Name: VENELIN DIMITROV

Student id: 6297262

Course: Computer Science

Module: 207SE Operating Systems, Security and Networks

Submission date: 27nd February

Portfolio 1

**Lab Activity 1 – Operating Systems Tasks and Programming**

a) Future of OS

For the last 25 years operating systems have changed multiple times. From being a bleeping line in a black screen in the command line, to a complicated graphical interface with multiple working environments and millions of software applications to choose from. Main goal for every OS developer has been creating a simplified interface so every user can control and customise their system by personal taste. Demands by users have changed and now just simplicity is not enough, everyone is using daily at least 4 different devices with every one of them having their own applications and just only a few of them having the functionality of multi-platforming.

* Next generation OS will have smaller size and running entirely in the cloud.
* Tracking of all applications and providing a single point of access for them
* Simplified upgrades, management and portability across platforms
* Having an artificial intelligence as an assistant to everyday tasks and researches.
* Encryption and security

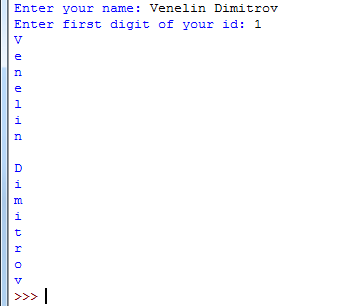
b) Programming activity  
  
**PYTHON**

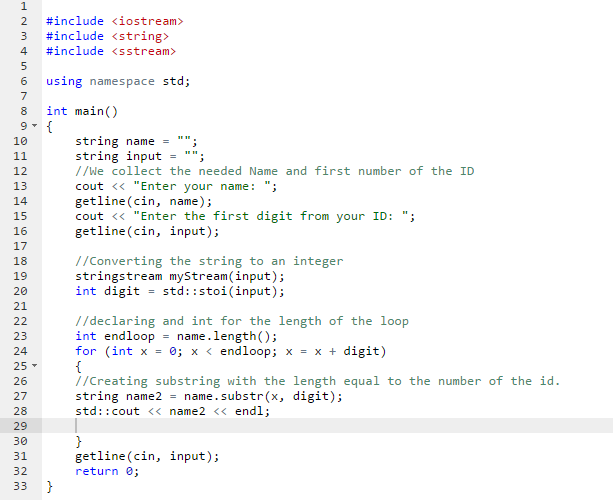
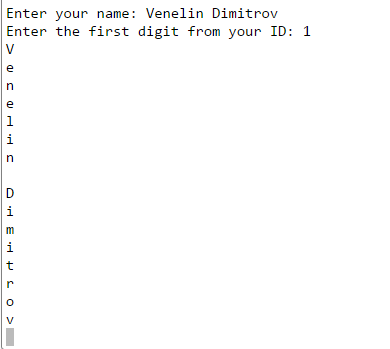
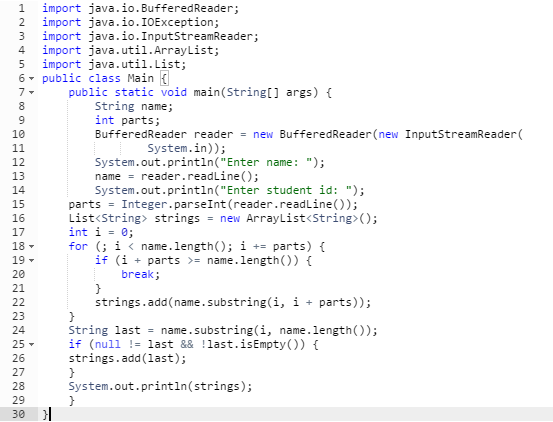
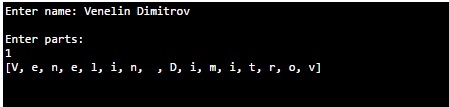
#Input for the student name and his ID  
student = input ("Enter the name of the student: ")   
id = input ("Enter the first digit of id: ")

#Loop takes the id number and splits the students name into parts equal to the number which was stored.

**for** i **in** range(0,len(student),int(id)):   
 #**prints and each split on a new line**

**print**(student[i:i+int(id)])   
  
**OUTPUT**

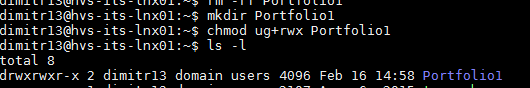


**C++** **OUTPUT** **JAVA** **OUTPUT  
**

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

1. How made Portfolio1 directory read/write/executable only for you and your group. That is, not for others. Show evidence of this with ls command.

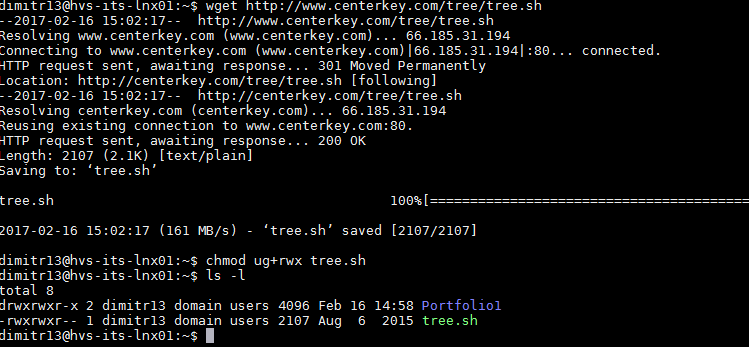
**mkdir Portfolio1**

**chmod ug+rwx Portfolio1** 

1. How downloaded the script http://www.centerkey.com/tree/tree.sh to your home directory using wget and make it executable.

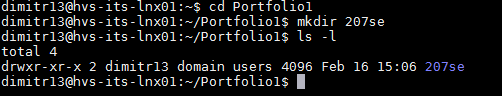
**wget http://www.centerkey.com/tree/tree.sh**

**chmod ug+rwx tree.sh**



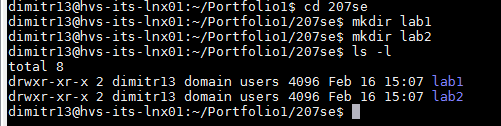
1. Making Directories

* How created a 207se directory in your Portfolio1 directory.  
    
  **cd Portfolio1  
  mkdir 207se**



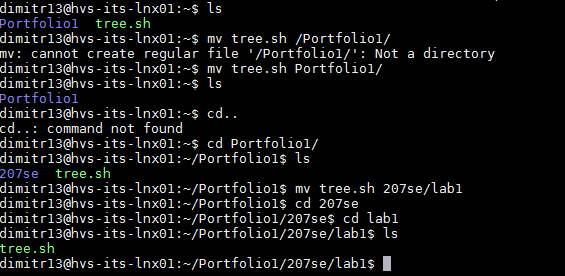
* How created numbered directories for the labs. i.e. lab1 and lab2 etc.

