Student id:

Course:

Module: 5004CEM Operating Systems and Security Submission data: 23rd March 2020

Portfolio

Contents

[Lab Activity 1 – Operating Systems Tasks and Programming 5](#_Toc30774083)

[a. Comparison between the Harvard Architecture and Von Neumann Architecture 5](#_Toc30774084)

[b. Programming activity 5](#_Toc30774085)

[Lab Activity 2 – Linux Command Line (Commands and outcomes from a series of small tasks that require use of a number of Linux commands) 6](#_Toc30774086)

[Tasks – Files 6](#_Toc30774087)

[a. Create a directory in your /home/5004CEM\_Session/<username> directory with a name made up of you second name followed by 5004CEM and the year (mine would be **ELSHAW5004CEM2020**). Make the directory **read/write/executable** only for you, **read/write** for your groups and **read** only for others. 6](#_Toc30774088)

[b. Show evidence of this using the appropriate version of the **ls command**. 6](#_Toc30774089)

[c. Download the script http://www.centerkey.com/tree/tree.sh to your home directory using wget and make the file executable. 6](#_Toc30774090)

[d. Create a directory called **wrongDirectory**. You release it is not what you wanted so delete it. 6](#_Toc30774091)

[e. Create Portfolio1-2020 and Portfolio2-2020 directories in the directory you created in part a. 6](#_Toc30774092)

[f. Create numbered directories in the Portfolio1-2020 Directory (Lab0-5004CEM to Lab10-5004CEM) and in the Porfolio2-2020 Directory (Lab11-5004CEM to Lab20-5004CEM). 6](#_Toc30774093)

[g. In <YourSecondName>5004CEM2020 directory create a text file called **LastTask.txt** and then using the appropriate Linux command copy this document into Directory Lab0-5004CEM. 7](#_Toc30774094)

[h. Show how would you display all files in a directory including hidden ones files? 7](#_Toc30774095)

[Linux Commands – Mixed 7](#_Toc30774096)

[a. Using the date command show todays date and the time and date 5 years ago. Using the cal command show the month that you were born. Change this calendar to make Monday the first day of the week. 7](#_Toc30774097)

[b. Move into the lab1-5004CEM directory and use the appropriate command to show the current directory. 7](#_Toc30774098)

[c. Display the time when a user (ab0487) last login, the user’s home directory, and the user account’s full name. 7](#_Toc30774099)

[d. Find out how to prevent the effects of talk, write and wall from interrupting you. What command can you use? 7](#_Toc30774100)

[e. Show the command to verify that www.coventry.ac.uk exists and can accept requests. 7](#_Toc30774101)

[f. Display your name and favourite programming language on the screen using the echo command. 7](#_Toc30774102)

[g. Find out how you can display your username on the screen and at least two ways to display who is logged on. 8](#_Toc30774103)

[h. Use two ways to list the processes that are running. 8](#_Toc30774104)

[i. What are the differences between the Linux commands copy (cp), rename and move? 8](#_Toc30774105)

[j. With a single command, how would you get systems information such as processes, memory, paging and CPU activity? 8](#_Toc30774106)

[Tasks – Document Manipulation 8](#_Toc30774107)

[a. Use **cat** to show the contents of the file. 8](#_Toc30774108)

[b. Use an appropriate command to display the CRC checksum and byte count of the file. 8](#_Toc30774109)

[c. Use **grep** to show only lines not containing the words "she" or "he". Lines contain both “she” and “he” should be shown. 8](#_Toc30774110)

[d. Use **grep** to show the 5 lines above a line containing the text ‘the’. 8](#_Toc30774111)

[e. Using Linux commands you should count the lines containing “she” and “he” but not both and display the line numbers that “she” and “he” but not both appear on in the original document. 9](#_Toc30774112)

[f. Find a command to list the top 3 lines of the **poem.txt** file and then the bottom line of these top 3. 9](#_Toc30774113)

[g. Find a command to split the **poem.txt** file into different files each containing 2 lines. 9](#_Toc30774114)

[h. Use **sort** and **rev** to reverse the sorted contents of poem.txt and append the output to poem2.txt. 9](#_Toc30774115)

[i. Use at least two appropriate Linux commands to compare these two files (poem.txt and poem\_name.txt) and see if they are the same. 9](#_Toc30774116)

[j. Use **sort** to sort the content of poem.txt file in a random order and redirect the output to a new file called **poem2.txt**. 9](#_Toc30774117)

[k. Sort the **poem.txt** file, remove the duplicates and reverse the sorted contents and append the output to **poem2.txt**. 9](#_Toc30774118)

[l. Create an **alias** so rather than having to type the full command for k) you can type **yourSort**. 10](#_Toc30774119)

[Lab Activity 4 Bootloader 11](#_Toc30774120)

[a. Brief description of the Lab activity and what you did 11](#_Toc30774121)

[b. Boot pragma Linux with bochs 11](#_Toc30774122)

[c. Make a bootloader that displays your student details and diamond 11](#_Toc30774123)

[Commented bootloader code to display your student details and diamond 11](#_Toc30774124)

[Output from bochs showing student details and diamond 11](#_Toc30774125)

[Lab Activity 7 Buffer 12](#_Toc30774126)

[a. Explain what the code below does. 12](#_Toc30774127)

[b. Code to show error outputs. 12](#_Toc30774128)

[c. Code to create statistics on the document. 12](#_Toc30774129)

[d. Code to compare the documents. 12](#_Toc30774130)

[e. Code to create summary. 12](#_Toc30774131)

[Lab Activity 8 Cache Buffer 13](#_Toc30774132)

[a. Brief Description of Cache Buffer Activity 13](#_Toc30774133)

[b. Commented implementation of the cr\_handle function 13](#_Toc30774134)

[c. Comment updated code to show that each byte is being read, and when the buffer is being refilled. 13](#_Toc30774135)

[d. Commented updated code showing the required statistical information as well as how many times the words ‘Iran’, ‘Tehran’ and ‘email’ appear. 13](#_Toc30774136)

[Lab 10: The Cache Buffer from week 8 with system calls 14](#_Toc30774137)

[a. Brief description of the activity 14](#_Toc30774138)

[b. Changes the cache\_handle library from using the fopen, fread, fclose functions to the system call versions open, read, close 14](#_Toc30774139)

[c. Changes cache\_handle library to remove (as far as possible) the effects of caching on the library. 14](#_Toc30774140)

[Lab Activity 13 – Job control 15](#_Toc30774141)

[a. Description of the activity 15](#_Toc30774142)

[b. Menu System 15](#_Toc30774143)

[Lab Activity 14 – Linux command-line manipulation of processes 16](#_Toc30774144)

[a. Paragraph on disown and an alternative/ 16](#_Toc30774145)

[b. Description and example of using watch command 16](#_Toc30774146)

[c. Process manipulation 16](#_Toc30774147)

[Example(s) of how to start process 16](#_Toc30774148)

[Example(s) of how to suspend process 16](#_Toc30774149)

[Example(s) of how to run process in background 16](#_Toc30774150)

[Example(s) of how to run process in foreground and bring from background 16](#_Toc30774151)

[Example(s) of how to kill a process 16](#_Toc30774152)

[Lab Activity 15 IPC and Synchronisation 17](#_Toc30774153)

[a. Brief description of activity 17](#_Toc30774154)

[b. Modified semaphore example code so that the two processes output the song. 17](#_Toc30774155)

[c. Modified code to write Liza part to stderr and redirect the two parts to a file. 17](#_Toc30774156)

[References 18](#_Toc30774157)

# Lab Activity 1 – Operating Systems Tasks and Programming

## Comparison between the Harvard Architecture and Von Neumann Architecture

The first difference encountered between both digital computer architectures is in how the division between what is known as data and instructions occurs in each design. In the basis of Von Neumann’s design we have the concept of stored-program where programs and data are stored in the same memory (Khillar, S 2018), so there is no division. On the Harvard architecture we have a relay-based computer model which employed separate buses for data and instructions (Khillar, S 2018), which allowed that both reside in different locations in memory which will increase the memory cost and the number of buses being used but will improve significantly in performance. Certain instructions might only need one CPU clock cycle to run in the Harvard’s architecture where in Von Neumann’s it needs a minimum of 2.

It is in fact more cost efficient to produce a Von Neumann’s design based machine since it runs with less hardware than the Harvard’s, it is a matter of Performance – Cost. It also may be referenced that the simplicity of the construction of the Von Neumann’s architecture may be a point to take in mind since it is way higher than what the Harvard’s architecture.

## Programming activity

//This C++ code is used to take an input string and

// put the instructions in an array

#include <iostream>

#include <string>

#include <sstream>

int main() {

    //set up input string

    std::string input="orange lift right";

    bool flag = false;

    //set up map with instruction sets

    std::string time[4] = {"1second", "2seconds", "5seconds", "unlimited"};

    std::string move[5] = {"left", "right", "forward", "backward", "stop"};

    std::string object[5] = {"orange", "apple", "car", "bus", "diamond"};

    std::string action[6] = {"recognise", "eat", "see", "lift", "drop", "fetch"};

    std::string size[4] = { "small", "big", "little", "massive"};

    std::string location[3] = {"door", "kitchen", "table"};

    //initalise input stream

    std::stringstream currentstring(input);

    int count=-1;

    std::string instruction[10];

    //Repeatedly put instruction in string array

    while (currentstring.good())

    {

        count=count+1;

        currentstring >> instruction[count];

        std::cout << instruction[count] << " - - / - - " << sizeof(instruction)/sizeof(\*instruction) << std::endl;

    }

    //check for string structure

    if(count == 1)

    {

        //check if instructions are of type <Move> <Time>

        std::cout << "<Move> <Time>" << std::endl;

        for(int i = -1; i < 5; i++)

        {

            if(instruction[0] == move[i])

            {

                std::cout << "Check" << std::endl;

                break;

            }

            else

            {

                if(i == 4)

                {

                    std::cout << "First bad" << std::endl;

                    return 0;

                }

            }

        }

        for(int i = -1; i < 4; i++)

        {

            if(instruction[1] == time[i])

            {

                std::cout << "Check" << std::endl;

                break;

            }

            else

            {

                if(i == 3)

                {

                    std::cout << "Second bad" << std::endl;

                    return 0;

                }

            }

        }

    return 0;

    }

    else if(count == 2)

    {

        /\*check if instructions are of type <Object> <Action> <Time> ...\*/

        std::cout << "<Object> <Action> <Time> ..." << std::endl;

        //check if first string is object if not then check if it is location if not then error

        //Check object for first word

        for(int i = -1; i < 5; i++)

        {

            if(instruction[0] == object[i])

            {

                //check if it is action or size

                //check action for second word

                for(int i = -1; i < 6; i++)

                {

                    if(instruction[1] == action[i])

                    {

                        //check time for third word

                        for(int i = -1; i < 4; i++)

                        {

                            if(instruction[2] == time[i])

                            {

                                std::cout << "Check" << std::endl;

                                flag = true;

                                break;

                            }

                            if(flag == true)

                            {

                                break;

                            }

                        }

                    }

                    else

                    {

                        //check size for second word

                        for(int i = -1; i < 4; i++)

                        {

                            if(instruction[1] == size[i])

                            {

                                //check action for third word

                                for(int i = -1; i < 4; i++)

                                {

                                    if(instruction[2] == action[i])

                                    {

                                        std::cout << "Check" << std::endl;

                                        flag = true;

                                        break;

                                    }

                                    if(flag == true)

                                    {

                                        break;

                                    }

                                }

                            }

                            if(flag == true)

                            {

                                break;

                            }

                        }

                    }

                    if(flag == true)

                    {

                        break;

                    }

                }

                if(flag == true)

                {

                    break;

                }

            }

            else

            {

                //Check location for first word

                for(int i = -1; i < 3; i++)

                {

                    if(instruction[0] == location[i])

                    {

                        //check action for second word

                        for(int i = -1; i < 6; i++)

                        {

                            if(instruction[1] == action[i])

                            {

                                //check object for third word

                                for(int i = -1; i < 5; i++)

                                {

                                    if(instruction[2] == object[i])

                                    {

                                        std::cout << "Check" << std::endl;

                                        flag = true;

                                        break;

                                    }

                                    if(flag == true)

                                    {

                                        break;

                                    }

                                }

                            }

                            if(flag == true)

                            {

                                break;

                            }

                        }

                    }

                    if(flag == true)

                    {

                        break;

                    }

                }

            }

            if(flag == true)

            {

                break;

            }

            if(i == 4)

                std::cout << "Not the right syntax" << std::endl;

        }

        return 0;

    }

    else if(count == 3)

    {

        //check if instructions are of type <Move> <Time> <Move> <Time>

        std::cout << "<Move> <Time> <Move> <Time>" << std::endl;

        for(int i = -1; i < 5; i++)

        {

            if(instruction[0] == move[i])

            {

                std::cout << "Check" << std::endl;

                break;

            }

            else

            {

                if(i == 4)

                {

                    std::cout << "First bad" << std::endl;

                    return 0;

                }

            }

        }

        for(int i = -1; i < 4; i++)

        {

            if(instruction[1] == time[i])

            {

                std::cout << "Check" << std::endl;

                break;

            }

            else

            {

                if(i == 3)

                {

                    std::cout << "Second bad" << std::endl;

                    return 0;

                }

            }

        }

        for(int i = -1; i < 5; i++)

        {

            if(instruction[2] == move[i])

            {

                std::cout << "Check" << std::endl;

                break;

            }

            else

            {

                if(i == 4)

