

**ECU178 Computer Science:  
207SE Operating Systems, Security and Networks  
Portfolio**

Due on Monday, December 15th, 2014

*Dr Mark Elshaw*

**Robert Rigler : 4939377**

## Contents

<b>Item 1 - Linux Command Line</b>	<b>3</b>
1. Logfile containing evidence of activities . . . . .	3
<b>Item 2 - Assembly Code</b>	<b>4</b>
<b>Item 3 - Bootloader</b>	<b>5</b>
<b>Item 4 - Inside Proc</b>	<b>6</b>
1. List the CPU Information using the Cat Command . . . . .	6
2. Show a table of the interruppts on the system . . . . .	7
3. Show number of CPUs, the producer of the CPUs and the CPU Model. . . . .	7
4. How the parameters that are passed to the kernel when starting up linux. . . . .	8
5. Show the name of the output devices and the number of megabytes read per second during the second sampled interval. . . . .	8
6. Menu based shell script. . . . .	9
<b>Item 5 - Buffer tutorial</b>	<b>10</b>
1. Commented version of the rovided code . . . . .	10
2. Evidence of compiled code . . . . .	11
3. Code adaptation to show how many cahracters were read in total and how many times the buffer was filled . . . . .	12
3a Evidence . . . . .	13
4. Altering the buffer size . . . . .	13
5. Adapt the code so that it is possible to compare if two files are the same. . . . .	14
5a. Evidence of comparison between review.txt and argo.txt . . . . .	15
5b. Evidence of comaprison between argo.txt and reviewobserver.txt . . . . .	15
<b>Item 6 - Cache tutorial</b>	<b>16</b>
1. Complete the cr_read_byte . . . . .	16
2.Prove the file is being buffered . . . . .	16
3. Provide some statistics . . . . .	16
<b>Item 7 - Kernell</b>	<b>20</b>

## Item 1 - Linux Command Line

### 1. Logfile containing evidence of activities

--

## Item 2 - Assembly Code

## Item 3 - Bootloader

## Item 4 - Inside Proc

### 1. List the CPU Information using the Cat Command

```
rob@rob-HP-ProBook-6470b: /proc
rob@rob-HP-ProBook-6470b:/proc$ cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 58
model name     : Intel(R) Core(TM) i5-3340M CPU @ 2.70GHz
stepping       : 9
microcode      : 0x16
cpu MHz        : 1200.000
cache size     : 3072 KB
physical id    : 0
siblings       : 4
core id        : 0
cpu cores      : 2
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx r
dtscp ln constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop
_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm
2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer
aes xsave avx f16c rdrand lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_
shadow vnmi flexpriority ept vpid fsgsbase smep erms
bogomips       : 5387.64
clflush size   : 64
cache_alignmen : 64
address sizes   : 36 bits physical, 48 bits virtual
power managemen:

processor       : 1
vendor_id      : GenuineIntel
cpu family     : 6
model          : 58
model name     : Intel(R) Core(TM) i5-3340M CPU @ 2.70GHz
stepping       : 9
microcode      : 0x16
cpu MHz        : 1200.000
```

## 2. Show a table of the interrupts on the system

```
rob@rob-HP-ProBook-6470b: /proc
rob@rob-HP-ProBook-6470b:/proc$ cat /proc/interrupts
```