**cd 207se  
mkdir lab1  
mkdir lab2**



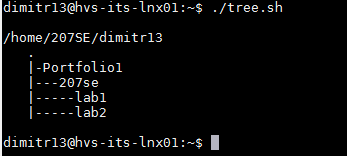
* Evidence of transferring lab1 activity into appropriate directory

**mv tree.sh Portfolio/207se/lab1**

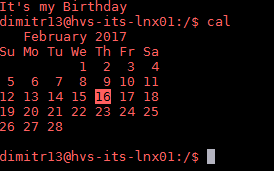
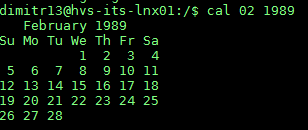


* Evidence of make directory activities using tree.sh

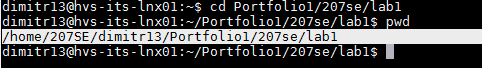
**~/Portfolio1/207se$ cd ~**



1. Display todays date and using the cal command show the month that you were born.

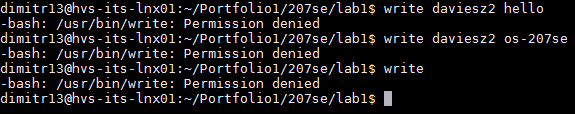
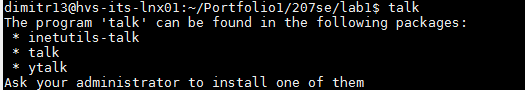
  
  


1. Move into the lab1 directory and use the appropriate command to show the current directory  
     
   **cd Portfolio1/207se/lab1  
   pwd**



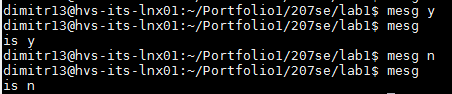
1. What is talk, write and wall are for and which ones work

**The write command is used to exchange private messages with other people connected to the same server. Just by simply typing “write (username)”. Sadly its not working in this server and has been disabled by admin.**

  
  
**The command talk copies lines from your terminal to that of another user specified. The other user sends a request to connect to his machine and after that the messages can be exchanged. Also not working on our servers.**  
  
  
  
**The command wall is the other one disabled which we cannot use. It is used to broadcast a message to all of the users and machines connected to the server by just typing “wall (message)”**

1. What command prevents the effects of those three commands from interrupting you?

**The command which disables and enables the messaging through servers is “mesg (y or n)” y= yes, n=no**



1. The song in song.txt.

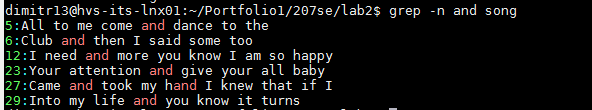
* Using wc the number of words and lines in the file.

**~/Portfolio1/207se/lab2  
wc song**



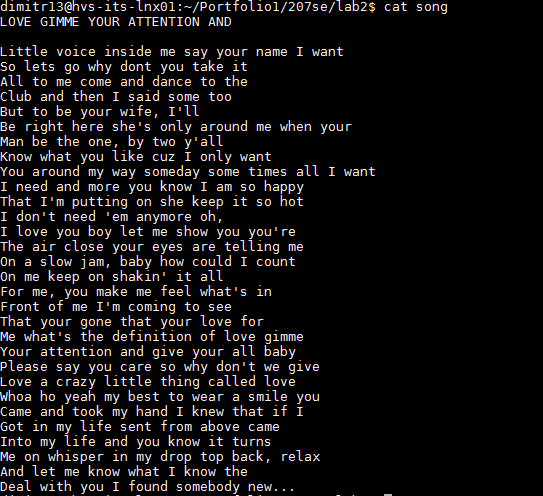
* Using grep to get the lines containing "and" and the number of the lines contain “and” in the document

**Grep –n and song**



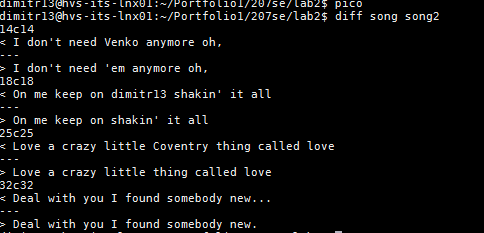
* Use cat to show the contents of the file.

**cat song**



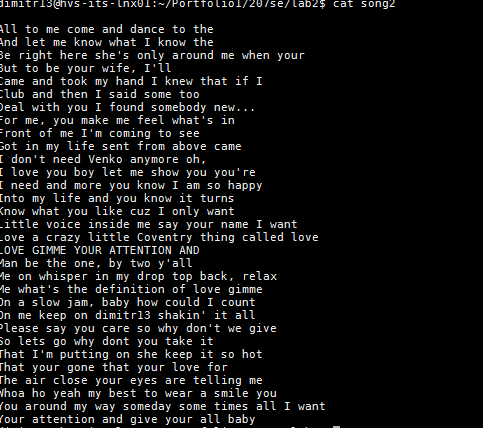
* Appropriate Linux command to see if the two files differ and how they differ.

**diff song song2**



* Use sort to sort the file and redirect the output to a new file called song2.txt

**sort song > song2.txt**

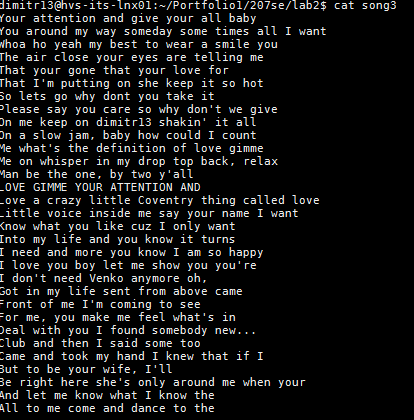


* Use sort and rev to reverse the sorted contents of song.txt and append the output to song2.txt

**dimitr13@hvs-its-lnx01:~/Portfolio1/207se/lab2$ sort -r song > song3**

**dimitr13@hvs-its-lnx01:~/Portfolio1/207se/lab2$ cat song3 >> song2**

**dimitr13@hvs-its-lnx01:~/Portfolio1/207se/lab2$ cat song3**



* Total memory used and the total memory available

**free –m**



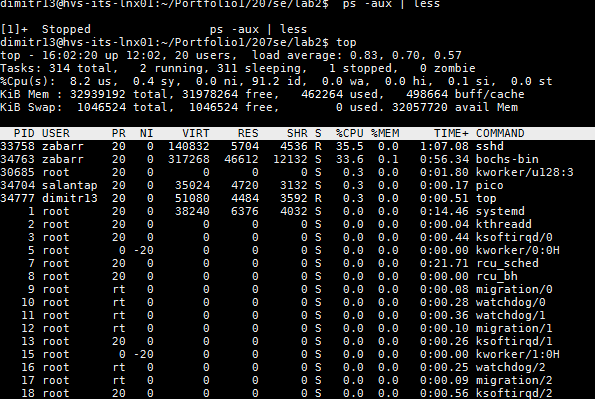
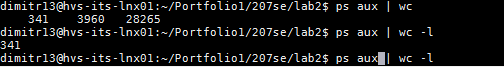
* Find out how you can display your username on the screen.