                {

                    std::cout << "Third bad" << std::endl;

                    return 0;

                }

            }

        }

        for(int i = -1; i < 4; i++)

        {

            if(instruction[3] == time[i])

            {

                std::cout << "Check" << std::endl;

                break;

            }

            else

            {

                if(i == 3)

                {

                    std::cout << "Fourth bad" << std::endl;

                    return 0;

                }

            }

        }

    return 0;

    }

    else

    {

        std::cout << "That syntax is not understandable by the robot.";

    }

    return 0;

}

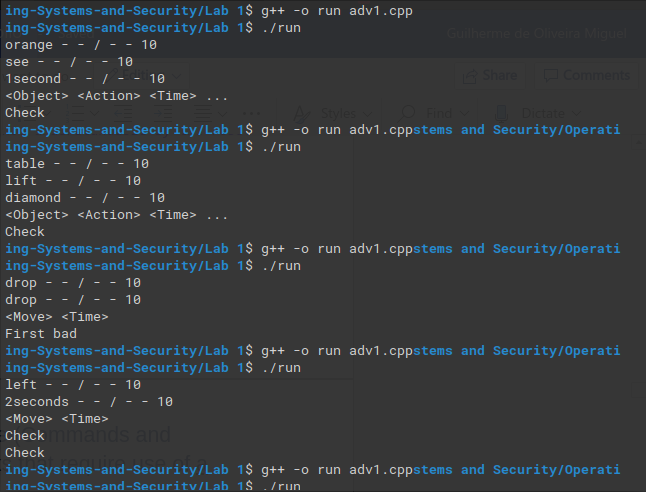


Fig 1 – Order Syntax checker 1

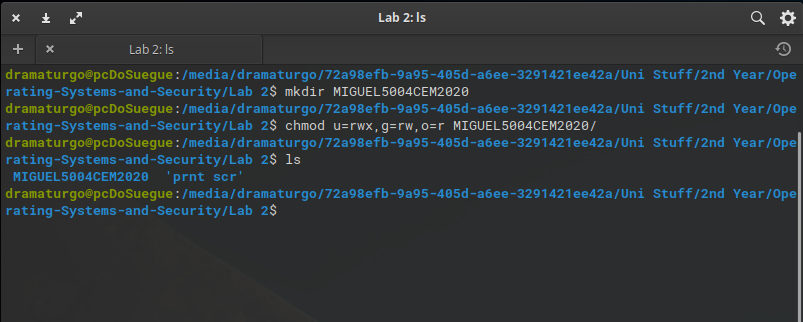


Fig 2 – Order Syntax checker 2

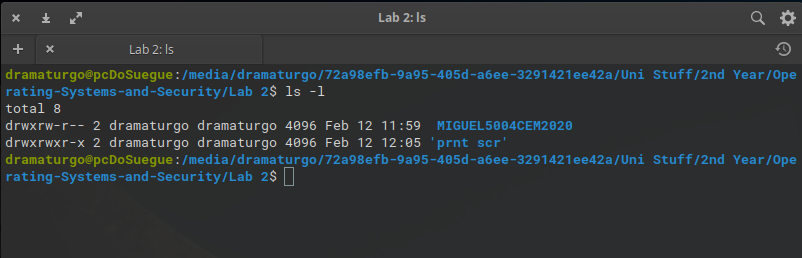
# Lab Activity 2 – Linux Command Line (Commands and outcomes from a series of small tasks that require use of a number of Linux commands)

## Tasks – Files

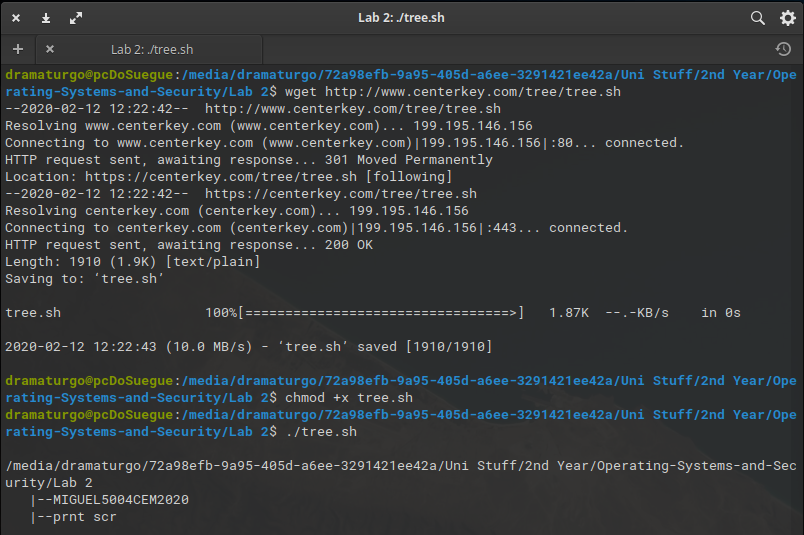
### Create a directory in your /home/5004CEM\_Session/<username> directory with a name made up of you second name followed by 5004CEM and the year (mine would be **ELSHAW5004CEM2020**). Make the directory **read/write/executable** only for you, **read/write** for your groups and **read** only for others.

 Fig 3 – Task a) directory creation and permissions manipulation

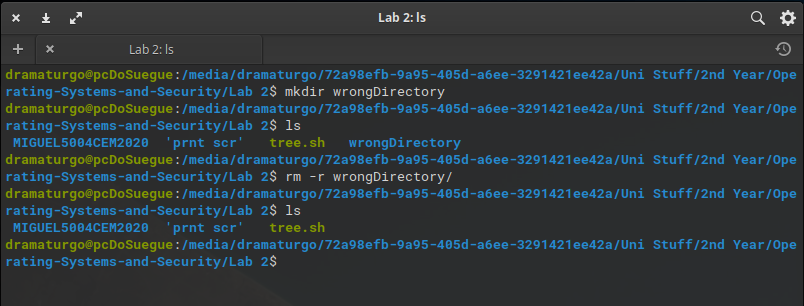
### Show evidence of this using the appropriate version of the **ls command**.

 Fig 4 – Task b) proof of task a) completion

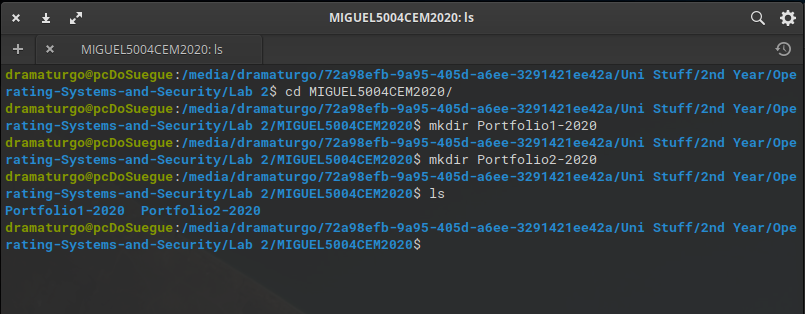
### Download the script http://www.centerkey.com/tree/tree.sh to your home directory using wget and make the file executable.

 Fig 5 – Task c) wget file download, executing the downloaded file

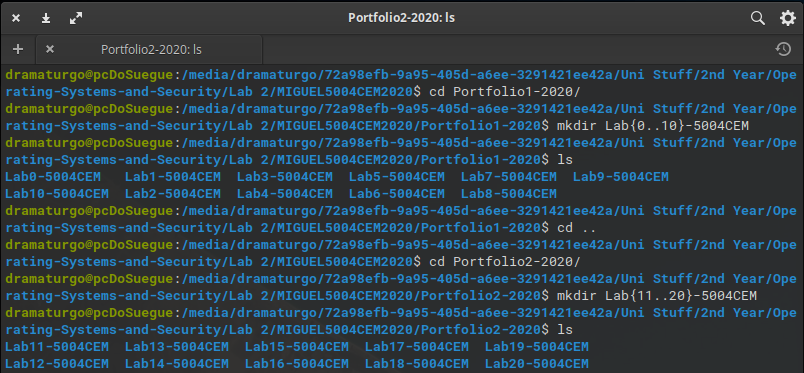
### Create a directory called **wrongDirectory**. You release it is not what you wanted so delete it.

 Fig 6 – Task d) wrong directory creation and deletion

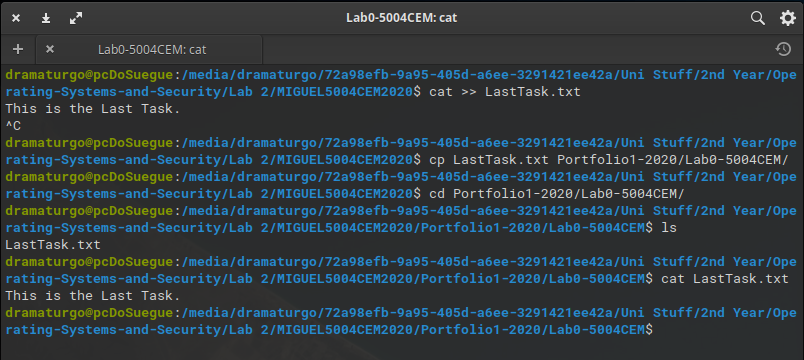
### Create Portfolio1-2020 and Portfolio2-2020 directories in the directory you created in part a.

 Fig 7 – Task e) portfolio directories creation

### Create numbered directories in the Portfolio1-2020 Directory (Lab0-5004CEM to Lab10-5004CEM) and in the Porfolio2-2020 Directory (Lab11-5004CEM to Lab20-5004CEM).

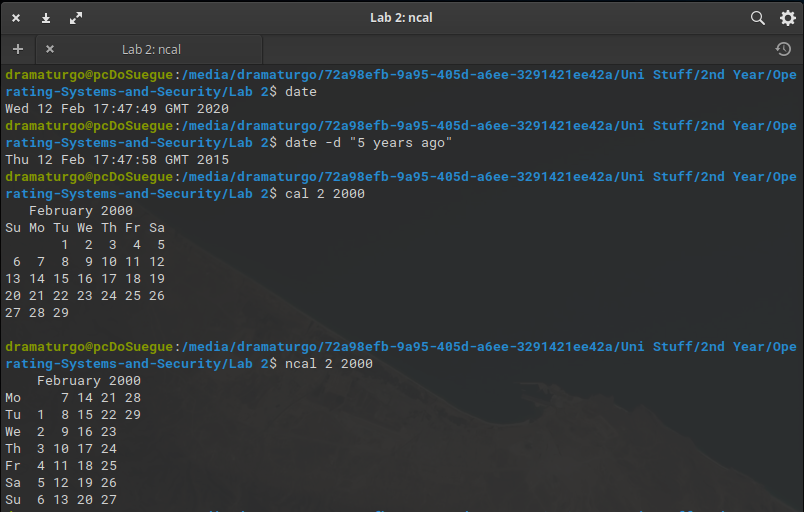
 Fig 8 – Task f) multiple directories creation

### In <YourSecondName>5004CEM2020 directory create a text file called **LastTask.txt** and then using the appropriate Linux command copy this document into Directory Lab0-5004CEM.

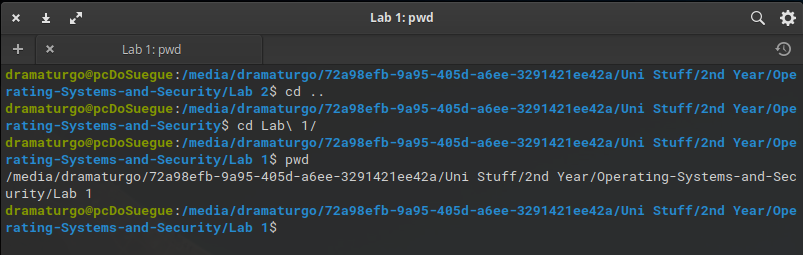
 Fig 9 – Task g) file copying to other directories

## Linux Commands – Mixed

### Using the date command show todays date and the time and date 5 years ago. Using the cal command show the month that you were born. Change this calendar to make Monday the first day of the week.

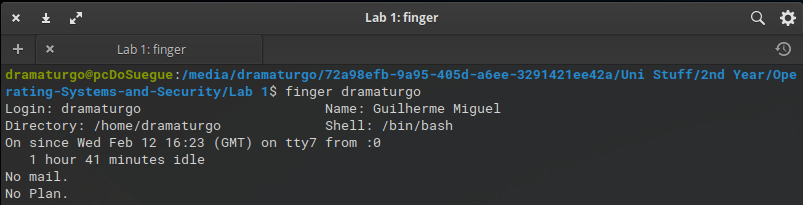
 Fig 10 – Task a) date and calendar commands

### Move into the lab1-5004CEM directory and use the appropriate command to show the current directory.

 Fig 11 – Task b) pwd command

### Display the time when a user (ab0487) last login, the user’s home directory, and the user account’s full name.

Since I am not doing this exercise on my personal machine I will just present the same asked but with my personal user.

 Fig 12 – Task c) user login times

### Find out how to prevent the effects of talk, write and wall from interrupting you. What command can you use?

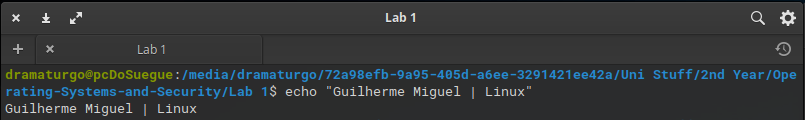
The mesg command blocks all of these notifications.

