

**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>
1. Right-Angled triangle . . . . .	4
2. Isosceles Triangle . . . . .	6
<b>Item 3 - Bootloader</b>	<b>9</b>
1. Make a Bootloader that displays my name . . . . .	9
2. Make a Bootloader that displays a triangle of dots . . . . .	10
<b>Item 4 - Inside Proc</b>	<b>11</b>
1. List the CPU Information using the Cat Command . . . . .	11
2. Show a table of the interrupts on the system . . . . .	12
3. Show number of CPUs, the producer of the CPUs and the CPU Model. . . . .	12
4. How the parameters that are passed to the kernel when starting up linux. . . . .	13
5. Show the name of the output devices and the number of megabytes read per second during the second sampled interval. . . . .	13
6. Menu based shell script. . . . .	14
<b>Item 5 - Buffer tutorial</b>	<b>15</b>
1. Commented version of the provided code . . . . .	15
2. Evidence of compiled code . . . . .	16
3. Code adaptation to show how many cahracters were read in total and how many times the buffer was filled . . . . .	17
3a Evidence . . . . .	18
4. Altering the buffer size . . . . .	18
5. Adapt the code so that it is possible to compare if two files are the same. . . . .	19
5a. Evidence of comparison between review.txt and argo.txt . . . . .	20
5b. Evidence of comaprison between argo.txt and reviewobserver.txt . . . . .	20
<b>Item 6 - Cache tutorial</b>	<b>21</b>
1. Complete the cr_read_byte function . . . . .	21
2. Prove the file is being buffered . . . . .	21
3. Provide some statistics . . . . .	21
<b>Item 7 - Kernel</b>	<b>25</b>
1. Description of the commands for loading and unloading Linux kernel modules. . . . .	25
2. List of the loaded modules . . . . .	25
3. Description of four loaded modules . . . . .	27

## Item 1 - Linux Command Line

### 1. Logfile containing evidence of activities

--

## Item 2 - Assembly Code

### 1. Right-Angled triangle

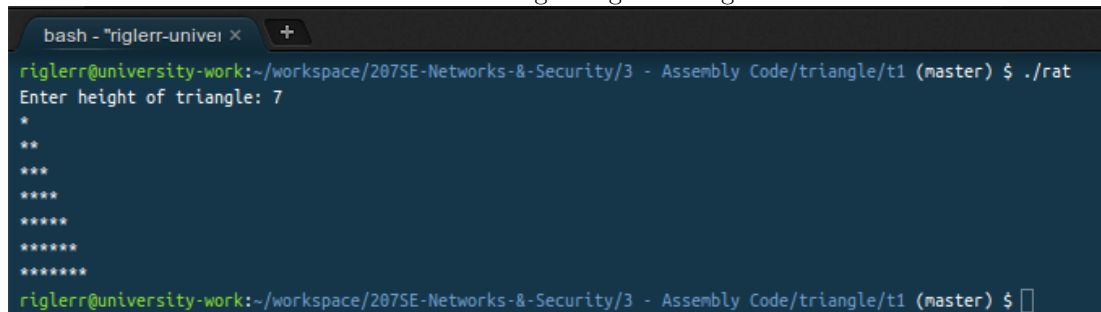
Using x86 Assembly code, draw a Right-angled triangle

Listing 1: Right angled triangle

```
1 section .data
2
3 Prompt db 'Enter height of triangle: '
4 pLen equ $-Prompt
5 chr db '*'
6 nl:     db "  ", 0x0a ; variable to draw a new line
7 nl_len equ $-nl ; length of new line variable
8
9 section .bss
10 num resb 2 ;reserve 2 bytes for the input variable
11
12 section .text
13 global _start
14 _start:
15
16 ;Ask user for size of triangle
17 mov eax,4
18 mov ebx,1
19 mov ecx,Prompt
20 mov edx,pLen
21 int 80h
22
23 ;store the variable
24 mov eax,3
25 mov ebx,0
26 mov ecx,num
27 mov edx,2
28 int 80h
29
30 mov ecx, [num] ;dereference input and store in ecx
31 sub ecx,'0' ;convert from ascii to decimal
32 xor ch,ch ; clear upper half of ecx
33
34 mov ebx,1
35
36 lo: ;outer loop, amount of lines in triangle
37     push ecx ; push outer loop count to stack
38     mov ecx,ebx ; place inner loop count in the loop counter
39     li: ; inner loop, amount of stars in line
40
41         push ecx
42         push ebx ;push ecx and ebx to stack so they can be used in drawing
43         mov eax,4
44         mov ebx,1
45         mov ecx,chr
46         mov edx,1
47         int 80h ;draw star
```

```
48         pop ebx
49         pop ecx ; pop ecx last, so it is has the correct loop counter value
50     loop li ; end of inner loop
51
52     push ebx ; push ebx to stack, so to use ebx in starting a new line
53     mov eax, 4
54     mov ebx, 1
55     mov ecx, nl
56     mov edx, nl_len
57     int 0x80 ;draws a new line
58
59     pop ebx ;pop inner loop count off stack
60     inc ebx ;increment inner loop count (to draw 1 more triangle next iteration)
61     pop ecx ;pop outler loop count off stack to use as counter for lo
62 loop lo ;end of outer loop
63     int 80h;
64
65 mov eax,1
66 mov ebx,0
67 int 80h;exit
```

