COMP1562 Logbook (Week 7)
Basic Information

1.1 Student name	**Trevor Kiggundu (001001720)**
1.2 Who did you work with? Name and/or id	**Maruf Hoque (001006731)**
1.3 Which lab topic does this document relate to?	File Systems
1.4 How well do you feel you have done?	I have completed the exercise and am totally satisfied with my work.
1.5 Briefly explain your answer to question 1.4	My group and I were able to successfully follow and complete the tasks. Proof of that is shown below.

Annotated screenshots demonstrating what you have achieved:

REPORT:

We were supposed to write a brief, coherent report (2-4 pages long) comparing, contrasting or describing the following disk monitoring and disk maintenance tools for Linux and Windows systems:

Linux and Windows Systems

fsck (Linux) vs **scandisk** (Windows)

fsck (Linux): fcsk (file system consistency check) is a tool that runs on unix/unix like systems like Linux and FreeBSD. In Linux however, it is mainly used a checking system, making sure that there are no errors in the file. It also attempts to repair fixable errors within the disk and generates a report of its findings. Fcsk is also used when a system fails to boot up and/or files or specific file location is compromised. Files are checked for inconsistencies that cause corrupted files.

scandisk (Windows): scandisk is a tool that runs on the Windows operating system. It is also used as an error checking system for softwares running the system. There are two main methods of error checking while using scandisk; error checking tool and SFC/SCANNOW. The error checking tool can be accessed using the File Explorer while SFC uses command lines to scan for files and resources.

Comparison between **fcsk** and **scandisk**: Both tools are quite similar, however scandisk can be limited as it only checks file systems supported in Windows. Fcsk is a bit more robust and much more optimised for deeper file checks.

mkfs (Linux) vs **format** (Windows)

mkfs (Linux): This is a command line line system used to build systems on the Linux operating systems. It is a directory based system, also creating sub-directories for the given storage device. The system does not create a directory hierarchy, allowing for the user to organize directories and files.

format (Windows): This is the preparation of a certain drive to be used by the Windows operating system (ex: formatting a hard drive). This is usually done using the disk management tool (often accessed by using a 'right-clicking' shortcut) or even in command prompt and control panel. This however, can only be done when the specified drive is not in use.

Comparison between **mkfs** and **format**: Once again, the Linux tool is a little bit more robust as it can use a variety of file systems, and can also uses up much less memory than Windows due to a process called 'Swap Partitioning'.

Logical Volume Manager (Linux) vs **Logical Disk Manager** (Windows).

Logical Volume Manager (Linux): This allows the user to view disk storage in greater detail, allowing the user to be able to make better decisions regarding storage use. It also allows the user to adjust ‘storage volumes’ that control multiple disk drives, making it as easy to do as if the user was only controlling smaller disk drives. This allows the user to also allocate larger groups of disk space to different drives as desired, giving the user an almost 100% customizable experience.

Logical Disk Manager (Windows): Similar to LDM in Linux, this allows the user to create and customize dynamic disks. These can be resized just like the ‘storage volumes’ in Linux; these are called dynamic volumes, of which there are 5 of:

● Simple - This dynamic volume is made from a single disk. A simple volume can be made from a single region on a disk or multiple regions on the same disk that are linked together.

● Mirrored - This dynamic volume is error free. If one of the disks fail, the data stored on the second disk and the system runs as usual.

● Spanned - This dynamic volume is made by using multiple physical disks. Size can be extended by using additional user created disks.

● Striped - This dynamic volume stores data in stripes on two or more physical drives. Data storage on these devices is done in ‘stripes’.

● RAID-5 - By far the most robust (except for Linux), it is error-free. ‘Parity’ allows errors and failures in the disks to be reconstructed. RAID-5 volumes can only be created on dynamic disks running Windows. It can also revive info from failed drives.

Comparison between **LVM** and **LDM**: The Windows based dynamic disks are a little more robust than the basic disks used in the LVM. Damaged disks can be replicated/reproduced without data loss, and the user can choose whether or not they would like to use the logical volume manager or the dynamic volume manager, allowing for more flexibility.

S.M.A.R.T. utility

SMART (Self-Monitoring, Analysis, and Reporting Technology) is a disk monitoring application used to scan hard drives to test operational and mechanical problems. The reason SMART utility is so coveted is because of its ability to display the individual attributes of the drive and specify exactly where it is failing. After this has been determined, it provides a report of where the drive failed and potential fixes if not already resolved. The fact that the system is self-testing means that some problems can be detected before they become too serious (ex: irreparable damage).

Comparison with others: SMART is very robust as it runs on modern and solid state disks. This allows the system to better detect/prevent problems with mechanical drives that are more prone to mechanical failures than software based drives.