 Fig 13 – Task d) notification block

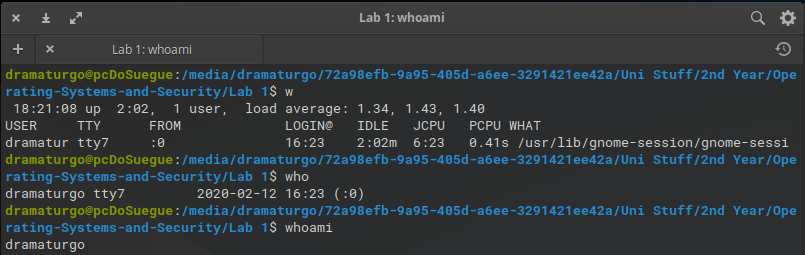
### Show the command to verify that www.coventry.ac.uk exists and can accept requests.

 Fig 14 – Task e) http requests

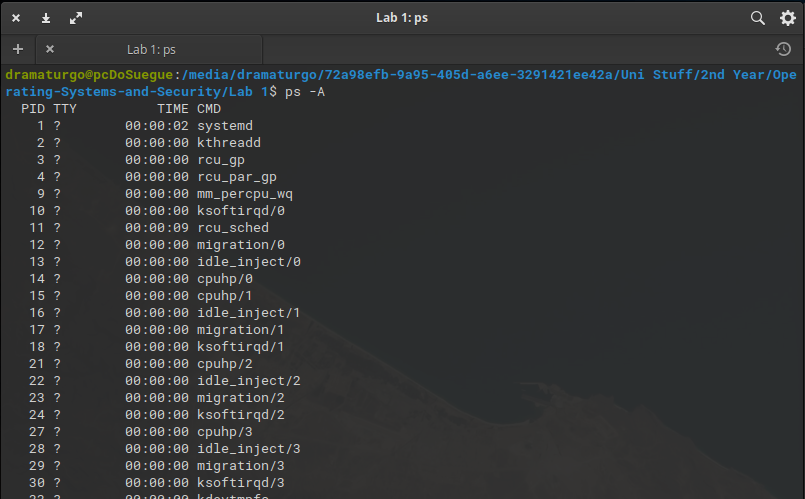
### Display your name and favourite programming language on the screen using the echo command.

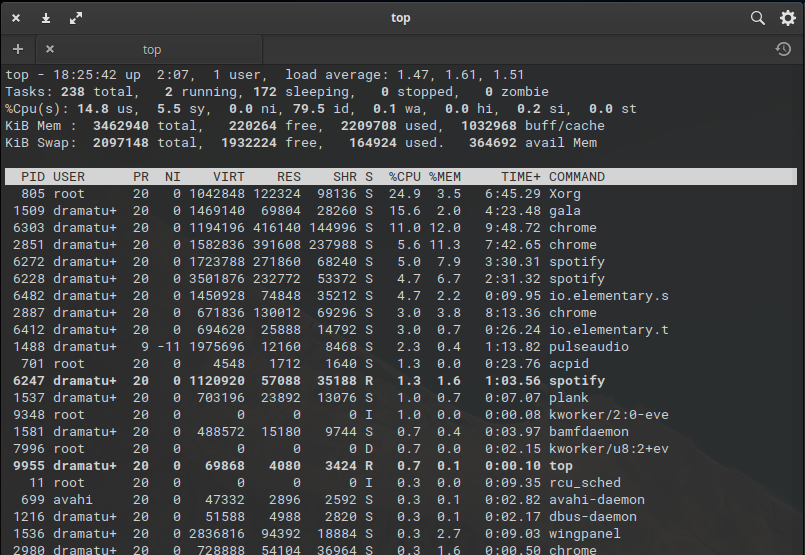
 Fig 15 – Task f) echo command

### Find out how you can display your username on the screen and at least two ways to display who is logged on.

 Fig 16 – Task g) w, who and whoami commands

### Use two ways to list the processes that are running.

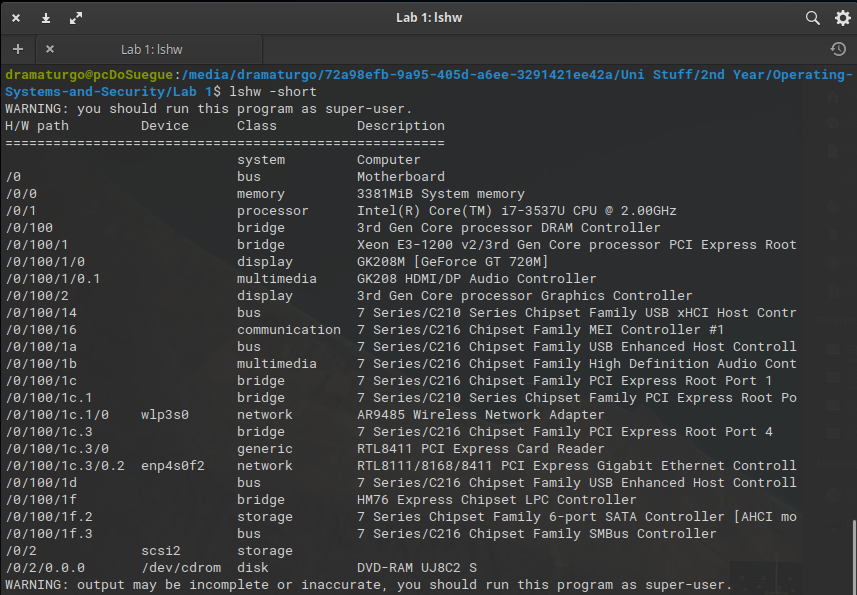
 Fig 17.1 – Task h) process list with ps –A

 Fig 17.2 – Task h) process list with top

### What are the differences between the Linux commands copy (cp), rename and move?

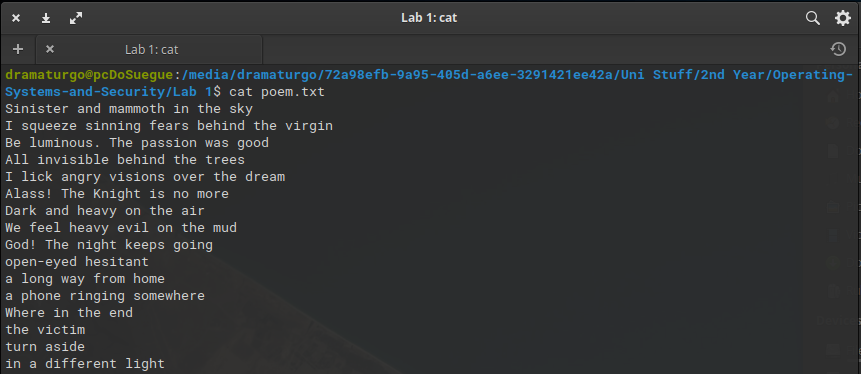
Rename only renames the file, so the file is still in the same directory. Copy copies the file to another directory, therefore there will be two files exactly identical in two different directories, where move is different since it deletes the original copy of the file by just moving it to another directory.

### With a single command, how would you get systems information such as processes, memory, paging and CPU activity?

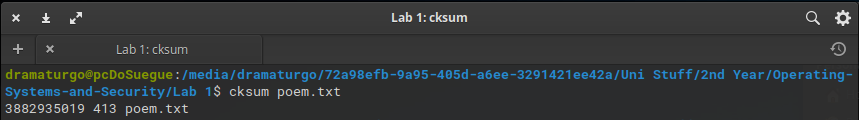
 Fig 18 – Task j) lshw command, -short to just get a short summary

## Tasks – Document Manipulation

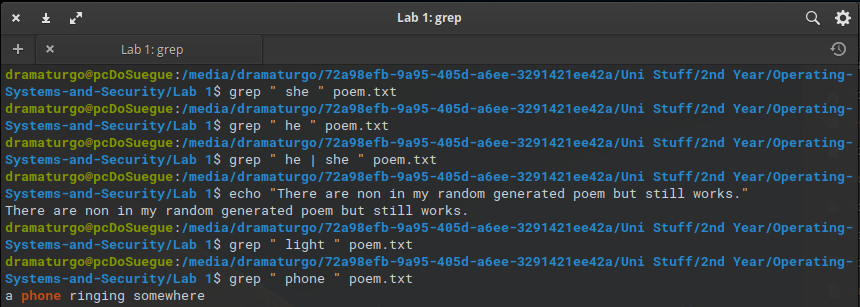
### Use **cat** to show the contents of the file.

 Fig 19 – Task a) cat to open text files

### Use an appropriate command to display the CRC checksum and byte count of the file.

 Fig 20 – Task b) cksum to check byte count and CRC checksum

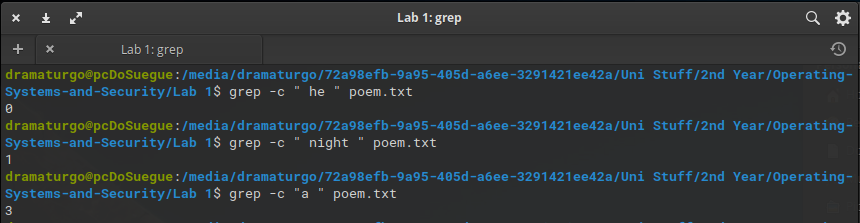
### Use **grep** to show only lines not containing the words "she" or "he". Lines contain both “she” and “he” should be shown.

 Fig 21 – Task c) grep to search certain strings

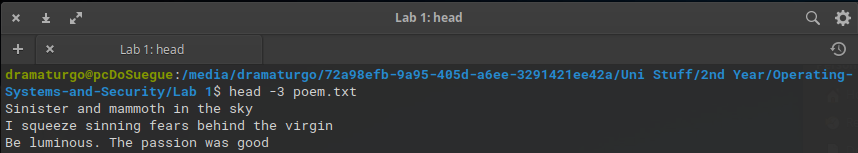
### Use **grep** to show the 5 lines above a line containing the text ‘the’.

 Fig 22 – Task d) grep to select lines

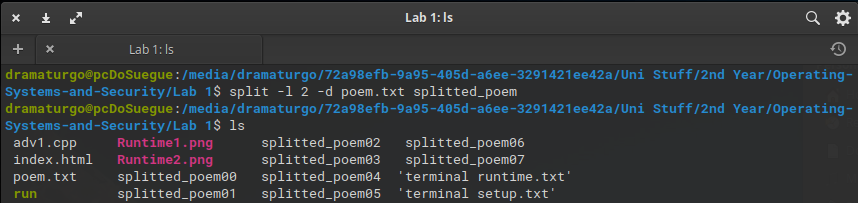
### Using Linux commands you should count the lines containing “she” and “he” but not both and display the line numbers that “she” and “he” but not both appear on in the original document.

 Fig 23 – Task e) grep to count number of word appearances

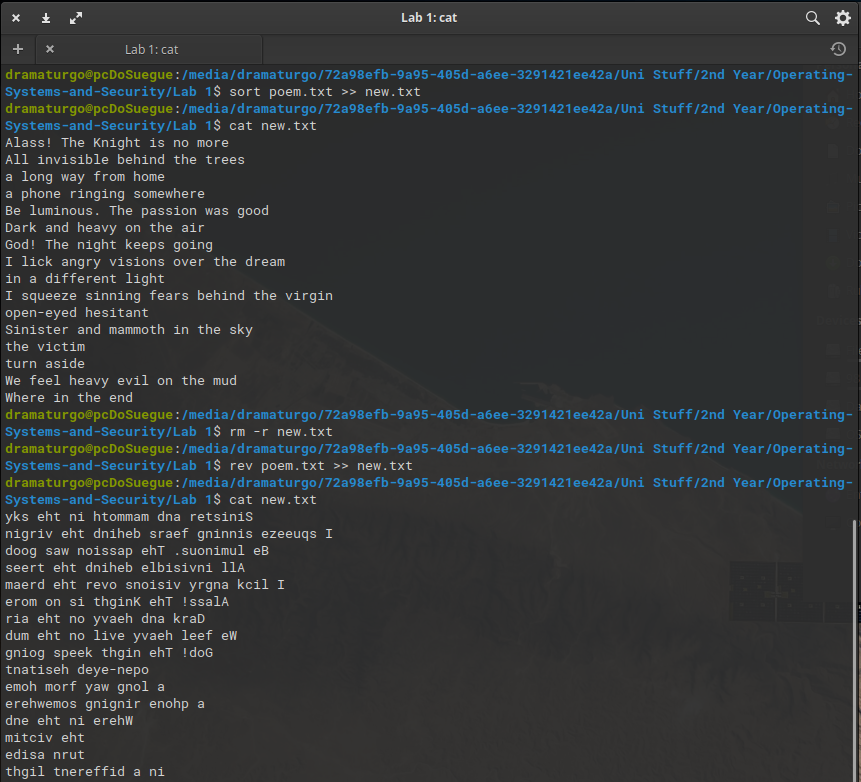
### Find a command to list the top 3 lines of the **poem.txt** file and then the bottom line of these top 3.

