## Lab 1 – Operating Systems Tasks and Programming:

## Basic Task:

Task: In one paragraph describe a possible future direction for operating systems (Windows , 2019).

Future OS will be unified and

(Microsoft, 2016)

## Advanced Task 1:

Write a simple program (or adapt the C++ program in Appendix A) to parse a string to determine if it fulfils one of the following grammar order rules. Below are rules related to the basic grammar. For example, in the grammar, a <Person Word> is followed by an <Action Word> and the <ActionWord>is follow by a <Thing Word>.

If an input does not fulfil a rule explain why.

Program screenshot with comments and unittesting included:

import unittest

def checkGrammar(w):#Function to check if word is correct in grammar

    #Pre-use checks for the string input

    s = w.lower()

    w = s.split()

    #Possible words in different arrays for different meaning.

    Person = ["baxter", "shila", "igor"]

    Position = ["left", "right", "forwards", "backwards"]

    Thing = ["screwdriver", "diamond", "person", "boat", "ball", "dog", "apple"]

    Action = ["recognise", "eat", "sees", "pick"]

    Pronoun = ["i", "you", "him", "we", "her"]

    Value = ["cheap", "expensive", "price-less"]

    #Making test cases for possible word combinations.

    test1 = ["perw", "aw", "tw"]

    test2 = ["perw", "aw", "vw", "tw"]

    test3 = ["perw", "aw", "pos"]

    test4 = ["pro", "aw", "tw"]

    test5 = ["pro", "aw", "pos"]

    #Test Cases

    if len(w) <=2:

        return False

    if len(w) >=5:

        return False

    elif w[0] not in Person:

        if w[0] not in Pronoun:

            print(w)

            print(Person)

            return False

    elif w[1] not in Action:

        return False

    elif len(w) <= 3:

        if w[2] not in Position:

            if w[2] not in Thing:

                return False

            else:

                return True

    elif len(w) == 4 and w[2] not in Value and w[3] not in Thing:

        print("fail 4")

        return False

    return True

class MyTest(unittest.TestCase):

    def test1(self):

        self.assertEqual(checkGrammar(s1), True)

    def test2(self):

        self.assertEqual(checkGrammar(s2), True)

    def test3(self):

        self.assertEqual(checkGrammar(s3), False)

    def test4(self):

        self.assertEqual(checkGrammar(s4), False)

    def test5(self):

        self.assertEqual(checkGrammar(s5), True)

    def test6(self):

        self.assertEqual(checkGrammar(s6), True)

    def test7(self):

        self.assertEqual(checkGrammar(s7), False)

    def test8(self):

        self.assertEqual(checkGrammar(s8), False)

if \_\_name\_\_ == '\_\_main\_\_':

    s1 = "Baxter sees boat"

    s2 = "I pick diamond"

    s3 = "Baxter Baxter"

    s4 = "Boat goes backwards"

    s5 = "We eat apple"

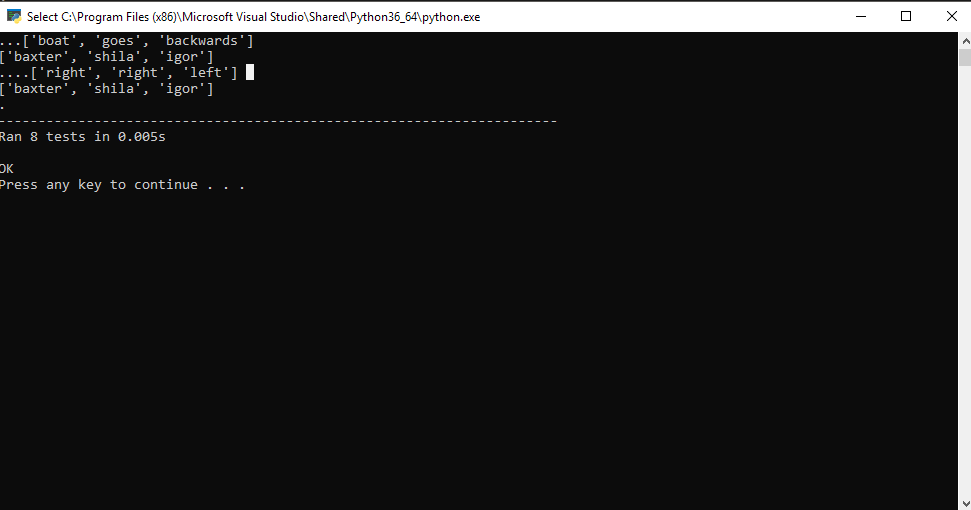
    s6 = "Shila recognise cheap ball "

    s7 = "Eat ball"

    s8 = "Right right left"

    unittest.main(exit=False)

Program compiled – with unit-testing in action:



# Lab 2 – Linux Command-Line:

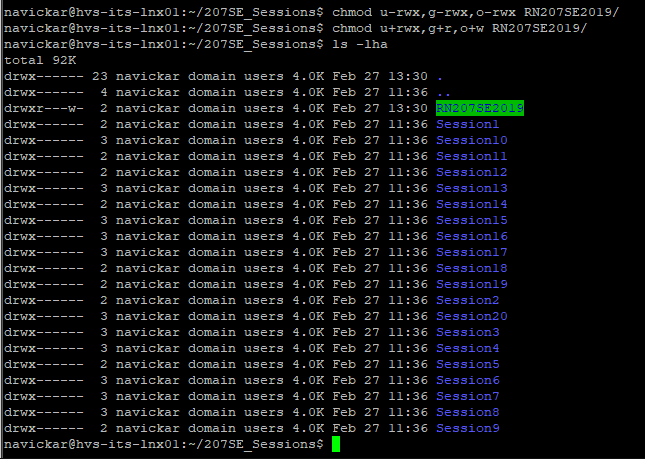
## File manipulation:

A – command used to create a directory was:

*mkdir ~/207SE\_Sessions/RN207SE2019*(Please see question B for a screenshot)

To change the permissions I have used chmod as per in picture, at first I had reset all permissions to all users so I can easier set the pattern of permissions. I found it easier way then deciding what permissions to add and what to take away.

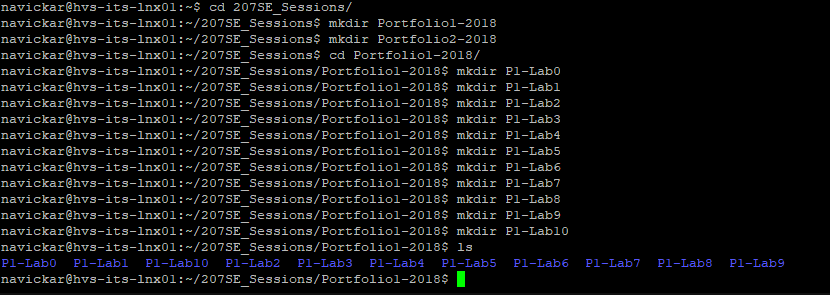
B - *ls –ha* was used to display permissions for different groups of users for different files.



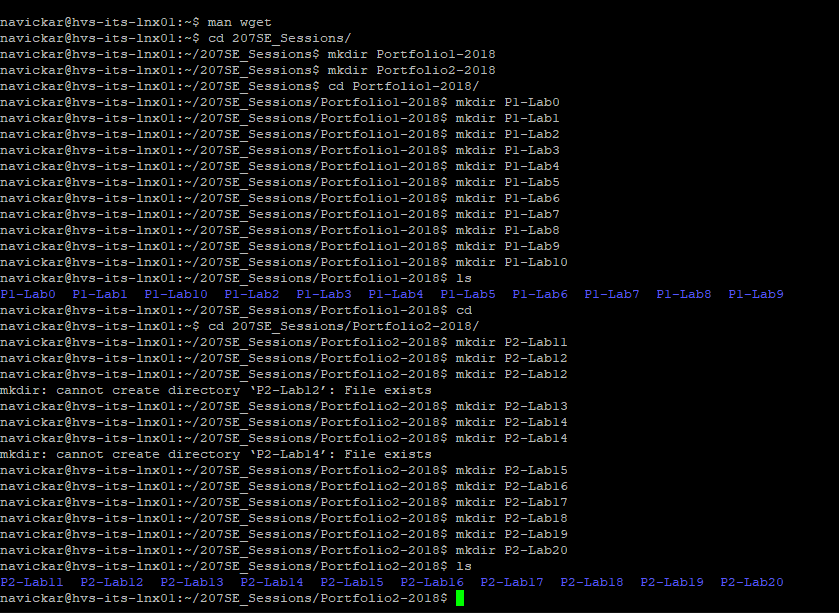
C - Sctipt wget was downloaded from the website using :  
wget “URL”And used wget to check how it works.

D – Portfolio1-2018 and Portfolio2-2018 directories were created in the directory /RN207SE2019

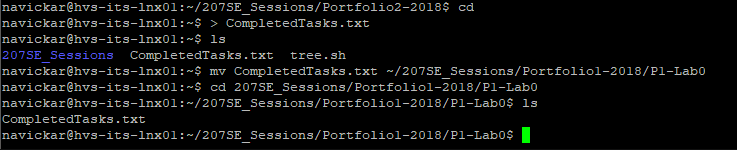
Using mkdir.



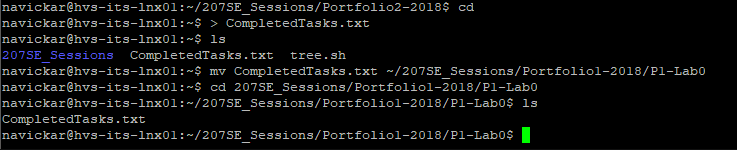
E – As shown in previous picture, different directories were created in Portfolio1 and Portfolio2 directories as required.



F – CompletedTasks.txt was created in home directory

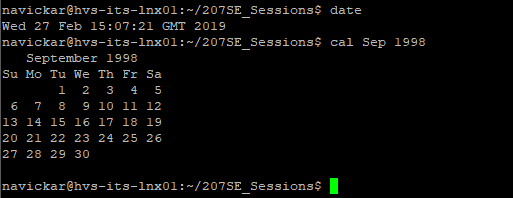


Then using mv command, file was moved to P1-Lab0 directory:

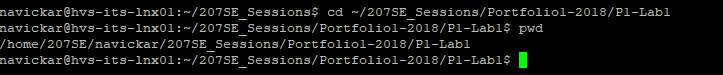


## Mixed Linux Commands

A – Todays date was displayed using *date* command; using *cal Sep 1998* I have displayed my month of birth.



B – I have used *cd*  command to jump between directories, that is what I used to move to P1-lab1 directory. To show a directory, I have used  *pwd*  command.