**whoami  
id –u –n**



* List the processes that are running.

ps –aux | less  
ps –aux | wc -l

* What are the differences between the Linux commands less, more and most.

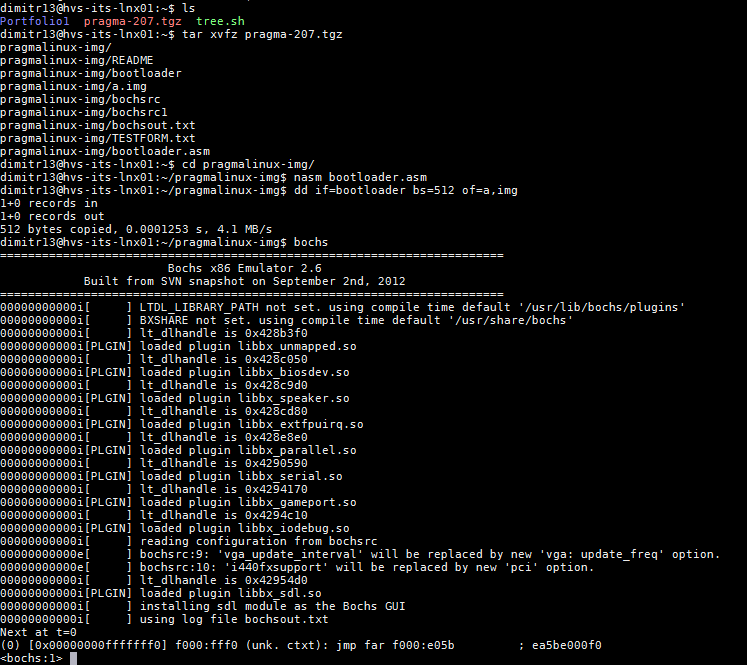
**“more” “most” and “less”** do mostly the same functions.  
**“more”** is an older version which shows a huge pile of text which can’t be displayed on the screen so it scrolls down every time you press a button but it can’t scroll back up which was a problem for most users.  
  
**“less”** was written by a user which was fed up by “more” incapability to scroll backwards in a file  
  
**“most”** most is considered an upgrade of “less” and the better version of those two. It can visualise multiple text files at once. It can hold the information about long lines and scrolls even further so they can be displayed so it also provides left and right scrolling as an included backwards and downwards. Also it has the opportunity to decompress gunzip files before reading them.

**Lab Activity 4 Bootloader**

1. Brief description of the Lab activity and what you did

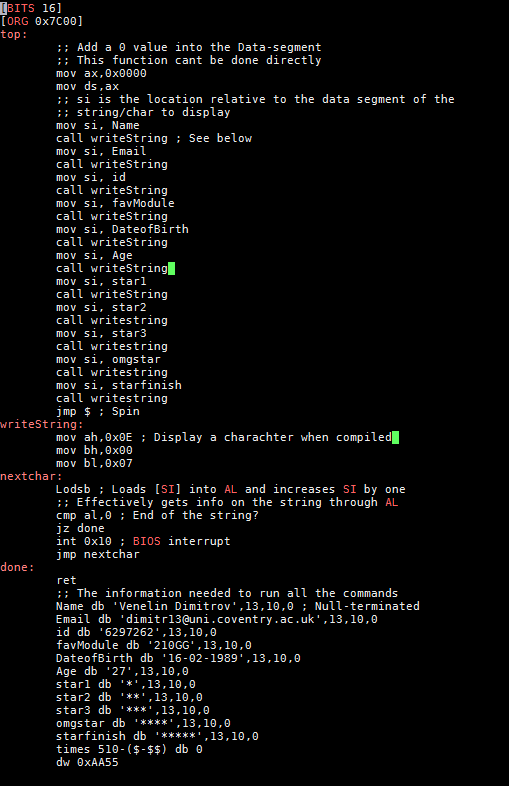
In this activity the main task was to create an empty file and run it as a bootloader. The file had written commands to be run and compiled through nasm and pragma. Task was to create a file which will display my names, student id, date of birth, email, second year favourite module and also my age all of these on separate lines.   
  
The outcome of this task is to learn and see how we can use a programming language to be capable to be used by microcontrollers or other programmable device to enhance or create new tasks after being self-programmable.