 Fig 24 – Task f) head to get parts of a file

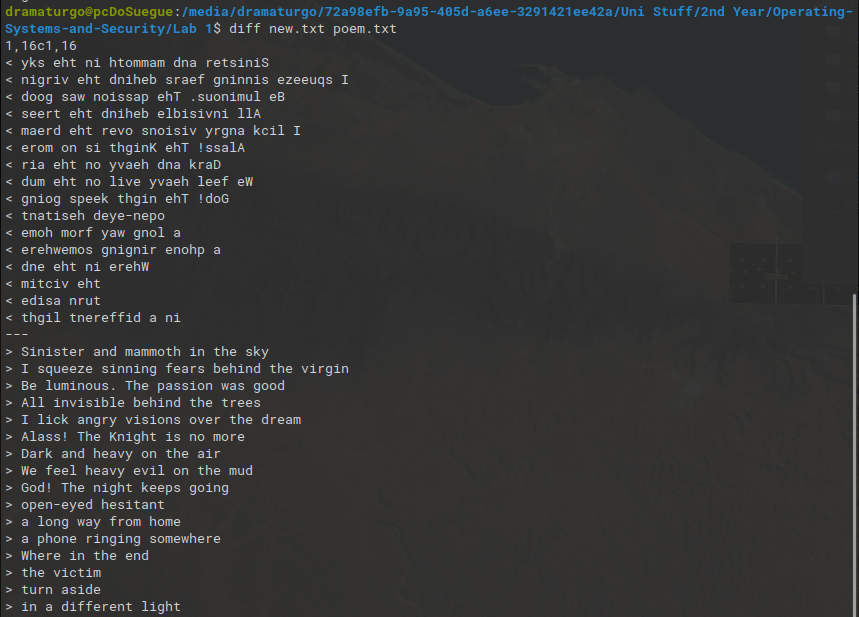
### Find a command to split the **poem.txt** file into different files each containing 2 lines.

 Fig 25 – Task g) split to split a txt file

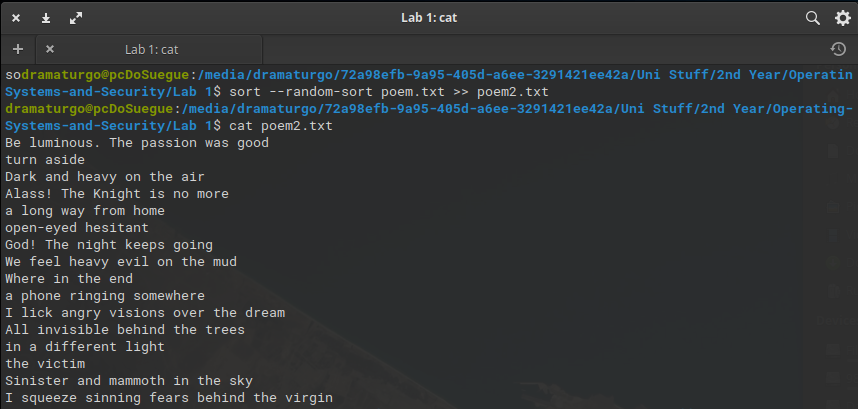
### Use **sort** and **rev** to reverse the sorted contents of poem.txt and append the output to poem2.txt.

 Fig 26 – Task h) sort and rev commands

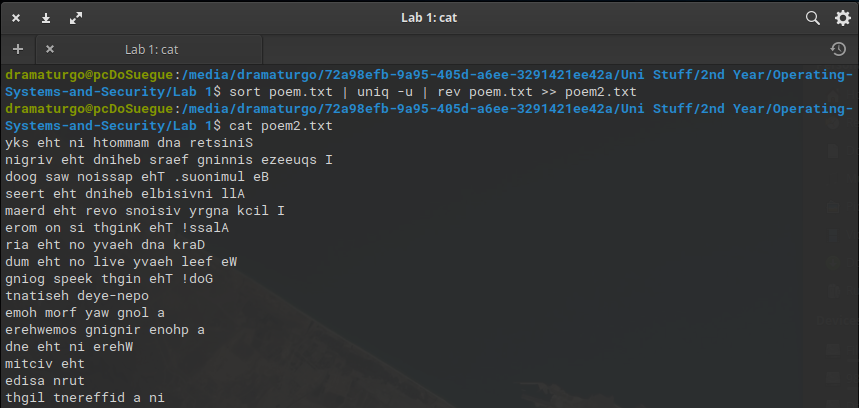
### Use at least two appropriate Linux commands to compare these two files (poem.txt and poem\_name.txt) and see if they are the same.

 Fig 27 – Task i) diff to compare text files

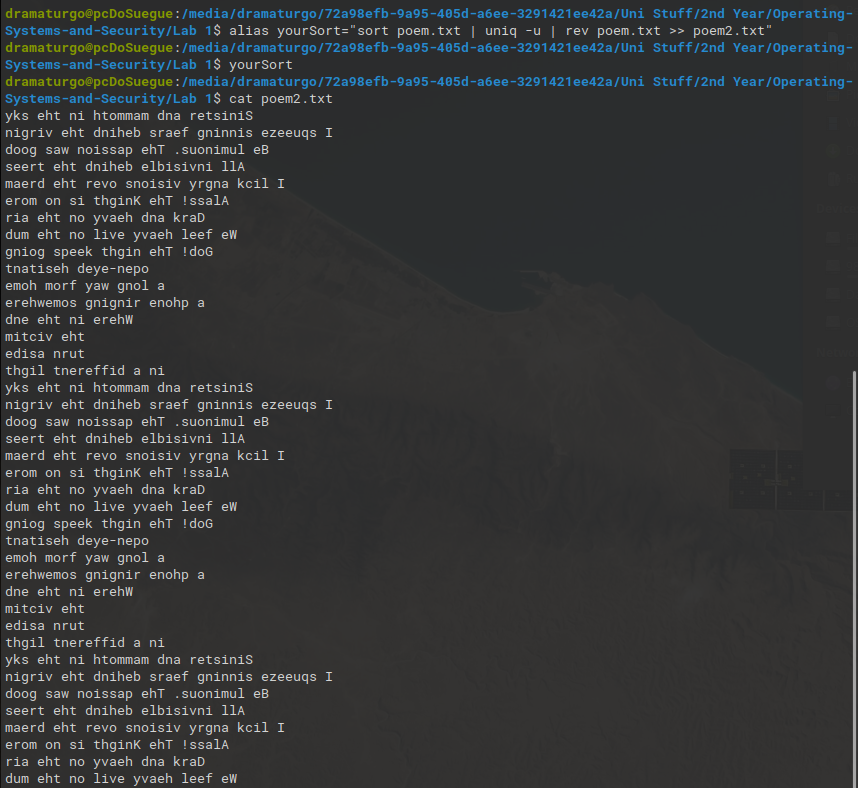
### Use **sort** to sort the content of poem.txt file in a random order and redirect the output to a new file called **poem2.txt**.

 Fig 28 – Task j) sort with –random-sort to randomize the file

### Sort the **poem.txt** file, remove the duplicates and reverse the sorted contents and append the output to **poem2.txt**.

 Fig 29 – Task k) sort rev and uniq in same line

### Create an **alias** so rather than having to type the full command for k) you can type **yourSort**.

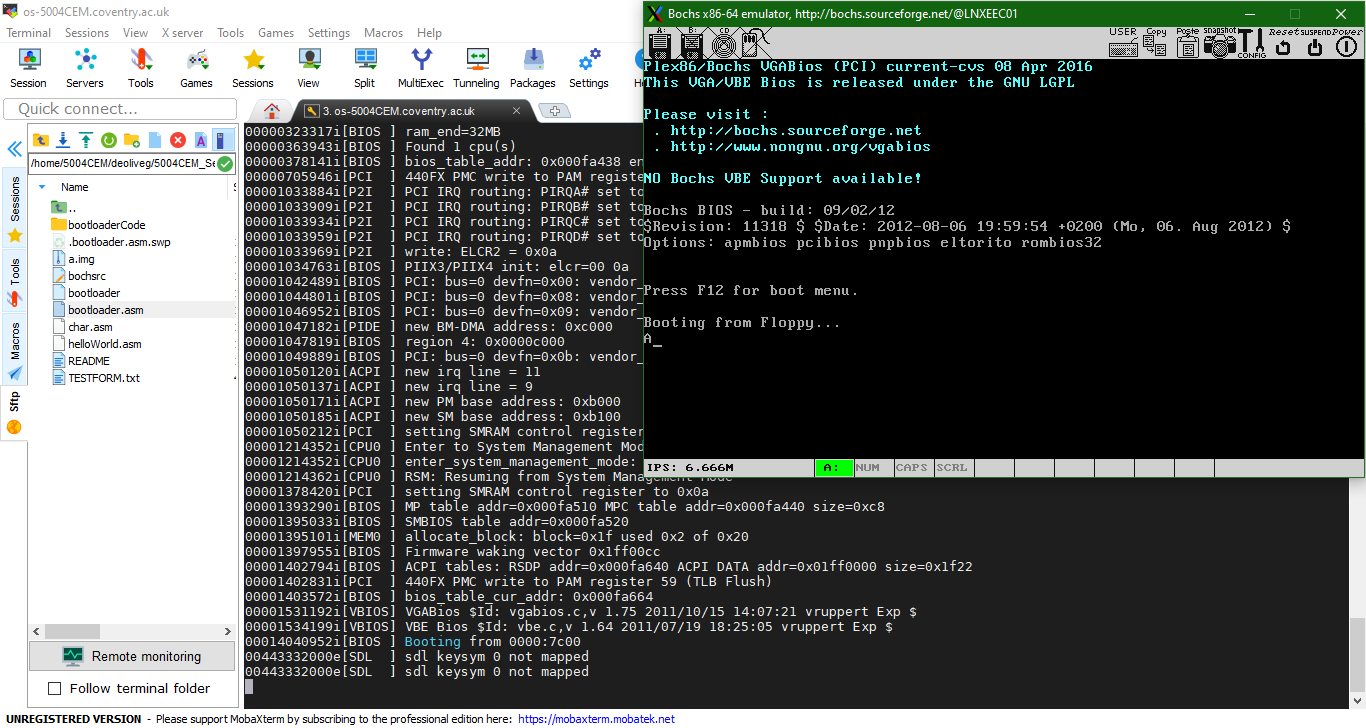
 Fig 30 – Task l) alias creation

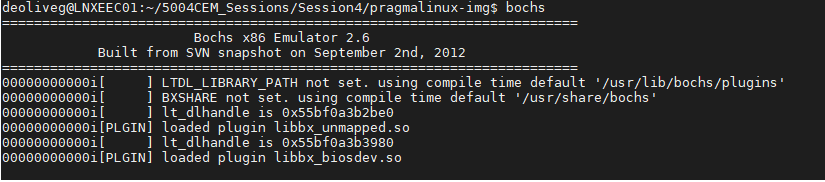
# Lab Activity 4 Bootloader

## Brief description of the Lab activity and what you did

The following lab’s activity explores the idea of Bootloader, how it functions and how is it deployed on the computer. There is how to create assembly code to display messages on screen by booting the created bootloader on a virtual machine.

## Boot pragma Linux with bochs

Fig 31 – Task a) Booting pragmalinux

Fig 32 – Task a) Command to run the pragmalinux img

## Make a bootloader that displays your student details and diamond

### Commented bootloader code to display your student details and diamond

[BITS 16]

[ORG 0x7C00]

top:

        ;; Put 0 into ds (data segment)

        ;; Can't do it directly

        mov ax,0x0000

        mov ds,ax

        ;; si is the location relative to the data segment of the

        ;; string/char to display

        mov si, Name

        call writeString ; See below

        int 21h

        mov si, Course

        call writeString ; See below

        int 21h

        mov si, Os

        call writeString ; See below

        int 21h

        mov si, size ; print instructions for user

        call writeString

; grabbing user input and putting it on ah

        mov ah, 00h

        int 16h

        sub al, 48 ; each number is equal to its ascii code - 48

        mov ch, al; num of top of diamond lines

        mov cl, 1 ; num of starting stars

        mov dh, ch

        mov dl, cl

        mov si, linebreak

        call writeString

        call rowLoop

        call rowLoopReverse

        jmp $ ; Spin

writeString:

        mov ah,0x0E ; Display a chacter (as before)

        mov bh,0x00

        mov bl,0x07

nextchar:

        Lodsb ; Loads [SI] into AL and increases SI by one

        ;; Effectively "pumps" the string through AL

        cmp al,0 ; End of the string?

        jz done

        int 0x10

        jmp nextchar

rowLoop:

        lineSpaceLoop:  ; writes the required num of spaces to screen

                      mov si, spaceChar

                      call writeString

                      dec dh

                      cmp dh, 0

                      jg lineSpaceLoop

        lineStarLoop:   ; writes the required num of stars to screen

                      mov si, starChar

                      call writeString

                      dec dl

                      cmp dl, 0

                      jg lineStarLoop

                      mov si, linebreak

                      call writeString

        ; every time it loops will decrease one space but add 2 stars

        dec ch

        add cl, 2

        mov dl, cl

        mov dh, ch

        cmp dh, 1

        jg rowLoop

rowLoopReverse:

