

LOGBOOK

Operating System



GROUP 21

USMAN BASHARAT - 000874782
UNIVERSITY OF GREENWICH

Groups are as followed:

Name	Surname	ID
Usman	Basharat	000874782
Mohamed	Aden	000874775
Yunus	Hassan	000880204
Derek	Mafohla	000743945

Coursework Header Sheet

234999-11



UNIVERSITY
of
GREENWICH

Course COMP1562: Operating Systems
Coursework COMp1562 Logbook
Tutor M Pelc

Course School/Level H/UG
Assessment Weight 40.00%
Submission Deadline 18/03/2017

Logbook based on weekly reports

Coursework is receipted on the understanding that it is the student's own work and that it has not, in whole or part, been presented elsewhere for assessment. Where material has been used from other sources it has been properly acknowledged in accordance with the University's Regulations regarding Cheating and Plagiarism.

000874782



Tutor's comments

Grade Awarded _____ *For Office Use Only* _____ *Final Grade* _____
Moderation required: yes/no *Tutor* _____ *Date* _____

Table of Contents

Groups are as followed:	2
Introduction	6
Task 1.1	6
Task 1.2	8
Reflection	9
Exercise 1	10
Exercise 2	11
Exercise 3	13
Task 2.1	13
Reflection	14
Example 1	15
Example 2	15
Example 3	16
Example 4	16
Task 3.1	17
Task 3.2	18
Task 3.3	18
Reflection	19
Task 4.1	20
Task 4.2	21
Reflection	23
Exercise One	24
First Come First Serve	24
Round Robin	25
Shortest Job First	27
Task 5.1	29
First Comes First Served	29
Gant Chart	29
Average Arrival Time	29
Average Turnaround Time	29
Task 5.2	30
Shortest Job	30
Average waiting time:	30
Average Turnaround Time	30

Gant Chart.....	30
Reflection	32
Task 6.1	33
First Fit	33
Best Fit	33
Worst Fit.....	33
Next Fit.....	33
Task 6.2	33
First Fit	33
Best Fit	33
Next Fit.....	33
Worst Fit.....	33
Reflection	34
Report	35
Fscck (Linux) vs Scandisk (Windows)	35
Mkfs (Linux) VS Format (Windows)	36
Logical Volume Manager (Linux) vs Logical Disk Manager (Windows).....	37
S.M.A.R.T. utility.....	37
Task 7.1	38
Reflection	39
Task 8	40
Reflection	43
Conclusion.....	44
References	45

Introduction

In this report, they are a number of tasks that are explained with an explanation of what is done with each task. I am going to do the following tasks processing programs, shell programming, scheduling, memory placement algorithms, file systems and selected aspects of Linux system administration.

Task 1.1

1.1. The hypothetical 16 bit processor in the PowerPoint slides also has two I/O instructions:

0011 = Load AC from

I/O 0101 = Add to AC (add data from the given memory location)

0111 = Store AC to I/O

Your system contains additionally two I/O device buffers at addresses 005h and 006h. Instructions should be placed in the memory locations 300h, 301h and 302h. Memory locations to store / get data are 900h, 901h and 902h. Assume initial values for I/O buffers 005h and 006h to be, respectively, 0001h and 0001h and initial values in the memory locations to be, respectively, 0100h, 0010h and 0001h. All registers values (PC, accumulator and IR, accordingly to the diagram for pseudo code execution) should be updated accordingly. There are three steps to be processed (each of the steps comprises of fetch and execute part):

- Load AC from the device buffer at address 005h.
- Add contents of memory location 901h.
- Store AC to device buffer at address 006h.

Step 1:		Step 2:	
Memory:	CPU Registers:	Memory:	CPU Registers:
300: 3005	PC: 300	300: 3005	PC: 301
301: 5901	AC:	301: 5901	AC: 0001
302: 7006	IR: 3005	302: 7006	IR: 3005
900: 0100		900: 0100	
901: 0010		901: 0010	
902: 0001		902: 0001	
I/O:		I/O:	
005: 0001		005: 0001	
006: 0001		006: 0001	

During this, we start by layout each stage of the process. This is used by putting in the correct values in this stage. Stage one consists of the program counter being started by using the first memory location and instruction register is enabling this in the instruction register. Accumulator is empty, because no instruction has been stored yet. Stage 2 is the part where it stores the instruction register by storing the memory by adding 1 to this into the accumulator. The program counter for stage 2 goes up to one because one instruction has been executed. This would be the same for the next stages to come too.

Step 3:		Step 4:	
Memory:	CPU Registers:	Memory:	CPU Registers:
300: 3005	PC: 301	300: 3005	PC: 302
301: 5901	AC: 0001	301: 5901	AC: 0011
302: 7006	IR: 5901	302: 7006	IR: 5901
900: 0100		900: 0100	
901: 0010		901: 0010	
902: 0001		902: 0001	
I/O:		I/O:	
005: 0001		005: 0001	
006: 0001		006: 0001	

Stage 3 is the next part of the process. Stage 3 and 4 consists of storing the next memory location which is 5901. This is done the same as Stage 2 and 3. Adding 5901 to the instruction register and for stage 4, is again, adding 1 to the instruction to know that this has been done. The program counter for stage 4 goes up to one because one instruction has been executed. Stage 3 is the same, because the accumulator has not gone one up to indicate that the instruction has not yet been stored.

Step 5:		Step 6:	
Memory:	CPU Registers:	Memory:	CPU Registers:
300: 3005	PC: 302	300: 3005	PC: 303
301: 5901	AC: 0011	301: 5901	AC: 0011
302: 7006	IR: 7006	302: 7006	IR: 7006
900: 0100		900: 0100	
901: 0010		901: 0010	
902: 0001		902: 0001	
I/O:		I/O:	
005: 0001		005: 0001	
006: 0001		006: 0011	

Stage 5 is the same as Stage 3. This means that the next instruction, which in this case, is 302 that is stored in the instruction register. However, for stage 6, it does not accumulate, because it needs to store the accumulator into the output, which is stored in 006. At this stage, it realises that everything is complete, and it stores it into the output. The only similarity between stage 6 and 4 is the program counter going up again.

Task 1.2

1.2. Consider a machine with:

- 1k words cache, access time 5ns.
- 1M words memory, access time 70ns.
- If the data is not in cache then the data is copied from memory.

Calculate (H is cache success ratio):

- Cache miss time.
- data access time for H=75%
- data access time for H=85%
- data access time for H=95%

Task 1.2 - Memory Access Time Calculations

Calculate memory access time for the three given H ratios. Enter the results respectively.

Cache miss time in nanoseconds:

75

Memory access time in nanoseconds for H=75%:

22.5

Memory access time in nanoseconds for H=85%:

15.5

Memory access time in nanoseconds for H=95%:

8.5

[V] Variable [S4M900] is correct!
[V] Variable [S3M901] is correct!
[V] Variable [S4M901] is correct!
[V] Variable [S3M902] is correct!
[V] Variable [S4M902] is correct!
[V] Variable [S3IO005] is correct!
[V] Variable [S4IO005] is correct!
[V] Variable [S3IO006] is correct!
[V] Variable [S4IO006] is correct!
[V] Variable [S5M300] is correct!
[V] Variable [S5PC] is correct!
[V] Variable [S6M300] is correct!
[V] Variable [S6PC] is correct!
[V] Variable [S5M301] is correct!
[V] Variable [S5AC] is correct!
[V] Variable [S6M301] is correct!
[V] Variable [S6AC] is correct!
[V] Variable [S5M302] is correct!
[V] Variable [S5IR] is correct!
[V] Variable [S6M302] is correct!
[V] Variable [S6IR] is correct!
[V] Variable [S5M900] is correct!
[V] Variable [S6M900] is correct!
[V] Variable [S5M901] is correct!
[V] Variable [S6M901] is correct!
[V] Variable [S5M902] is correct!
[V] Variable [S6M902] is correct!
[V] Variable [S5IO005] is correct!
[V] Variable [S6IO005] is correct!
[V] Variable [S5IO006] is correct!
[V] Variable [S6IO006] is correct!
[V] Variable [CacheMissTime] is correct!
[V] Variable [MAT75] is correct!
[V] Variable [MAT85] is correct!
[V] Variable [MAT95] is correct!

Group [21] score for task[1]: [100.000000%]

Cache miss time is calculated by adding the cache access time, which is 5ns, and the memory access time, which is 70ns. By adding these two together, we get 75nanoseconds which is the final answer for the cache miss time. Memory access time for 75% is calculated by $0.75 \times 5 + 0.25 \times (70 + 5) = 22.5$. Memory access time for 85% is calculated by $0.85 \times 5 + 0.15 \times (70 + 5) = 15.5$. Memory access time for 95% is calculated by $0.95 \times 5 + 0.05 \times (70 + 5) = 8.5$.

Figure 1 shows the result we received for this lab which is 100%.

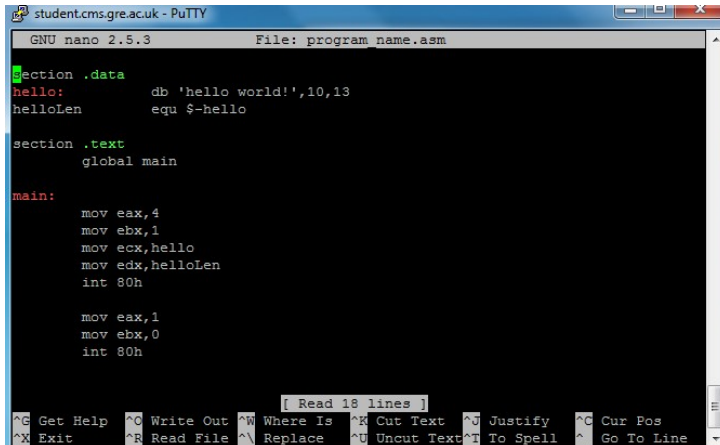
Figure 1 shows the result of these tasks

Reflection

During the tasks for this laboratory, I found these tasks pretty easy as soon as I found out the figures of Task 1.1. I felt that by using the lecture notes from the previous week was a huge step stone to the answers that I have currently got from them. Task 1.2 was easy too, because an example of the calculations were given in the lecture notes. I found that by using the examples on the lecture notes for both of the examples was huge. Looking back at these tasks, I felt that this has given me knowledge of how registers go amount with communicating with each other of storing values within the specific locations and how to calculate the cache time. Overall, I felt that I have understood the basic understanding and importance of how to answer the given calculations and what needs to be loaded in each step of a 16 bit processor.