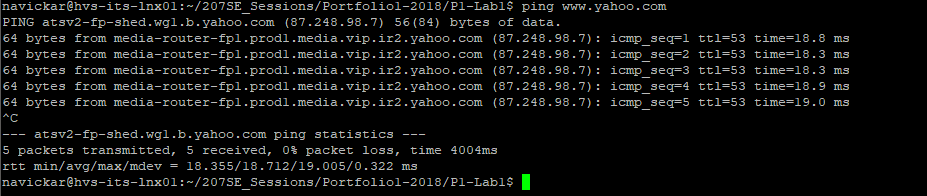
C- Using command  *uptime* I have displayed system’s running time since last reboot.



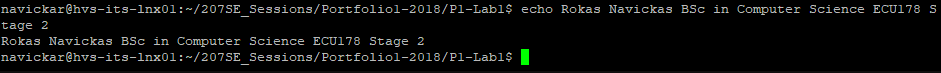
D – Effects of talk, write and wall can be disabled using *mesg n*  command



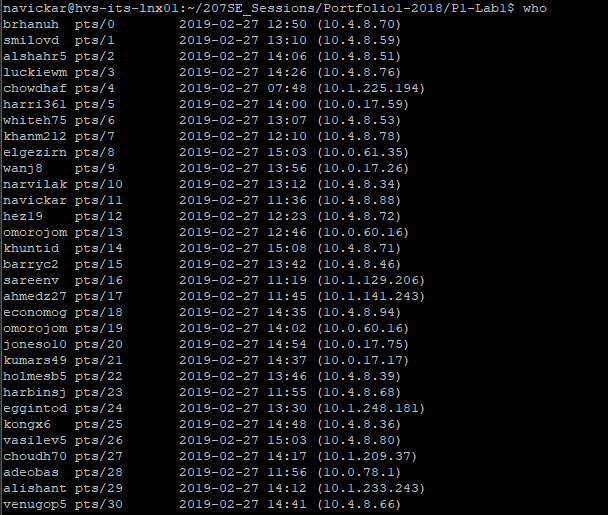
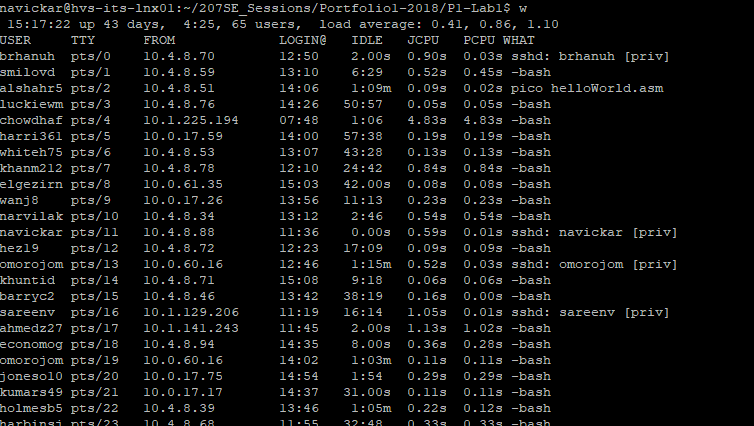
E - Using  *ping*  I can check weather another side of connection can receive/transmit packages



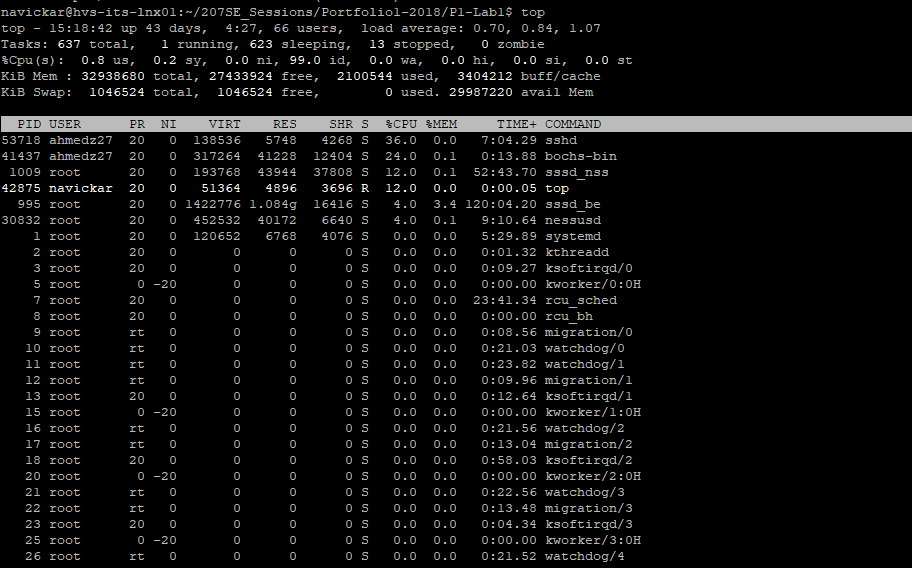
F – Using *echo* command I have displayed required details.



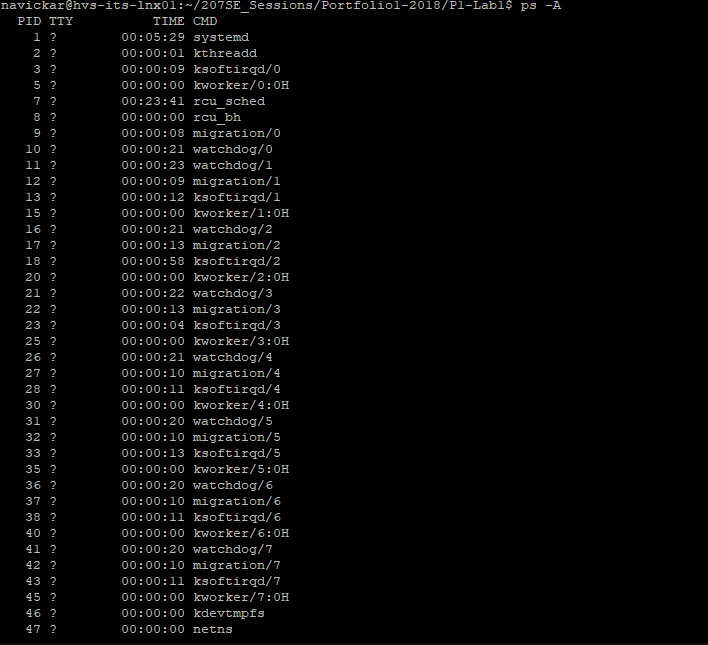
G – My username can be displayed by simply writing *whoami* in to CLI interface; One way to check who is logged in is *w*; another is *who*



H – One way to list processes that are running is to type *top* into CLI



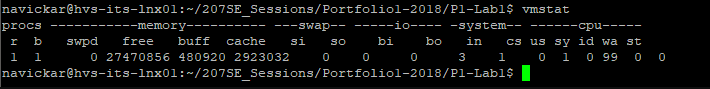
Another is *ps –A:*



I – mv command takes the original file and moves it to the destination required, but does not leave the original file in the original location. However, if cp is being used, the file stays in original location as well as in the location it’s being moved to. Basically, mv works as a right click – cut in windows interface and cp works as copy.

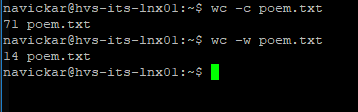
NB: There is no difference for the timestamps what command to use as it does not affect the timestamps.

J – Using single command of *vmstat*  I have displayed the system information requested.

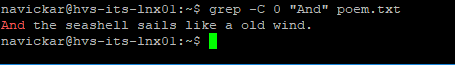


## Document Manipulation

A – I have used *wc –c*  and *wc -w*  to count required parts of poem.



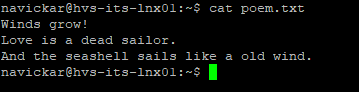
B – Using *grep –C 0 “And” poem.txt* I have displayed line that contains word “and”



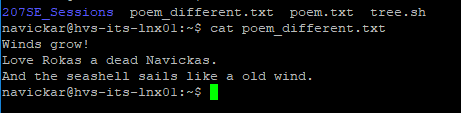
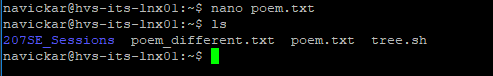
C – I have used grep –E ‘and|And|it|It’ poem.txt to filter for words “and” and “it” in file and display a line that contains the word.

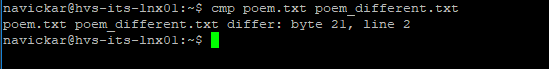


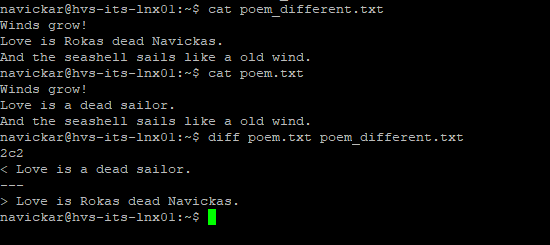
D – Used cat [filename] to check contents of the file



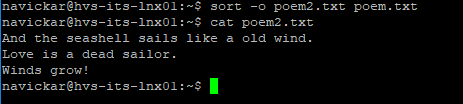
E - I have replaced two words in an editor using nano and saved to different file called poem\_different.txt



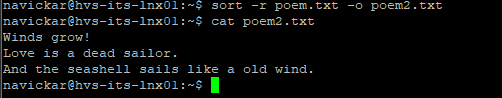
F – I have used *diff and cmp*  commands to compare two files. One is comparing word by word, another is comparing byte by byte. Since the exercise did not state that command had to compare word by word, I have chosed these two methods of doing it. 



G – I have used  *sort*  -o poem2.txt poem.txt to sort lines from file poem.txt and output that to poem2.txt



H – *sort –r poem.txt –o poem2.txt*  was used to reverse sort the file.