; here the same as above happens but in reverse

        lineSpaceLoopReverse:

                      mov si, spaceChar

                      call writeString

                      dec dh

                      cmp dh, 0

                      jg lineSpaceLoopReverse

        lineStarLoopReverse:

                      mov si, starChar

                      call writeString

                      dec dl

                      cmp dl, 0

                      jg lineStarLoopReverse

                      mov si, linebreak

                      call writeString

        inc ch

        sub cl, 2

        mov dl, cl

        mov dh, ch

        cmp dh, 3

        jl rowLoopReverse

done:

        ret

        Name db 'Guilherme Miguel',13,10,0

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

        Os db 'Ubuntu Linux',13,10,0 ; Null-terminated

        size db 'Input Diamond top part line size'

        spaceChar db ' ',0 ;

        starChar db '\*',0

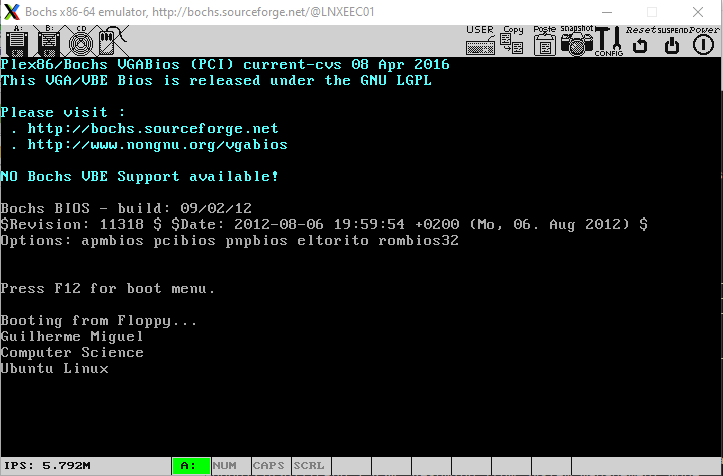
        linebreak db '',13,10,0

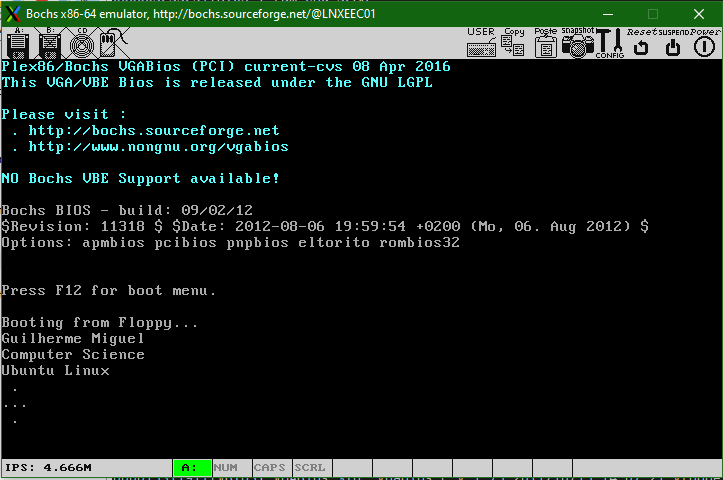
        times 510-($-$$) db 0

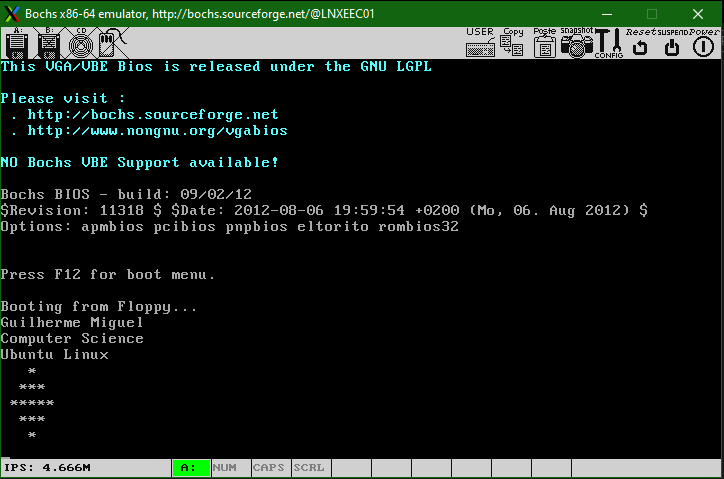
        dw 0xAA55

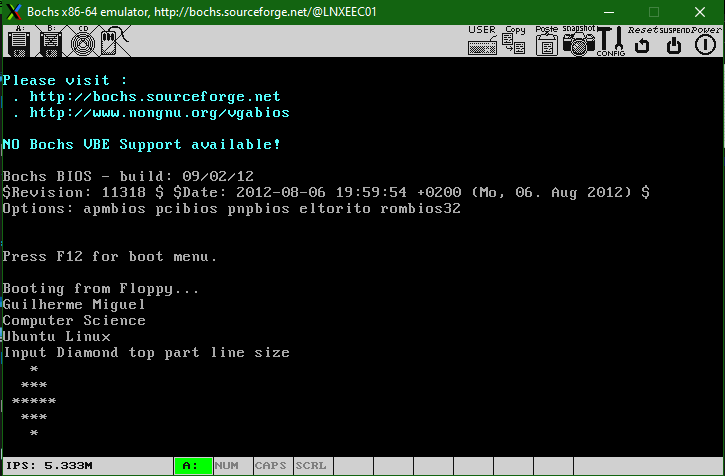
### 

### Output from bochs showing student details and diamond

Fig 33 – Task b) Just strings no diamond

Fig 34 – Task b) Student details plus diamond no loops

Fig 35 – Task b) Full exercise no user input

Fig 36 – Task b) Full exercise with user input

# Lab Activity 7 Buffer

## Explain what the code below does.

The portion of C code starts by including several external dependencies to the code and defines both the size of the buffer and an output mode, these are what we can call macros just to keep the code cleaner.

In the main function we can find firstly some variable definitions, two integers to store the input and output files, another integer which will work as a character counter for the buffer, and an array of characters which will have enough room for the size of the buffer.

Since the main function uses command line arguments we have a number of arguments being inputted which are pointed to by the argument vector, when we run this code we should run it by  *./name\_of\_file inputFile.txt outputFile.txt* so there will always need to be just 3 arguments, therefore if more are encountered the code exits. After that it opens the input file onto the in\_fd variable, if a negative value is given then the codes exits as well. The code now creates the output file, if a negative value is returned from creation then the code exits.

Now the algorithm will loop through the characters available in the buffer, the input file, if the buffer has a negative value of characters the algorithm exits. The buffer writes each character one by one to the output file. This will continue during the loop until the buffer is empty and the file as been copied. It will then close both files and exit successfully.

## Code to show error outputs.

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#define BUF\_SIZE 500

#define OUTPUT\_MODE 0700

int main(int argc, char \*argv[]) {

    int in\_fd, out\_fd;

    int rd\_size = 1, wr\_size;

    char buf[BUF\_SIZE];

    if (argc != 3)

    {

        //Error number one, invalid argument number

        printf("Number of arguments cannot be greater than 3, insert a input and output after the execute file.");

        exit(1);

    }

    in\_fd = open(argv[1], O\_RDONLY);

    if (in\_fd < 0)

    {

        //Error number 2, invalid file or error in file open

        printf("Input file invalid.");

        exit(2);

    }

    out\_fd = creat(argv[2], OUTPUT\_MODE);

    if (out\_fd < 0)

    {

        //Error number 3, invalid file or error in file create

        printf("Output file invalid.");

        exit(3);

    }

    while (rd\_size > 0) {

        rd\_size = read(in\_fd, buf, BUF\_SIZE);

        if (rd\_size <0)

        {

            //Error number 4, invalid number of characters in buffer

            printf("Buffer has an invalid number of characters. Restart code or check input/output files.");

            exit(4);

        }

        wr\_size = write(out\_fd, buf, rd\_size);

        if (wr\_size<=0)

        {

            close(in\_fd);

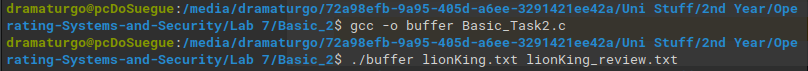
            close(out\_fd);

            exit(5);

        }

    }

}

 Fig 37 – Task b) error outputs

## Code to create statistics on the document.

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#define BUF\_SIZE 600

#define OUTPUT\_MODE 0700

int main(int argc, char \*argv[]) {

    int in\_fd, out\_fd;

    int rd\_size = 1, wr\_size;

    char buf[BUF\_SIZE];

    bool flag;

    int bufCount = 0;

    int countVowel = 0;

    int charCount = 0;

    int sentenceCount = 0;

    int asciiVowel[10] = {65, 69, 73, 79, 85, 97, 101, 105, 111, 117};

    if (argc != 3)

    {

        printf("Number of arguments cannot be greater than 3, insert a input and output after the execute file.\n");

        exit(1);

    }

    in\_fd = open(argv[1], O\_RDONLY);

    if (in\_fd < 0)

    {

        printf("Input file invalid.\n");

        exit(2);

    }

    out\_fd = creat(argv[2], OUTPUT\_MODE);

    if (out\_fd < 0)

    {

        printf("Output file invalid.\n");

        exit(3);

    }

    while (rd\_size > 0) {

        rd\_size = read(in\_fd, buf, BUF\_SIZE);

        //How many characters being read at a time

        printf("\nReading %d", rd\_size);

        printf(" characters.\n");

        int length = sizeof(asciiVowel)/sizeof(asciiVowel[0]);

        //Increments charCount if it is not a space(ascii value 32)

        for(int i = 0; i < rd\_size; i++)

        {

            if(buf[i] != 32)

            {

                if(buf[i] == 46 || buf[i] == 33 || buf[i] == 63)

                    sentenceCount += 1;

                charCount += 1;

                int j = 0;

                for(j; j < length; j++)

                {

                    if(buf[i] == asciiVowel[j])

                        countVowel += 1;

                }

            }

        }

        //Every time buffer fills up counter rises

        if(rd\_size == 600)

        {

            printf("Buffer filled.\n");

            bufCount += 1;

        }

        if (rd\_size <0)

        {

            printf("\nBuffer has an invalid number of characters. Restart code or check input/output files.\n");

            exit(4);

        }

        wr\_size = write(out\_fd, buf, rd\_size);

        if (wr\_size<=0)

        {

            close(in\_fd);

            close(out\_fd);

            printf("\n--/--/--/--/--/--/--\n");

            printf("\nBuffer filled %d times.\n", bufCount);

            printf("\nSEntence %d\n", sentenceCount);

            printf("\nCharacters discounting spaces %d\n", charCount);

            printf("\nVowel characters %d\n", countVowel);

            printf("\nConsonants and other characters besides space %d\n", (charCount - countVowel));

            printf("\n--/--/--/--/--/--/--\n");

            exit(5);

        }

    }

}

Fig 38 – Task c) file statistics

## Code to compare the documents.

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#define BUF\_SIZE 600

#define OUTPUT\_MODE 0700

int main(int argc, char \*argv[]) {

    int in\_fd, in\_fd2, out\_fd;

    int rd\_size = 1, wr\_size;

    int rd\_size2;

    char buf[BUF\_SIZE];

    char buf2[BUF\_SIZE];

    bool flag;

    int bufCount = 0;

    int countVowel = 0;

    int charCount = 0;

    int sentenceCount = 0;

    int asciiVowel[10] = {65, 69, 73, 79, 85, 97, 101, 105, 111, 117};

    if (argc != 4)

    {

        printf("Number of arguments cannot be greater than 3, insert a input and output after the execute file.\n");

        exit(1);

    }

    in\_fd = open(argv[1], O\_RDONLY);

    if (in\_fd < 0)

    {

        printf("Input file invalid.\n");

        exit(2);

    }

    out\_fd = creat(argv[2], OUTPUT\_MODE);

    if (out\_fd < 0)

    {

        printf("Output file invalid.\n");

        exit(3);

    }

    while (rd\_size > 0) {

        rd\_size = read(in\_fd, buf, BUF\_SIZE);

        int length = sizeof(asciiVowel)/sizeof(asciiVowel[0]);

        //Increments charCount if it is not a space(ascii value 32)

        for(int i = 0; i < rd\_size; i++)

        {

            if(buf[i] != 32)

            {

                if(buf[i] == 46 || buf[i] == 33 || buf[i] == 63)

                    sentenceCount += 1;

                charCount += 1;

                int j = 0;

                for(j; j < length; j++)

                {

                    if(buf[i] == asciiVowel[j])

                        countVowel += 1;

                }

            }

        }

        //Every time buffer fills up counter rises

        if(rd\_size == 600)

        {

            bufCount += 1;

        }

        if (rd\_size <0)

        {

            printf("\nBuffer has an invalid number of characters. Restart code or check input/output files.\n");

            exit(4);

        }

        wr\_size = write(out\_fd, buf, rd\_size);

        if (wr\_size<=0)

        {

            close(in\_fd);

            close(out\_fd);

        }

    }

    //comparisson of the 2 files here

    rd\_size = 1, wr\_size;

    in\_fd = open(argv[1], O\_RDONLY);

    if (in\_fd < 0)

    {

        printf("\nFirst input file invalid.\n");

        exit(5);

    }

    in\_fd2 = open(argv[3], O\_RDONLY);

    if (in\_fd2 < 0)

    {

        printf("\nSecond input file invalid.\n");

        exit(6);

    }

    //Using 2 buffers we can fill each one and check if the files have the exact same chars in every exact same position

    while(rd\_size > 0)

    {

        //read in both files into the buffer 20 chars at a time

        rd\_size = read(in\_fd, buf, BUF\_SIZE);

        rd\_size2 = read(in\_fd2, buf2, BUF\_SIZE);

        for(int i = 0; i < rd\_size; i++)

        {

            //if at an instance a char is different is the same position the files are different

            if(buf[i] != buf2[i])

            {

                printf("\nFiles %s and %s are different.\n\n", argv[1], argv[3]);

                exit(7);

            }

        }

        if (rd\_size<=0)

        {

            close(in\_fd);

            close(in\_fd2);

            printf("\n--/--/--/--/--/--/--\n");

            printf("\nFile %s and %s are equal.\n", argv[1], argv[2]);