TASK 7.1:

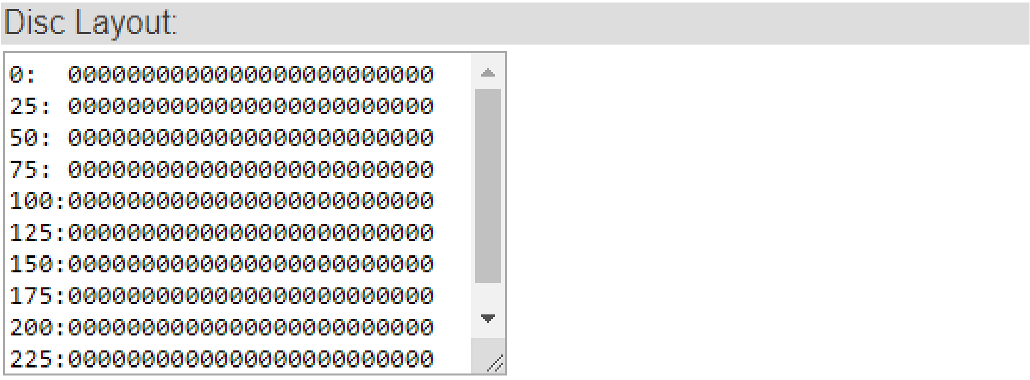
This task required us to allocate 3 files using the contiguous file allocation algorithm. We were also given 3 pre-allocated files that had already been allocated:

Pre-allocate 3 files are:
File X: size 15B, content: ‘%’, start block: 10
File Y: size 10B, content: ‘^’, start block: 30
File Z: size 33B, content: ‘&’, start block: 66

Number of files to allocate: 3
File A size: 14B, content: ‘a’,
File B size: 30B, content: ‘b’,
File C size: 12B, content: ‘c’.

Shown below is the default disk space template, pre-allocated disk space layout and the final layout that was uploaded to Scriptcheck:

Disk space template:
0:0000000000000000000000000
25:0000000000000000000000000
50:0000000000000000000000000
75:0000000000000000000000000
100:0000000000000000000000000
125:0000000000000000000000000
150:0000000000000000000000000
175:0000000000000000000000000
200:0000000000000000000000000
225:0000000000000000000000000



Disc Layout:

```
0:  0000000000000000000000000
25: 0000000000000000000000000
50: 0000000000000000000000000
75: 0000000000000000000000000
100:0000000000000000000000000
125:0000000000000000000000000
150:0000000000000000000000000
175:0000000000000000000000000
200:0000000000000000000000000
225:0000000000000000000000000
```

Pre-allocated disk space layout:
0:   00 00 00 00 00 %% %% %% %% %% %% %% %

```
25: 00 00 0^ ^^ ^^ ^^ ^^ ^0 00 00 00 00 0
50: 00 00 00 00 00 00 00 00 && && && && &
75:  && && && && && && && && && && && && 0
100: 00 00 00 00 00 00 00 00 00 00 00 00 0
125: 00 00 00 00 00 00 00 00 00 00 00 00 0
150: 00 00 00 00 00 00 00 00 00 00 00 00 0
175: 00 00 00 00 00 00 00 00 00 00 00 00 0
200: 00 00 00 00 00 00 00 00 00 00 00 00 0
225: 00 00 00 00 00 00 00 00 00 00 00 00 0
```

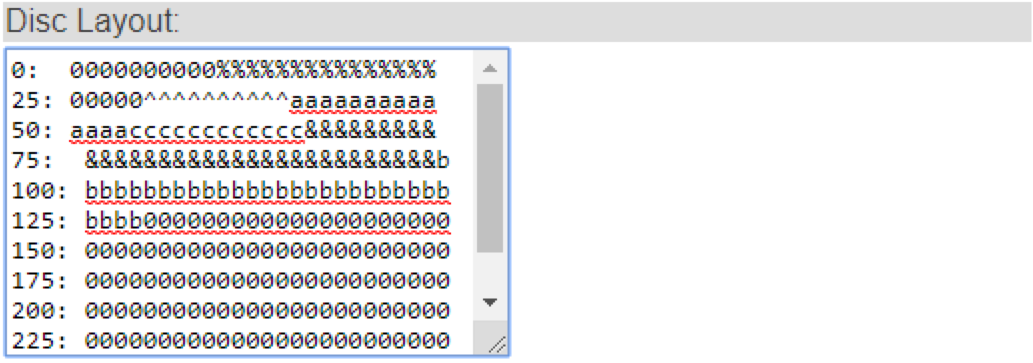
Final/Scriptcheck Layout:

```
0:  0000000000%%%%%%%%%%%%%%%
25: 00000^^^^^^^^^^^aaaaaaaaa
50: aaaacccccccccccc&&&&&&&&&
75:  &&&&&&&&&&&&&&&&&&&&&&&&b
100: bbbbbbbbbbbbbbbbbbbbbbbbb
125: bbbb000000000000000000000
150: 0000000000000000000000000
175: 0000000000000000000000000
200: 0000000000000000000000000
225: 0000000000000000000000000
```