1. Boot pragma linux with bochs

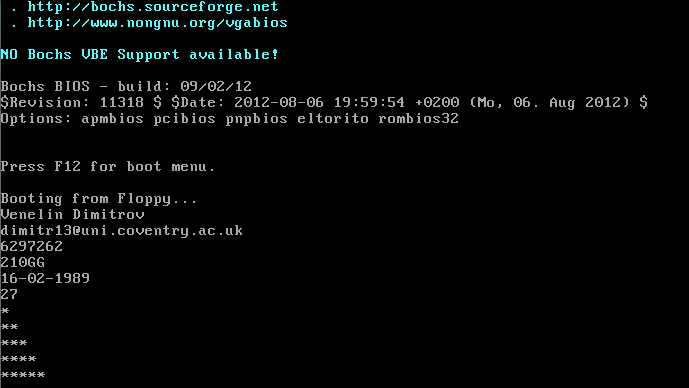


1. Make a bootloader that displays your student details and triangle

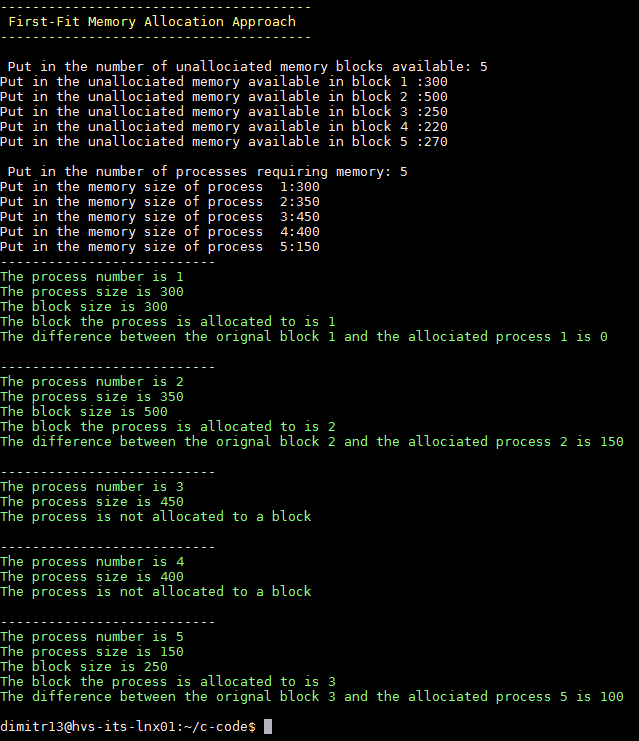
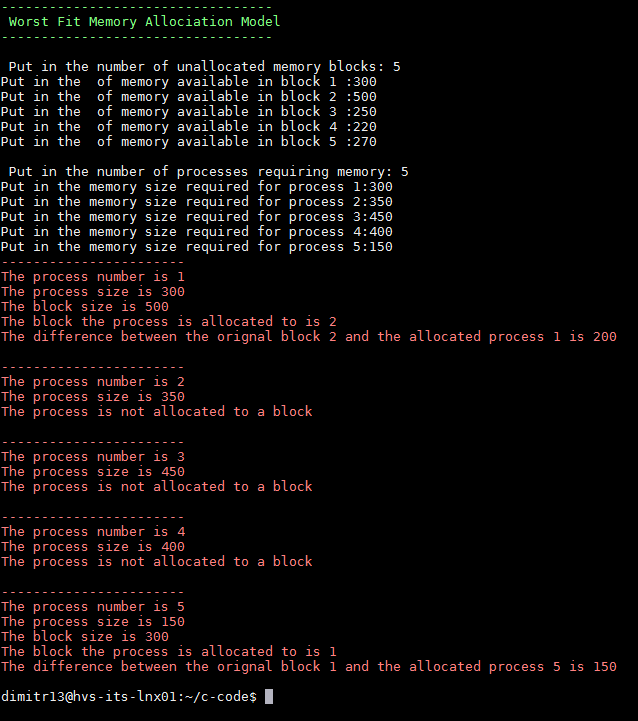
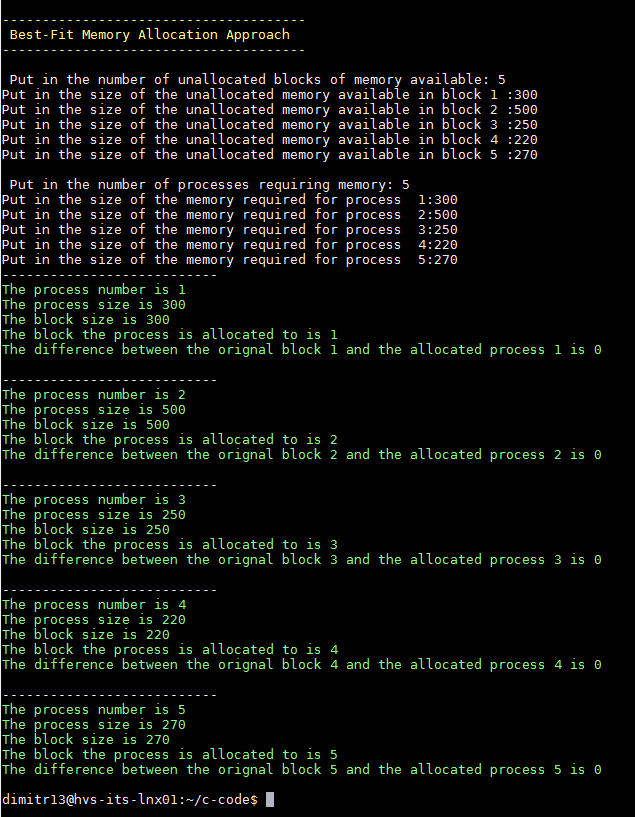
* Commented bootloader code to display your student details and triangle



* Output from Bochs showing student details and triangle



**Lab Activity 6 Outside the processor**

1. Memory Allocation Activities  
     
   
2. 
3. 

Which approaches allocates all of the processes and with the least fragmentation.

**Best fit**

M1 (300)

M5 (270)

M4 (220)

M3 (250)

M2 (500)

**Worst fit**

M2 (500)

M1 (300)

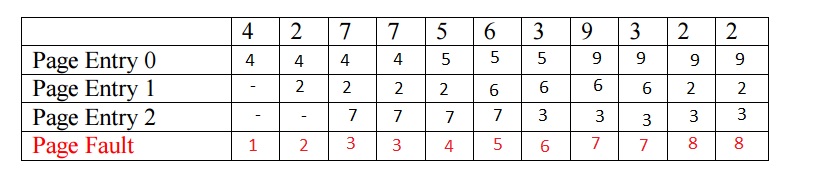
M5 (270)

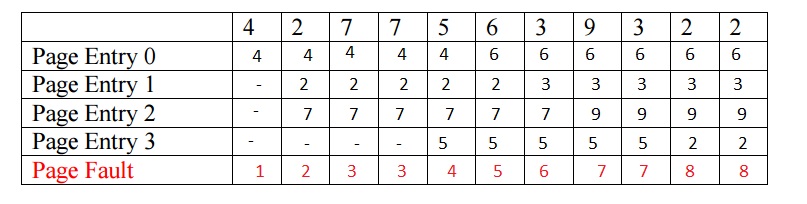
M4 (220)

M3 (250)

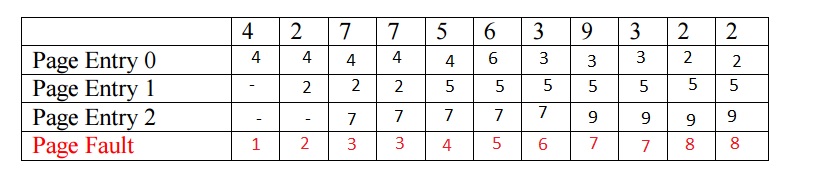
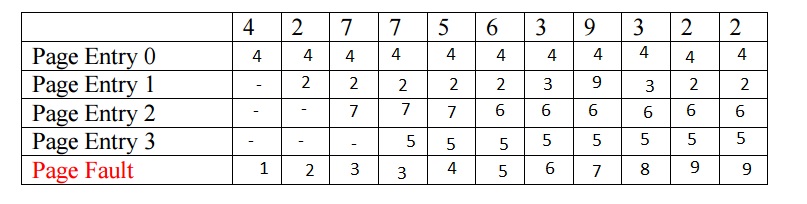
The process which fragments the least of the two is the **best fit**. It allocates all the memory units in the processes and does least task for fragmentation.

Paging Activities  
  
**FIRST IN FIRST OUT**



  
  
The results for both of them are not that much different from each other after the algorithm has passed from the first half of the table it changes drastically as all the values for each entry of the steps have a different result because of the existing 4th page entry on the second table. The value just pushes into the next step and thus resulting in a different sequence. Page fault score stayed the same and completely identical to both tables.

Repeat the above process for the random page allocation approach.  
**RANDOM PAGE ALLOCATION**

The results for both random page allocation with 3 and 4 page entries resulted in a completely different from each other tables. The values given by the user were identical but as there was a 4 page values for the second table the result ended with some interesting results. Page fault score did change but only after the 9th step in the sequence.