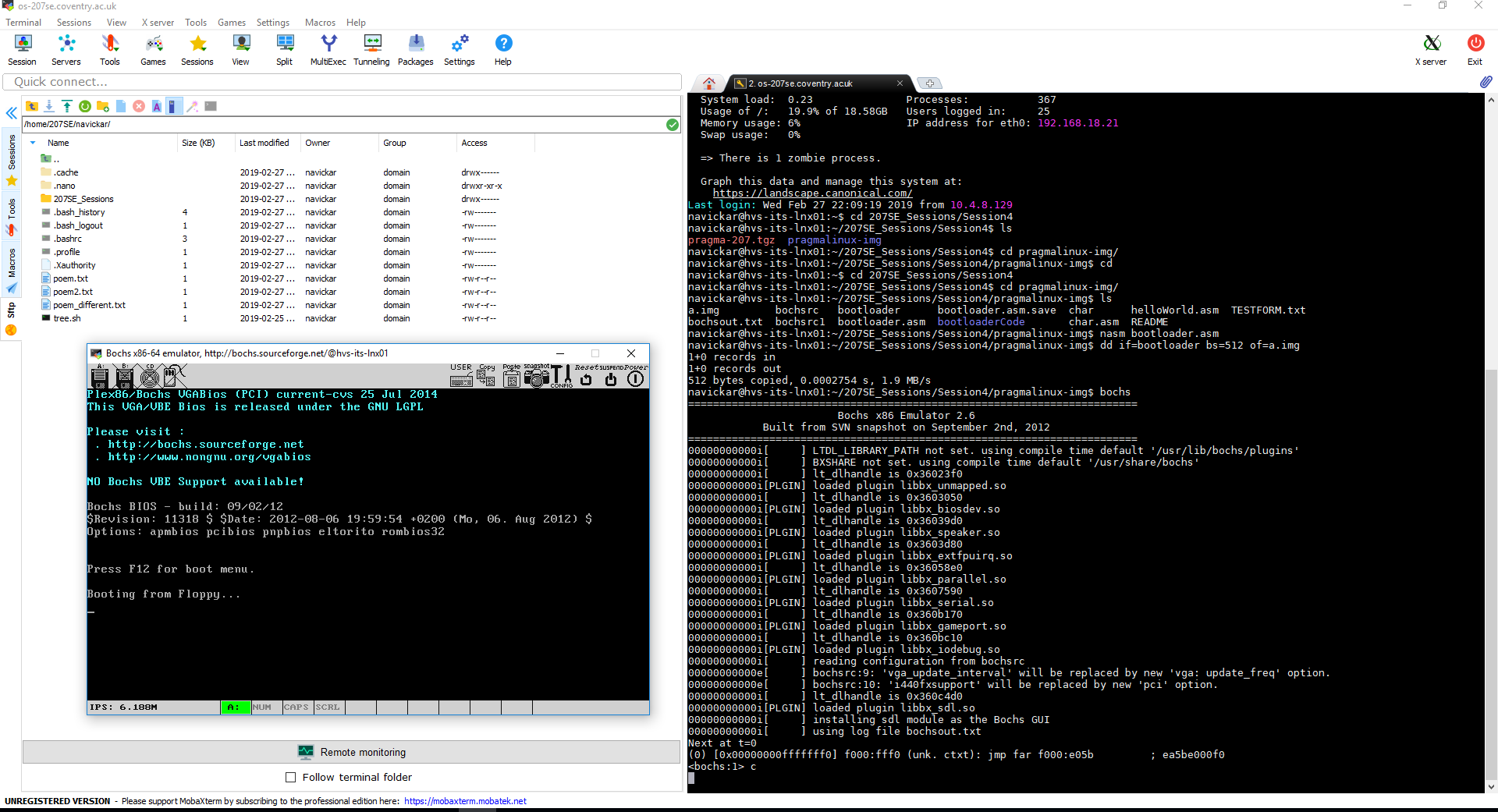
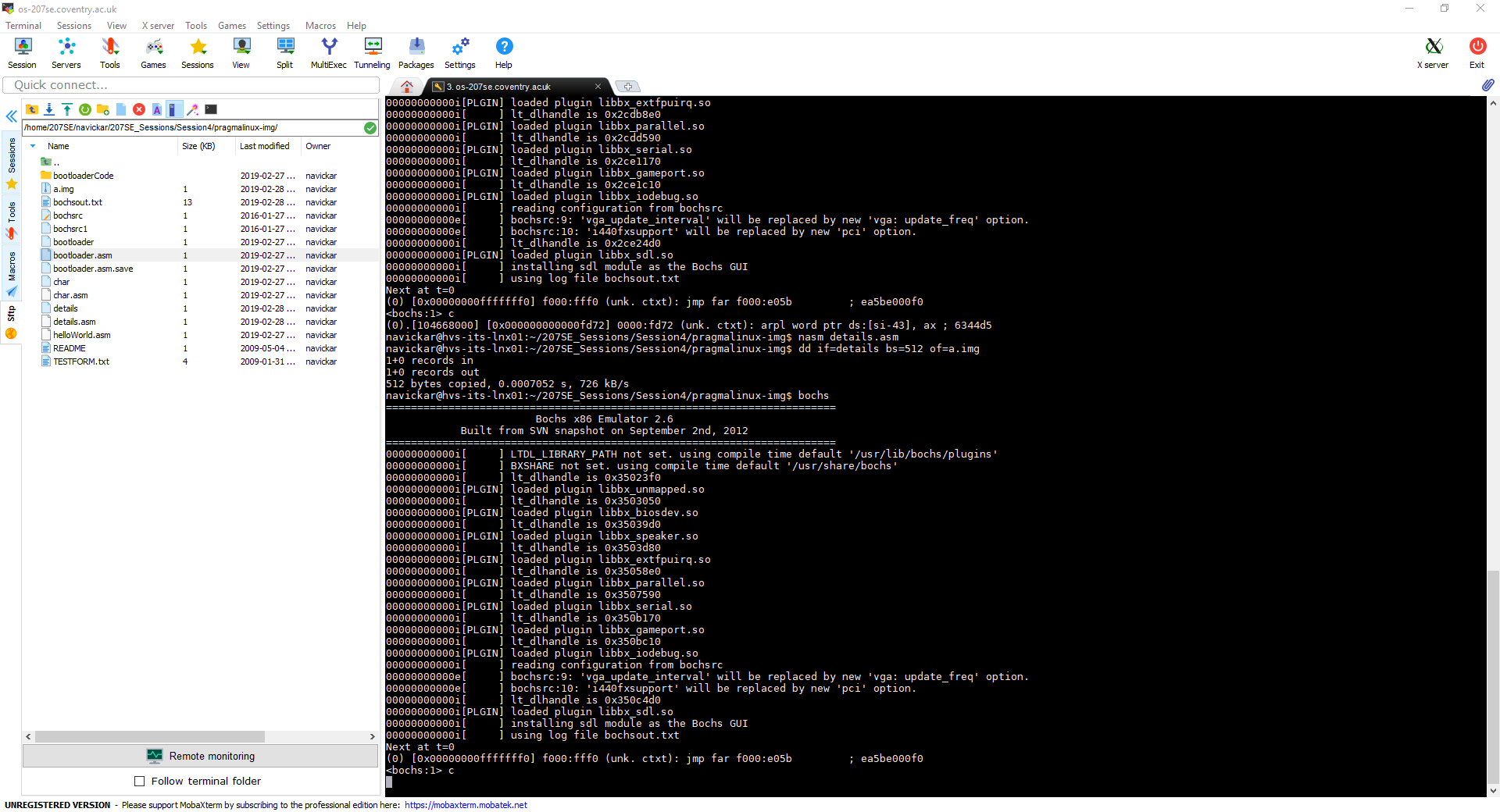
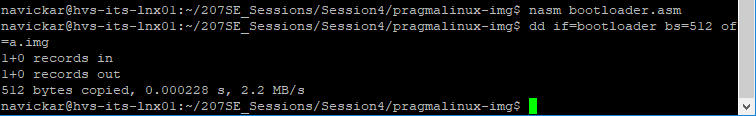
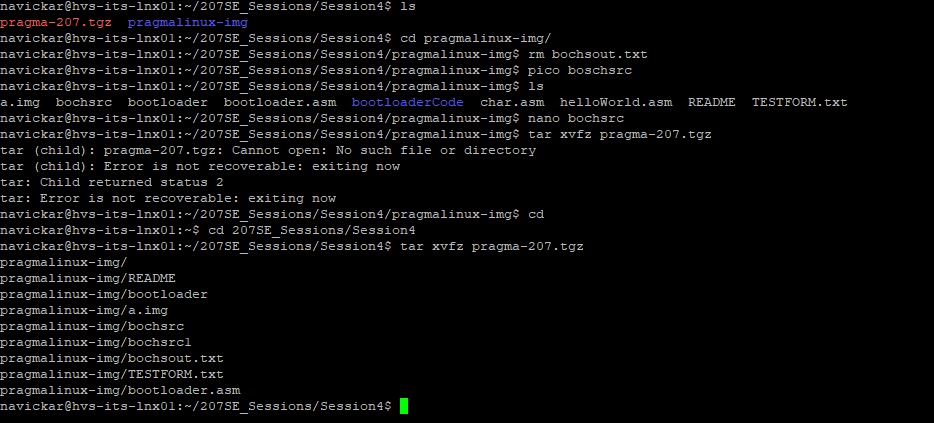
I – Using *alias mysort=”sort –r poem.txt –o poem2.txt”*  I have create shortcut for a command stated in brackets that is being called by writing *mysort*.



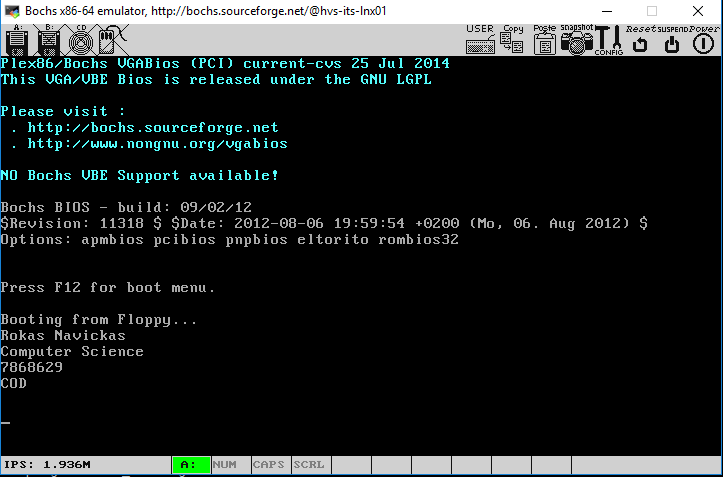
# Lab 4 – Booting

## Creating a Bootloader

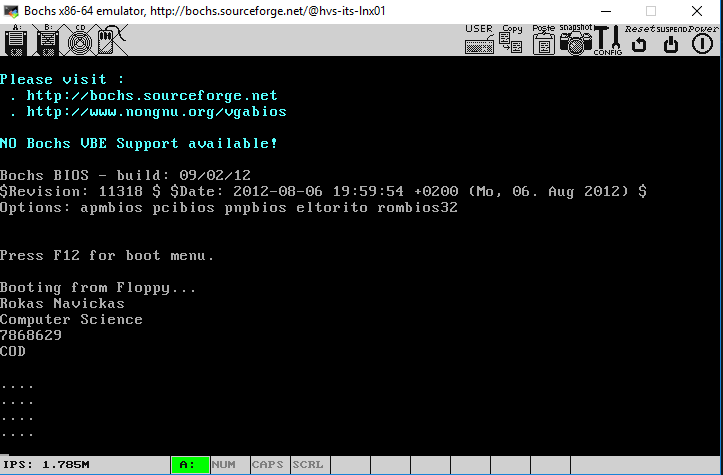
Basic task A – Evidence of booting pragma linux:



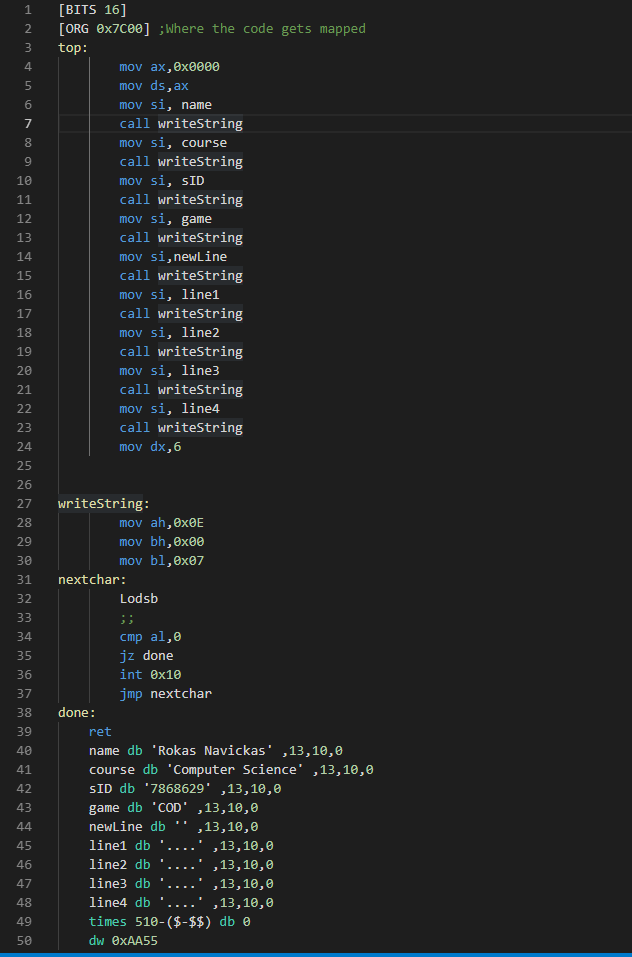
Basic Task B – Bootloader was edited to display information required:



Basic Task C – I have created a bootloader that creates squares of dots without using loops as well as displaying info that was displayed in previous bootloader configuration.

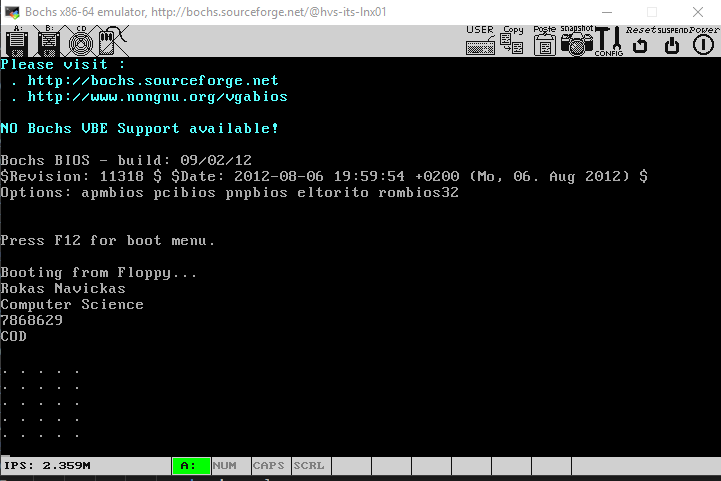
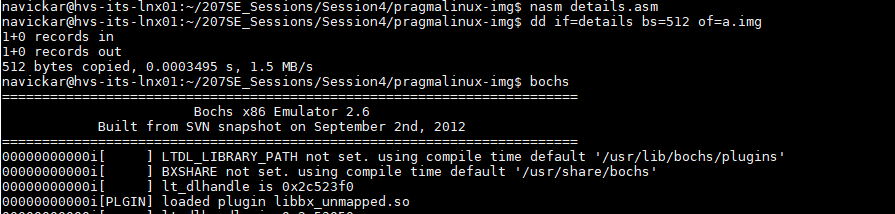


Code for the bootloader to display things required:



## Advanced Tasks:

A – I have created a bootloader that displays square of dots using loops:



Code for the bootloader:

[BITS 16]

[ORG 0x7C00] ;Where the code gets mapped

top:

mov ax,0x0000

mov ds,ax

mov si, name

call writeString

mov si, course

call writeString

mov si, sID

call writeString

mov si, game

call writeString

mov si,newline

call writeString

mov dx,5 ;Number of lines to be printed

outer\_loop:

mov cx,5 ;Numbers of dots to put in line

inner\_loop:

mov si, dot

call writeString

dec cx

cmp cx,0

jne inner\_loop

mov si,newLine

call writeString

dec dx

cmp dx,0

jne outer\_loop

jmp done

writeString:

mov ah,0x0E

mov bh,0x00

mov bl,0x07

nextchar:

Lodsb

;;

cmp al,0

jz done

int 0x10

jmp nextchar

done:

ret

name db 'Rokas Navickas' ,13,10,0

course db 'Computer Science' ,13,10,0

sID db '7868629' ,13,10,0

game db 'COD' ,13,10,0

newLine db '' ,13,10,0

newline db '' ,13,10,0

dot db ' .' ,0

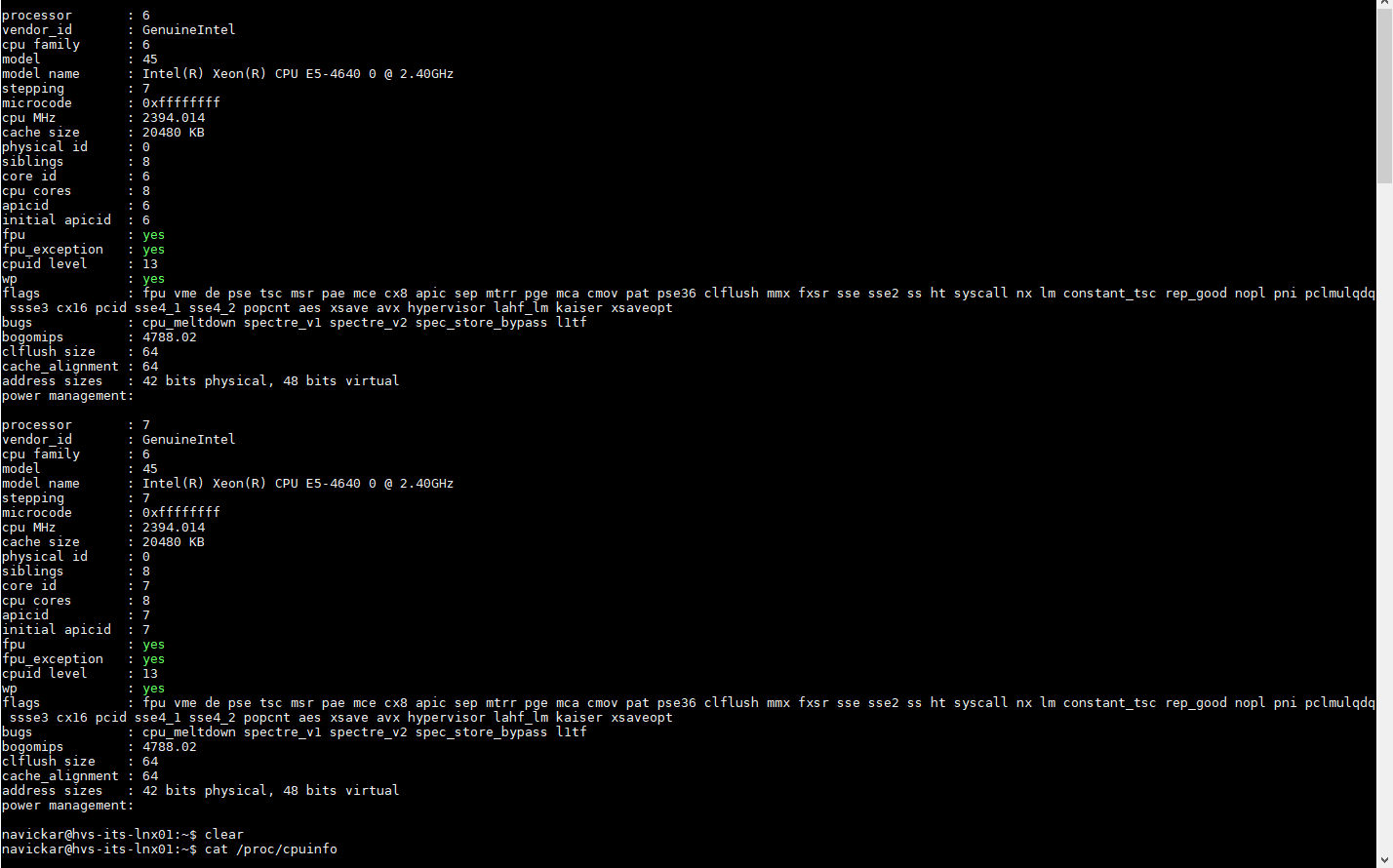
times 510-($-$$) db 0

dw 0xAA55

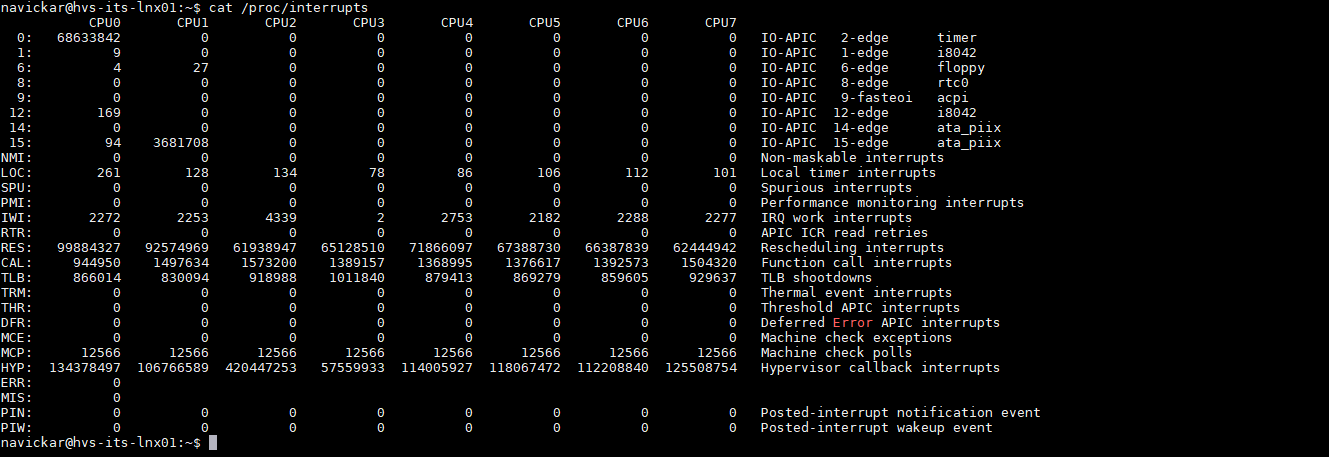
# Lab 5 – Outside the CPU

## Basic Tasks

A – used cat /proc/cpuinfo to view information about computer CPUs



B - I have used *cat /proc/interrupts*  to view different interrupts available on the system since boot time.