	CPU0	CPU1	CPU2	CPU3		
0:	17	0	0	0	IO-APIC-edge	timer
1:	127	965	84	114	IO-APIC-edge	i8042
5:	1	0	0	0	IO-APIC-edge	parpor
8:	0	0	0	1	IO-APIC-edge	rtc0
9:	161	666	69	45	IO-APIC-fastest	acpi
12:	17919	140293	11353	10149	IO-APIC-edge	i8042
16:	156	160	13	16	IO-APIC-fastest	ehci_h
18:	2	0	0	0	IO-APIC-fastest	firewl
23:	0	0	0	0	IO-APIC-edge	lis3lv
40:	0	0	0	0	PCI-MSI-edge	PCIe P
41:	0	0	0	0	PCI-MSI-edge	PCIe P
42:	0	0	0	0	PCI-MSI-edge	PCIe P
43:	0	0	0	0	PCI-MSI-edge	PCIe P
44:	0	0	0	0	PCI-MSI-edge	xhci_h
46:	13082	7835	8087	8869	PCI-MSI-edge	ahci
47:	11	0	1	3	PCI-MSI-edge	mei_me
48:	137	253	45647	54	PCI-MSI-edge	twlwl
49:	4755	31559	2541	2296	PCI-MSI-edge	i915
50:	1179	80	17	57	PCI-MSI-edge	snd_hd
NMI:	0	0	0	0	Non-maskable interrupts	
LOC:	45464	43595	48260	38011	Local timer interrupts	
SPU:	0	0	0	0	Spurious interrupts	
PMI:	0	0	0	0	Performance monitoring i	
IWI:	2279	1883	1820	1826	IRQ work interrupts	
RTR:	2	0	0	0	APIC ICR read retrles	
RES:	20170	18531	19566	18119	Rescheduling interrupts	
CAL:	494	594	574	529	Function call interrupts	

## 3. Show number of CPUs, the producer of the CPUs and the CPU Model.

```
rob@rob-HP-ProBook-6470b: /proc
rob@rob-HP-ProBook-6470b:/proc$ clear

rob@rob-HP-ProBook-6470b:/proc$ grep model /proc/cpuinfo
model          : 58
model name     : Intel(R) Core(TM) i5-3340M CPU @ 2.70GHz
model          : 58
model name     : Intel(R) Core(TM) i5-3340M CPU @ 2.70GHz
model          : 58
model name     : Intel(R) Core(TM) i5-3340M CPU @ 2.70GHz
model          : 58
model name     : Intel(R) Core(TM) i5-3340M CPU @ 2.70GHz
rob@rob-HP-ProBook-6470b:/proc$
```

4. How the parameters that are passed to the kernel when starting up linux.

```
rob@rob-HP-ProBook-6470b: /proc
rob@rob-HP-ProBook-6470b:/proc$ clear

rob@rob-HP-ProBook-6470b:/proc$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-3.13.0-39-generic root=UUID=cada5b07-62dd-4282-b91
b-46d583d8b2ab ro quiet splash vt.handoff=7
rob@rob-HP-ProBook-6470b:/proc$ █
```

5. Show the name of the output devices and the number of megabytes read per second during the second sampled interval.

```
rob@rob-HP-ProBook-6470b: /proc
rob@rob-HP-ProBook-6470b:/proc$ clear

rob@rob-HP-ProBook-6470b:/proc$ awk '{ print $3, $4}' /proc/diskstats | gre
p sda
sda 20786
sda1 166
sda2 2
sda3 162
sda4 164
sda5 14715
sda6 161
sda7 5229
rob@rob-HP-ProBook-6470b:/proc$ █
```



## 6. Menu based shell script.

Listing 1: Bash Script

```
#!/bin/bash

# DISPLAYS A MENU

5 while true;
do
echo "1. Display information about the CPU. "
echo "2. Display the interrupts system. "
echo "3. Display a process PID for a process on the system and its status. "
10 echo "4. exit. "

read input_variable
#STORES THE INPUT INTO A VARIABLE CALLED "input_variable"

15 echo "Your choice was $input_variable"

#CASE STATEMENT TO DIFFERENTIATE OUTPUT RESPECTIVE TO THE USER'S CHOICE.

case "$input_variable" in
20 1) #Display CPU info
    echo Displaying CPU information
    grep model /proc/cpuinfo
    ;;
25 2) #Display the interrupts info
    echo Displaying interrupts
    cat /proc/interrupts
    ;;
3) #Display the PID and its status.
30 echo Enter PID
    read input2
    ps -p "$input2"
    ;;
4) #stop the script
35 break
    ;;
#END CASE STATEMENT
esac
#END OF WHILE LOOP
40 done

echo "Exiting"
```