**Lab Activity 7 Buffer**

1. Brief description of the Buffer Activity

First of we have to edit and comment different lines of the buffer.c code so we show that we understand what is happening in the code. For the first editing part we have to include a print error messages in case something goes wrong and we have information about it. Possible errors could be that we don’t have the two inserts, not a valid input or problem with copying the file.  
  
The code below is just creating a buffer file which collects information from the input.txt file and doubles it in another output file different from the original one. Changing the size of the buffer at the beginning, changes how many bytes of characters are filled to the buffer at one time. The bigger the size of the buffer the more quicker it can get filled.

1. Commented Buffer.c code

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

//defining the size of the buffer and its output mode

#define BUF\_SIZE 500

#define OUTPUT\_MODE 0700

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

{

int in\_fd, out\_fd; //input and output values defined

int rd\_size = 1, wr\_size;

char buf[BUF\_SIZE]; //defining a variable to be stored corresponding to the size of the buffer

//if statement to check if all 3 of the arguments are present

if (argc != 3)

exit(1);

//checks for a valid input file

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

if (in\_fd < 0)

exit(2);

//checks if the file is copied

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

if (out\_fd < 0)

exit(3);

while (rd\_size > 0) {

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

if (rd\_size <0)

exit(4);

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

if (wr\_size<=0){

close(in\_fd);

//closes all opened files

close(out\_fd);

exit(5);

}

}

}

1. Update the code to so that it prints if an error has occurred or if a file is successfully created with the content of the review in it.

After running code what is in hamlet.txt

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

//defining the size of the buffer and its output mode

#define BUF\_SIZE 500

#define OUTPUT\_MODE 0700

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

{

int in\_fd, out\_fd; //input and output values defined

int rd\_size = 1, wr\_size;

char buf[BUF\_SIZE]; //defining a variable to be stored corresponding to the size of the buffer

//if statement to check if all 3 of the arguments are present

if (argc != 3) {

printf("You 2 or more inputs");

exit(1);

}

//checks for a valid input file

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

if (in\_fd < 0) {

print("Sorry you have not selected an input file");

exit(2);

}

//checks if the file is copied

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

if (out\_fd < 0) {

printf("Sorry, file was not copied");

exit(3);

}

while (rd\_size > 0) {

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

if (rd\_size <0)

exit(4);

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

if (wr\_size<=0){

close(in\_fd);

//closes all opened files

close(out\_fd);

exit(5);

}

}

}

After the code has been compiled and runed in the terminal we receive two files. One of hamlet.txt and one of the latest production of Hamlet by Shakespeare

1. Updated buffer.c code to show how many character are read to buffer, how many character read at a time into the buffer, how many words in the document and how many times the buffer is filled

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

//defining the size of the buffer and its output mode

#define BUF\_SIZE 500

#define OUTPUT\_MODE 0700

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

{

int characters = 0;

int words = 0;

int i = 0;

int in\_fd, out\_fd; //input and output values defined

int rd\_size = 1, wr\_size;

char buf[BUF\_SIZE]; //defining a variable to be stored corresponding to the size of the buffer

//if statement to check if all 3 of the arguments are present

if (argc != 3) {

printf("You need two or more inputs");

exit(1);

}

//checks for a valid input file

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

if (in\_fd < 0) {

printf("Sorry you have not selected an input file");

exit(2);

}

//checks if the file is copied

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

if (out\_fd < 0) {

printf("Sorry, file was not copied");

exit(3);

}

while (rd\_size > 0) {

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

if (rd\_size <0) {

exit(4);}

else {  
characters +=rd\_size;  
printf("%d\n",rd\_size);

}

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

if (buf[i]==' ')

words++;

{

if (rd\_size > 0){

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

printf("Words: %d\n", words);

}

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

if (wr\_size<=0){

close(in\_fd);

//closes all opened files

close(out\_fd);

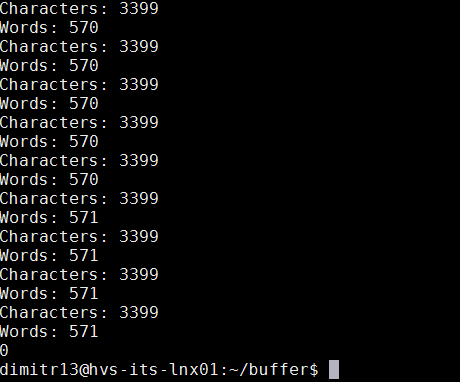
printf("File Succesfully Copied");}

}

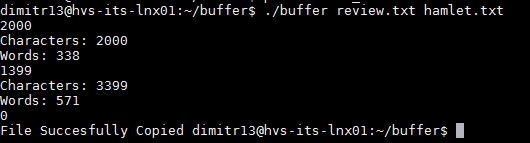
}

}

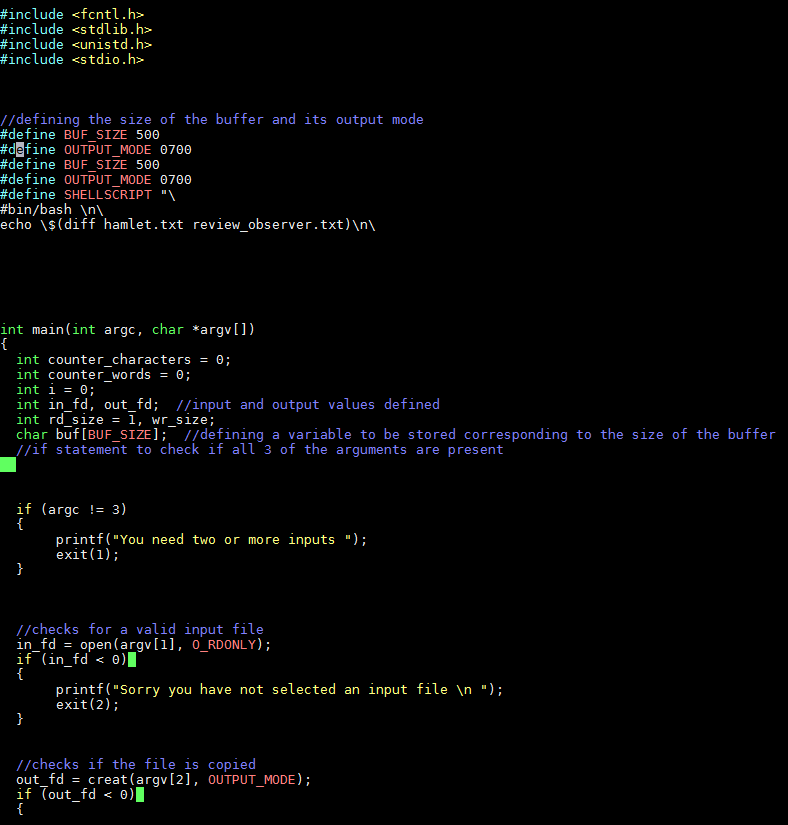
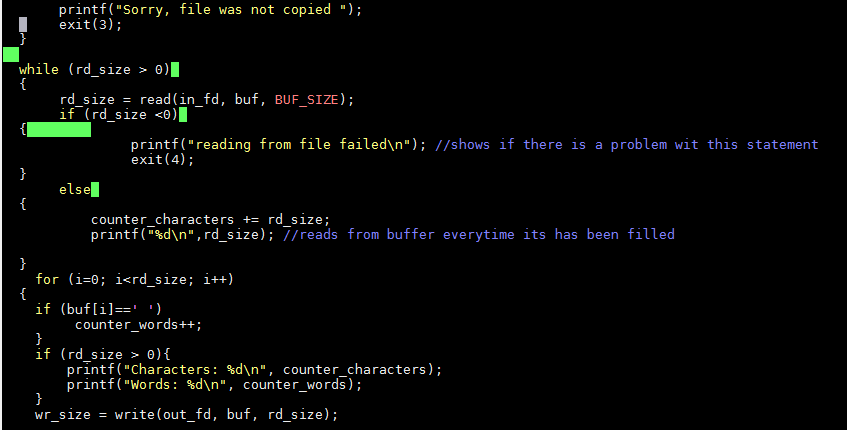
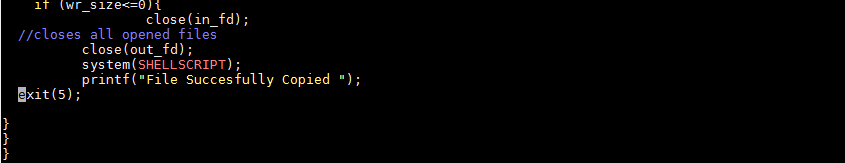
}