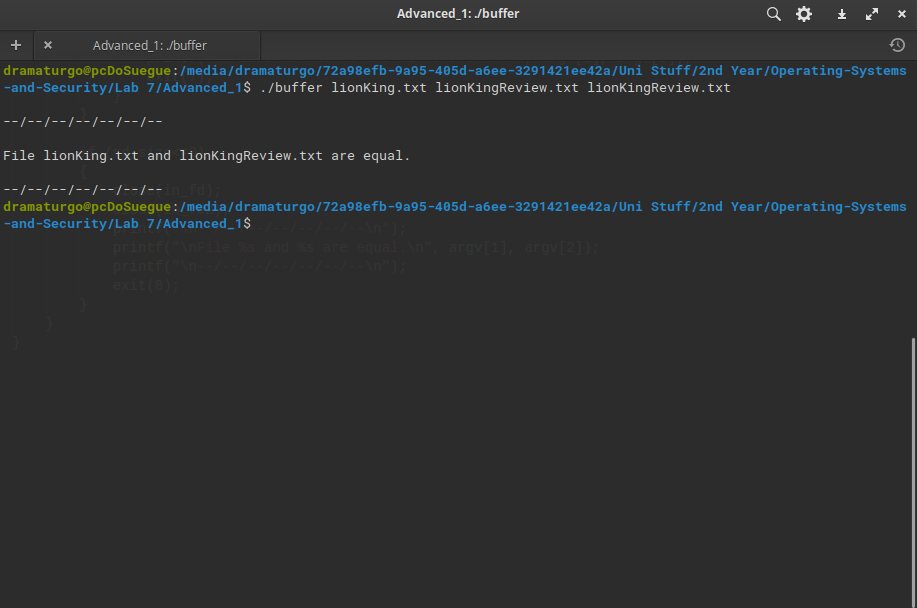
            printf("\n--/--/--/--/--/--/--\n");

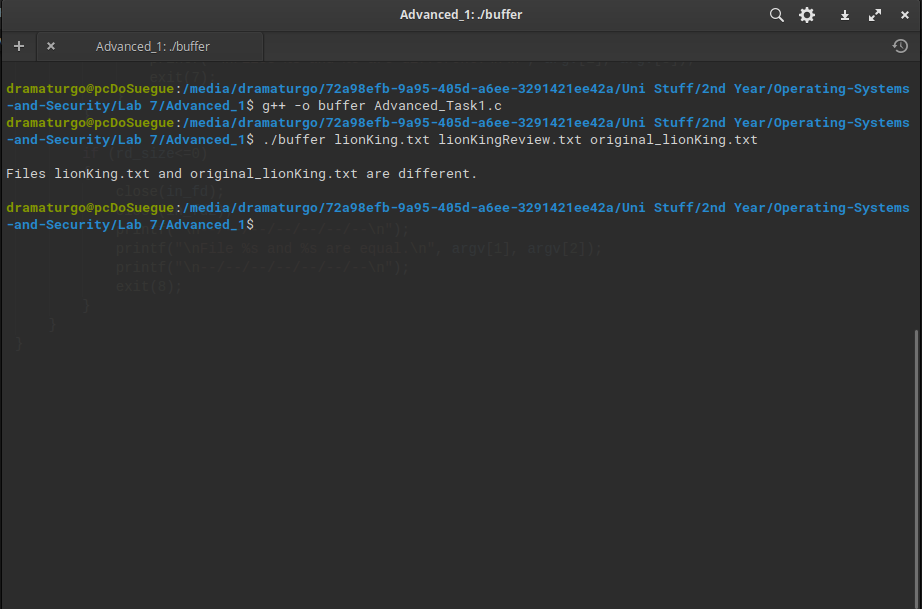
            exit(8);

        }

    }

}

 Fig 39 – Task Advanced a) file equality

 Fig 40 – Task Advanced b) changing a char to create file different output

## Code to create summary.

undone

# Lab Activity 8 Cache Buffer

## Brief Description of Cache Buffer Activity

This lab reviews the idea of Cache and Buffer by showing how both work in a code environment. It shows how to access a file and print each character on screen by using a buffer to fill in the characters to print. It also contains code used to analyse the file and check certain info such as words or byte count.

## Commented implementation of the return\_character function

//FILE CACHE\_HANDLE.H

#include <stdio.h>

#include <stdlib.h>

//The internals of this struct aren't important

//from the user's point of view

typedef struct{

  FILE\* file;        //File being read

  int bufferLength;  //Fixed buffer length

  int alongBuffer;  //Current point in the buffer

  int charToRet;    //alongBuffer when it is being returned

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

                     //  same length as "bufferlength"

} bufferStruct;

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

bufferStruct\* file\_open(char\* filename, int buffersize);

//Close an open file

void file\_close(bufferStruct\* buff);

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

char return\_character(bufferStruct\* buff);

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

//Refill an empty buffer.  Not intended for users

int buffer\_refill(bufferStruct\* buff);

//FILE CACHE\_HANDLE.C

#include "cache\_handle.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 buffer\_refill(bufferStruct\* buff){

  //Refills a buffer

  //Only works when completely used buffer

  if(buff->alongBuffer!=buff->bufferLength)

    return 0;

  else{

    buff->alongBuffer=0;

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

    //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 file\_close(bufferStruct\* buff){

  free(buff->buffer);

  fclose(buff->file);

}

bufferStruct\* file\_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;

  }

  bufferStruct\* initBuffer=(bufferStruct\*)malloc(sizeof(bufferStruct));

  initBuffer->file=f;

  initBuffer->bufferLength=buffersize;

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

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

  buffer\_refill(initBuffer);

  return initBuffer;

}

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

char return\_character(bufferStruct\* buff){

  // First we need to check if the current point in the buffer is the last one that the buffer contains

  // If so then we need to reload the buffer with the next 20 characters and get alongBuffer to point back to the

  //beggining of the buffer

  if(buff -> alongBuffer == buff -> bufferLength)

  {

    buffer\_refill(buff);

    buff -> alongBuffer = EOF;

  }

  // If it is not the last character in the buffer then we need to go to the next one so it can be

  //printed on the next run

  else

    buff -> alongBuffer += 1;

  // In the end we just need to return the character that was being pointed to at the beggining

  return buff -> buffer[(buff -> alongBuffer) - 1];

}

//FILE CACHE\_PRINTER.C

#include "cache\_handle.h"

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

int main(){

  char character;

  //Open a file

  bufferStruct\* f = file\_open("text",20);

  printf("\n\n");

  //While there are useful bytes coming from it

  while((character=return\_character(f))!=EOF)

    //Print them

    printf("%c",character);

  printf("\n\n");

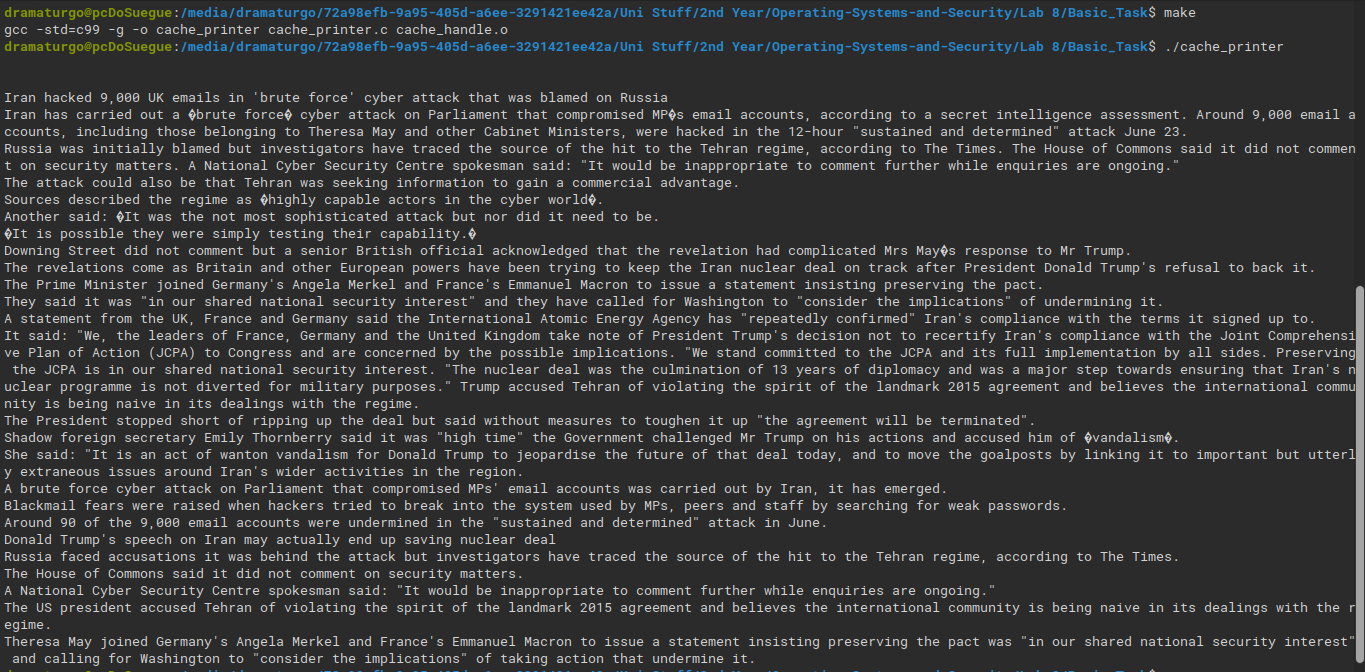
  //Then close the file

  file\_close(f);

  //And finish

  return 0;

}

 Fig 41 – Task a) printing file contents

## Comment updated code to show that each byte is being read, and when the buffer is being refilled.

//FILE CACHE\_HANDLE.H

#include <stdio.h>

#include <stdlib.h>

//The internals of this struct aren't important

//from the user's point of view

typedef struct{

  FILE\* file;        //File being read

  int bufferLength;  //Fixed buffer length

  int alongBuffer;  //Current point in the buffer

  int charToRet;    //alongBuffer when it is being returned

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

                     //  same length as "bufferlength"

  int refillCount;

  int bytesRead;

} bufferStruct;

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

bufferStruct\* file\_open(char\* filename, int buffersize);

//Close an open file

void file\_close(bufferStruct\* buff);

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

char return\_character(bufferStruct\* buff);

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

//Refill an empty buffer.  Not intended for users

int buffer\_refill(bufferStruct\* buff);

//FILE CACHE\_HANDLE.C

#include "cache\_handle.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 buffer\_refill(bufferStruct\* buff){

  //Refills a buffer

  //Only works when completely used buffer

  if(buff->alongBuffer!=buff->bufferLength)

    return 0;

  else{

    buff->alongBuffer=0;

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

    //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!

    printf("\nBuffer refilled\n");

    buff->refillCount += 1;

    return len;

  }

}

void file\_close(bufferStruct\* buff){

  free(buff->buffer);

  fclose(buff->file);

}

bufferStruct\* file\_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;

  }

  bufferStruct\* initBuffer=(bufferStruct\*)malloc(sizeof(bufferStruct));

  initBuffer->file=f;

  initBuffer->bufferLength=buffersize;

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

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

  initBuffer->refillCount = 0;

  initBuffer->bytesRead = 0;

  buffer\_refill(initBuffer);

  return initBuffer;

}

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

char return\_character(bufferStruct\* buff){

  // First we need to check if the current point in the buffer is the last one that the buffer contains

  // If so then we need to reload the buffer with the next 20 characters and get alongBuffer to point back to the

  //beggining of the buffer

  if(buff -> alongBuffer == buff -> bufferLength)

  {

    buffer\_refill(buff);

    buff -> alongBuffer = EOF;

  }

  // If it is not the last character in the buffer then we need to go to the next one so it can be

  //printed on the next run

  else

    buff -> alongBuffer += 1;

  //  Counting the number of bytes being read into the buffer

  buff->bytesRead += sizeof((buff->alongBuffer) - 1);

  // In the end we just need to return the character that was being pointed to at the beggining

  return buff -> buffer[(buff -> alongBuffer) - 1];

}

//FILE CACHE\_PRINTER.C

#include "cache\_handle.h"

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

int main(){

  char character;

  //Open a file

  bufferStruct\* f = file\_open("text",20);

  printf("\n\n");

  //While there are useful bytes coming from it

  while((character=return\_character(f))!=EOF)

    //  Print them, the star tells if jus one character was printed, if it has more than one char between \*

    //it has a bug in the code

    printf("%c\*",character);

  //Printing number of bytes that were read in total and buffer refills

  printf("\n\n %d Bytes read.\n", f->bytesRead);

  printf("\n Buffer refilled %d times.\n", f->refillCount);

  printf("\n\n");

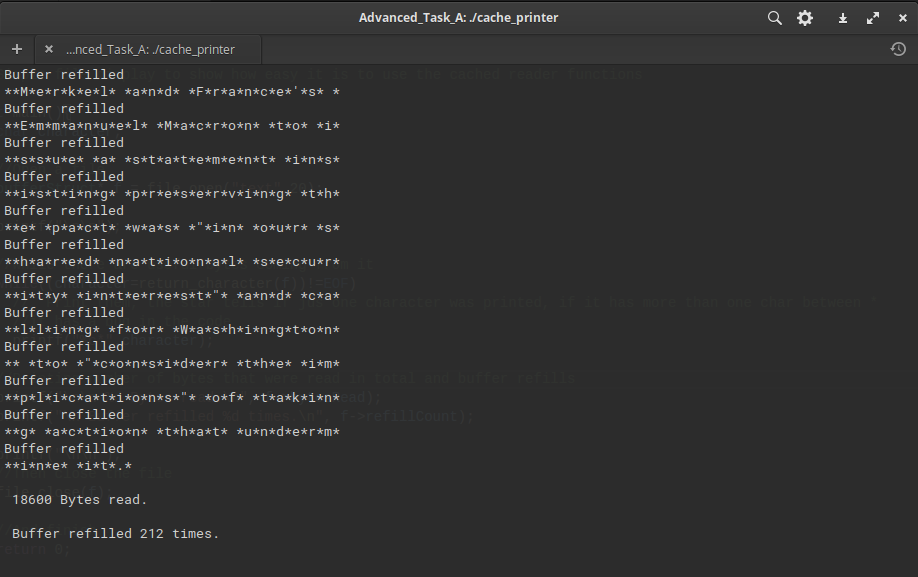
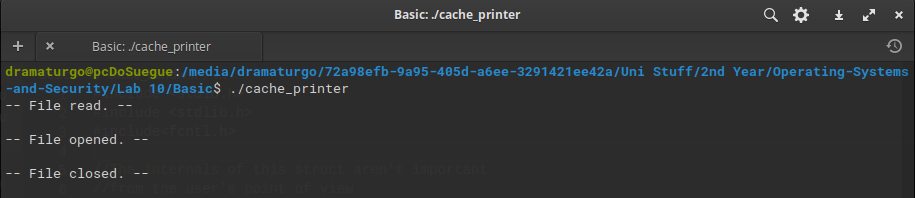
  //Then close the file

  file\_close(f);

  //And finish

  return 0;

}

## Commented updated code showing the required statistical information as well as how many times the words ‘Iran’, ‘Tehran’ and ‘email’ appear.