## Item 5 - Buffer tutorial

### 1. Commented version of the rovided code

Listing 2: Commented Buffer Code

```
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h> //Define header files
#include <stdio.h>

5
#define BUF_SIZE 500 //Define Buffer size as 500.
#define OUTPUT_MODE 0700 //Define file permission.

int main(int argc, char *argv[])
10 {
    int in_fd, out_fd;
    int rd_size = 1, wr_size;
    char buf[BUF_SIZE]; //Declare buffer.

15    if (argc != 3)
        exit(1);

    in_fd = open(argv[1], O_RDONLY); //Open input file.
    if (in_fd < 0)
20        exit(2);

    out_fd = creat(argv[2], OUTPUT_MODE); //Create output file.
    if (out_fd < 0)
        exit(3);

25    while (rd_size > 0) {

        rd_size = read(in_fd, buf, BUF_SIZE); // Continuously read from input file
                                                //into buffer.

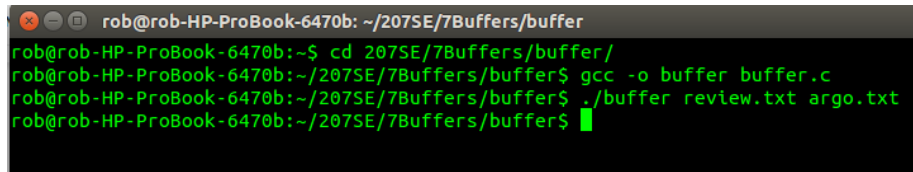
30        if (rd_size < 0)
            exit(4);

        wr_size = write(out_fd, buf, rd_size); // Continuously write from buffer into
                                                //the output file.

35        if (wr_size <= 0) {
            close(in_fd); //Close both of the files.
            close(out_fd);
            exit(5);
        }

40    }
}
```

## 2. Evidence of compiled code

A terminal window with a black background and green text. The window title is 'rob@rob-HP-ProBook-6470b: ~/207SE/7Buffers/buffer'. The terminal shows the following commands and output:

```
rob@rob-HP-ProBook-6470b:~$ cd 207SE/7Buffers/buffer/  
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ gcc -o buffer buffer.c  
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ ./buffer review.txt argo.txt  
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$
```

Argo.txt contains the exact same text that was in review.txt

### 3. Code adaptation to show how many cahracters were read in total and how many times the buffer was filled

Listing 3: Adpated Code

```

#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
5 //Define header files

#define BUF_SIZE 500 //Define Buffer size as 500.
#define OUTPUT_MODE 0700 //Define file permission.

10 int main(int argc, char *argv[])
{
    int in_fd, out_fd;
    int buf_count=0,rd_count=0;
    int rd_size = 1, wr_size;
15    char buf[BUF_SIZE]; //Declare buffer.

    if (argc != 3)
        exit(1);

20    in_fd = open(argv[1], O_RDONLY); //Open input file.
    if (in_fd < 0)
        exit(2);

    out_fd = creat(argv[2], OUTPUT_MODE); //Create output file.
25    if (out_fd < 0)
        exit(3);

    while (rd_size > 0) {

30        rd_size = read(in_fd, buf, BUF_SIZE); //Continuously read from input file
                                                //into buffer.

        rd_count+= rd_size;
        if(rd_size > 0)
            buf_count +=1; //Counts the number of times the buffer
35                                //is filled (only if rd_size is > 0

        exit(4);

        wr_size = write(out_fd, buf, rd_size); //Continuously write from buffer into
                                                //output file.

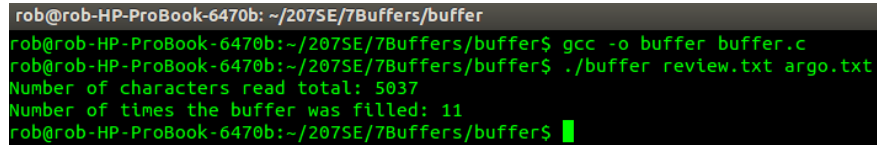
40        if (wr_size<=0){
            close(in_fd); //Close input file.
            close(out_fd); //Close output file

45        printf("Number of characters read total: %d\n",rd_count -1 );
            //Prints how many Characters were read.
        printf("Number of times the buffer was filled: %d\n",buf_count);
            //Prints how many times the buffer was filled