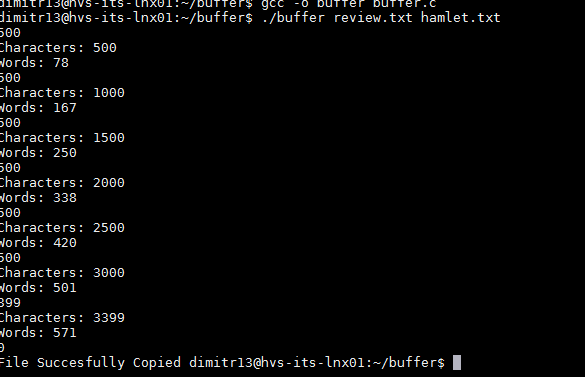
1. Impact of changing buffer size

  
  
  
If we double the size of the buffer from 500 to 2000 we receive the following output. There is less buffer refils this way

1. Updated buffer.c code to compare if two files are the same

* Comparison of review.txt and hamlet.txt

  
There is no output from the program because the files are identical

* Comparison of hamlet.txt and review\_observer.txt



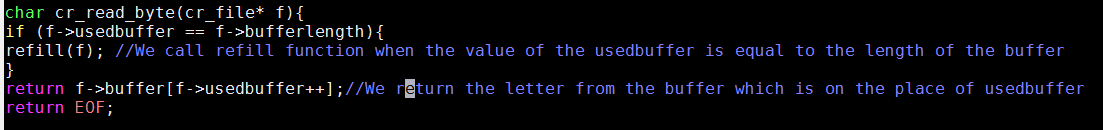
We receive an output from both files because they are different content from each other and the text has merged.

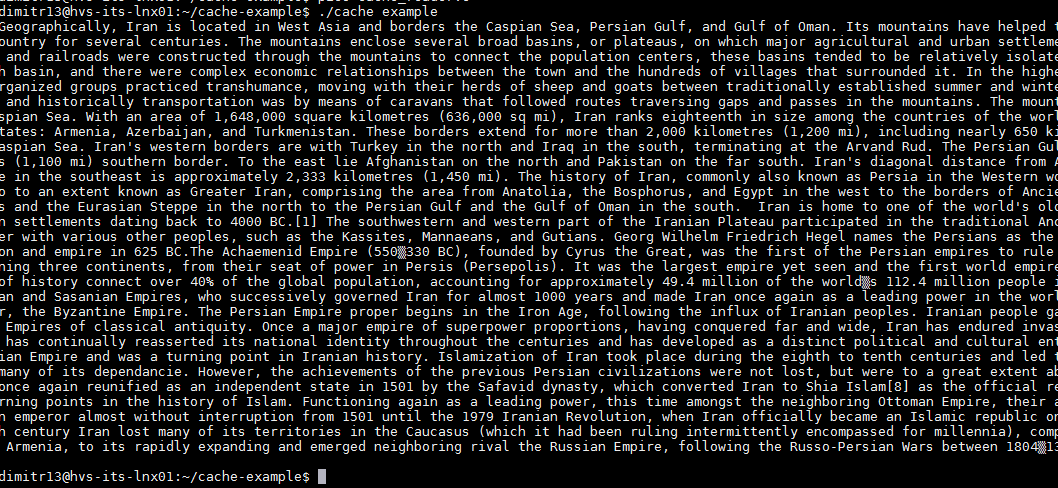
**Lab Activity 8 Cache Buffer**

1. Brief Description of Cache Buffer Activity

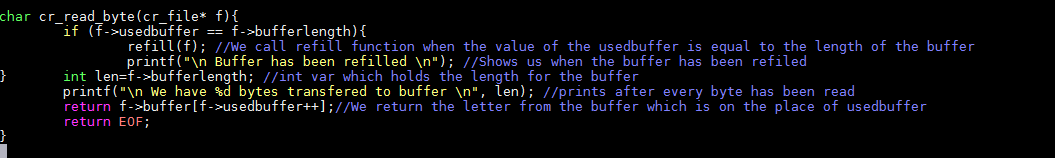
The first task of this lab is to implement the red\_byte function into a code. It should read the bytes created and transfer them to the cache buffer. When all the bytes are transferred the buffer should refill. For the next task we need to edit the code, so we can view when the buffer is refiled and when a byte of data has been read. This will involve using external ints, so we can share them between the three files created. Last change to the code will be to make it print the amount of bytes read and the number of times the buffer has been refiled.