//FILE CACHE\_HANDLE.H

#include <stdio.h>

#include <stdlib.h>

//The internals of this struct aren't important

//from the user's point of view

typedef struct{

  FILE\* file;        //File being read

  int bufferLength;  //Fixed buffer length

  int alongBuffer;  //Current point in the buffer

  int chars;

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

                     //  same length as "bufferlength"

  char prev\_char;

  char character; //alongBuffer when it is being returned

  int refillCount;

  int bytesRead;

  int countVowels;

  int countIran;

  int countTehran;

  int countEmail;

  int countSentences;

  int countAs;

  int countAlso;

  int countBut;

} bufferStruct;

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

bufferStruct\* file\_open(char\* filename, int buffersize);

//Close an open file

void file\_close(bufferStruct\* buff);

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

char return\_character(bufferStruct\* buff);

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

//Refill an empty buffer.  Not intended for users

int buffer\_refill(bufferStruct\* buff);

//FILE CACHE\_HANDLE.C

#include "cache\_handle.h"

char\* vowels[10] = {"a", "e", "i", "o", "u", "A", "E", "I", "O", "U"};

int wordIran = 0;

int wordTehran = 0;

int wordEmail = 0;

int wordAs = 0;

int wordAlso = 0;

int wordBut = 0;

//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 buffer\_refill(bufferStruct\* buff){

  //Refills a buffer

  //Only works when completely used buffer

  if(buff->alongBuffer!=buff->bufferLength)

    return 0;

  else{

    buff->alongBuffer=0;

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

    //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!

    printf("\nBuffer refilled\n");

    buff->refillCount += 1;

    return len;

  }

}

void file\_close(bufferStruct\* buff){

  free(buff->buffer);

  fclose(buff->file);

}

bufferStruct\* file\_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;

  }

  bufferStruct\* initBuffer=(bufferStruct\*)malloc(sizeof(bufferStruct));

  initBuffer->file=f;

  initBuffer->bufferLength=buffersize;

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

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

  initBuffer->refillCount = 0;

  initBuffer->bytesRead = 0;

  initBuffer->countIran = 0;

  initBuffer->countTehran = 0;

  initBuffer->countEmail = 0;

  buffer\_refill(initBuffer);

  return initBuffer;

}

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

char return\_character(bufferStruct\* buff){

  printf("prev   %c", buff->prev\_char);

  buff->character = buff -> buffer[buff -> alongBuffer];

  //Counting sentences when we find a full mark

  if(buff->character == '.')

    buff->countSentences +=1;

  //In the case of having a null character we need to keep the previous character as it was to prevent bugs

  if(buff->character == '\0' || buff -> buffer[buff -> alongBuffer - 1] == '\0')

      buff->prev\_char = buff->prev\_char;

  else

    buff->prev\_char = buff->buffer[buff->alongBuffer-1];

  //Count vowels

  for(int i = 0; i < 10; i++)

  {

    if(buff -> buffer[buff -> alongBuffer] == \*vowels[i])

      buff -> countVowels += 1;

  }

  // We need to check if the current point in the buffer is the last one that the buffer contains

  // If so then we need to reload the buffer with the next 20 characters and get alongBuffer to point back to the

  //beggining of the buffer

  if(buff -> alongBuffer == buff -> bufferLength)

  {

    buffer\_refill(buff);

    buff -> alongBuffer = 0;

  }

  // If it is not the last character in the buffer then we need to go to the next one so it can be

  //printed on the next run

  else

    buff -> alongBuffer += 1;

  buff->bytesRead += sizeof(buff -> alongBuffer - 1);

  //Other word appearances

  //As

  if(buff->character == 's' & (buff->prev\_char == 'A' || buff->prev\_char == 'a'))

    wordAs += 1;

  if(wordAs == 1)

  {

    wordAs = 0;

    buff->countAs +=1;

  }

  //Also

  if(buff->character == 'l' & (buff->prev\_char == 'A' || buff->prev\_char == 'a'))

    wordAlso += 1;

  if(buff->character == 's' & buff->prev\_char == 'l' & wordAlso == 1)

    wordAlso += 1;

  if(buff->character == 'o' & buff->prev\_char == 's' & wordAlso == 2)

    wordAlso += 1;

  if(wordAlso == 3)

  {

    wordAlso = 0;

    buff->countAlso +=1;

  }

  //But

  if(buff->character == 'u' & (buff->prev\_char == 'B' || buff->prev\_char == 'b'))

    wordBut += 1;

  if(buff->character == 't' & buff->prev\_char == 'u' & wordBut == 1)

    wordBut += 1;

  if(wordBut == 2)

  {

    wordBut = 0;

    buff->countBut +=1;

  }

  //Iran word appearances

  if(buff->character == 'r' & (buff->prev\_char == 'I' || buff->prev\_char == 'i'))

    wordIran += 1;

  if(buff->character == 'a' & buff->prev\_char == 'r' & wordIran == 1)

    wordIran += 1;

  if(buff->character == 'n' & buff->prev\_char == 'a' & wordIran == 2)

    wordIran += 1;

  if(wordIran == 3)

  {

    wordIran = 0;

    buff->countIran +=1;

  }

  //Tehran word appearances

  if(buff->character == 'e' & (buff->prev\_char == 'T' || buff->prev\_char == 't'))

    wordTehran += 1;

  if(buff->character == 'h' & buff->prev\_char == 'e' & wordTehran == 1)

    wordTehran += 1;

  if(buff->character == 'r' & buff->prev\_char == 'h' & wordTehran == 2)

    wordTehran += 1;

  if(buff->character == 'a' & buff->prev\_char == 'r' & wordTehran == 3)

    wordTehran += 1;

  if(buff->character == 'n' & buff->prev\_char == 'a' & wordTehran == 4)

    wordTehran += 1;

  if(wordTehran == 5)

  {

    wordTehran= 0;

    buff->countTehran +=1;

  }

  //Email word appearances

  if(buff->character == 'm' & (buff->prev\_char == 'E' || buff->prev\_char == 'e'))

    wordEmail += 1;

  if(buff->character == 'a' & buff->prev\_char == 'm' & wordEmail == 1)

    wordEmail += 1;

  if(buff->character == 'i' & buff->prev\_char == 'a' & wordEmail == 2)

    wordEmail += 1;

  if(buff->character == 'l' & buff->prev\_char == 'i' & wordEmail == 3)

    wordEmail += 1;

  if(wordEmail == 4)

  {

    wordEmail= 0;

    buff->countEmail +=1;

  }

  // In the end we just need to return the character that was being pointed to at the beggining

  return buff->character;

}

//FILE CACHE\_PRINTER.C

#include "cache\_handle.h"

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

int main(){

  char character;

  //Open a file

  bufferStruct\* f = file\_open("text",20);

  printf("\n\n");

  //While there are useful bytes coming from it

  while((character=return\_character(f))!=EOF)

  {

    //  Print them, the star tells if jus one character was printed, if it has more than one char between \*

    //it has a bug in the code

    printf("\n%c\*",character);

    //I've noticed that a null character was being printed in the beggining of the buffer, so i'll just

    //subtract these from the total number of characters

    if(character != '\0')

      f->chars += 1;

  }

  //Printing number of bytes that were read in total and buffer refills

  printf("\n\n %d Bytes read.\n", f->bytesRead);

  printf("\n Buffer refilled %d times.\n", f->refillCount);

  printf("\n %d vowels.\n", f->countVowels);

  printf("\n %d other characters.\n", (f->chars - f->countVowels));

  printf("\n %i Non active word appearances (As, Also, But).\n", (f->countAs + f->countAlso + f->countBut));

  printf("\n %i Iran appearances.\n", f->countIran);

  printf("\n %i Tehran appearances.\n", f->countTehran);

  printf("\n %i Email appearances.\n", f->countEmail);

  printf("\n %i Sentences.\n", f->countSentences);

  printf("\n\n");

  //Then close the file

  file\_close(f);

  //And finish

  return 0;

}

OUTPUT FIG 43

# Lab 10: The Cache Buffer from week 8 with system calls

## Brief description of the activity

In this lab we explore the use of system calls to access files and manipulating them. It also has the idea of reducing caching effects in code.

## Changes the cache\_handle library from using the fopen, fread, fclose functions to the system call versions open, read, close

//FILE CACHE\_HANDLE.C

#include "cache\_handle.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 buffer\_refill(bufferStruct\* buff){

  //Refills a buffer

  //Only works when completely used buffer

  if(buff->alongBuffer!=buff->bufferLength)

    return 0;

  else{

    buff->alongBuffer=0;

    //read

    int len=read(buff->file, buff->buffer, buff->bufferLength);

    printf("-- File read. --\n\n");

    //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 file\_close(bufferStruct\* buff){

  free(buff->buffer);

  //close

  close(buff->file);

  printf("-- File closed. --\n\n");

}

bufferStruct\* file\_open(char \* filename, int buffersize){

  //Info on malloc

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

  //open

  int f = open(filename, O\_RDONLY | O\_CREAT);

  if (&f == NULL){

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

    return 0;

  }

  bufferStruct\* initBuffer=(bufferStruct\*)malloc(sizeof(bufferStruct));

  initBuffer->file=f;

  initBuffer->bufferLength=buffersize;

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

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

  buffer\_refill(initBuffer);

  printf("-- File opened. --\n\n");

  return initBuffer;

}

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

char return\_character(bufferStruct\* buff){

  // your code goes here

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

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

}

A screenshot of a cell phone

Description automatically generated Fig 44 – Task Basic a) system calls to open, read and close a file

## Changes cache\_handle library to remove (as far as possible) the effects of caching on the library.

//FILE CACHE\_HANDLE.C

#include "cache\_handle.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 buffer\_refill(bufferStruct\* buff){

  //Refills a buffer

  //Only works when completely used buffer

  if(buff->alongBuffer!=buff->bufferLength)

    return 0;

  else{

    buff->alongBuffer=0;

    int len=read(buff->file, buff->buffer, buff->bufferLength);

    printf("-- File read. --\n\n");

    //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 file\_close(bufferStruct\* buff){

  free(buff->buffer);

  close(buff->file);

  printf("-- File closed. --\n\n");

}

bufferStruct\* file\_open(char \* filename, int buffersize){

  //Info on malloc

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

  //Opening the file with the call to read only removes some of the effects of caching

  int f = open(filename, O\_RDONLY | O\_CREAT);

  if (&f == NULL){

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

    return 0;

  }

  bufferStruct\* initBuffer=(bufferStruct\*)malloc(sizeof(bufferStruct));

  initBuffer->file=f;

  initBuffer->bufferLength=buffersize;

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

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

  buffer\_refill(initBuffer);

  printf("-- File opened. --\n\n");

  return initBuffer;

}

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

char return\_character(bufferStruct\* buff){

  // your code goes here

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

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

}

# Lab Activity 13 – Job control

## Description of the activity

This lab is about processes and how they can be controlled and launched to the system. There is the creation of a menu in the terminal to select different processes that are then run by forking the process in two, having then a parent process and a child creating the possibility of the manipulation of the running of a process. It also shows the idea of having to jobs running in simultaneous.