Exercise 1

During this task, we had to follow each step to print out the code that was given to give us, "Hello World!" The first step we had to do is start up putty and login through the server stulinux.cms.gre.ac.uk. We could use the option of nano, pico or vim, so I decided to use nano for the rest of the tasks. Referring to Figure 1, this is where the code put in to execute by following the next that were steps given. Referring to Figure 2, this shows the results of when exercise one was executed. Referring to Figure 3 and 4, they both show the relevant screenshots for this exercise.



```
student.cms.gre.ac.uk - PuTTY
GNU nano 2.5.3      File: program_name.asm

section .data
hello:              db 'hello world!',10,13
helloLen            equ $-hello


section .text
global main

main:
    mov eax,4
    mov ebx,1
    mov ecx,hello
    mov edx,helloLen
    int 80h

    mov eax,1
    mov ebx,0
    int 80h

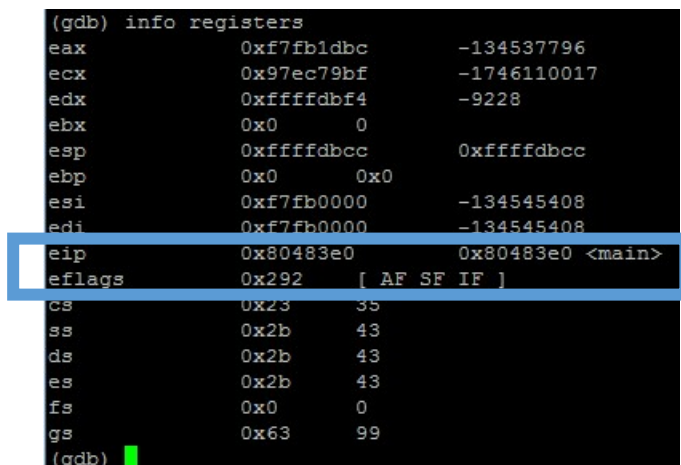
Read 18 lines
Get Help  Write Out  Where Is  Cut Text  Justify  Cur Pos
Exit      Read File  Replace  Uncut Text  To Spell  Go To Line
```

Figure 1 shows the place of the code where it was written



```
student:~> ./program_name
hello world!
student:~>
```

Figure 2 shows the code being execute and the result of it



```
(gdb) info registers
eax             0xf7fb1dbc      -134537796
ecx             0x97ec79bf      -1746110017
edx             0xffffdbf4      -9228
ebx             0x0             0
esp             0xffffdbcc      0xffffdbcc
ebp             0x0             0x0
esi             0xf7fb0000      -134545408
edi             0xf7fb0000      -134545408
eip             0x80483e0        0x80483e0 <main>
eflags          0x292          [ AF SF IF ]
cs              0x23           35
ss              0x2b           43
ds              0x2b           43
es              0x2b           43
fs              0x0            0
gs              0x63           99
(gdb)
```

Figure 3 shows the EIP and EFLAGS registers

These two screenshots show two different outcomes of the code that is in. Figure 3 shows all the registers being displayed for it and what is stored in them for this code.

Figure 4, which is on the next page, shows the next instruction being implemented. As you can see in the code, eax register holds a value of four. After the next instruction is implemented, it shows the value is in place in hexadecimal

```
(gdb) print/x $esp
$1 = 0xffffdbcc
(gdb)
```

```
(gdb) print/x $eax
$2 = 0xf7fb1dbc
(gdb)
```

```
(gdb) nexti
0x080483e5 in main ()
```

```
(gdb) print/x $esp
$3 = 0xffffdbcc
(gdb)
```

```
(gdb) print/x $eax
$4 = 0x4
(gdb)
```

Figure 4 shows the registers before and after the NEXTI instruction

Exercise 2

This exercise was the same as exercise one. However, the difference was that two numbers was being added together instead of being printed out hello world. Therefore, the same steps were followed and the screenshots below was the proof of the outcome of the following code that was given. Figure 6 shows us the code being executed.

```
section .data
NUMBER1:      dw 2
NUMBER2:      dw 4

section .text
global main

main:
    mov ax,[NUMBER1]
    add ax,[NUMBER2]
    mov [NUMBER2],ax
    mov eax,1
    mov ebx,0
    int 80h
```

[Read 14 lines]

Get Help Write Out Where Is Cut Text Justify Cur Pos
Exit Read File Replace Uncut Text To Spell Go To Line

Figure 5 shows exercise two code being put in.

```
student:~> chmod +x do_asm
student:~> ./do_asm program_two
student:~> ./program_two
student:~>
```

Figure 6 shows the result of exercise two

```
(gdb) info registers
eax            0xf7fb1dbc      -134537796
ecx            0x2a18fd48      706280776
edx            0xffffdbf4      -9228
ebx            0x0            0
esp            0xffffdbcc      0xffffdbcc
ebp            0x0            0x0
esi            0xf7fb0000      -134545408
edi            0xf7fb0000      -134545408
eip            0x80483e0        0x80483e0 <main>
eflags         0x292          [ AF SF IF ]
cs             0x23            35
ss             0x2b            43
ds             0x2b            43
es             0x2b            43
fs             0x0            0
gs             0x63            99
(gdb)
```

Figure 7 shows the EIP and EFLAGS registers for exercise two

These two screenshots show two different outcomes of the code that is in. Figure 7 shows all the registers being displayed for it and what is stored in them.

Figure 8 shows what the specific register is stored before and after the NEXTI instruction is being executed. This means that after it being executed it should hold a different value. I assumed that \$eax should hold the value of 0x6 and I entered the same code displayed in the lab document. However, the outcome was completely different.

```
(gdb) print/x $esp
$1 = 0xffffdbcc
(gdb) print/x $eax
$2 = 0xf7fb1dbc
(gdb) nexti
0x080483e6 in main ()
(gdb) print/x $esp
$3 = 0xffffdbcc
(gdb) print/x $eax
$4 = 0xf7fb0002
(gdb) next
Single stepping until exit from function main,
which has no line number information.
[Inferior 1 (process 29267) exited normally]
(gdb) info stack
No stack.
```

Figure 8 shows the NEXTI registers before and after it is executed for exercise two.

Exercise 3

For this task, we were asked to assemble the code and debug it to show the correct answer. The code asked us to divide number 2 by number 1, which equals to 2. For this, to be able to be correct, the code being put in needs to be correct in order to do this. Therefore, Figure 9 shows that the code being used to divide the two numbers together. Figure 10 shows us that once we debugged the steps above, it showed us the correct result.

```
section .data
NUMBER1:      dw 2
NUMBER2:      dw 4
section .text
globl main
main:
    mov ax, [NUMBER1]
    mov bl, [NUMBER2]
    div bl
    mov [NUMBER2], ax

    mov eax, 1
    mov ebx, 0
    int 80h
```

Figure 9 shows the assembled code used to divide the two numbers together

```
$3 = 0x2
(gdb)
```

Figure 10 shows us the result of it

Task 2.1

This one was similar to exercise three; however, it was more complex. This asked us to complete an equation of numbers being added in. We were asked to solve the equation below by representing the following numbers as a = 3, b = 7 and h = 4. The answer is 20, which was our aim.

$$result = \frac{a + b}{2} \times h$$

```
section .data
a:      dw 3
b:      dw 7
h:      dw 4
result:  dw 1

section .text
global main

main:
    mov ax, [a]
    add ax, [b]
    mov [b], ax

    mov ax, [b]
    mov bl, 2
    div bl
    mov [result], ax

    mov ax, [h]
    mov bx, [result]
    mul bx
    mov [result], ax
    mov word[result], ax
```

Figure 11 shows the assembled code of the equation above

Figure 11 shows us the assembled code being put in. I felt that the previous three exercises helped for this exercise being complete. The adding part above was done by using the exercise one.

The second part and the final part of the code helped by the example being given in the document. I implemented this in our example and I checked this code and it worked.

Task [2] results for group [21]

--- Marking results ---

[V] Compilation was successful!

Execution results: [00020]

[V] Execution results correct!

Group [21] score for task[2]: [100.000000%]

Your current score [100.000000%] is group's best [0.000000%]. Your result is saved as group's.

Figure 12 shows us that the score given for this task.

Reflection

After completing these tasks, I found that it was not as simple as it seemed for the above tasks. I found that each tasks were hugely important as the others. For example, when I did Task 2.1, I went back to previous exercises that I did before to help me add a+b. An example was given in the laboratory about multiplying 5^2 ; therefore, I used both of these examples to implement the last task. These challenges took time and effort to be completed and I felt that with my group members, we were able to complete this task and get the result above. Overall, I felt that I have understood the basic understanding and importance of how to debug, run and produce the code effectively.

Example 1

```
#!/bin/bash
if [ $# -lt 2 ]
then
    echo "-----"
    echo -e "\tScript <$0> should be executed with at least 2 arguments"
    echo "-----"
else
    echo "-----"
    echo -e "\tScript <$0> executed with $# parameters: $@"
    echo "-----"
fi
```

Figure 1 shows the code for Example 1

```
student:~> ./example1 3 2 4
-----
-e \tScript <./example1> executed with 3 parameters: 3 2 4
-----
student:~> ./example1 3 2 2
-----
-e \tScript <./example1> executed with 3 parameters: 3 2 2
-----
```

Figure 2 shows the code being executed

This example shows us whenever the user types in 3 arguments. Referring to Figure 2, the result recognises this and displays the return of how many has been typed in and the result of this. This screenshot shows us how the code being executed. When this was ran at the start, there was errors at the start. I had to change the quotation marks for it to go yellow.

Example 2

```
student.cms.gre.ac.uk - PuTTY
GNU nano 2.5.3 File: example2.sh

#!/bin/bash
while [ -z "$arg1" ]           #will be repeated until a data will be entered
do
    read -p "Enter argument1: " arg1
done

while [ -z "$arg2" ]           #will be repeated until a data will be entered
do
    read -p "Enter argument2: " arg2
done

echo "You've entered: arg1=$arg1 and arg2=$arg2"
let "addition=arg1+arg2"
let "subtraction=arg1-arg2"
let "multiplication=arg1*arg2"
let "division=arg1/arg2"
let "power=arg1 ** arg2"
echo -e "results:\n"
echo "$arg1+$arg2=$addition"
echo "$arg1-$arg2=$subtraction"
echo "$arg1*$arg2=$multiplication"
echo "$arg1**$arg2=$power"
```

Figure 3 shows us the code for Example 2

```

student:~> ./example2
student:~> ./example2.sh
Enter argument1: 4
Enter argument2: 2
You've entered: arg1=4 and arg2=2
-e results:\n
4+2=6
4-2=2
4*2=8
4**2=16

```

Figure 4 shows us the code being executed

This example shows whenever we type in two numbers, it does the calculations for us in one list. As you can see in Figure 4, it clearly shows us the results of 2 arguments being entered. Figure 3 is the whole code of Figure 3 being executed. When the original code was run, there was errors for the reminder. Therefore, this was replaced with the power of two numbers. When this was ran at the start, there was errors at the start. I had to change the quotation marks for it to go yellow.