Also, using *cat /proc/interrupts | wc –l*  I have produced count of lines which essentially shows how many different interrupts have there been since boot time.

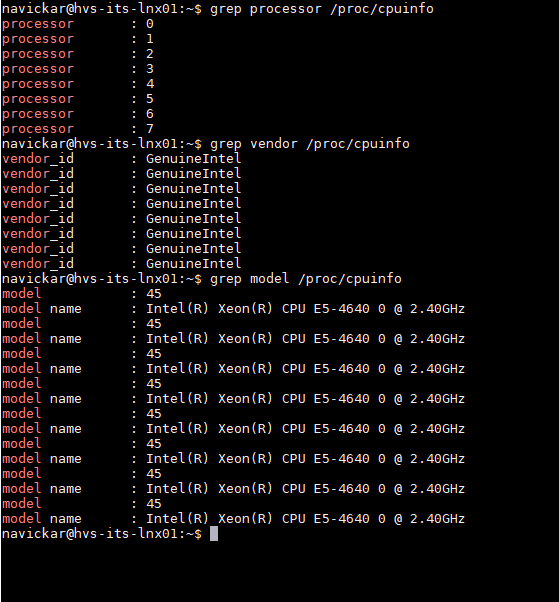


C – I have showed number of CPU’s, producer of the cpu and CPU model using:

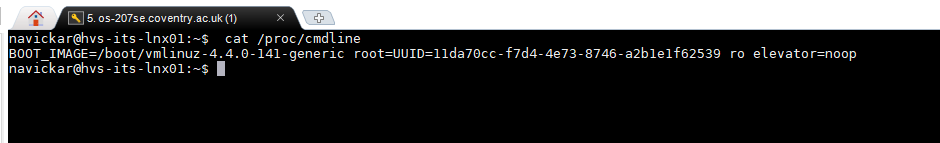
*grep processor /proc/cpuinfo*

*grep vendor /proc/cpuinfo*

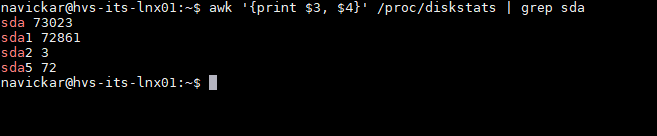
*grep model proc/cpuinfo*



D – You can see parameters that are being passed to kernel during boot up using *cat proc/cmdline*

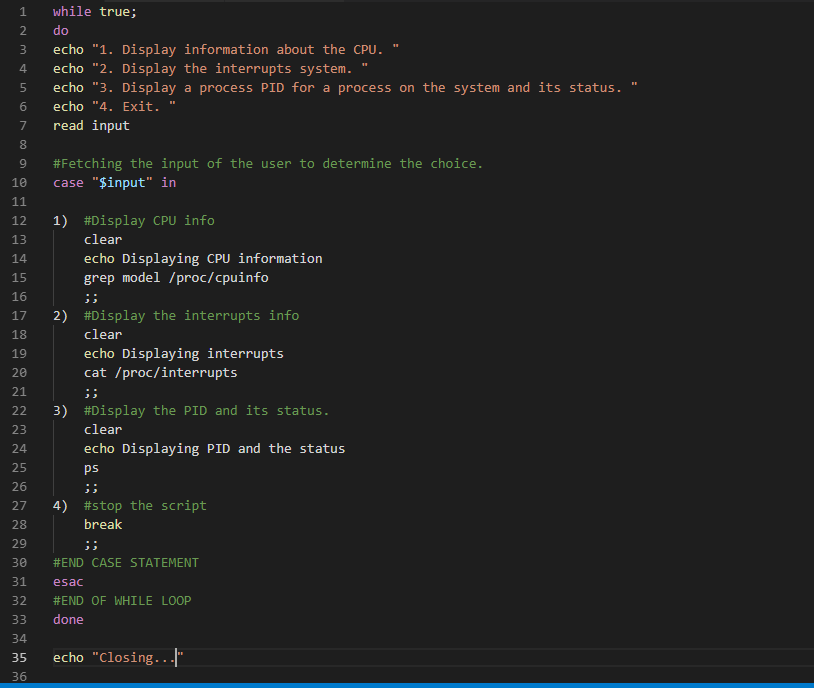
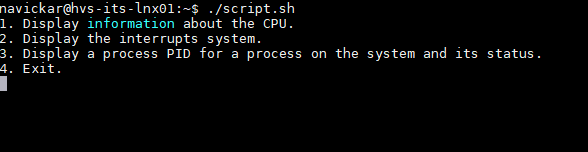


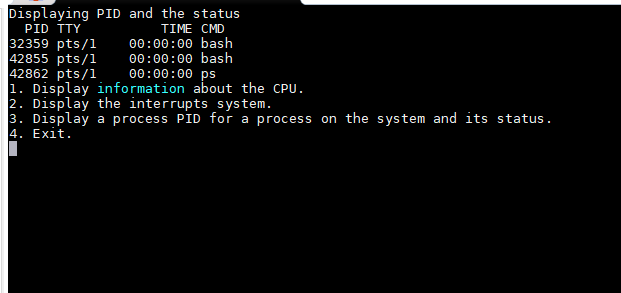
E - Used awk ‘{print $3, $4}’ /proc/diskstats | grep sda to display parameters required

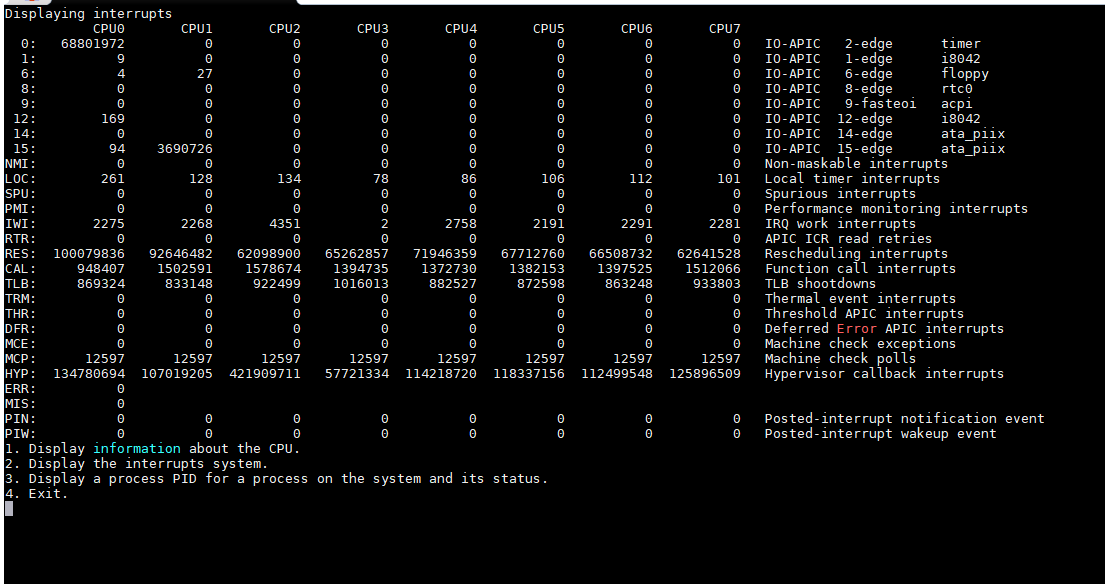
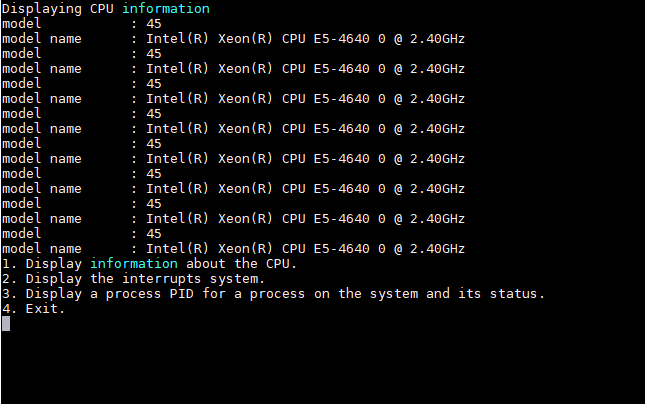


## Advanced tasks:

The menu-based graphical user interface was created using shell script. Please see evidence provided:







# Lab 7 – Simple Buffer

## Basic Task 1:

Commented Code:

#include <fcntl.h> //Define header files

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#define BUF\_SIZE 500   //Define Buffer size as 500.