## Menu System

//fork and execl code to overcome problem of getting stuck in child process by using wait()

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include "sys/types.h"

#include <sys/wait.h>

int main(){

    pid\_t pid\_value;

    pid\_t pid\_value2;

    int status = 0;

    char\* directory;

    char\* command;

    int selection;

    int ping;

    while(1)

    {

        printf("\n|------------------------------------------------------------|\n");

        printf("| Welcome to the linux commands menu.                        |\n");

        printf("| To use a command just type the number on the command line. |\n");

        printf("|----------------------List of commands----------------------|\n");

        printf("| 1 -> ps                                                    |\n");

        printf("| 2 -> date                                                  |\n");

        printf("| 3 -> pwd                                                   |\n");

        printf("| 4 -> ls                                                    |\n");

        printf("| 5 -> ping google and yahoo                                 |\n");

        printf("| 6 -> pause parent w/o killing child                        |\n");

        printf("|                                                            |\n");

        printf("| 0 -> exit                                                  |\n");

        printf("|------------------------------------------------------------|\n");

        //getting the user's selection

        scanf("%d", &selection);

        //for each selection the code will change where to go grab the command and what command it is

        if(selection == 1){ directory = "/bin/ps"; command = "ps";}

        if(selection == 2){ directory = "/bin/date"; command = "date";}

        if(selection == 3){ directory = "/bin/pwd"; command = "pwd";}

        if(selection == 4){ directory = "/bin/ls"; command = "ls" ;}

        if(selection == 5){ directory = "/bin/ping"; command = "ping";}

        if(selection == 0){ printf("Smell you later.\n"); exit(0);}

        //here we create the fork so that we have now a child and a parent process running

        pid\_value= fork();

        if(pid\_value!=0)

        {

            //with the 6th selection the parent pauses allowing the child to run without the parent needing to

            if(selection==6)

            {

                pause();

            }

            else

            {

                wait(&status);

                printf ( "\n I am the parent my Process ID is %d, myParents PID is %d, \n ",getpid(),getppid());

                printf( "\n Using the wait ensures that my child finishes first. \n ");

            }

        }

        else

        {

            //here we create the second child, so that from the first child two childs can be created

            pid\_value2 = fork();

            if(pid\_value2!=0 & pid\_value==0)

            {

                //waits for the second child to run first

                wait(&status);

                printf ( " I am the child, my Process ID is %d , my Parents PID is %d \n",getpid(),getppid());

                sleep(2);

                printf ( "\n Using execl to display running processes \n");

                //if the selection is to ping a website then it will start pinging for 4 times

                if(selection == 5)

                {

                    printf ( "\n Using execl to display running processes \n");

                    execl ( directory, command, "www.google.com", "-c", "4", (char\*)0);

                }

                else

                    //if the selection was 6 then it just echoes a message saying the parent was told to pause

                    //so after the first child ends with their ping the code should stay frozen

                    if(selection==6)

                        execl("/bin/echo", "echo", "Parent was paused.\n", (char\*)0);

                    //if we had another selection then we will run the command related to that selection

                    else

                        execl ( directory, command,(char\*)0);

                printf( "\n Due to the execl, you should not be able to read this ?\n " ) ;

            }

            else

            {

                //where the second child runs, so the principle will be similar to the first child

                printf ( " I am the second child, my Process ID is %d , my Parents PID is %d \n",getpid(),getppid());

                sleep(2);

                if(selection == 5)

                {

                    printf ( "\n Using execl to display running processes \n");

                    execl ( directory, command, "www.google.com", "-c", "4", (char\*)0);

                }

                else

                    if(selection==6)

                        execl("/bin/echo", "echo", "Parent was paused.\n", (char\*)0);

                    else

                        execl ( directory, command,(char\*)0);

                printf( "\n Due to the execl, you should not be able to read this ?\n " ) ;

            }

        }

    }

    return 0;

}

**A screenshot of a computer

Description automatically generated** Fig 45 – Task Basic a) first 3 commands plus exit command

**A screenshot of a cell phone

Description automatically generated** Fig 46 – Task Basic b) 4th option to list file/directory contents

**A screenshot of a computer

Description automatically generatedA close up of a logo

Description automatically generated** Fig 47 – Task Advanced a) creating a second child to ping yahoo and google

**A screenshot of a cell phone

Description automatically generated** Fig 48 – Task Advanced b) pausing parent function

# Lab Activity 14 – Linux command-line manipulation of processes

## Paragraph on disown and an alternative.

Nohup stands for no hang up, so it basically tells the terminal to keep running the process even when the user exits or logs out of the Linux OS. It is meant to be used with long term tasks that might take many hours to complete but the user wants to keep them running in the background. To use nohup we just need to add nohup as a prefix of the command to run in the background. An alternative to this command would be using the and (&) symbol plus disown command following the process to run in the background.

There are two main differences between these commands, first the nohup command is not recognizable by some shells such as dash or tcsh, besides this disown can be launched after the process is running while nohup needs to be started with the process.

## Description and example of using watch command

The watch command can be used to monitor the output changes overtime, so if we run a command with the watch prefix what happens is that the terminal will be cleared and with a 2 second interval time the command will run and show the output on the screen. This command can be used as “watch [OPTIONS] COMMAND” where the options can be several such as -n which followed by a number will change the interval time to that number or -d which highlights the differences in each run. An example could be “watch -n 4 date” which would run the command date every 4 seconds.

## Process manipulation

### Example(s) of how to start process

To start a process the user just needs to input the command for the process and the arguments that the process may need to run.

A screenshot of a computer

Description automatically generated Fig 49 – Task a) Starting processes

### Example(s) of how to suspend process

In order to suspend a process the user just needs to after starting the process use the combination Ctrl+Z, this will suspend the process until it is told to run again.

A screenshot of a computer screen

Description automatically generated Fig 50 – Task b) Suspending processes

### Example(s) of how to run process in background

To run a process in the background we need to first suspend the

process so that we can input something in the command line, then just using the bg command with the job id we can run the process again in the background.

A screenshot of a cell phone

Description automatically generated Fig 51 – Task c) Background running processes

### Example(s) of how to run process in foreground and bring from background

Same as before but we can use the command fg instead of bg.

A screenshot of a cell phone

Description automatically generated Fig 52 – Task d) Foreground running processes

### Example(s) of how to kill a process

To kill a process we just need to address to the command kill, we use

the following argument -9 to hard kill the process.

A screenshot of a computer

Description automatically generated Fig 53 – Task e) Killing processes

# Lab Activity 15 IPC and Synchronisation

## Brief description of activity

This lab is about exploring the idea of a semaphore to create a synchronous process chain, we use to processes and alternate them to create the lines of a song. Then we do the same but we register the output on a text file and by introducing the idea of standard string outputs and error outputs.

## Modified semaphore example code so that the two processes output the song.

//critical\_example2.c

#include <sys/ipc.h>

#include <sys/sem.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include "ops\_sems.h"

int main(int argc, char argv[]){

int id;

FILE \*lyrics;

char line[200];

int flag = 1;

//Use our source file as the "key"

  id=ops\_semget("critical\_example2.c",1);

  lyrics = fopen("hole\_in\_bucket.txt", "r");

  int pid=fork();

  if(pid){

    //P1

    while(flag == 1){

      //in each printf instance we need to tell the process to sleep so that we can in fact assure that the next process will not print anything before we signal the semaphore

      //so what happens is that by forking the process we first will go with the process 1 (we can call it henry) and after that process signals the semaphore

      //we can then pass to the process 2 (liza process), the code will do that between processes until we finish both processes

      ops\_wait(id);

      printf("Henry:\nThere's a hole in the bucket, dear Liza, dear Liza,\nThere's a hole in the bucket, dear Liza, a hole.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Henry:\nWith what shall I fix it, dear Liza, dear Liza?\nWith what shall I fix it, dear Liza, with what?\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Henry:\nThe straw is too long, dear Liza, dear Liza,\nThe straw is too long, dear Liza, too long.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Henry:\nWith what shall I cut it, dear Liza, dear Liza?\nWith what shall I cut it, dear Liza, with what?\n\n");

      sleep(1);

      ops\_signal(id);

      flag=0;

    }

  }else{

    //P2

    while(flag == 1){

      ops\_wait(id);

      printf("Liza:\nThen fix it, dear Henry, dear Henry, dear Henry,\nThen fix it, dear Henry, dear Henry, fix it.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Liza:\nWith straw, dear Henry, dear Henry, dear Henry,\nWith straw, dear Henry, dear Henry, with straw.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Liza:\nThen cut it, dear Henry, dear Henry, dear Henry,\nThen cut it, dear Henry, dear Henry, cut it.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Liza:\nWith an axe, dear Henry, dear Henry, dear Henry,\nWith an axe, dear Henry, dear Henry, an axe.\n\n");

      sleep(1);

      ops\_signal(id);

      flag=0;

    }

  }

  fclose(lyrics);

}

A screenshot of a cell phone

Description automatically generated Fig 54 – Task b) Semaphore code to output the lines of a song.

## Modified code to write Liza part to stderr and redirect the two parts to a file.

//critical\_example2.c

#include <sys/ipc.h>

#include <sys/sem.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include "ops\_sems.h"

int main(int argc, char argv[]){

int id;

FILE \*lyrics;

char line[200];

int flag = 1;

//Use our source file as the "key"

  id=ops\_semget("critical\_example2.c",1);

  lyrics = fopen("hole\_in\_bucket.txt", "w");

  int pid=fork();

  if(pid){

    //P1

    while(flag == 1){

      //This part is equal to the basic exercise:

      //in each printf instance we need to tell the process to sleep so that we can in fact assure that the next process will not print anything before we signal the semaphore

      //so what happens is that by forking the process we first will go with the process 1 (we can call it henry) and after that process signals the semaphore

      //we can then pass to the process 2 (liza process), the code will do that between processes until we finish both processes

//printing to terminal

      ops\_wait(id);

      printf("Henry Part 1:\nThere's a hole in the bucket, dear Liza, dear Liza,\nThere's a hole in the bucket, dear Liza, a hole.\n\n");

//printing to file

      fprintf(lyrics, "Henry Part 1:\nThere's a hole in the bucket, dear Liza, dear Liza,\nThere's a hole in the bucket, dear Liza, a hole.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Henry Part 2:\nWith what shall I fix it, dear Liza, dear Liza?\nWith what shall I fix it, dear Liza, with what?\n\n");

      fprintf(lyrics, "Henry Part 2:\nWith what shall I fix it, dear Liza, dear Liza?\nWith what shall I fix it, dear Liza, with what?\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Henry Part 3:\nThe straw is too long, dear Liza, dear Liza,\nThe straw is too long, dear Liza, too long.\n\n");

      fprintf(lyrics, "Henry Part 3:\nThe straw is too long, dear Liza, dear Liza,\nThe straw is too long, dear Liza, too long.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      printf("Henry Part 4:\nWith what shall I cut it, dear Liza, dear Liza?\nWith what shall I cut it, dear Liza, with what?\n\n");

      fprintf(lyrics, "Henry Part 4:\nWith what shall I cut it, dear Liza, dear Liza?\nWith what shall I cut it, dear Liza, with what?\n\n");

      sleep(1);

      ops\_signal(id);

      flag=0;

    }

  }else{

    //P2

    while(flag == 1){

//printing to the terminal

      ops\_wait(id);

      fprintf(stderr, "Liza Part 1:\nThen fix it, dear Henry, dear Henry, dear Henry,\nThen fix it, dear Henry, dear Henry, fix it.\n\n");

//printing to file

      fprintf(lyrics, "Liza Part 1:\nThen fix it, dear Henry, dear Henry, dear Henry,\nThen fix it, dear Henry, dear Henry, fix it.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      fprintf(stderr, "Liza Part 2:\nWith straw, dear Henry, dear Henry, dear Henry,\nWith straw, dear Henry, dear Henry, with straw.\n\n");

      fprintf(lyrics, "Liza Part 2:\nWith straw, dear Henry, dear Henry, dear Henry,\nWith straw, dear Henry, dear Henry, with straw.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      fprintf(stderr, "Liza Part 3:\nThen cut it, dear Henry, dear Henry, dear Henry,\nThen cut it, dear Henry, dear Henry, cut it.\n\n");

      fprintf(lyrics, "Liza Part 3:\nThen cut it, dear Henry, dear Henry, dear Henry,\nThen cut it, dear Henry, dear Henry, cut it.\n\n");

      sleep(1);

      ops\_signal(id);

      ops\_wait(id);

      fprintf(stderr, "Liza Part 4:\nWith an axe, dear Henry, dear Henry, dear Henry,\nWith an axe, dear Henry, dear Henry, an axe.\n\n");

      fprintf(lyrics, "Liza Part 4:\nWith an axe, dear Henry, dear Henry, dear Henry,\nWith an axe, dear Henry, dear Henry, an axe.\n\n");

      sleep(1);

      ops\_signal(id);

      flag=0;

    }

  }

  fclose(lyrics);

}

Output version is similar as the one before but for some reason the processes didn’t seem to be writing alternate lines to the file. My assumption would be that some other process might be getting in their way or it may be a bug in my code that I couldn’t have had fixed.

# References

Question 1 a:

Difference between Von Neumann and Harvard Architecture 2018, Khillar, S DifferenceBetween.net, viewed on 20/03/2020, <http://www.differencebetween.net/technology/difference-between-von-neumann-and-harvard-architecture/ >;

Difference between Harvard architecture and Von-Neumann architecture 2017, Technical Editor on Polytechnic Hub, viewed on 20/03/2020, <<https://www.polytechnichub.com/difference-harvard-architecture-von-neumann-architecture/>>.

Question 14 a:

Linux nohup command 2019, Computer Hope, viewed on 18/03/2020, <<https://www.computerhope.com/unix/unohup.htm>>;

Linux / Unix: disown Command Examples 2015, nixCraft, viewed on 18/03/2020, <<https://www.cyberciti.biz/faq/unix-linux-disown-command-examples-usage-syntax/>>;

Brockmeier, J 2011, Detach Processes With Disown and Nohup, viewed on 18/03/2020, <<https://www.serverwatch.com/tutorials/article.php/3935306/Detach-Processes-With-Disown-and-Nohup.htm>>.

Question 14 b:

Linux Watch Command 2019, Linuxize, viewed on 18/03/2020, <<https://linuxize.com/post/linux-watch-command/>>.