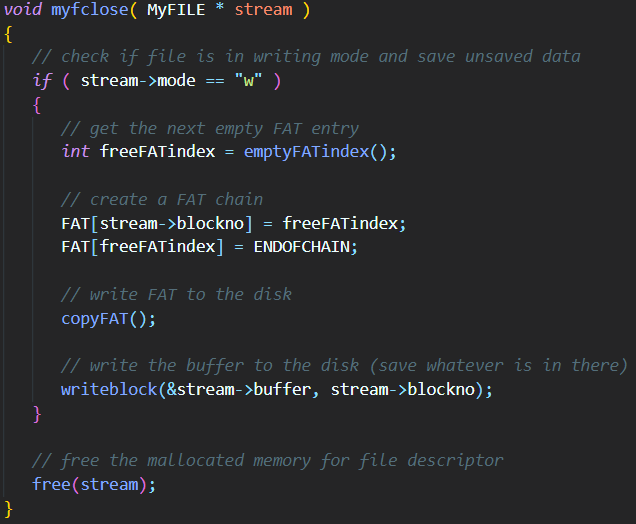
## Function myfclose()

If file is in writing mode:

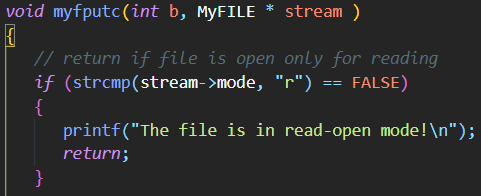
* Find the next free entry in FAT
* Set the FAT entry of the current block to point at the newly found free entry
* Mark the new FAT entry as ENDOFCHAIN indicating the end of chain in FAT for the file that is being closed
* Write the contents of FAT to the disk
* Write the current state of the buffer block of the file descriptor to the disk – save unsaved contents of the file

For a file in any mode:

* Free the memory allocated for the file descriptor



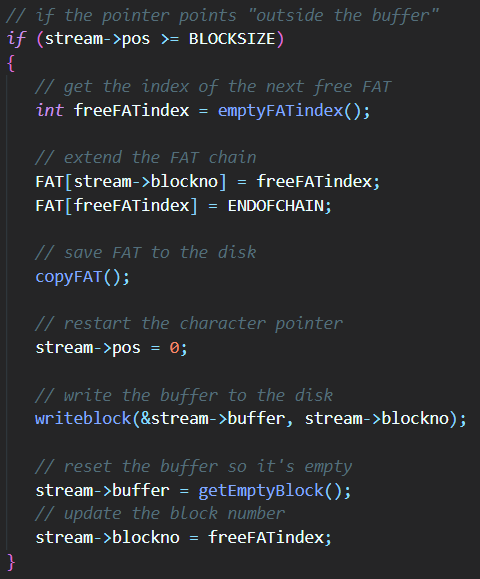
## Function myfputc()



* Display an error message if the myfputc() function is called on a read-only file
* Exit the function

Case when the file descriptor buffer is full:

* Checking, if the position counter is pointing outside the block – if it is larger than 1024 (size of a block)
* Find the next empty FAT entry
* Extend the FAT chain
* Mark the next free FAT entry as the ENDOFCHAIN
* Save FAT to the disk

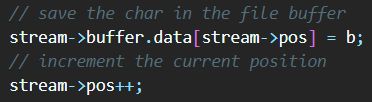


* Reset the buffer by assigning a new, empty block to it
* Update the block number to be pointing at the next free FAT entry
* Save the full buffer to the disk
* Reset the position pointer to point at the 0th index in the buffer

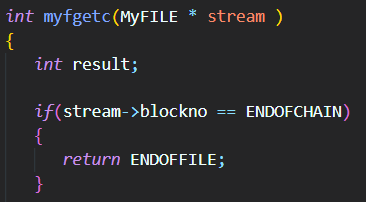
Case when buffer is not full or has just been cleaned:

* Save the passed character ‘b’ to the buffer on the current position
* Increment the position pointer to be pointing at the next free entry in the file descriptor buffer

v



## Function myfgetc()

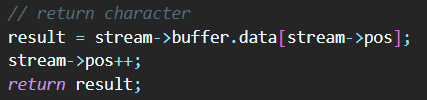


* Save a character on the current position of the buffer to a previously declared variable ‘result’
* Increment the position pointer
* Return the character

v

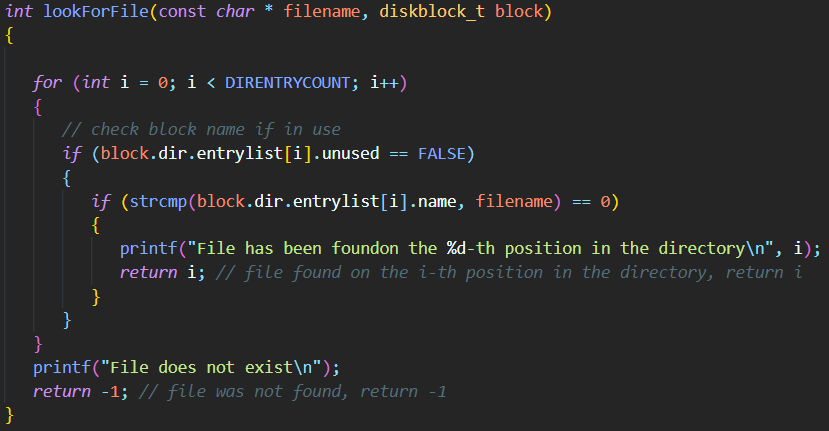
* Declare a variable for saving the character in
* Check if it’s the end of the file; if so, return ENDOFFILE character

v



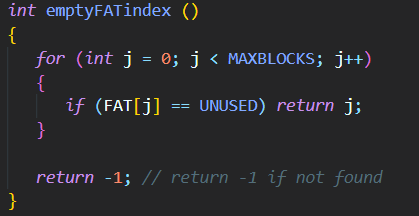
## Helper functions

#### lookForFile() – returns the location of the file in the directory



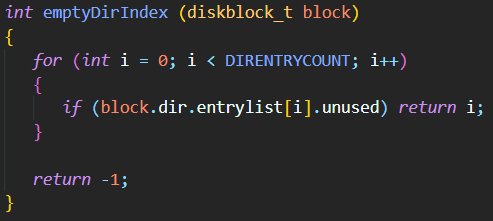
* Loop iterates over directory entries
* Firstly, it is checked whether a block is in use
* Then, it is checked whether the name attribute of the block is the same as ‘filename’
* If the file is found, its index in the directory entrylist is returned
* ‘-1’ is returned if there is no such file

#### emptyFATindex() – returns the index of the first free entry in the FAT



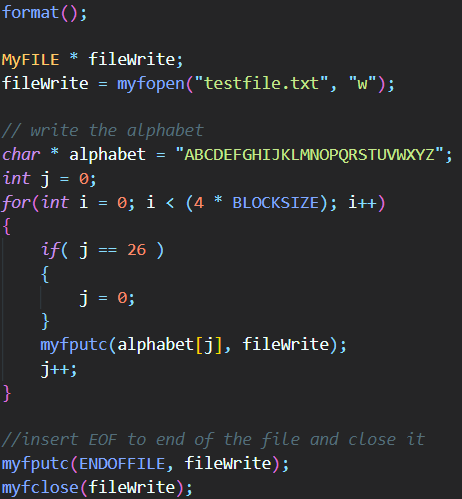
* Loop iterates over all FAT entries
* If an entry is marked as UNUSED (it’s empty), its index is returned
* ‘-1’ is returned if the FAT if full

#### emptyDirIndex() – returns the index of the next free directory entry



* Loop iterates over all directory entries
* If an entry is marked as UNUSED (it’s empty), its index is returned
* ‘-1’ is returned if there are no free entries