#define OUTPUT\_MODE 0700  //Define file permission.

int main(int argc, char \*argv[]) //Main function

{

int in\_fd, out\_fd; // define input and output files

int rd\_size = 1, wr\_size; //Declare read and write sizes

char buf[BUF\_SIZE]; //Declare buffer.

if (argc != 3)// if argumet size is not 3 - exit the program

{

   exit(1);

}

in\_fd = open(argv[1], O\_RDONLY); //Open input file

if (in\_fd < 0)//If the output file cannot be oppened, in\_fd will be negative, hence display error and exit

{

exit(2);

}

out\_fd = creat(argv[2], OUTPUT\_MODE); //Create output file.

if (out\_fd < 0)//If the file cannot be created, the out\_fd will fall to negative

{

exit(3);

}

while (rd\_size > 0)// While read size is higher than none, do loop

{

rd\_size = read(in\_fd, buf, BUF\_SIZE);// Read from input file into buffer.

if (rd\_size<0)//if there is nothing to read:

{

exit(4);

}

wr\_size = write(out\_fd, buf, rd\_size); // Write from buffer into output file.

if (wr\_size<=0)//if there are nothing to write:

{

close(in\_fd); //Close input file

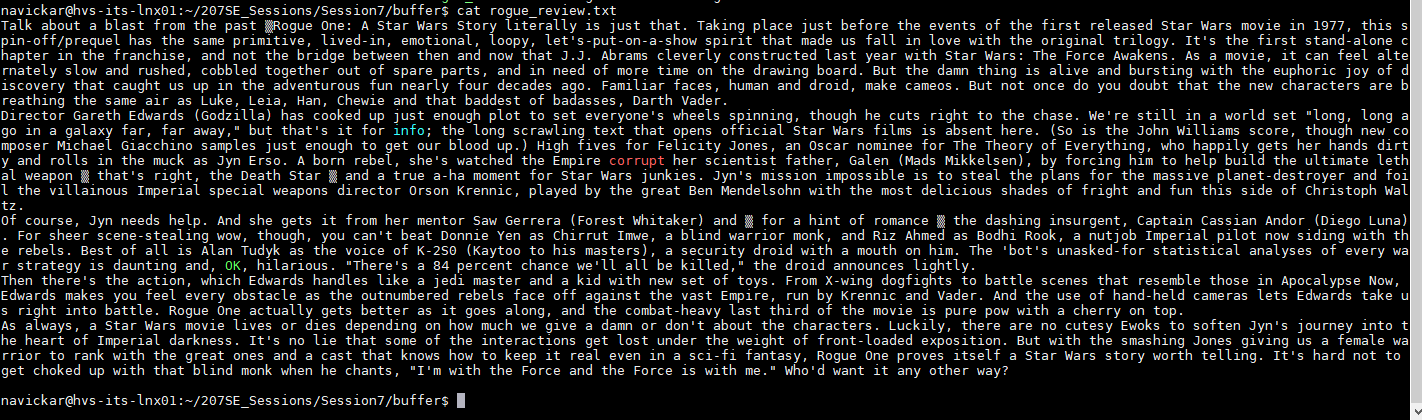
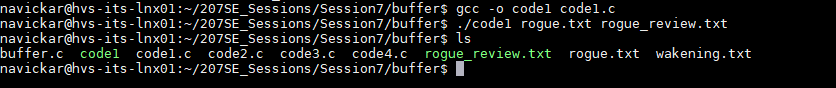
close(out\_fd); //Close output file

exit(5); //Exit the program

}

}

}



## Basic Task 2

Code that prints when and what kind of error occurs and also it prints when the file creation was successful:

#include <fcntl.h> //Define header files

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#define BUF\_SIZE 500   //Define Buffer size as 500.

#define OUTPUT\_MODE 0700  //Define file permission.

int main(int argc, char \*argv[]) //Main function

{

int in\_fd, out\_fd; // define input and output files

int rd\_size = 1, wr\_size; //Declare read and write sizes

char buf[BUF\_SIZE]; //Declare buffer.

int characters = 0;

int x;

int word\_count = 0;

int sent\_count = 0

int times\_filled = 0;

int max = 0;

if (argc != 3)// if argumet size is not 3 - exit the program

{

printf("No three arguments were found!\n");

   exit(1);

}

in\_fd = open(argv[1], O\_RDONLY); //Open input file

if (in\_fd < 0)//If the output file cannot be oppened, in\_fd will be negative, hence display error and exit

{

printf("Input file not found!\n");

exit(2);

}

out\_fd = creat(argv[2], OUTPUT\_MODE); //Create output file.

if (out\_fd < 0)//If the file cannot be created, the out\_fd will fall to negative

{

printf("Cannot create output file!\n");

exit(3);

while (rd\_size > 0)// While read size is higher than none, do loop

{

rd\_size = read(in\_fd, buf, BUF\_SIZE);// Read from input file into buffer.

if (rd\_size<0)//if there is nothing to read:

{

printf("There is nothing to read!\n");

exit(4);

}

wr\_size = write(out\_fd, buf, rd\_size); // Write from buffer into output file.

if (wr\_size<=0)//if there are nothing to write:

{

close(in\_fd); //Close input file

close(out\_fd); //Close output file

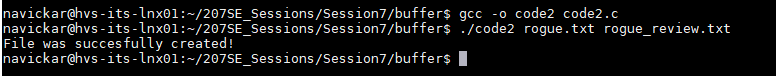
printf("File was succesfully created!\n")

exit(5); //Exit the program

}

}

}



## Basic Task 3

Code that shows how many characters were read in total, how many characters were read from buffer at a time, how many words and sentences read and how many times buffer was filled.

#include <fcntl.h> //Define header files

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#define BUF\_SIZE 1125   //Define Buffer size as 500.

#define OUTPUT\_MODE 0700  //Define file permission.

int main(int argc, char \*argv[]) //Main function

{

int in\_fd, out\_fd; // define input and output files

int rd\_size = 1, wr\_size; //Declare read and write sizes

char buf[BUF\_SIZE]; //Declare buffer.

int characters = 0;

int x;

int word\_count = 0;

int sent\_count = 0;

int times\_filled = 0;

int at\_time = 0;

if (argc != 3)// if argumet size is not 3 - exit the program

{

printf("No three arguments were found!\n");

   exit(1);

}

in\_fd = open(argv[1], O\_RDONLY); //Open input file

if (in\_fd < 0)//If the output file cannot be oppened, in\_fd will be negative, hence display error and exit

{

printf("Cannot open input file!\n");

exit(2);

}

out\_fd = creat(argv[2], OUTPUT\_MODE); //Create output file.

if (out\_fd < 0)//If the file cannot be created, the out\_fd will fall to negative

{

printf("Cannot open output file!\n");

exit(3);

}

while (rd\_size > 0)// While read size is higher than none, do loop

{

rd\_size = read(in\_fd, buf, BUF\_SIZE);// Read from input file into buffer.

if (rd\_size>0)

{

characters += rd\_size;

times\_filled += 1;

printf("%d characters were read at a time!\n",rd\_size);

}

if (rd\_size<0)

{

exit(4);

}

for(x = 0; x<rd\_size; x++)//Counts words and sentences in the read sequence

{

if(buf[x]==' ')

{

word\_count += 1;

}

else if (buf[x] == '.')

{

word\_count += 1;

sent\_count += 1;

x++;

}

}

wr\_size = write(out\_fd, buf, rd\_size); // Write from buffer into output file.

if (wr\_size<=0)//if there are nothing to write:

{

close(in\_fd); //Close input file

close(out\_fd); //Close output file

printf("File was succesfully created!\n");

printf("Characters read in total: %d\n",characters);

printf("Words read: %d\n",word\_count);

printf("Sentences read: %d\n", sent\_count);

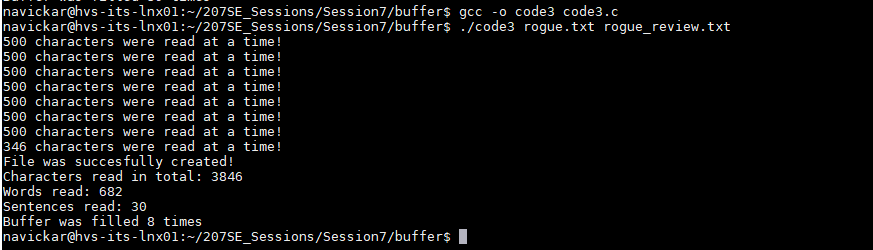
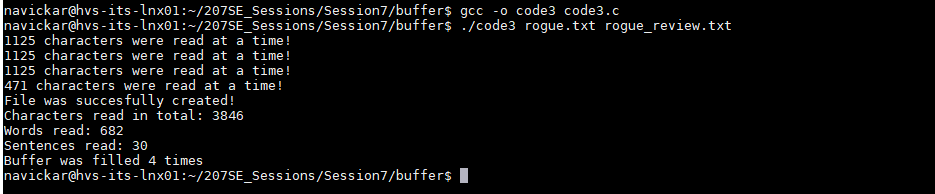
printf("Buffer was filled %d times\n",times\_filled);

exit(5); //Exit the program

}

}

}



## Advanced Task:

A – See attached code for file comparison:

#include <fcntl.h> //Define header files

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#define BUF\_SIZE 500   //Define Buffer size as 500.

#define OUTPUT\_MODE 0700    //Define file permission.

int main(int argc, char \*argv[])//Main function

{

int in\_fd, in0\_fd; // define input files

int rd\_size = 1; //Declare read size

char buf[BUF\_SIZE]; // //Declare first buffer.

char buf0[BUF\_SIZE]; //Declare second buffer.

if (argc != 3)// if argumet size is not 3 - exit the program

exit(1);

in\_fd = open(argv[1], O\_RDONLY); //Open 1st file.

if (in\_fd < 0)//If the input file cannot be oppened, in\_fd will be negative, hence display error and exit

exit(2);

in0\_fd = open(argv[2], O\_RDONLY); //Open 2nd file.

if (in0\_fd < 0)//If the input file cannot be oppened, in0\_fd will be negative, hence display error and exit

exit(3);

while (rd\_size > 0) // While read size is higher than none, do loop

{

rd\_size = read(in\_fd, buf, BUF\_SIZE); // Read From 1st file into 1st                                    //buffer

if (rd\_size <0)

{//if there is nothing to read:

exit(4);

}

rd\_size = read(in0\_fd, buf0, rd\_size); //Read from 2nd file into 2nd                                    //buffer

for (int i =0; i < BUF\_SIZE; i++)//Loop through the contents of each buffer.

{

if (buf[i] == buf0[i])// If buffer contents are equal, go to next buffer item

{

continue;

}

else //If buffer contets are not the same, close the files and exit the program

{

close(in\_fd); //Close input file.

close(in0\_fd); //Close output file

printf("Files are not the same. \n");

exit(5);

}

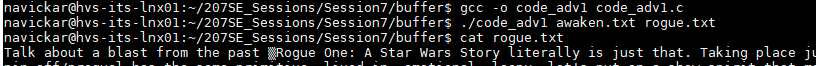
}

}

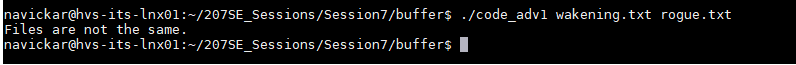
printf("Files are the same \n"); //If it does pass tests, print message

}

Program in action – Since file awaken.txt was not existing:



I have thought that the program does not work. After a while I have noticed that the file name is a bit different than provided -



When ran with wakening.txt, program worked fine.

B – Please see code for adapted version of buffer.c to create a summary that is less than 30 words.

#include <fcntl.h> //Define header files

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#define BUF\_SIZE 4500   //Define Buffer size as 3500.