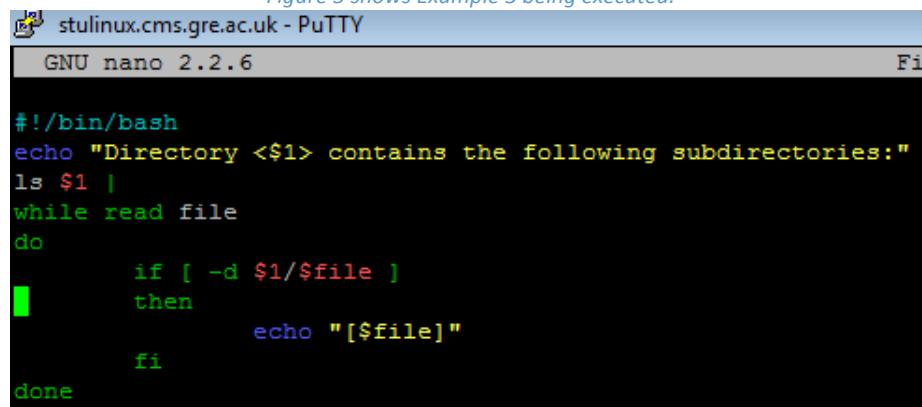
Example 3

```

student:~> ./exercise3 do_asm
Directory <do_asm> contains the following subdirectories:
student:~>

```

Figure 5 shows Example 3 being executed.



```

#!/bin/bash
echo "Directory <$1> contains the following subdirectories:"
ls $1 |
while read file
do
    if [ -d $1/$file ]
    then
        echo "[$file]"
    fi
done

```

Figure 6 shows the code for Example 3

Example 4

```

#!/bin/bash
echo "Directory <$1> contains the following filenames of odd size:"
ls -l $1 |
while read file_parm
do
    size=`echo $file_parm | cut -f 5 -d " "`
    name=`echo $file_parm | cut -f 9 -d " "`
    let "div=size%2"
    if [ ! -d $name ]
    then
        if [ $div -ne 0 ]
        then
            echo "[$name : $size]"
        fi
    fi
done

```

Figure 7 shows Example 4 code


```

Suspended (signal)
student:~> ./example4
Directory <> contains the following filenames of odd size:
[do_asm.save : 51]
[example1.asm : 531]
[example.1.save : 1]
[example1.save.1 : 107]
[example1.save.1.save : 107]
[example1.save.2 : 407]
[example1.save.3 : 335]
[example1.save.4 : 223]

```

Figure 8 shows the code being executed.

Example 3 was showing us a file and its subdirectory. Example 4 showed us the list of files and its sizes. This was later implemented to be used for Task 3.3. All of the examples were great and a start ahead of the later tasks that were complete. This helped us and gave us a good idea of how to start and correct all the errors that came up at the start. Figure 8 is a list. The list is too long. Therefore, I put in the first few and showing the example being executed.

Task 3.1

```

#!/bin/bash
let "addition=$1+$2"
let "subtraction=$1-$2"
let "multiplication=$1*$2"
let "division=$1/$2"

if [ $# -eq 3 ] && [ $3 = "+" ]; then
echo "$addition"

elif [ $# -eq 3 ] && [ $3 = "-" ]; then
echo "$subtraction"

elif [ $# -eq 3 ] && [ $3 = "/" ]; then
echo "$division"

elif [ "$3" = "*" ]; then
echo "$multiplication"

elif [ $# -eq 3 ] && [ $3 = "^" ]; then
echo "$1^$2" | bc
fi

```

Figure 9 shows Task 3.1 code

Task 3.1 was complete by modifying Example 2. As I was doing this, I realised that I needed to put if statements in for each one to execute if it is not the right one. However, I realised that after I did the first few, the multiplication and powering the two numbers was the one that was wrong. After an intense amount of researching the reason, I realised that I needed to put test the multiplication another way to see if the code is correct. For powering, I researched a way of using the bc command. This manually by using a library does the calculation together for both of the arguments. This works and I found this way easier than the loop.

```

--- Marking Script [task3.1.sh] ---
Result of [Addition] operation: [6]
[V] Result of [Addition] operation correct!
Result of [Subtraction] operation: [0]
[V] Result of [Subtraction] operation correct!
Result of [Multiplication] operation: [9]
[V] Result of [Multiplication] operation correct!
Result of [Division] operation: [1]
[V] Result of [Division] operation correct!
Result of [Power] operation: [27]
[V] Result of [Power] operation correct!

```

Figure 10 shows that this code is correct through Script Check.

```

student:~> ./task3.1.sh 4 2 ^
16

```

Task 3.2

```
#!/bin/bash
ls -l $1 | (
while read file_parm
do
    size=`echo $file_parm | cut -f 5 -d " "`
    name=`echo $file_parm | cut -f 9 -d " "`
    let "div=size%2"
    if [ "$2" == "odd" ]
    then
        if [ ! -d $name ]
        then
            if [ $div -ne 0 ]
            then
                let "total=total+size"
            fi
        elif [ ! -d $name ]
        then
            let "div=size%2"
            if [ "$2" == "even" ]
            then
                if [ $div -eq 0 ]
                then
                    let "total=total+size"
                fi
            fi
        fi
    fi
done
echo $total
)
```

Figure 11 shows the code for Task 3.2

```
--- Marking Script [task3.2.sh] ---
[V] Size (odd) calculated correctly!
[V] Size (even) calculated correctly!
```

Figure 12 shows that this code is correct through Script Check.

Referring to Figure 11, Task 3.1 shows us the total of all of the files of the choice of directory and if it is odd or even. For this to be done, I used if statements. This is done by the odd being size being divided by 2 and the remainder being recognised by it being odd if it is not equal. This is the same for even. Whilst the if statement is going for both even and odd, it adds both of them together. Once this is complete, "echo \$total" displays the total number. The same process is done for both even and odd.

Figure 12 shows the result of the code running in through script check.

```
student:~> ./task3.2.sh ~/Desktop odd
1532
student:~> ./task3.2.sh ~/Desktop even
1450
```

Task 3.3

```
#!/bin/bash
if [ -d "$@" ]; then
    echo "$(find "$@" -type f | wc -l)"
fi
```

Figure 13 shows the code for Task 3.3

```
--- Marking Script [task3.3.sh] ---
[V] Number of files calculated correctly!
```

Figure 14 shows that this code is correct through Script Check.

```
student:~> ./task3.3.sh ~/Desktop
3
```

Figure 13 shows the code of how it was implemented. This code finds how many files are there in the example and displays it as a number. As you can see, I showed an example of this on my Desktop drive and it says 3. Figure 14 shows that this is correct.

--- Marking results ---

The file task3.1.sh has been uploaded.

The file task3.2.sh has been uploaded.

The file task3.3.sh has been uploaded.

--- Marking Script [task3.1.sh] ---

Result of [Addition] operation: [6]

[V] Result of [Addition] operation correct!

Result of [Subtraction] operation: [0]

[V] Result of [Subtraction] operation correct!

Result of [Multiplication] operation: [9]

[V] Result of [Multiplication] operation correct!

Result of [Division] operation: [1]

[V] Result of [Division] operation correct!

Result of [Power] operation: [27]

[V] Result of [Power] operation correct!

--- Marking Script [task3.2.sh] ---

[V] Size (odd) calculated correctly!

[V] Size (even) calculated correctly!

--- Marking Script [task3.3.sh] ---

[V] Number of files calculated correctly!

Group [21] score for task[3]: [100.000000%]

Your current score [100.000000%] is group's best [75.000000%]. Your result is saved as group's.

Reflection

After completing this laboratory, it gave us different challenges along the way. I felt that it gave us different knowledge each time we started doing another task. However, it was a laboratory that we did together after continuous problems occurred, we solved the problems. During this time and effort I spent on this, I felt that I did much research on all the examples for me to understand the each task. I felt that the examples helped very much, because it gave me a platform to work on. The one I struggled on was the first one the most. I felt that I spent much time on this the most, because it was the most challenging one out of them all. These challenges took time and effort to be completed and I felt that with my group members, we were able to complete this task and get the result above. Overall, I felt that I have understood the basic understanding and importance of how to run and produce the code effectively.

Task 4.1

```
GNU nano 2.5.3 File: task4.1.sh
find . -xtype l | xargs rm
```

Figure 10 shows the code for Task 4.1

Figure 4.1 shows the code to execute broken symbolic links. As soon as the piping method is used to search for the broken links within the directory, it removes it. As I was doing this, I made a mistake of searching for symbolic links, and removing them. After reading the laboratory document, I realised it is the complete opposite. Therefore, when I ran the line of code through script check, it worked and I moved on to the next task. The screenshots below are the following of how this lab has been executed. The first movements were to create a directory and make an empty file. Once this was created, we need to create the link. Once the link was created, we make it broken by removing it and then we tested our code to remove it and it worked shown on Figure 6

```
student:~> mkdir tests
student:~> dir
answer.save    example5.save    questions.save  task3.2
answer.sh      lab4link         questions.sh    task3.2.sh
answer.txt     nano.save        questions.txt   task3.2.sh.save
ccs            noninteractive.sh quiz2.sh        task3.3
contains       noninteractive.txt quiz.sh         task3.3.sh
Desktop        program_name     quiz.sh.save   task4.1.sh
dicwords.txt   program_name.asm quiz.sh.save.1 task4.1.sh.save
do_asm         program_name.asm.save quiz.sh.save.2 task4.2.sh.save
do_asm.save    program_name.lst quiz.txt        task4.2.sh.save.1
documents      program_name.o   task3.1         tests
Documents      public_html      task3.1.sh
student:~> ls -l
total 57
```

Figure 11 shows making a new directory

```
student:~> cd tests
student:~/tests> nano lab4
```

Figure 12 shows creating a new file

```
student:~/tests> ln -s lab4 lab3link
```

Figure 13 shows creating a symbolic link between the new file created

```
student:~/tests> dir
lab3link
```

Figure 14 shows the broken link once the new file has been created

```
student:~/tests> cd
student:~> ./task4.1.sh .
student:~> cd tests
student:~/tests> dir
student:~/tests> ls -l
total 0
```

Figure 15 shows the symbolic link removed

Task 4.2

```
#!/bin/bash
right=0
wrong=0
question=10
i=0
if [ $# -eq 2 ]
then
for((i=0; i<question;)) {
i=$((i + 1))
y=$i
echo "$(awk "NR==$y" $1)"
read -p "Enter Answer: " arg1
result=$(awk "NR==$y" $2)

if [ "$arg1" = "$result" ]
then
right=$((right + 1))
else
wrong=$((wrong + 1))
fi
}
echo "Number_of_correct_answers " $right
echo "Number_of_wrong_answers " $wrong
fi
```

```
if [ $# -eq 3 ]
then
for((i=0; i<question;)) {
i=$((i + 1))
y=$i
ans=$(awk "NR==$y" $2)
resultfile=$(awk "NR==$y" $3)

if [ "$resultfile" = "$ans" ]
then
right=$((right + 1))
else
wrong=$((wrong + 1))
fi
}
echo $right
fi
```

The code is split between in two. This works by the first section highlighted in blue is recognised by the interactive mode of the quiz. This means that the user has to type in the answer they feel and they would receive the result. Once the user has typed in the result, this is matched by the correct answers that has been typed in. Each answer that is wrong, \$wrong is added one and the same with \$right.