## Program shell

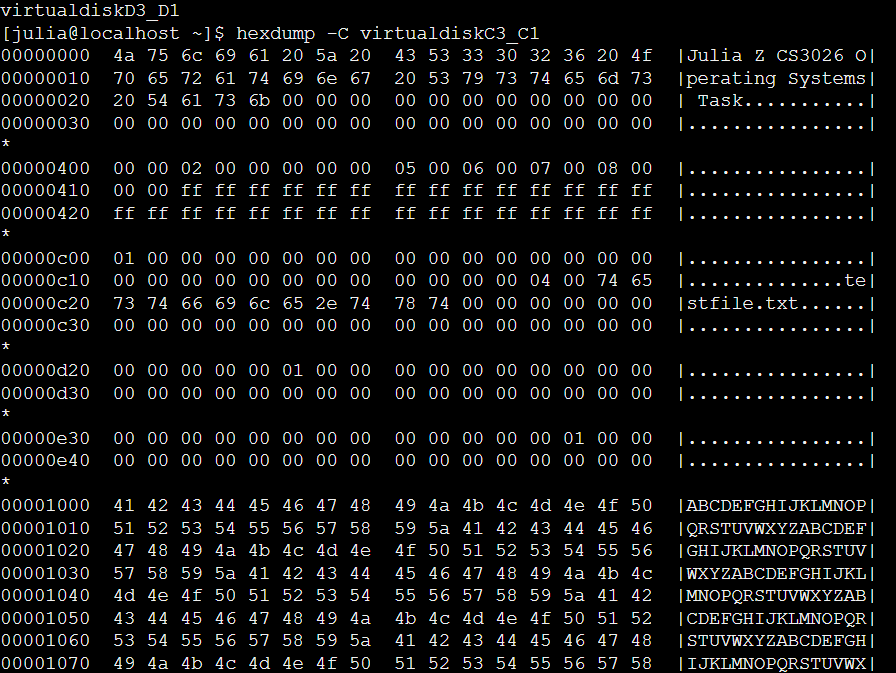


* After writing 4kB of characters into the file, ENDOFFILE character is also saved to the file
* The file is closed using myfclose() function
* Declare a string containing the alphabet
* The loop calls myfputc() functions 4096 times and saves the alphabet to the file
* The ‘j’ variable is used to iterate over the alphabet string – it is incremented after each loop and if it exceeds 26 (the number of letters in the alphabet) it is set to 0 again
* Call the format() function
* Initialise a new file descriptor and open a “testfile.txt” in a writing mode using [myfopen()](#_Function_myfopen())

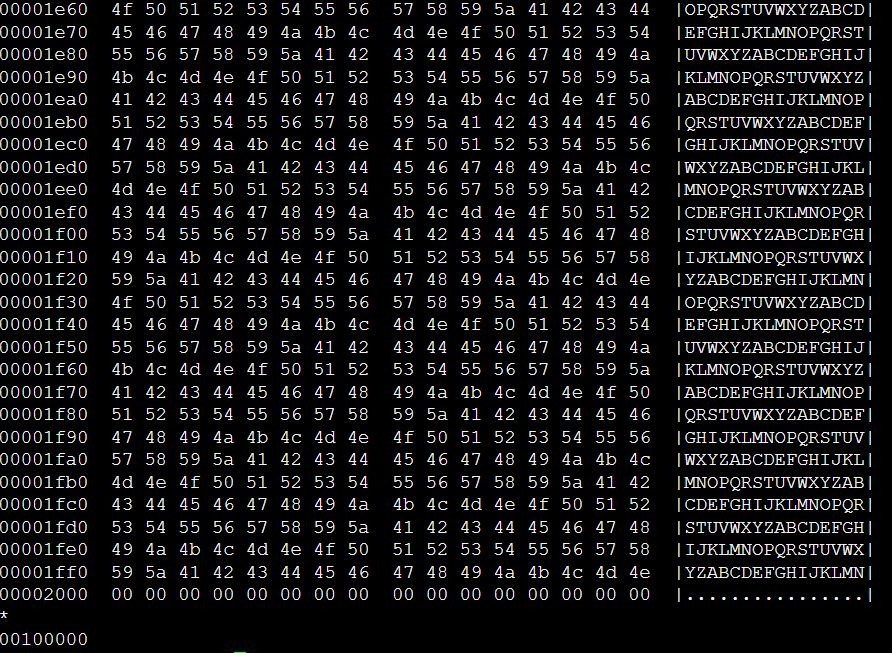


* The ‘fileRead’ is closed using myfclose() function
* The real file is closed
* Virtual disk is saved
* Loop iterates 4096 times or until ENDOFFILE character is encountered
* During each iteration a character is read from the “testfile.txt” on the virtual disk using myfgetc() function
* Then, the character is written to the real file
* Declare a new file using myfopen() – this time for reading
* Declare a real file in order to copy the data from the ‘fileRead’ into it

## Hexdump output



Continuation of the output



In addition to the content of the virtual disk from the CGS\_D3\_D1 part:

The file “testfile.txt” has been created in the root directory in the block #3 starting at 0xc00.

The FAT has been updated and a block chain is visible: FAT[4] = 05 00, FAT[5] = 06 00, FAT[6] = 07 00,   
FAT[8] = 09 00, and FAT[9] = ENDOFCHAIN.

The numbers are hexadecimal representation of characters stored in the virtual disk memory.

# CGS\_B3\_B1

## Function mymkdir()

## Function mylistdir()

## Helper functions

No additional helper functions were implemented for this task.

## Hexdump output

In addition to the content of the virtual disk from parts CGS\_D3\_D1 and CGS\_C3\_C:

# CSG\_A5\_A1