#define OUTPUT\_MODE 0700  //Define file permission.

int main(int argc, char \*argv[]) //Main function

{

int in\_fd, out\_fd; // define input and output files

int rd\_size = 1, wr\_size; //Declare read and write sizes

char buf[BUF\_SIZE]; //Declare buffer.

int word\_count = 0;

int sent\_count = 0;

int stop, finish;

if (argc != 3)// if argumet size is not 3 - exit the program

{

printf("No three arguments were found!\n");

   exit(1);

}

in\_fd = open(argv[1], O\_RDONLY); //Open input file

if (in\_fd < 0)//If the output file cannot be oppened, in\_fd will be negative, hence display error and exit

{

printf("Cannot open input file!\n");

exit(2);

}

out\_fd = creat(argv[2], OUTPUT\_MODE); //Create output file.

if (out\_fd < 0)//If the file cannot be created, the out\_fd will fall to negative

{

printf("Cannot open output file!\n");

exit(3);

}

while (rd\_size > 0)// While read size is higher than none, do loop

{

rd\_size = read(in\_fd, buf, BUF\_SIZE);// Read from input file into buffer.

if (rd\_size<0)

{

exit(4);

}

for(int x = 0; x<rd\_size; x++)//Counts words and sentences in the read sequence

{

if(word\_count >= 30)

{

finish = stop+1;

buf[finish+1]='\n';

break;

}

else if(buf[x]==' ')

{

word\_count += 1;

}

else if (buf[x] == '.' || buf[x] == '"' || buf[x] == ';')

{

word\_count += 1;

sent\_count += 1;

stop = x;

}

}

wr\_size = write(out\_fd, buf, finish+1); // Write from buffer into output file.

close(in\_fd); //Close input file

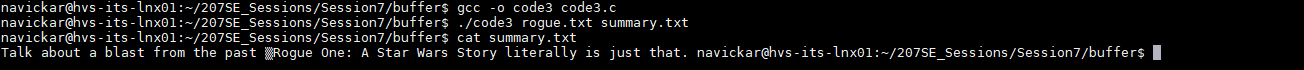
close(out\_fd); //Close output file

exit(5); //Exit the program

}

}

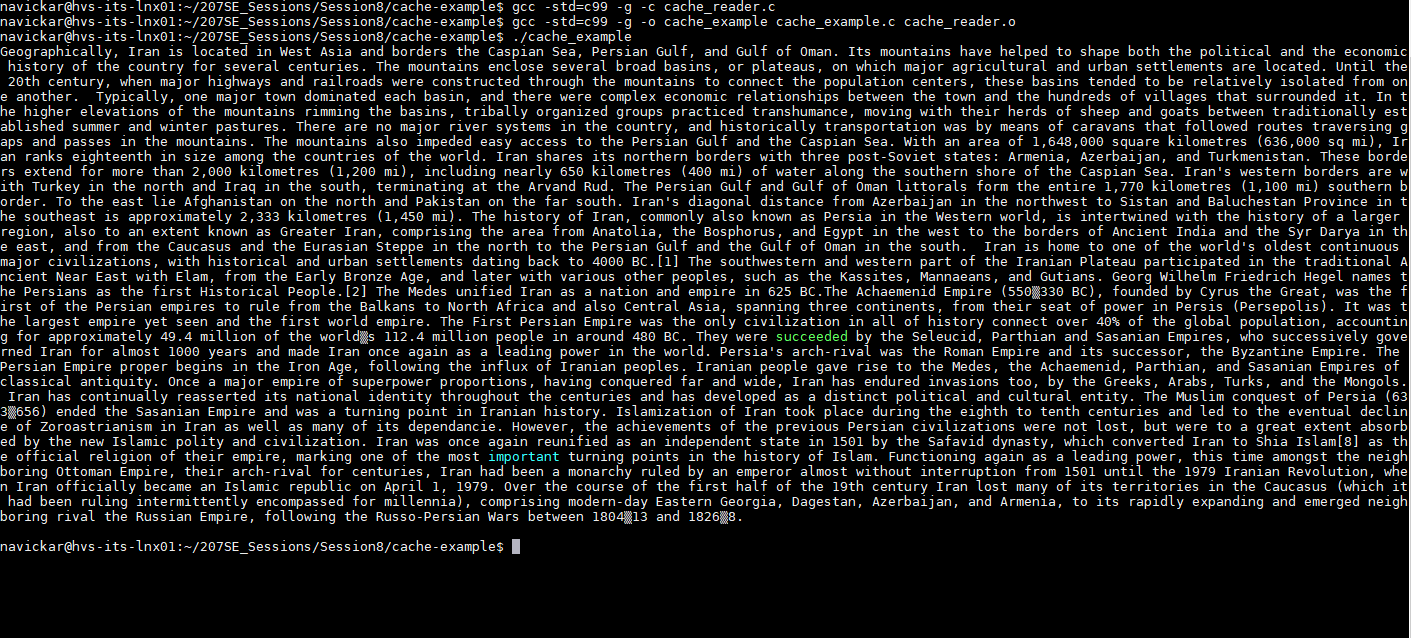
That is the output of the program:



# Lab 8 – Cache Library

## Basic Task – *cr\_read\_byte:*

I have modified cr\_read\_byte to display the next byte in the buffer and to change current position of usedBuffer.



//------------------------------------------------------------------

char cr\_read\_byte(cr\_file\* f){

printf("a byte has been read\n");

if (f->usedbuffer>= f->bufferlength)//Buffer check, if it's at the end of the buffer - refill

{

refill(f);

}

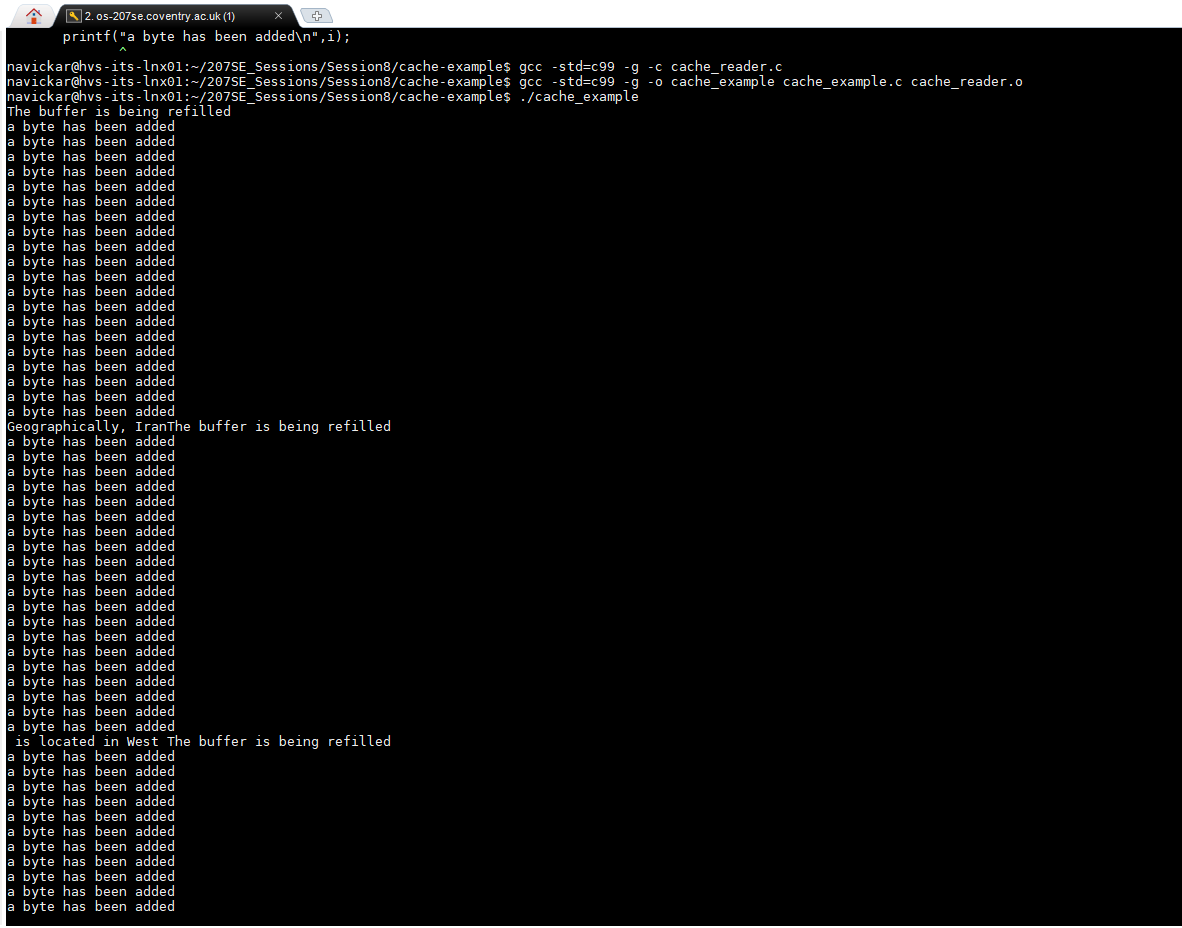
return f->buffer[f->usedbuffer++];//Returns the next character in the buffer

return EOF; // this is just so the compile works...

}

## Advanced Tasks

A – I have modified the library to print a message when each byte is being read and when the buffer is being refilled.



That is code to show how it works:

#include "cache\_reader.h"

//http://www.phim.unibe.ch/comp\_doc/c\_manual/C/SYNTAX/struct.html

//http://vergil.chemistry.gatech.edu/resources/programming/c-tutorial/structs.html

int refill(cr\_file\* buff){

printf("The buffer has been refilled");

//Refills a buffer

//Only works when completely used buffer

if(buff->usedbuffer!=buff->bufferlength)

return 0;

else{

buff->usedbuffer=0;

int len=fread(buff->buffer, sizeof(char), buff->bufferlength, buff->file);

//If we didn't fill the buffer, fill up with EOF

for (int i=0; i<len; i++)//Counts how many bytes has been read to a buffer.

