**Running the application**

I did this project under Unix so to run it in Linux or MacOSX you have to extract cs3008\_assessment\_u07ev11.zip file to selected directory. Then you have to open Terminal in your Operating System and select that directory. Extracted file should contain:

* report
* shell.c
* filesys.c
* filesys.h
* virtualdisk9\_11
* virtualdisk12\_14
* virtualdisk12\_14\_copy.txt
* trace12\_14.txt
* Makefile

To run program you can use provided Makefile. It is lauched by typing into Terminal: **make shell**. To compile and run application manually you have to write in your Terminal: **gcc \*c** and then you have to run the compiled program by typing to your Terminal: **./a.out**. If you want to see what the file contains write: **hexdump –C virtualdisk9\_11** for CAS9-11 part and **hexdump –C virtualdisk12\_14** for CAS12-14 part. Moreover, to redirect my shell’s output from my shell program into trace12\_14.tx file write in your terminal: **hexdump -C virtualdisk12\_14 > trace12\_14.txt.**

**Implementation description**

**For CAS9-11:**

First of all I got familiar with filesys.c and filesys.h files and their data structures. Also with functions I had to use to implement following tasks virtual disc’s layout.

To start with I had to implement format() function in filesys.c class. Before that I wrote test program shell.c containing main function. At the very beginning I included <stdio.h> library and "filesys.h". Then in main function I called readdisk("virtualdisk9\_11") function which is in filesys.c class to read virtual disk from physical disk. I also called format() function to format that virtual disk and then writedisk("virtualdisk12\_14") to write it back in physical disk.

When I was finished with shell.c I had to declare void format() function in file.h and after that started implementing format() function.

* At the begining I had prepare block 0. I filled it with 0. Then I used strcpy() to copy some text to it for test purposes and wrote block 0 to virtual disk. To prepare fat table I filled all it’s entries starting from 4th one to unused because first one is used by block 0 which is reserved for information about file system. Following two are used by block 1 and 2 that are occupied by the FAT (we need 2 blocks, because each entry is a short integer, occupying 2 bytes of disk space, and with 1024 entries in the FAT it needs 2048 bytes, which are 2 blocks of disk space). Third is used by root directory. Then set FAT own chain and wrote first 512 FAT entries to first fat block 1 and other 512 to fat block 2. To prepare root directory I had to fill all block with 0’s and write root directory block to virtual disk. To finish formatting I formatted rest of blocks by filling all block with 0’s and writting them to virtual disk.
* **For CAS12-14:**

I had to implement the following functions:

* myfopen()
* myfput()
* myfget()
* myfclose()

I started with extending shell.c test program.I created a file on my virtual disk and opened it using myfopen() function with mode set to write. To put some text to file I had to create a char array of 4\*BLOCKSIZE length and fill it with some text. To do that I iterated through that array and wrote all its entries one by one to file with myfput() function. I closed the file using myfclose() function.

To get files content from virtual disk I had to open that file with read mode. The following step was to make for loop of 4\*BLOCKSIZE length and get all content of text piece by piece. To write all received data to real file on my physical disk I had to open it and put every piece to that file. Finaly I closed it.

After declaring all 4 functions, that I had to implement, in filesys.h I moved to filesys.c implementation. I started with myfopen() function . I recorded files index from directory block’s entry list to fileindex variable if file with given filename exists in root directory. After that I checked if file's mode is readable. If condition was positive I had to read disk block and copy it to my virtual files buffer (disk block). Then I had to set file’s position (byte within block) to 0 and return file. But if file with given name was not located in root directory, I had to get next available entry in root directory's entrylist and assign its value to fileindex variable along with increasing next available entry rootDir.nextEntry by one. Then I had to prepare directory entry by setting its entrylength, making entry used, setting it as data block and copying files name to entry’s name. To finish I had to write directory block to my virtual file system and return the file.

The following task was to put content to file piece by piece as I described it in shell.c program. To do that I had get current file's position (byte within block) and check if it is smaller or equal to BLOCKSIZE. If condition was positive I had to put byte (entry from array list in shell.c) to block (files buffer) by file's current position (byte within block) and increase it by 1. If current file's position (byte within block) was grater then BLOCKSIZE I had to set current file's position (byte within block) to 0. Then wrote the block by its number, which is currently 4 to virtual disk and increased it by 1. Moreover, I deleted buffer's content and set it's content to 0. Then I updated FAT by setting FAT block chain and writing blocks 1 and 2 to virtual disk as I did when formatting.

To get content from the file I had to get current file's position (byte within block) and check if it is smaller or equal to BLOCKSIZE. Within IF statement I got byte (entry within block) from block (files buffer) by file's current position (byte within block) and increased it by 1. If current file's position (byte within block) was grater then BLOCKSIZE I had to set current file's position (byte within block) to 0. Then I set buffer’s content to 0 to delete its content. Moreover, I read the block by its number, which was currently 4 to virtual disk and increase it by 1. Then I got byte (entry within block) from block (files buffer) by file's current position (byte within block). At the end of this function I returned the byte I just got from the file.

The last function myfclose() have to close the file but I did not find how to implement it correctly so I just freed allocated memory for file.