However, the other section highlighted in green is the non-interactive part. This part shows that there are non-interactive answers that have been already typed in shown in Figure 9 and is compared to the actual answers shown in Figure 8. This only displays the correct answer. Once it executed, it matches the two and it displays how many answers are correct.

How the code is run for both is similar. The first loop runs 10 times, matching how many questions there are. Each time it runs, it uses the arguments to display the question and to match the answer. This is a clever loop, because once it is executed, it matches the arguments and selects which one is needed to execute.

Figure 10 and Figure 11 shows the difference between the two codes.

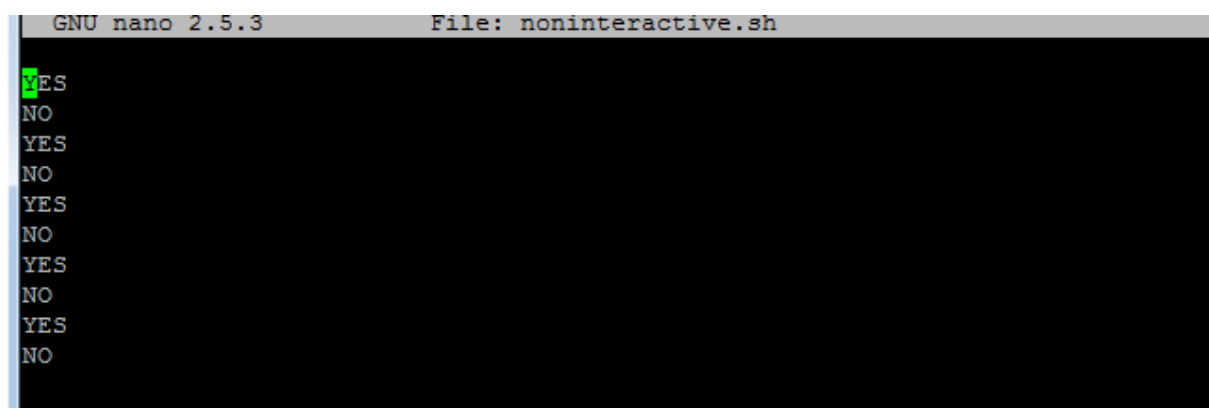
```
GNU nano 2.5.3                               File: questions..
Can Humans fly?
Is it 2018?
Is orange juice yellow?
Is Game of Thrones better than Walking Dead??
Have you got a phone?
Are you a student?
Do you speak English?
Is it raining today?
The opposite of yes, it is?
Is the sky blue?
```

Figure 16 shows the questions used for this task



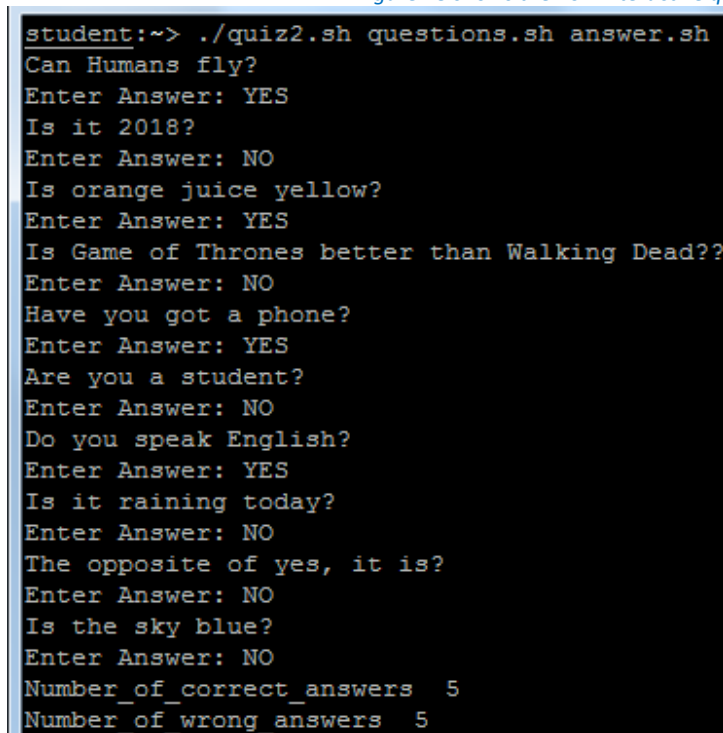
```
student.cms.gre.ac.uk - PuTTY
GNU nano 2.5.3 File: answer.sh
NO
NO
NO
YES
YES
YES
YES
NO
NO
YES
```

Figure 17 shows the answers to the questions for this task



```
GNU nano 2.5.3 File: noninteractive.sh
YES
NO
YES
NO
YES
NO
YES
NO
YES
NO
```

Figure 18 shows the non-interactive questions used.



```
student:~> ./quiz2.sh questions.sh answer.sh
Can Humans fly?
Enter Answer: YES
Is it 2018?
Enter Answer: NO
Is orange juice yellow?
Enter Answer: YES
Is Game of Thrones better than Walking Dead??
Enter Answer: NO
Have you got a phone?
Enter Answer: YES
Are you a student?
Enter Answer: NO
Do you speak English?
Enter Answer: YES
Is it raining today?
Enter Answer: NO
The opposite of yes, it is?
Enter Answer: NO
Is the sky blue?
Enter Answer: NO
Number_of_correct_answers 5
Number_of_wrong_answers 5
```

Figure 19 shows the interactive quiz.

```
student:~> ./quiz2.sh questions.sh answer.sh noninteractive.sh
4
```

Figure 20 shows the non-interactive quiz

Task [4] results for group [21]

--- Marking results ---

The file task4.1.sh has been uploaded.

The file quiz2.sh has been uploaded.

The file questions.sh has been uploaded.

The file answer.sh has been uploaded.

--- Marking Script [task4.1.sh] ---

[V] Your script [task4.1.sh] worked correctly!

--- Marking Script [quiz2.sh] ---

Test result: [3]

Verifying your script [quiz2.sh]:

Q[1]: X

Q[2]: V

Q[3]: X

Q[4]: X

Q[5]: X

Q[6]: X

Q[7]: V

Q[8]: X

Q[9]: V

Q[10]: X

Verification result: Your script [quiz2.sh] correctly marked [3] out of 10 answers!

Group [21] score for task[4]: [100.000000%]

Your current score [100.000000%] is group's best [40.000000%]. Your result is saved as group's.

Reflection

During the time for this laboratory, I found this task tough again. I felt that the first one was not as tough as the second one, because we had to link three files together. It was hard to put the non-interactive part of the quiz in, but when I understood the first part of how I did it, it was easier to understand the non-interactive part. I felt that I spent much time on this the most, because it was the most challenging one out of them all. These challenges took time and effort to be completed and I felt that with my group members, we were able to complete this task and get the result above. Overall, I felt that I have understood the basic understanding and importance of how to run and produce the code effectively.

Exercise One

First Come First Serve

Process 1 30ms

Process 2 30ms

Scheduling Algorithms - Introductory

Process Configuration

	CPU Intense	Balanced	IO Intense	Runtime (Milliseconds)
Process 1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	30
Process 2	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	30

System Configuration

Number of Processors: 1

Quantum size: 10 Milliseconds

IO device latency (constant): 15 Milliseconds

Priority: Fixed, all processes equal

Turn OFF Icon Flashing

Runtime State Display

RUN

READY

BLOCKED

COMPLETED

Scheduler Configuration

☐ Round Robin

☒ First Come First Served

☐ Shortest Job First

Animation Control

Simulation speed

Fastest Slowest

System Statistics

Elapsed Time (Milliseconds)

60

Balanced processes perform IO every 15 ms of Runtime

IO-intensive processes perform IO every 7 ms of Runtime

Runtime Statistics (Milliseconds)

	Runtime Used	Runtime Remaining	Wait time	IO Time (Blocked)	Total time in system
Process 1	30	0	0	0	30
Process 2	30	0	30	0	60

Free Run

Pause

Reset Simulation

Done

Process 1 300ms

Process 2 30ms

Scheduling Algorithms - Introductory

Process Configuration

	CPU Intense	Balanced	IO Intense	Runtime (Milliseconds)
Process 1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	300
Process 2	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	30

System Configuration

Number of Processors: 1

Quantum size: 10 Milliseconds

IO device latency (constant): 15 Milliseconds

Priority: Fixed, all processes equal

Turn OFF Icon Flashing

Runtime State Display

RUN

READY

BLOCKED

COMPLETED

Scheduler Configuration

☐ Round Robin

☒ First Come First Served

☐ Shortest Job First

Animation Control

Simulation speed

Fastest Slowest

System Statistics

Elapsed Time (Milliseconds)

330

Balanced processes perform IO every 15 ms of Runtime

IO-intensive processes perform IO every 7 ms of Runtime

Runtime Statistics (Milliseconds)

	Runtime Used	Runtime Remaining	Wait time	IO Time (Blocked)	Total time in system
Process 1	300	0	0	0	300
Process 2	30	0	300	0	330

Free Run

Pause

Reset Simulation

Done

Process 1 30ms

Process 2 300ms

Scheduling Algorithms - Introductory

Process Configuration

	CPU Intense	Balanced	IO Intense	Runtime (Milliseconds)
Process 1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	30
<input checked="" type="checkbox"/> Process 2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	300

System Configuration

Number of Processors: 1


Quantum size: 10 Milliseconds

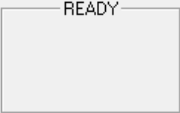
IO device latency (constant): 15 Milliseconds