```

```
50 | exit(5);  
    | }  
    | }  
    | }
```

### 3a Evidence



```
rob@rob-HP-ProBook-6470b: ~/207SE/7Buffers/buffer  
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ gcc -o buffer buffer.c  
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ ./buffer review.txt argo.txt  
Number of characters read total: 5037  
Number of times the buffer was filled: 11  
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ █
```

### 4. Altering the buffer size

- Doubling the buffer size to 1000, the program filled the buffer 6 times. This is half of the original value + 1.
- Doubling the buffer size again to 2000 , the program filled the buffer 3 times which is half of 6.
- Raising the the buffer size to 10000, the program filled the buffer 1 time, indicating that the entire text was placed into the buffer.

There is a direct linear correlation between the buffer size and the amount of times that the buffer was filled.

## 5. Adapt the code so that it is possible to compare if two files are the same.

Listing 4: Adapted code

```
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
5 //Define header files

#define BUF_SIZE 500 //Define Buffer size as 500.
#define OUTPUT_MODE 0700 //Define file permission.

10 int main(int argc, char *argv[])
{
    int in_fd, in0_fd; // Create integers to hold file handles.
    int rd_size = 1; // Create integer to hold the amount of bytes in the buffer.
    char buf[BUF_SIZE]; //Declare 1st buffer.
15 char buf0[BUF_SIZE]; //Declare 2nd buffer.

    if (argc != 3)
        exit(1);

20

    in_fd = open(argv[1], O_RDONLY); //Open 1st file.
    if (in_fd < 0)
        exit(2);

25

    in0_fd = open(argv[2], O_RDONLY); //Open 2nd file.
    if (in0_fd < 0)
        exit(3);

30 while (rd_size > 0) {

    int i;
    rd_size = read(in_fd, buf, BUF_SIZE); // Read From 1st file into 1st buffer

35

    if (rd_size < 0)
        exit(4);
    rd_size = read(in0_fd, buf0, rd_size); //Read from 2nd file into 2nd buffer

    for (i = 0; i < BUF_SIZE; i++){//Loop through the contents of each buffer.

40

        if (buf[i] == buf0[i])// If buffer contents are equal, go to next buffer element.
            continue;
        else { //If buffer contents are not the same,
            //close the files and display a message
            //and exit the program.
45

            close(in_fd); //Close input file.
            close(in0_fd); //Close output file
            printf("Files are not the same. \n");
50
            exit(5);
        }
    }
}
```

```
        } //end else
    } //end for
} //end while

55 printf("Files are the same \n");// Display this message if the files are the same

} //end main
```

### 5a. Evidence of comparison between review.txt and argo.txt

```
rob@rob-HP-ProBook-6470b: ~/207SE/7Buffers/buffer
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ ./bufcomp review.txt argo.txt
Files are the same
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ █
```

### 5b. Evidence of comparison between argo.txt and reviewobserver.txt

```
rob@rob-HP-ProBook-6470b: ~/207SE/7Buffers/buffer
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ ./bufcomp argo.txt review_observer.txt
Files are not the same.
rob@rob-HP-ProBook-6470b:~/207SE/7Buffers/buffer$ █
```

## Item 6 - Cache tutorial

### 1. Complete the cr\_read\_byte

Please see the provided code in cache\_reader.c

### 2. Prove the file is being buffered

To prove the code is being buffered. I included `printf("\n");` on line 58 in the cache\_reader.c file . The program now starts a new line every time it reaches the end of the buffer ( in this example 20).