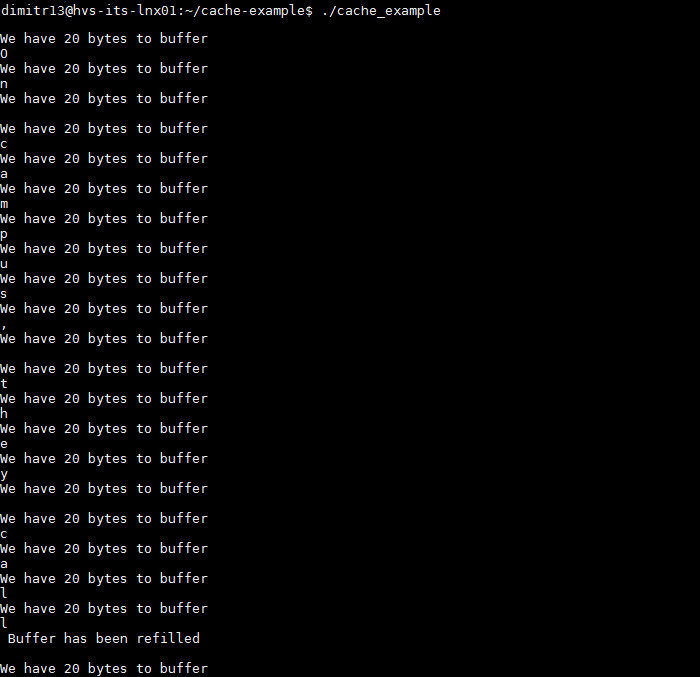
1. Commented implementation of the cr\_read\_byte function





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





1. Commented updated code showing to show how many bytes were read in total, and how many times the buffer was refilled

GNU nano 2.5.3 File: cache\_reader.c Modified

#include "cache\_reader.h"

#include "cache\_reader.c"

//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 bytes = 0; //var for bytes

int refills=0; //var for refills

int refill(cr\_file\* buff)

{

//Refills a buffer

//Only works when completely used buffer

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

return 0;

else{

refills++; // increasing refills with each loop

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

if(len<buff->bufferlength)

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

buff->buffer[i]=EOF;

return len, refills;

}

}

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 expec$

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

refill(a);

return a;

}

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

if (f->usedbuffer == f->bufferlength){

refill(f); //We call refill function when the value of the usedbuffer is$

printf("\n Buffer has been refilled \n"); //Shows us when the buffer has$

}

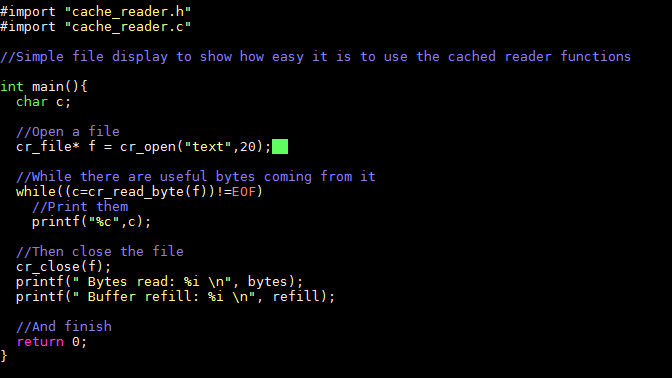
bytes++; //inc bytes counter

int len=f->bufferlength; //int var which holds the length for the buffer

printf("\n We have %d bytes transfered to buffer \n", len); //prints after every$

return f->buffer[f->usedbuffer++];//We return the letter from the buffer which i$

return EOF;

}  
  
  


OUTPUT

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

1. Brief description of the activity

For our last lab on this coursework we are exploring how can we change the old code for cache buffer and add system calls to it. From the slides and lab exercises the instructions showed us that we have to swap the already existing open, read, close with the ones fopen, fread, fclose. A few libraries have to be added “fcntl.h”,”unistd.h”,”sys/stat.h” – all need for the code to be able to send system calls to the machine (read, close and open calls). For the last task the FILE\* pointer has to be changed to a simple integer. Caching on the library has to be reduced and thus performance of the buffer will get better. All of this can be seen in the outputs in the screenshots provided for this task.

1. Changes the cache\_reader library from using the fopen, fread, fclose functions to the system call versions open, read, close

int refill(cr\_file\* buff){

//Refills a buffer

//Called when a buffer has been refiled

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

return 0;

else{

buff->usedbuffer=0;

int len=read(buff->file, buff->buffer, buff->bufferlength); //changing the fread with one for read

if(len<buff->bufferlength)

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

buff->buffer[i]=EOF; //access the buffer as an array

return len;

}

}

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

free(f->buffer);

close(f->file); //fclose is changed with close

}

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

int f; //using the int declared in the header

if ((f = open(filename, O\_RDONLY, O\_DIRECT)) == 0) // fopen function changed to the system call function “open”. using O\_DIRECT to prevent caching

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

return 0;

}

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

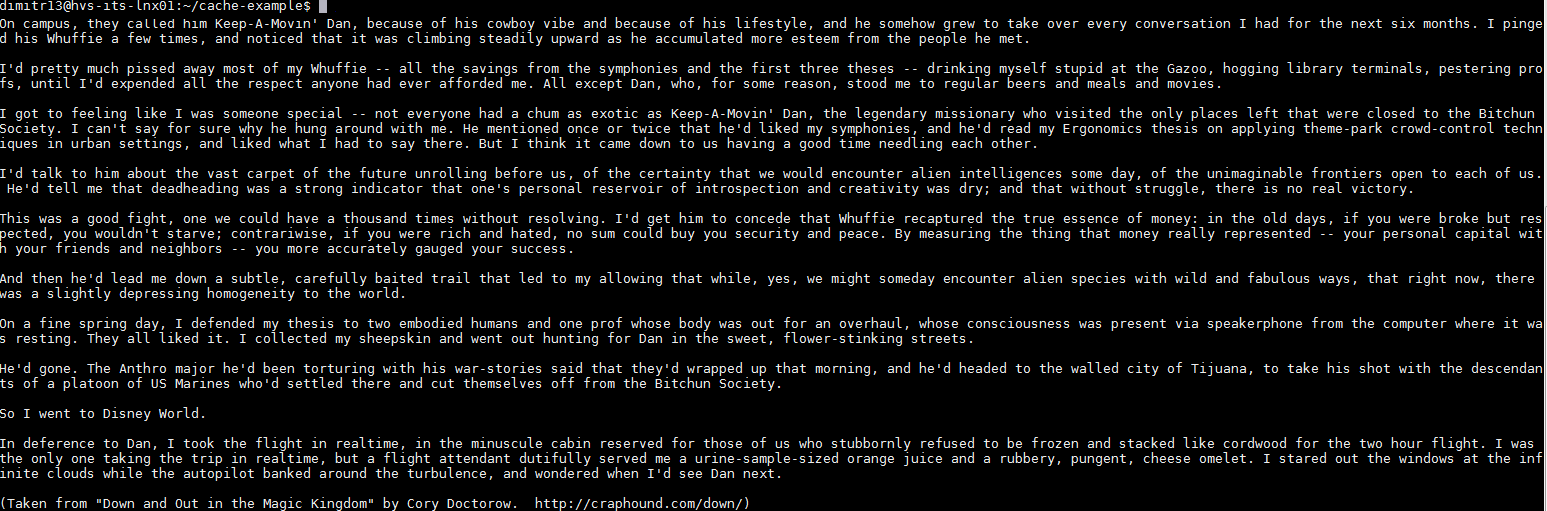
a->file=f;

a->bufferlength=buffersize;

a->usedbuffer=buffersize; //When opening a file, the file is empty so refil is going to be called

a->buffer=(char\*)pvalloc(sizeof(char)\*buffersize); // pvalloc insted of malloc to prevent faults in segmentation   
refill(a);

return a;  
}



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

cache\_reader.h file:

#include <stdio.h>

#include <stdlib.h>

// func below are added to provide access to the needed system calls

#include <fcntl.h> // for the open call

#include <unistd.h> //allows us to call the close and read

#include <sys/stat.h> //allows us open call

typedef struct

{

int file;

int bufferlength;

Int usedbuffer; // current point in the buffer

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

//has same length as "bufferlength"

} cr\_file;

  
  
Reducing the cache drastically reduces performance and it can be seen how the buffer reacts to the information received and the statistic it prints wit refills

# Bibliography

*CS UIC EDU.* [Online]   
Available at: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9\_VirtualMemory.html  
[Accessed 25 02 2017].

*krishna's Operating System.* [Online]   
Available at: https://books.google.co.uk/books?id=DuXvCF0EfDIC&pg=SA6-PA92&lpg=SA6-PA92&dq=cache+buffer+c&source=bl&ots=vjXdiCn43g&sig=VYV0FZavaeUN4Utr80vFqKsOtqM&hl=en&sa=X&ved=0ahUKEwjIjpb0i7bSAhUoIcAKHSPoAAIQ6AEIPzAF#v=onepage&q=cache%20buffer%20c&f=false  
[Accessed 25 02 2017].

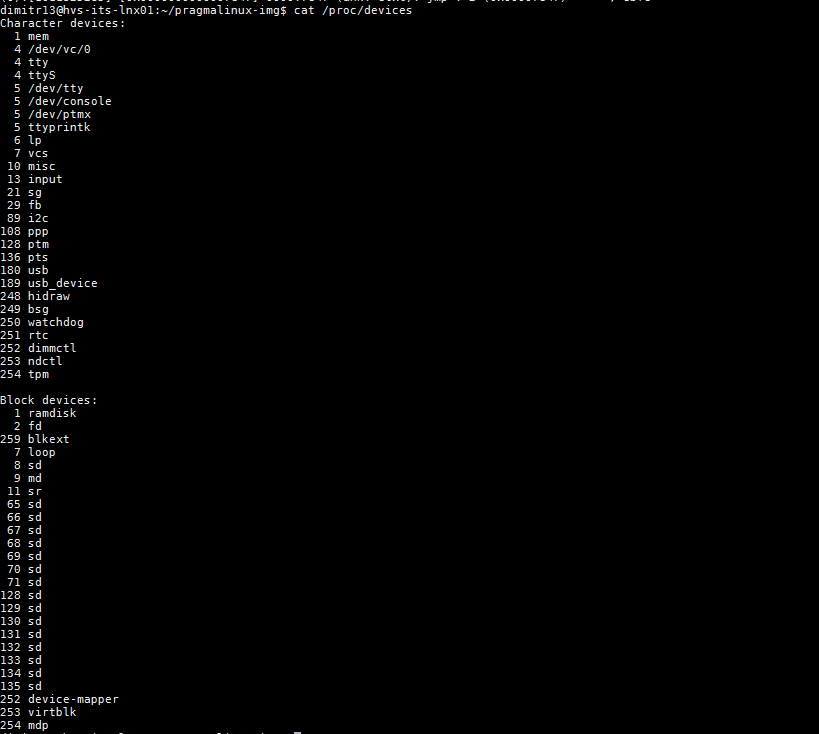
*Matt How to C.* [Online]   
Available at: https://matt.sh/howto-c  
[Accessed 22 02 2017].

*Stackoverflow - What does it mean by buffer?.* [Online]   
Available at: http://stackoverflow.com/questions/648309/what-does-it-mean-by-buffer  
[Accessed 20 02 2017].

*Tutorials on point.* [Online]   
Available at: https://www.tutorialspoint.com/cprogramming/c\_pointers.htm  
[Accessed 23 02 2017].

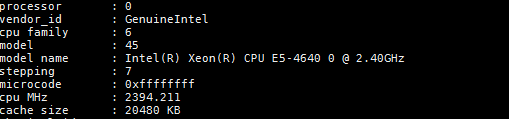
Lab5

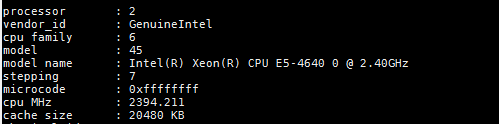
**dimitr13@hvs-its-lnx01:~/pragmalinux-img$ cat /proc/devices**

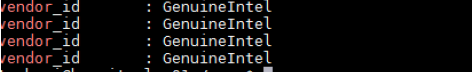


Show the number of CPUs, the producer of the CPUs and the CPU model

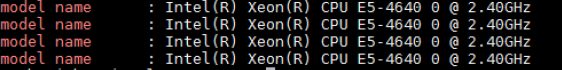
**cat /proc/cpuinfo**



  
  
**dimitr13@hvs-its-lnx01:/proc$ grep "vendor" cpuinfo**



**dimitr13@hvs-its-lnx01:/proc$ grep "model name" cpuinfo**



Show the parameters that are passed to the kernel when starting up Linux

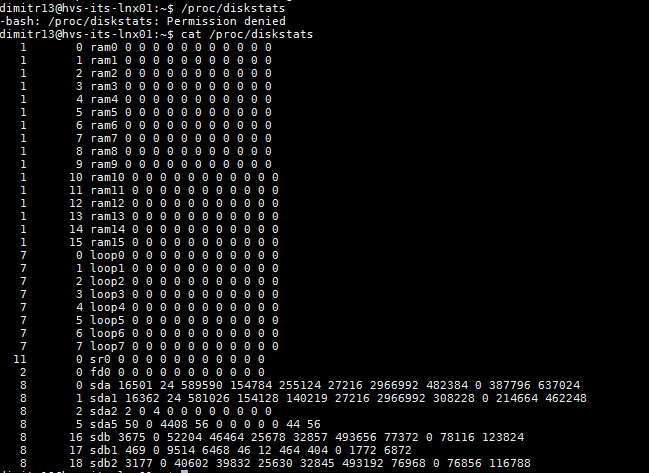


Using /proc/loadavg show the CPU utilisation in the last 5 and 10 minutes and the id of the last process used.



**The first three fields are the one-minute, five-minute, and fifteen-minute load averages.**

Using the /proc/diskstats show the name of the output devices and the number of megabytes read per second during the sampled interval.



**We are mostly interested in devices sda , sda1 and sdb as this are the only physical devices found on the**