Priority: Fixed, all processes equal


Turn OFF Icon Flashing

Runtime State Display

RUN: 

READY: 

BLOCKED: 

COMPLETED: 

Scheduler Configuration

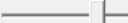
☐ Round Robin

☒ First Come First Served

☐ Shortest Job First

Animation Control

Simulation speed

Fastest  Slowest

System Statistics

Elapsed Time (Milliseconds)

330

Balanced processes perform IO every 15 ms of Runtime

IO-intensive processes perform IO every 7 ms of Runtime

Runtime Statistics (Milliseconds)

	Runtime Used	Runtime Remaining	Wait time	IO Time (Blocked)	Total time in system
Process 1	30	0	0	0	30
Process 2	300	0	30	0	330

Free Run

Pause

Reset Simulation

Done

Round Robin

Process 1 30ms

Process 2 30ms

Scheduling Algorithms - Introductory

Process Configuration

	CPU Intense	Balanced	IO Intense	Runtime (Milliseconds)
Process 1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	30
<input checked="" type="checkbox"/> Process 2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	30

System Configuration

Number of Processors: 1


Quantum size: 10 Milliseconds


IO device latency (constant): 15 Milliseconds


Priority: Fixed, all processes equal


Turn OFF Icon Flashing

Runtime State Display

RUN: 

READY: 

BLOCKED: 

COMPLETED:  

Scheduler Configuration


☒ Round Robin

☐ First Come First Served

☐ Shortest Job First

Animation Control

Simulation speed

Fastest  Slowest

System Statistics

Elapsed Time (Milliseconds)

60

Balanced processes perform IO every 15 ms of Runtime

IO-intensive processes perform IO every 7 ms of Runtime

Runtime Statistics (Milliseconds)

	Runtime Used	Runtime Remaining	Wait time	IO Time (Blocked)	Total time in system
Process 1	30	0	20	0	50
Process 2	30	0	30	0	60

Free Run

Pause

Reset Simulation

Done

Process 1 300ms

Process 2 30ms

Scheduling Algorithms - Introductory

Process Configuration

	CPU Intense	Balanced	IO Intense	Runtime (Milliseconds)
Process 1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	300
Process 2	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	30

System Configuration

Number of Processors: 1
Quantum size: 10 Milliseconds
IO device latency (constant): 15 Milliseconds
Priority: Fixed, all processes equal

Turn OFF Icon Flashing

Runtime State Display

RUN

READY

BLOCKED

COMPLETED

Scheduler Configuration

☒ Round Robin
☐ First Come First Served
☐ Shortest Job First

Animation Control

Simulation speed

Fastest | Slowest

System Statistics

Elapsed Time (Milliseconds)

330

Balanced processes perform IO every 15 ms of Runtime
IO-intensive processes perform IO every 7 ms of Runtime

Runtime Statistics (Milliseconds)

	Runtime Used	Runtime Remaining	Wait time	IO Time (Blocked)	Total time in system
Process 1	300	0	30	0	330
Process 2	30	0	30	0	60

Free Run | Pause | Reset Simulation | Done

Process 1 30ms

Process 2 300ms

Scheduling Algorithms - Introductory

Process Configuration

	CPU Intense	Balanced	IO Intense	Runtime (Milliseconds)
Process 1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	30
Process 2	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	300

System Configuration

Number of Processors: 1
Quantum size: 10 Milliseconds
IO device latency (constant): 15 Milliseconds
Priority: Fixed, all processes equal

Turn OFF Icon Flashing

Runtime State Display

RUN

READY

BLOCKED

COMPLETED

Scheduler Configuration

☒ Round Robin
☐ First Come First Served
☐ Shortest Job First

Animation Control

Simulation speed

Fastest | Slowest

System Statistics

Elapsed Time (Milliseconds)

330

Balanced processes perform IO every 15 ms of Runtime
IO-intensive processes perform IO every 7 ms of Runtime

Runtime Statistics (Milliseconds)

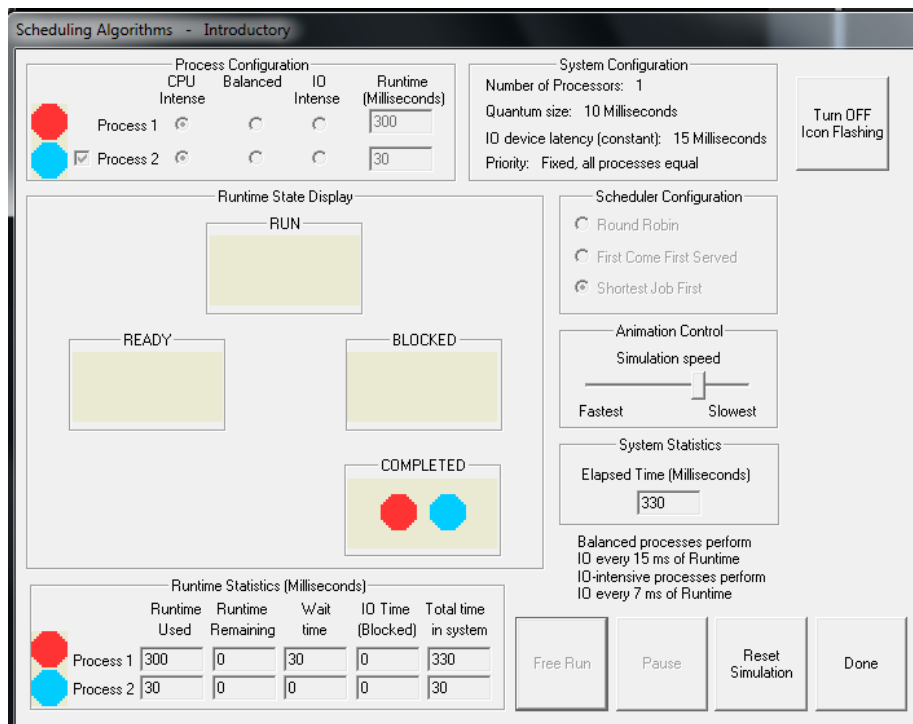
	Runtime Used	Runtime Remaining	Wait time	IO Time (Blocked)	Total time in system
Process 1	30	0	20	0	50
Process 2	300	0	30	0	330