{

printf("a byte has been added\n");

}

if(len<buff->bufferlength)

for(int i=len;i<buff->bufferlength;i++)

  buff->buffer[i]=EOF; //Accessing like an array!

return len;

}

}

void cr\_close(cr\_file\* f){

free(f->buffer);

fclose(f->file);

}

cr\_file\* cr\_open(char \* filename, int buffersize){

//Info on malloc

//http://www.space.unibe.ch/comp\_doc/c\_manual/C/FUNCTIONS/malloc.html

FILE\* f;

if ((f = fopen(filename, "r")) == NULL){

fprintf(stderr, "Cannot open %s\n", filename);

return 0;

}

cr\_file\* a=(cr\_file\*)malloc(sizeof(cr\_file));

a->file=f;

a->bufferlength=buffersize;

a->usedbuffer=buffersize; //Start off with no characters, so refill will work as expected

a->buffer=(char\*)malloc(sizeof(char)\*buffersize);

refill(a);

return a;

}

//------------------------------------------------------------------

char cr\_read\_byte(cr\_file\* f){

if (f->usedbuffer>= f->bufferlength)//Buffer check, if it's at the end of the buffer - refill

{

refill(f);

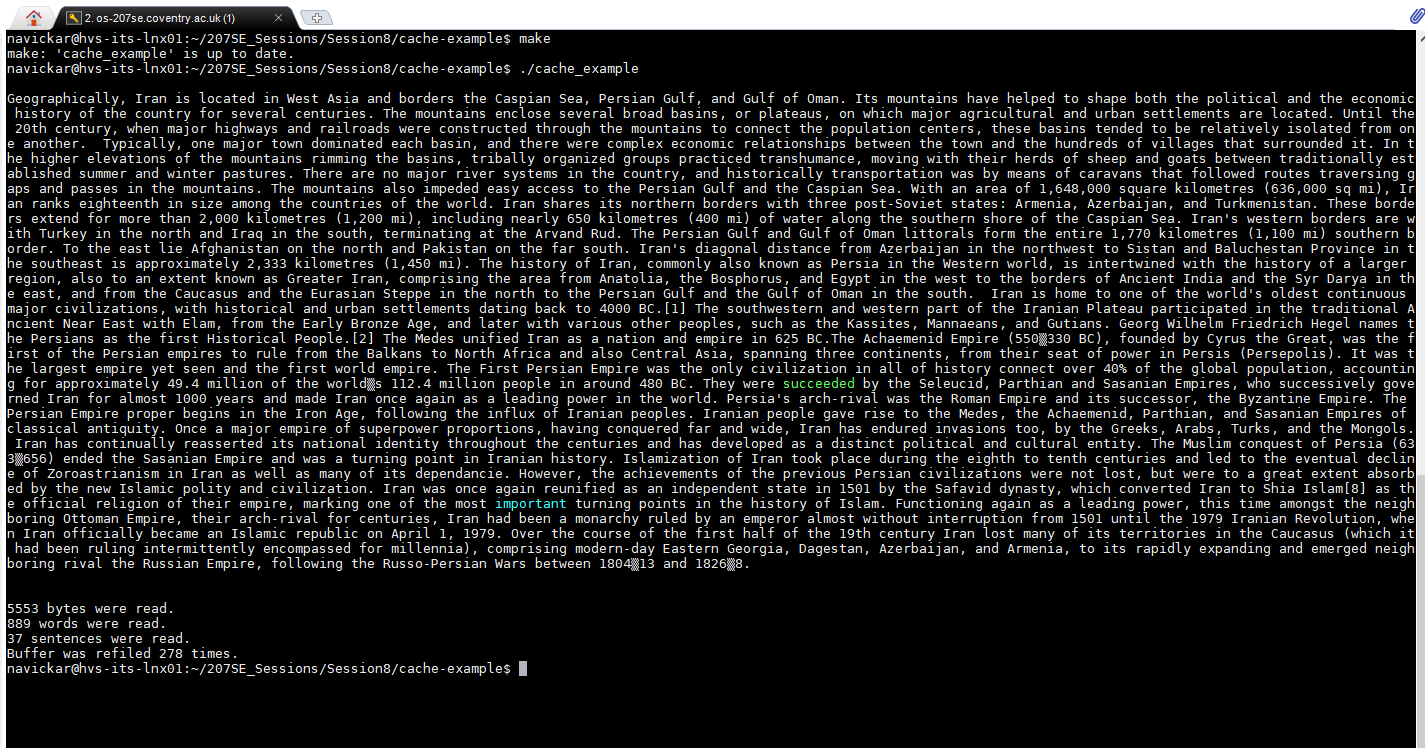
}

return f->buffer[f->usedbuffer++];//Returns the next character in the buffer

return EOF; // this is just so the compile works...

}

B - I have modified the code to show how many bytes, words, sentences were read and how many times buffer was refilled:



Code to show how it works:

#include "cache\_reader.h"

//Simple file display to show how easy it is to use the cached reader functions

int main(){

char c;

int buffer\_size = 20 ;

int refill\_count = 0, byte\_count = 0, word\_count = 0, sent\_count = 0;

for( int i =0; i<6; i++)

{

}

//Open a file

cr\_file\* f = cr\_open("text",buffer\_size);

//While there are useful bytes coming from it

printf("\n");

while((c=cr\_read\_byte(f))!=EOF)

{

byte\_count += 1;

//Print them

if( c == ' ')

{

word\_count += 1;

}

if(c == '.' || c == ';' || c == '!' || c == '?')

{

sent\_count += 1;

}

printf("%c",c);

}

//Then close the file

cr\_close(f);

if (byte\_count%buffer\_size == 0)

{

refill\_count = byte\_count/buffer\_size;

}

else

{

refill\_count = byte\_count/buffer\_size + 1;

}

//And finish

printf("\n%d bytes were read.\n", byte\_count);

printf("%d words were read.\n",word\_count);

printf("%d sentences were read.\n",sent\_count);

printf("Buffer was refiled %d times.\n",refill\_count);

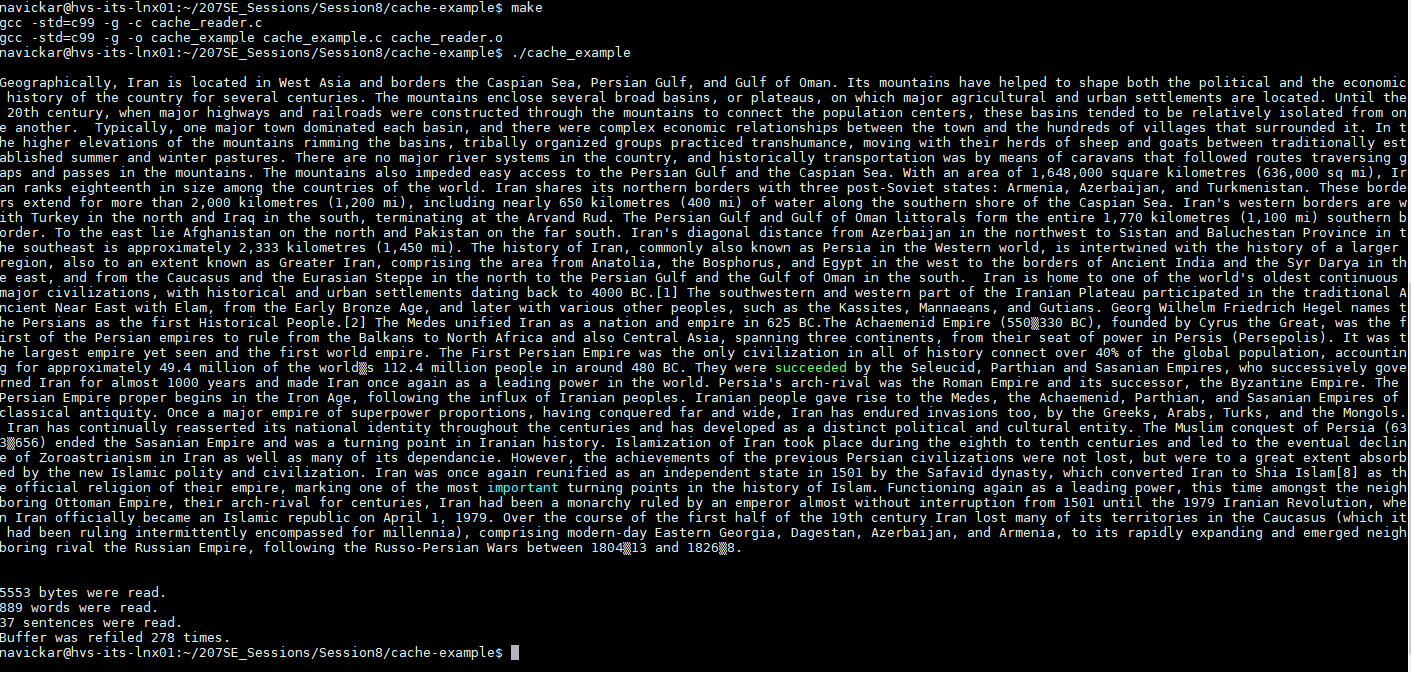
return 0;

}

# Lab 10 – Kernel II System Calls

## Basic Task

I have changed the code to work with open(),read() and write() functions instead of fopen,fclose and fread:



Code header file:

#include <stdio.h>

#include <stdlib.h>

//-----------------------------

#include<sys/types.h>

#include<sys/stat.h> //Includes needed for open() to work in C

#include <fcntl.h>

//------------------------

#include <unistd.h> //Include needed for read() and write() to work in C

//The internals of this struct aren't important

//from the user's point of view

typedef struct{

int file; //File being read(int is being used on open())

int bufferlength; //Fixed buffer length

int usedbuffer; //Current point in the buffer

char\* buffer; //A pointer to a piece of memory

} cr\_file;

//Open a file with a given size of buffer to cache with

cr\_file\* cr\_open(char\* filename, int buffersize);

//Close an open file

void cr\_close(cr\_file\* f);

//Read a byte. Will return EOF if empty.

char cr\_read\_byte(cr\_file\* f);

code.c file:

#include "cache\_reader.h"

int refill(cr\_file\* buff){

//Refills a buffer

//Only works when completely used buffer

if(buff->usedbuffer!=buff->bufferlength)

return 0;

else{

buff->usedbuffer=0;

int len=read(buff->file, buff->buffer, buff->bufferlength); //reads bytes from file starting at buff.

//If we didn't fill the buffer, fill up with EOF

if(len<buff->bufferlength)

for(int i=len;i<buff->bufferlength;i++)

  buff->buffer[i]=EOF; //Accessing like an array!

return len;

}

}

void cr\_close(cr\_file\* f){

free(f->buffer);

close(f->file);

}

cr\_file\* cr\_open(char \* filename, int buffersize)

{

int f;

if ((f = open(filename, O\_RDONLY)) < 0){ //Opens a file in read\_only, while oppening a file the value changes to non-negative.

printf("Cannot open %s\n", filename);

return 0;

}

cr\_file\* a=(cr\_file\*)malloc(sizeof(cr\_file));

a->file=f;

a->bufferlength=buffersize;

a->usedbuffer=buffersize; //Start off with no characters, so refill will work as expected

a->buffer=(char\*)malloc(sizeof(char)\*buffersize);

refill(a);

return a;

}

//------------------------------------------------------------------

char cr\_read\_byte(cr\_file\* f){

// your code goes here

if (f->usedbuffer>= f->bufferlength)//Buffer check, if it's at the end of the buffer - refill

{

refill(f);

}

return f->buffer[f->usedbuffer++];//Returns the next character in the buffer

// remember that this needs to return a char (a byte, put another way..)

return EOF; // this is just so the compile works...

}

# References

Windows . (2019). *Windows Subsystem for Linux Installation Guide for Windows 10*. Retrieved from Windows Subsystem for Linux Installation Guide for Windows 10.