Personal Reflection:

This task was on of the more enjoyable yet irritable ones, as the task at hand was not straight forward at first. We had two theories we it came to allocating the letters; My partner tried to replace the pre-allocated files with the new ones, while I tried to ‘integrate’ them with the new ones. The upload was a bit tricky because not matter what we tried, the last 3 lines of code would always be correct because the output was to be ‘0000000000000000000000000’. We then used the process of elimination and concluded that the way I was doing the task was correct, and we began integrating the files. A few checks on scriptcheck to confirm our answers and we were finished with the task. The report was pretty straight forward as we were given the terms to research. I am happy to have completed this task, and I feel like I have definitely furthered my understanding of Operating Systems because of it.

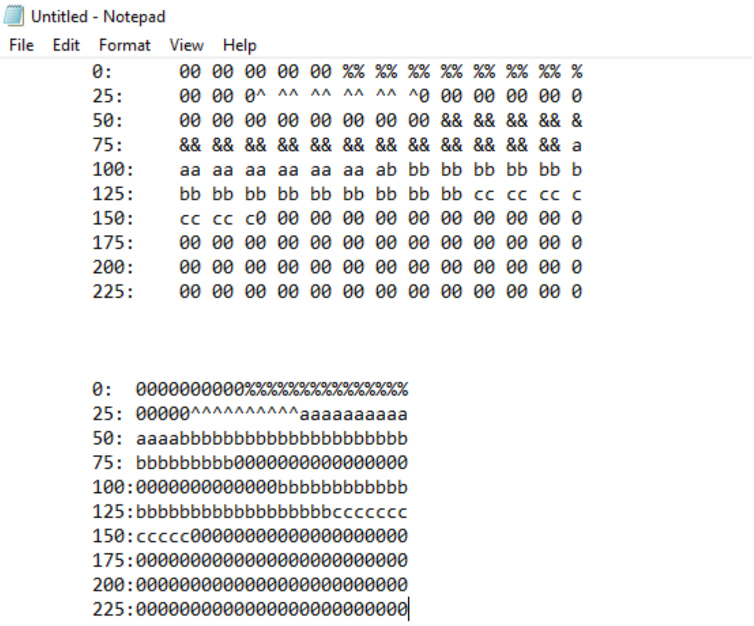
As usual, initial scriptcheck uploads were problematic as usual, but after we removed the the extra spaces between lines, the lines were able to function correctly. Editing the lines in notepad then transferring them helped:



Disc Layout:

```
0:   0000000000%%%%%%%%%%%%%%%
25:  00000^^^^^^^^^^aaaaaaaaaa
50:  aaaacccccccccccc&&&&&&&&&
75:   &&&&&&&&&&&&&&&&&&&&&&&&b
100: bbbbbbbbbbbbbbbbbbbbbbbbb
125: bbbb000000000000000000000
150: 0000000000000000000000000
175: 0000000000000000000000000
200: 0000000000000000000000000
225: 0000000000000000000000000
```

Typing the code in:



```
      0:        00 00 00 00 00 %% %% %% %% %% %% %% %
      25:       00 00 0^ ^^ ^^ ^^ ^^ ^0 00 00 00 00 0
      50:       00 00 00 00 00 00 00 00 && && && && &
      75:       && && && && && && && && && && && && a
      100:      aa aa aa aa aa aa ab bb bb bb bb bb b
      125:      bb bb bb bb bb bb bb bb bb cc cc cc c
      150:      cc cc c0 00 00 00 00 00 00 00 00 00 0
      175:      00 00 00 00 00 00 00 00 00 00 00 00 0
      200:      00 00 00 00 00 00 00 00 00 00 00 00 0
      225:      00 00 00 00 00 00 00 00 00 00 00 00 0


      0:  0000000000%%%%%%%%%%%%%%%
      25: 00000^^^^^^^^^^aaaaaaaaaa
      50: aaaabbbbbbbbbbbbbbbbbbbbb
      75: bbbbbbbbb0000000000000000
      100:0000000000000bbbbbbbbbbbb
      125:bbbbbbbbbbbbbbbbbbccccccc
      150:ccccc00000000000000000000
      175:0000000000000000000000000
      200:0000000000000000000000000
      225:0000000000000000000000000
```

```
0:  0000000000%%%%%%%%%%%%%%%
25: 00000^^^^^^^^^^aaaaaaaaaa
50: aaaacccccccccccc&&&&&&&&&
75: &&&&&&&&&&&&&&&&&&&&&&&&b
100:bbbbbbbbbbbbbbbbbbbbbbbbb
125:bbbb000000000000000000000
150:0000000000000000000000000
175:0000000000000000000000000
200:0000000000000000000000000
225:0000000000000000000000000
```