Free Run | Pause | Reset Simulation | Done

Shortest Job First

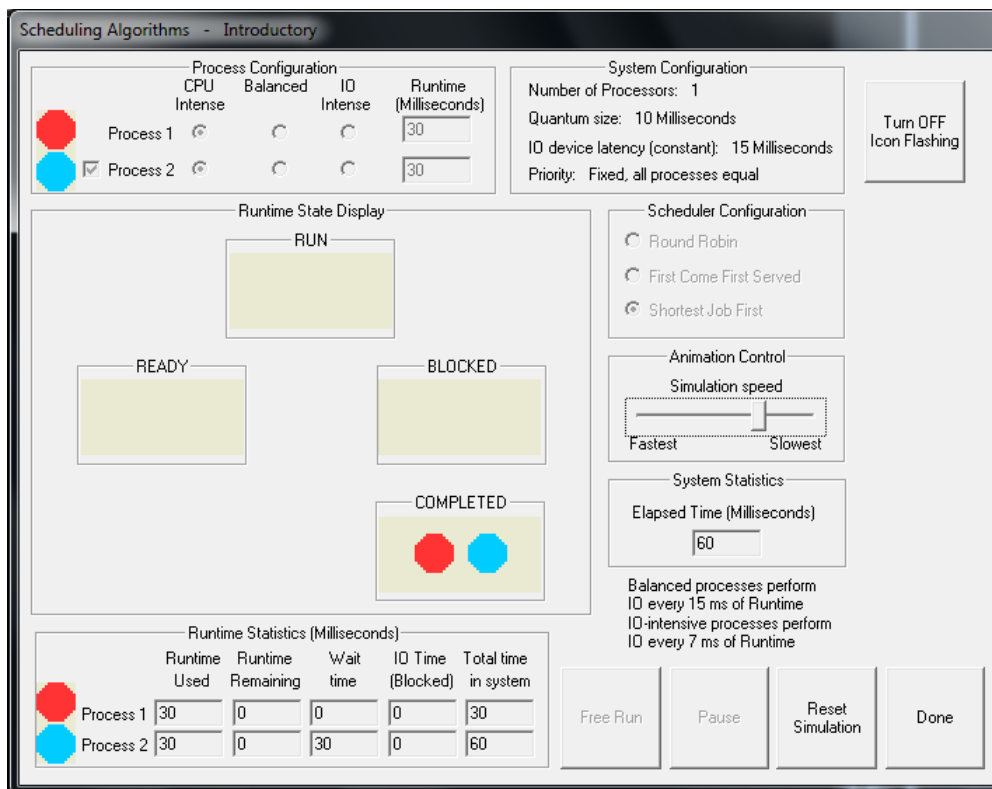
Process 1 300ms

Process 2 30ms



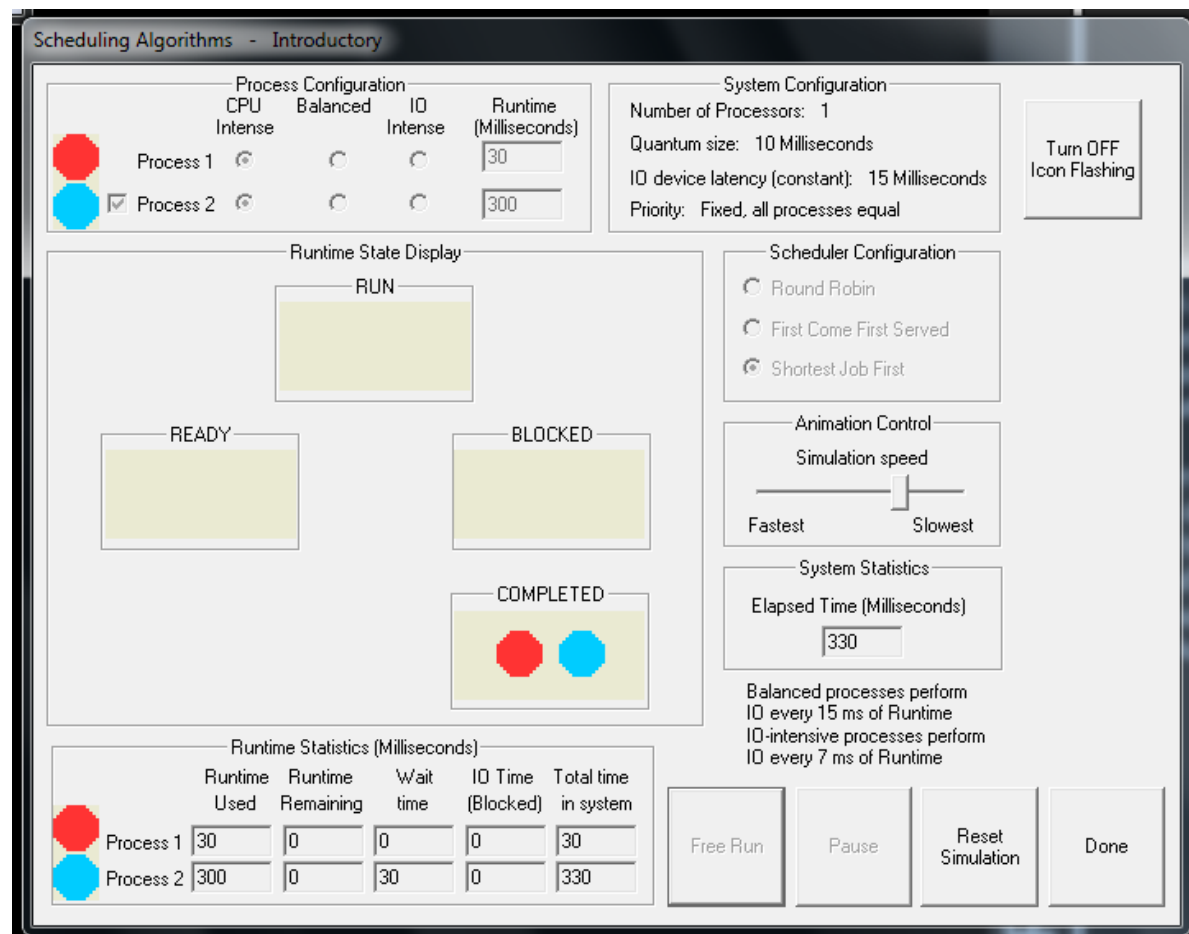
Process 1 30ms

Process 2 30ms



Process 1 30ms

Process 2 300ms



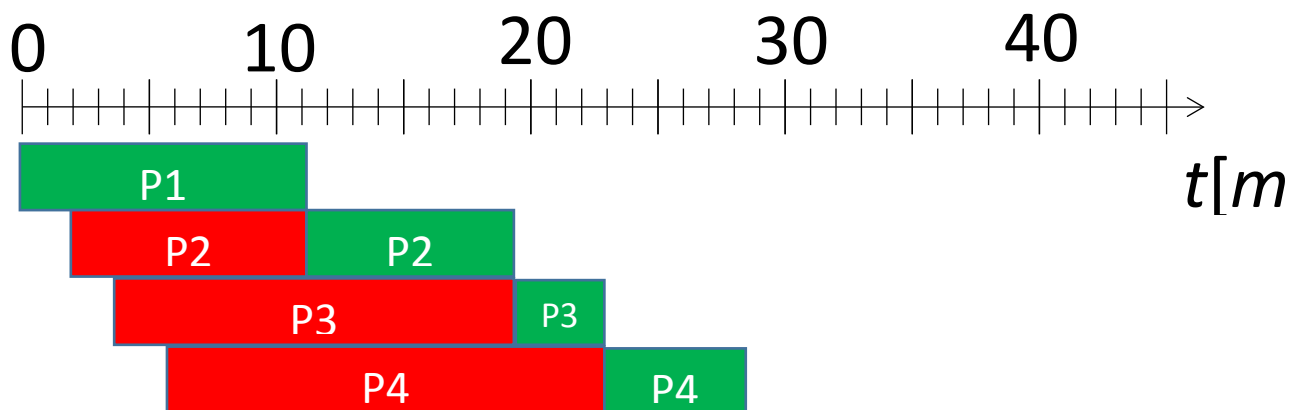
As you can see, the above screenshots are from the Operating System Workbench. Three different algorithms that need to be simulating. They are round robin, first come first served and shortest job first. Each time was monitored for each algorithm. Each algorithms acts differently with different runtimes. For example, as you can see for the screenshots above, shortest job first has a wait time that totals up time for process two. Round robin has a wait time for each process. Different algorithms have different waiting times for each process to be complete. The difference is the waiting time for each of them.

Task 5.1

First Comes First Served.

Process	Arrival Time [ms]	Service Time [ms]
P1	0	11
P2	2	8
P3	4	4
P4	6	6

Gant Chart



Average Arrival Time

$$= 0 + (11-2) + (19-4) + (23-6)$$

$$= 41$$

$$= 41/4$$

$$= 10.25$$

Average Turnaround Time

$$= 11 + 17 + 19 + 23$$

$$= 70$$

$$= 70/4$$

$$= 17.5$$

Process 1 rrrrrrrr
 Process 2 --wwwrrrrrrr
 Process 3 ----wwwrrrrr
 Process 4 -----wwwrrrrrr

Shortest Job

0-11	0
21-29	2
11-15	4
15-21	6

[illegible]

0 10 20 30 40 $t[ms]$

P1 P3 P2 P2 P3 P4 P4



UNIVERSITY
of
GREENWICH

Task 1

Task 2

Task 3

Task 4

Task 5

Task 6

Task 7

Task 8

Feedback

21

Please enter each line separately for each process. Use "-" to indicate that a process is not running yet, "w" to indicate a process is waiting, "r" to indicate the process is running.

Process 1: rrrrrrrrrr

```
Process 2: --wwwwwwwwwrrrrrrrr
```

Process 3: ----wwwwwwwwwwwwwwwrrrr

```
Process 4: -----wwwwwwwwwwwwwwwwwwwwwrrrrrr
```

Average Waiting Time in milliseconds:

10.25

Average Turnaround Time in milliseconds:

17.5

- Task 2
- Task 3
- Task 4
- Task 5
- Task 6
- Task 7
- Task 8
- Feedback

Gantt Chart:

Process 1:

```
Process 2: --wwwwwwwwwwwwwwwwwwwwwwwwwwwrrrrrrrr
```

```
Process 3: ----wwwwwwrrrr
```

```
Process 4: -----wwwwwwrrrrrr
```

Average Waiting Time in milliseconds:

8.75

Average Turnaround Time in milliseconds:

16

Task [5] results for group [21]

--- Marking results ---

[M] [P11] value is correct !
[M] [P12] value is correct !
[M] [P13] value is correct !
[M] [P14] value is correct !
[M] [T11] value is correct !
[M] [T12] value is correct !
[M] [P21] value is correct !
[M] [P22] value is correct !
[M] [P23] value is correct !
[M] [P24] value is correct !
[M] [T21] value is correct !
[M] [T22] value is correct !

Group [21] score for task[5]: [100.000000%]

Your current score [100.000000%] is group's best [75.000000%]. Your result is saved as group's.

Reflection

Overall, I felt that this laboratory, it was better than some of the last previous ones that we have completed recently. Even so, we need to take time and effort into any laboratory, because it was the most challenging one out of them all. These challenges took time and effort to be completed and I felt that with my group members, we were able to complete this task and get the result above. Overall, I felt that I have understood the basic understanding and importance of how to run and produce the answers effectively.

Task 6.1

First Fit

P1-M3,P2-M2,P3-M4,P4-M1,P5-M5

Best Fit

P1-M3,P2-M2,P3-M5,P4-M1,P5-M4

Worst Fit

P1-M3,P2-M4,P3-M5,P4-M2,P5-M1

Next Fit

P1-M3,P2-M4,P3-M5,P4-M1,P5-M2

Task 6.2

First Fit

P1-M3,P2-M4,P3-M1,P4-M2,P5-M5

Best Fit

P1-M3,P2-M5,P3-M1,P4-M2,P5-M4

Next Fit

P1-M3,P2-M4,P3-M5,P4-M2,P5-M1

Worst Fit

P1-M3,P2-M4,P3-M5,P4-M2,P5-M1

21

Task 6.1 - Memory Placement Algorithms - Scenario 1

Please enter solutions for each placement algorithm separately. Data should be entered accordingly to the lab instruction as a comma separated sequence stating process number and the memory gap the process will be placed into (for example P1-M1,P2-M4,...).

Memory Placement Graph:

Best Fit:

Worst Fit:

First Fit:

Next Fit:

Task 6.2 - Memory Placement Algorithms - Scenario 2

Please enter solutions for each placement algorithm separately. Data should be entered accordingly to the lab instruction as a comma separated sequence stating process number and the memory gap the process will be placed into (for example P1-M1,P2-M4,...).

Memory Placement Graph:

Best Fit:

Worst Fit:

First Fit:

Next Fit:

Task [6] results for group [21]

--- Marking results ---

[V] [A11] is correct!

[V] [A12] is correct!

[V] [A13] is correct!

[V] [A14] is correct!

[V] [A21] is correct!

[V] [A22] is correct!

[V] [A23] is correct!

[V] [A24] is correct!

Group [21] score for task[6]: [100.000000%]

Your current [100.000000%] score is NOT group's best [100.000000%]. Your result will NOT be saved.

Reflection

During this laboratory, we had to understand how memory allocations works. I felt that this was interesting and how memory allocation works gave me a better understanding. We were given 4 examples. The four was best fit, worst fit, first fit, next fit. We had two tasks that we needed to complete during this laboratory. As I was going through this, I felt that each of them were allocated differently. The easy of them four were best and worst. However, the only one I got quite stuck on was the next fit. I thought that it was the next fit to fit the actual memory. However, it was how I placed it during it. As I went on to do further research, I realised that it started from the actual P3 onwards. Overall, I felt that I have understood the basic understanding and importance of how to allocate memory allocations.

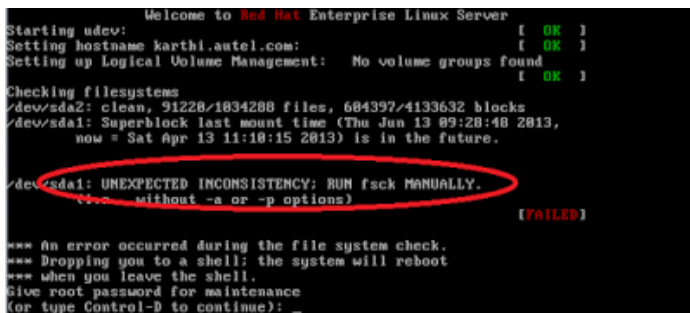
Report

Fsck (Linux) vs Scandisk (Windows)

FSCK is a file system checking tool that consistently is used to check and repairs any files within the Linux system. Typically, fsck is automatically run as soon as the operating system is booted. However, the user has an option to run this as a command if the system user feels that the user believes that there is an issue with the files on the system. This would initiate a list of errors within the disk and give the user an option to choose which the best option is for the issue. For example, this could be if any corrupted files should be removed, the user has the decision to make. This is the interactive mode. The non-interactive mode does check for all problems it tries to find in the file system, without user response, however it has a disadvantage of removing any corrupted files it detects along the way.