### 3. Provide some statistics

To count the number of bytes read, I created a variable called `byte_tot` in the `cr_file` structure (line12) in the cache\_reader.h file. This variable is used in the `Refill()` method (line 15). Every time the `Refill()` method is called, it adds the value of `len` (which contains the number of bytes currently being read) to itself.

The amount of times

Listing 5: cache\_example.c

```
#include "cache_reader.h"

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

5 int main() {
    char c;
    int refill_count=0;
    int byte_count=0;
    //Open a file
10 cr_file* f = cr_open("text",20);

    //While there are useful bytes coming from it
    while((c=cr_read_byte(f))!=EOF) {
        //Print them
15 printf("%c",c);

    }

20 //Then close the file
    printf("\nByte Count: %d",f->byte_tot);
    // Displaying the total number of bytes read.

    printf("\nRefill Count: %d\n",f->byte_tot/f->bufferlength);
25 //Displaying the total number of times the buffer was filled.
    // (No_of_bytes / buffersize).
    cr_close(f);

    //And finish
30 return 0;
}
```



Listing 6: cache\_reader.h

```
#include <stdio.h>
#include <stdlib.h>

//The internals of this struct aren't important
//from the user's point of view
5 typedef struct{
    FILE* file;           //File being read
    int bufferlength;     //Fixed buffer length
    int usedbuffer;       //Current point in the buffer
10    char* buffer;        //A pointer to a piece of memory
                        // same length as "bufferlength"
    int byte_tot;        //Integer to store the total amount of bytes that were read
                        //from the file.
15 } cr_file;

//Open a file with a given size of buffer to cache with
20 cr_file* cr_open(char* filename, int buffersize);

//Close an open file
void cr_close(cr_file* f);
25

//Read a byte. Will return EOF if empty.
char cr_read_byte(cr_file* f);

30

//-----

//Refill an empty buffer. Not intended for users
int refill(cr_file* buff);
```

Listing 7: cache\_reader.c

```
#include "cache_reader.h"

int refill(cr_file* buff){
    //Refills a buffer
    //Only works when completely used buffer
5    if(buff->usedbuffer!=buff->bufferlength)
        return 0;
    else{
        buff->usedbuffer=0;
10        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!
15        buff->byte_tot +=len; //Adding len to the byte total.
        return len;
    }
}

20

25
void cr_close(cr_file* f){
    free(f->buffer);
    fclose(f->file);
}

30

cr_file* cr_open(char * filename, int buffersize){

    //Info on malloc
35    //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;
40    }

    cr_file* a=(cr_file*)malloc(sizeof(cr_file));
    a->file=f;
    a->bufferlength=buffersize;
    a->usedbuffer=buffersize; //Start off with no characters,
45    // so refill will work as expected
    a->buffer=(char*)malloc(sizeof(char)*buffersize);
    a->byte_tot =0;
    refill(a);
50    return a;
}
```

```
//-----  
char cr_read_byte(cr_file* f){  
55  
    char btoRet; // byte to hold the character to return.  
    if (f->usedbuffer >= f->bufferlength){ // if the buffer is all used, refill()  
        printf(" \n "); // starts a new line very time the buffer needs to be refilled.  
        refill(f);  
60  
    }  
    else{ // If buffer hasn't been fully used, return the chracter and increase  
        // the usedBuffer position by 1.  
65  
        btoRet = f->buffer[f->usedbuffer]; //Place next character in the  
            //btoRet variable.  
        f->usedbuffer +=1; //Move the buffer position up by 1.  
        return btoRet; //return the varibale.  
70  
    }  
}
```

## **Item 7 - Kernell**