Scandisk is a diagnostic utility tool on Windows that checks for any for any errors on the hard drive of the system. Some of the files it checks for is log files, cross linked files, log file fragments, check files. Scandisk usually highlights any errors that it has found on the drive and it repairs those files that have been damaged initially. An additional feature that scandisk usually has it is summaries and it displays to the user which of the errors have been fixed and which are remaining. However, the disadvantage of Scandisk is it typically takes a huge amount of time depending on how big the drive is. The bigger the drive is, the longer it will take. When running Scandisk, it is usually safe to close and save any data to prevent any data loss during this process.

Comparing Scandisk to fsck, it could be agreed that scandisk has a lot more functionalities to fsck. It has and provides more features for the user. However, it does make the user wait for it to complete whilst fsck has a non-interactive mode where it does it automatically. Even though fsck can be efficient, as stated above, it can remove any corrupted files. Whilst scandisk is running, it visually and accurately shows the user what it has done so far even though it only scans for the Windows disk and files.



```
Welcome to Red Hat Enterprise Linux Server
Starting udev: [ OK ]
Setting hostname karthi.autel.com: [ OK ]
Setting up Logical Volume Management: No volume groups found [ OK ]
Checking filesystems
/dev/sda2: clean, 91228/1834288 files, 684397/4133632 blocks
/dev/sda1: Superblock last mount time (Thu Jun 13 09:28:48 2013,
now = Sat Apr 13 11:18:15 2013) is in the future.
/dev/sda1: UNEXPECTED INCONSISTENCY: RUN fsck MANUALLY.
(= without -a or -p options) [FAILED]

*** An error occurred during the file system check.
*** Dropping you to a shell; the system will reboot
*** when you leave the shell.
Give root password for maintenance
(or type Control-D to continue): _
```

Figure 21 shows the Linux version of fsck

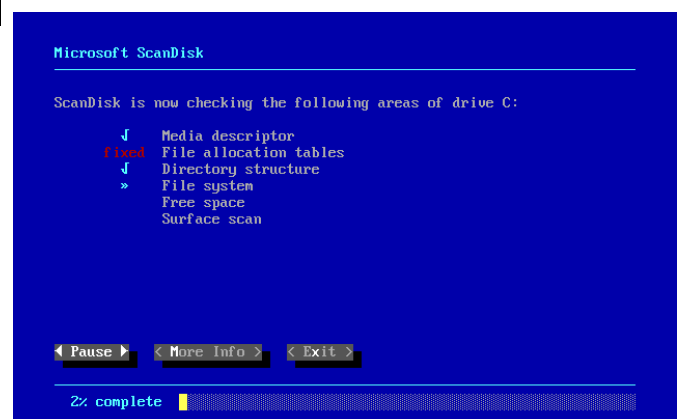


Figure 22 shows the Windows version of ScanDisk

Mkfs (Linux) VS Format (Windows)

In Linux, MKFS is a command that can be used to format a block storage device with a specific file system. This allows user to create partitions from spare free storage for different uses. In addition, features for the partition information is required from the user to specific which type of partition the user would like to require. Other information the user should be aware of is that mkfs checks for any bad blocks to create. Overall, mkfs is very complex and requires an ability to learn, but when the user has taught the user's skills to use mkfs, it can be quick and efficient when partitioning disks.

Format function on Windows can be done on different levels. Some reasons can to format a drive is to clean any unnecessary files or to get rid of viruses. However, on some occasions, when formatting a drive it does not get rid of the virus. Once getting rid of the current files, Windows would enable the partition of the selected drives. Formatting the disk is up to the administrator to choose if the hard disk is in need of partition and the administrator has an option to reformat the hard drive. However, reformatting the hard drive and reinstalling the operating system is time consuming.

Comparing the two on different formats is complex. This is because on Windows, you have to require knowledge and background on what you are doing. Whereas, on Linux, you have to require commands to format the drive. Figure 3 is shown the difference between the two of the two new drives created. Having said that they both require some sort of knowledge, Windows would be the easier option. Some users have their option, but having been a Windows user, I would prefer this option.

Device	Mount Point/ RAID/Volume	Type	Format	Size (MB)	Start	End
LINUX FILE SYSTEM						
Hard Drives						
/dev/sda						
/dev/sda1	/	ext3	✓	1027	1	131
/dev/sda2	/usr	ext3	✓	8001	132	1151
/dev/sda3		swap	✓	3498	1152	1597
/dev/sda4						
		Extended		7946	1598	2610
/dev/sda5	/disk3	ext3	✓	2000	1598	1852
/dev/sda6	/disk2	ext3	✓	2000	1853	2107
/dev/sda7	/disk1	ext3	✓	2000	2108	2362
/dev/sda8	/var	ext3	✓	996	2363	2489
/dev/sda9	/flash	ext3	✓	949	2490	2610
Hide RAID device/LVM Volume Group members						
WINDOWS FILE SYSTEM						
OS (C:) 231.77 GB NTFS Healthy (Boot, Page File, Crash)						
New Volume (D:) 64.03 GB NTFS Healthy (Logical Drive)						
New Volume (J:) 80.00 GB NTFS Healthy (Logical Drive)						
■ Primary partition ■ Extended partition ■ Free space ■ Logical drive						

Figure 23 showing the difference between Windows and Linux

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

Logical Volume Manager (LVM) is used in Linux that is a device mapper that allows the administrator to provide to the volume management. A device mapper is an interface that works by passing data from one device to another. LVM allows functions of not only adding, but replacing hard disk in order to manager the partition properly for each drive. The clever feature for LVM is that it does not disrupt any other service whilst performing its own task. Another feature that LVM allows is for it to have backup's method. Overall, the efficiency for LVM allows it to be efficient by making the job done.

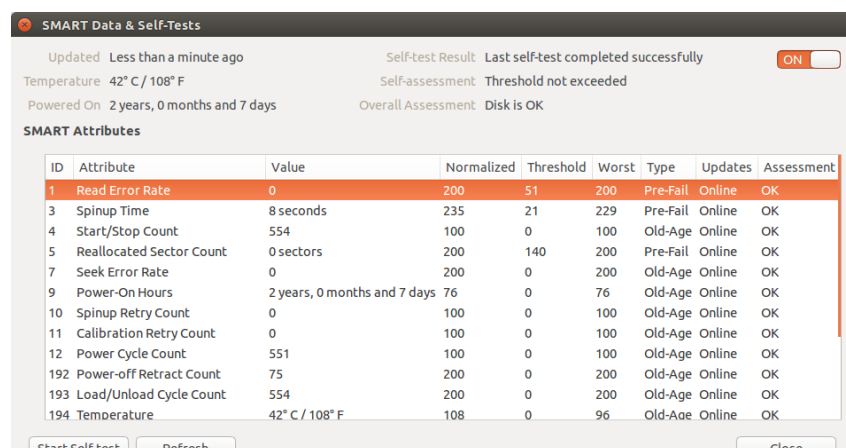
Logical Disk Manager (LDM) for Windows is a service that allows disk management to manage hard disk. The service is similar to Linux. The basic for LDM is to be involved in the partition of dividing disk and extending any if necessary. As same as LVM, LDM has a backup method that allows the data to be backed up on another drive on a reformatted disk drive to make sure any data that is lost, it can be retrieved. Another specification for this is that the basic disks can be upgraded to dynamic. However, if any administrator would like to go back to basic, it would be hard to do. It can be degraded, however, it would not be easily done.

Comparing the two services together is that they are huge differences between the two. LDM tends to be much simpler and easier for an administrator to use. The simplicity enables the results to be efficient and the administrator to understand this easier. This enables the user to navigate and make decisions quicker, enabling actions and jobs that need done easier and faster. However, LVM is much more of the opposite towards LDM, even though the services run similar. It requires more knowledge and depth as for the other services to know for formatting and partitioning a disk drive.

S.M.A.R.T. utility

S.M.A.R.T utility stands for Self-Monitoring, Analysis and Reporting Technology utility that is built in and it reports any problems the drive is having. For example, it would report a drive if it has any electrical, or mechanical problems and it can indicate it if it is failing to indicate to the administrator what steps he needs to do to either replace, or fix the problem. Another feature SMART utility has that differentiates it from others is that it allows it to be scanned in the background and have a SMART status that allows administrator to see what the current status is of the selected drive. SMART Utility is mostly used on MAC computers and this service is specifically designed for this operating system. This displays information such as the drive model, capacity, powers on hours, temperature, bad sector count, any errors and what type they are.

SMART Utility is unique and different from others because it has features to examine the device and storage and the ability to do self-tests enables the problem to be detected sooner. For example, Figure 4 shows an example of self-tests.



ID	Attribute	Value	Normalized	Threshold	Worst	Type	Updates	Assessment
1	Read Error Rate	0	200	51	200	Pre-Fail	Online	OK
3	Spinup Time	8 seconds	235	21	229	Pre-Fail	Online	OK
4	Start/Stop Count	554	100	0	100	Old-Age	Online	OK
5	Reallocated Sector Count	0 sectors	200	140	200	Pre-Fail	Online	OK
7	Seek Error Rate	0	200	0	200	Old-Age	Online	OK
9	Power-On Hours	2 years, 0 months and 7 days	76	0	76	Old-Age	Online	OK
10	Spinup Retry Count	0	100	0	100	Old-Age	Online	OK
11	Calibration Retry Count	0	100	0	100	Old-Age	Online	OK
12	Power Cycle Count	551	100	0	100	Old-Age	Online	OK
192	Power-off Retract Count	75	200	0	200	Old-Age	Online	OK
193	Load/Unload Cycle Count	554	200	0	200	Old-Age	Online	OK
194	Temperature	42° C / 108° F	108	0	96	Old-Age	Online	OK

Figure 24 shows the self-tests for SMART Utility

Task 7.1

```
0:0000000000%%%%%%%%%
25:00000^^^^^^^^^aaaaaaaa
50:aaaaccccccccccc&&&&&&&&
75:&&&&&&&&&&&&&&&&&&&b
100:bbbbbbbbbbbbbbbbbbbbbb
125:bbbb000000000000000000
150:0000000000000000000000
175:0000000000000000000000
200:0000000000000000000000
225:0000000000000000000000
```

--- Marking results ---

[V] Line [0:0000000000%%%%%%%%%] is correct!
[V] Line [25:00000^^^^^^^^^aaaaaaaa] is correct!
[V] Line [50:aaaaccccccccccc&&&&&&&&] is correct!
[V] Line [75:&&&&&&&&&&&&&&&&&&&b] is correct!
[V] Line [100:bbbbbbbbbbbbbbbbbbbbbb] is correct!
[V] Line [125:bbbb000000000000000000] is correct!
[V] Line [150:0000000000000000000000] is correct!
[V] Line [175:0000000000000000000000] is correct!
[V] Line [200:0000000000000000000000] is correct!
[V] Line [225:0000000000000000000000] is correct!

Group [21] score for task[7]: [100.000000%]

Your current score [100.000000%] is group's best [50.000000%]. Your result is saved as group's.

Task 7 - Files Allocation Algorithm

Please enter solutions accordingly to the lab instruction. Each line of the text area indicates as to which sectors of the hard drive are referred to. For each line enter sequence of characters represengint the disc contents. Disc area has been pre-filled in for you with zeros so you should change only these sectors where necessary.

Disc Layout:

```
0:0000000000%%%%%%%%%
25:00000^^^^^^^^^aaaaaaaa
50:aaaaccccccccccc&&&&&&&&
75:&&&&&&&&&&&&&&&&&&&b
100:bbbbbbbbbbbbbbbbbbbbbb
125:bbbb000000000000000000
150:0000000000000000000000
175:0000000000000000000000
200:0000000000000000000000
225:0000000000000000000000
```

Reflection

For this laboratory, I felt that it requires time and effort for this to be complete. During this laboratory, I felt that it gave me an interesting insight into how hard drive and disk management works with both operating system of Windows and Linux. I felt that this gave me a better understanding and developed more knowledge with the extensive research on both operating systems. For File Allocation Management, I felt that this gave me a rough understanding of content can be allocated in different positions. I felt that this laboratory gave me a broad understanding of two different topics into one laboratory.

Overall, I felt that Task 7.1 was the much easier to understand than the report. For the report, I felt that it took time and a lot of research to understand the importance of partitioning and formatting disk and SMART utility. Another issue, I felt that comparing the two together is another load of research that I felt I needed to do. Overall, I felt that I have understood the basic understanding and importance of file allocation algorithm.

Task 8

1. Which file you would use to redefine existing system variable **PATH** (provide full path)?
/etc/bashrc
2. How you would redefine the **PATH** variable to add path to **/home/john/bin** directory?
export PATH=\$PATH:/home/john/bin
3. Which file you would use to limit maximum number of logins for a user **mariusz** to 3 logins at a time (provide full path)?
/etc/security/limits.conf
4. How you would modify the above file to achieve the result (please show how the line you would add to the file would look like)?
mariusz - maxlogins 3
5. You were requested to add to the system new user called **tatiana**. You want to execute the command adding the user to the system so that default shell for **tatiana** was be **/bin/sh** and her initial / primary group was **students**. How the command would look like?
useradd -g students -s /bin/sh tatiana
6. One of the system users asked you to change his login name from **pm75** to **mariusz**. How you would achieve this (provide appropriate command)?
usermod -l mariusz pm75
7. You want to add user **mariusz** to ACL (Access Control List) of file **/usr/share/ccsm** so that this user had read permission (assume the user does NOT belong to the file group nor he is the file owner).
setfacl -m u:mariusz:r /usr/share/ccsm
8. You want to change current permissions for file **/usr/share/ccsm** to **rwxr--r--**. How the command allowing you to achieve the goal would look like.
chmod 744 /usr/share/ccsm
9. You want to check your file system supports ACL. Which command would tell you this?
ls -l
10. You want to create in the current directory symbolic link **ptr** to file **/usr/share/ccsm**. How the command allowing you to achieve this would look like?
ln -s /usr/share/ccsm ptr
11. One of the system users **clare** forgot her password. How the command initialising her password change would look like? Assume you execute the command as root (in other words: what command would root execute to change clare's password).
passwd clare
12. Which file you would modify to add a new DNS server (provide full path)?
/etc/resolv.conf
13. How would the entry (line) to the above file look like if the DNS server you wanted add to the system was **217.173.13.13**?
nameserver 217.173.13.13
14. You want to execute **/usr/share/ccsm** script **every Sunday at 2pm**. How the **crontab** entry allowing you to achieve this would look like?
0 14 * * 7 /usr/share/ccsm

15. As system admin (root) you copied **/usr/share/ccsm** file to the home directory of user **ben**. Assuming user ben has his home directory in the default location for CentOS system, show how you would make the file to be owned by **ben** and belong to system group **students**?

chown ben:students /usr/share/ccsm

16. You want immediately lock account for student **ben** (so that he was unable to login to the system) as it turned out he has fee hold status. How the command allowing you to achieve this would look like?

passwd -l ben

17. You want to check which process(es) running in the system is / are owned by user **ben**. How the command allowing you to achieve this may look like?

ps -u ben

18. Suppose you have just finished a completely fresh Linux installation. However, after first system boot it turned out that it by default takes you to the graphical user interface login prompt (meaning the system boots by default into runlevel 5). Which file defines default system runlevel (provide full path)?

/etc/inittab

19. How you would modify the above file (provide whole line) to change the default runlevel to **3**.

id:3:initdefault:

20. You want to change default location for the users home directory from **/home** to **/home/users**. Which file you would use to make it a default setting (provide full path)?

/etc/default/useradd

21. How the entry (line) to the above file would look like?

HOME=/home/users

22. You want your Linux server to request network settings for interface **eth0** from **DHCP** server. Which file you would use to force this (provide full path)?

/etc/network/interfaces

23. How the entry (line) in the above file would look like?

BOOTPROTO=dhcp

24. As normal / standard Linux system user **mariusz** you spotted that your default shell is Korn shell **/bin/ksh**. You definitely prefer BASH **/bin/bash** shell over Korn shell (**/bin/ksh**) so you want it to get changed **permanently**. However, you do not want to engage system admin to achieve this as you are perfectly capable to do it on your own. Which file you would modify (provide full path assuming default location for the user's home directory)?

~/ .bash_profile

25. How the line allowing you to change your default shell would look like.

export SHELL=/bin/bash

26. How would you interpret Heisenberg Uncertainty Principle in the context of wavelet analysis?

Heisenberg Uncertainty Principle is a principle that was introduced by Werner Heisenberg in 1927. The principle states that it is in an act of measuring a variable of particles. The principle states that the more precisely the position is determined, the less precisely the momentum is known in this instant and vice versa. The principle works the other way. Therefore, the more precise the momentum of the particle, the harder it is to predict.

In terms of wavelet, it is interpreted in terms of wave. Therefore, the position of the particle can be anywhere and the chance of finding the position as stated above, for the precision of the monument, can be exactly the same. The reason for this is because finding the probability of finding the particle like a wave format is impossible due to the particles being spread out.

Overall the uncertainty principle tells us that it is impossible for us to know how accurate the position is for the particle. The best we can do is roughly guess how it is, but to be certain, we do not know, hence the reason of it being called “uncertainty principle”.

Task [8] results for group [21]

--- Marking results ---

[M] Line [/etc/bashrc] is correct!
[M] Line [export PATH=\$PATH:/home/john/bin] is correct!
[M] Line [/etc/security/limits.conf] is correct!
[M] Line [mariusz - maxlogins 3] is correct!
[M] Line [useradd -g students -s /bin/sh tatiana] is correct!
[M] Line [usermod -l mariusz pm75] is correct!
[M] Line [setfacl -m u:mariusz:r /usr/share/ccsm] is correct!
[M] Line [chmod 744 /usr/share/ccsm] is correct!
[M] Line [ls -l] is correct!
[X] Line [ln -s /usr/share/ccsm ptr] is incorrect!
[M] Line [ln -s /usr/share/ccsm ptr] is correct!
[M] Line [passwd clare] is correct!
[M] Line [/etc/resolv.conf] is correct!
[M] Line [nameserver 217.173.13.13] is correct!
[X] Line [0 14 * * 7 /usr/share/ccsm] is incorrect!
[M] Line [0 14 * * 7 /usr/share/ccsm] is correct!
[M] Line [chown ben:students /usr/share/ccsm] is correct!
[M] Line [passwd -l ben] is correct!
[M] Line [ps -u ben] is correct!
[M] Line [/etc/inittab] is correct!
[M] Line [id:3:initdefault:] is correct!
[M] Line [/etc/default/useradd] is correct!
[M] Line [HOME=/home/users] is correct!
[X] Line [/etc/network/interfaces] is incorrect!
[M] Line [/etc/network/interfaces] is correct!
[M] Line [BOOTPROTO=dhcp] is correct!
[X] Line [~/.bash_profile] is incorrect!
[M] Line [~/.bash_profile] is correct!
[M] Line [export SHELL=/bin/bash] is correct!

Group [21] score for task[8]: [100.000000%]

Your current [100.000000%] score is NOT group's best [100.000000%]. Your result will NOT be saved.

Reflection

After completing this laboratory, I felt that it took a heavy amount of researching from all parts of the so it needed us to work as a group. During this time, I felt that there was too many questions for the research to take place. To make the process quicker, I felt that we needed to split the work to make it easier. Therefore, we split the questions into three. Aden did 6, I did 10 and Yunus did 10. However, any questions that that could not be answered, we all helped each other to understand and solve the question. This made it longer, however, by splitting the work between us made it easier. After completing this task, I felt that I this lab was essential in helping me develop even further of understanding different codes on the Linux system.

Conclusion

While carrying out this week's laboratory, I felt that I have implemented what I have learnt previously and gradually discovering what I have learnt to be put into practice by answering the questions above. I believe that this week's laboratory was essential gave me an opportunity to learn different aspects of Linux that I had not previously discovered. Overall, I felt that all these laboratories has given me a better understanding of each topic by learning as a group. I felt that I have picked up more information by learning from others than I would have done if I did this by myself. I felt that I have understood the basic understanding and importance of file allocation algorithm.

References

Linux FSCK, [Online] Available at: http://4.bp.blogspot.com/-F9yzz5t_Hw/VP10RDHMUul/AAAAAAAAA4Y/SHD_Bg-V3_A/s1600/fsck_linux.png
(Accessed 8 March 2017).

Windows, (2017) *Microsoft ScanDisk*, [Online] Available at:
[https://upload.wikimedia.org/wikipedia/en/6/63/Microsoft_Scandisk_\(Windows_98\).png](https://upload.wikimedia.org/wikipedia/en/6/63/Microsoft_Scandisk_(Windows_98).png)
(Accessed 8 March 2017).

Linux, Windows, (2017) *Linux File System and Windows File System Difference*, [Online] Available at:
<https://linuxexplore.com/2012/10/01/linux-file-system-and-windows-file-system-difference/>
(Accessed 8 March 2017).