## Evidence of Right angled triangle



```
bash - "riglerr-univer x +
riglerr@university-work:~/workspace/207SE-Networks-&-Security/3 - Assembly Code/triangle/t1 (master) $ ./rat
Enter height of triangle: 7
*
**
***
****
*****
*****
*****
riglerr@university-work:~/workspace/207SE-Networks-&-Security/3 - Assembly Code/triangle/t1 (master) $
```

## 2. Isosceles Triangle

Using x86 Assembly code, draw an Isosceles triangle

Listing 2: Isosceles Triangle

```
1 section .data
2
3 Prompt db 'Enter height of triangle: '
4 pLen equ $-Prompt
5 chr db '*'
6 nl:      db "  ", 0x0a ; variable to draw a new line
7 nl_len  equ $-nl ; length of new line variable
8 ns: db " "; variable to draw a space
9 ns_l equ $-ns ; length of space variable
10
11 section .bss
12 num resb 2 ; reserve 2 bytes for the input variable
13 width resb 2
14
15 section .text
16 global _start
17 _start:
18
19 ; Ask user for size of triangle
20 mov eax, 4
21 mov ebx, 1
22 mov ecx, Prompt
23 mov edx, pLen
24 int 80h
25
26 ; store the variable
27 mov eax, 3
28 mov ebx, 0
29 mov ecx, num
30 mov edx, 2
31 int 80h
32
33 mov ecx, [num] ; dereference input and store in ecx
34 sub ecx, '0' ; convert from ascii to decimal
35 xor ch, ch ; clear upper half of ecx
36
37 mov ebx, 1
38 mov eax, ecx
39
40 ; this block places w=2n-1 into eax
41 push ebx ; push inner loop to stack
42 mov ebx, 2 ; move 2 into ebx
43 mul ebx ; multiply eax by 2
44 sub eax, 1 ; subtract 1 from eax
45 pop ebx
46 mov [width], eax ; place width value into width variable
47
48 lo: ; outer loop, amount of lines in triangle
49
```

```
50     push ecx ; push outer loop count to stack
51
52     ;This block works out the number of spaces to print before drawing
53     ;noOfSpace = (width-noOfAsterisks)/2
54     mov ecx,[width] ; make loop counter equal to the width of the triangle
55     sub ecx,ebx ;subtract number of asterisks
56     shr ecx,1 ; shift right, divides ecx by 2^1 (2)
57     jz l2 ; jump to the l2 label if the result of the division was 0
58
59     l3:
60     push ebx ; push ebx to stack, so to use ebx in starting a new line
61     push ecx
62     mov eax, 4
63     mov ebx, 1
64     mov ecx, ns
65     mov edx, ns_1
66     int 0x80 ;draws a new line
67     pop ecx ;pop inner loop count off stack
68     pop ebx
69
70     loop l3
71
72     l2:
73
74     mov ecx,ebx ; place inner loop count in the loop counter
75     li: ; inner loop, amount of stars in line
76
77         push ecx
78         push ebx ;push ecx and ebx to stack so they can be used in drawing
79         mov eax,4
80         mov ebx,1
81         mov ecx,chr
82         mov edx,1
83         int 80h ;draw star
84         pop ebx
85         pop ecx ; pop ecx last, so it is has the correct loop counter value
86     loop li ; end of inner loop
87
88     push ebx ; push ebx to stack, so to use ebx in starting a new line
89     mov eax, 4
90     mov ebx, 1
91     mov ecx, nl
92     mov edx, nl_len
93     int 0x80 ;draws a new line
94
95     pop ebx ;pop inner loop count off stack
96     ; add two to the inner loop counter,
97     ;each line always has an odd number of asterisks
98     add ebx,2
99
100
101     pop ecx ;pop outler loop count off stack to use as counter for lo
102 loop lo ;end of outer loop
```





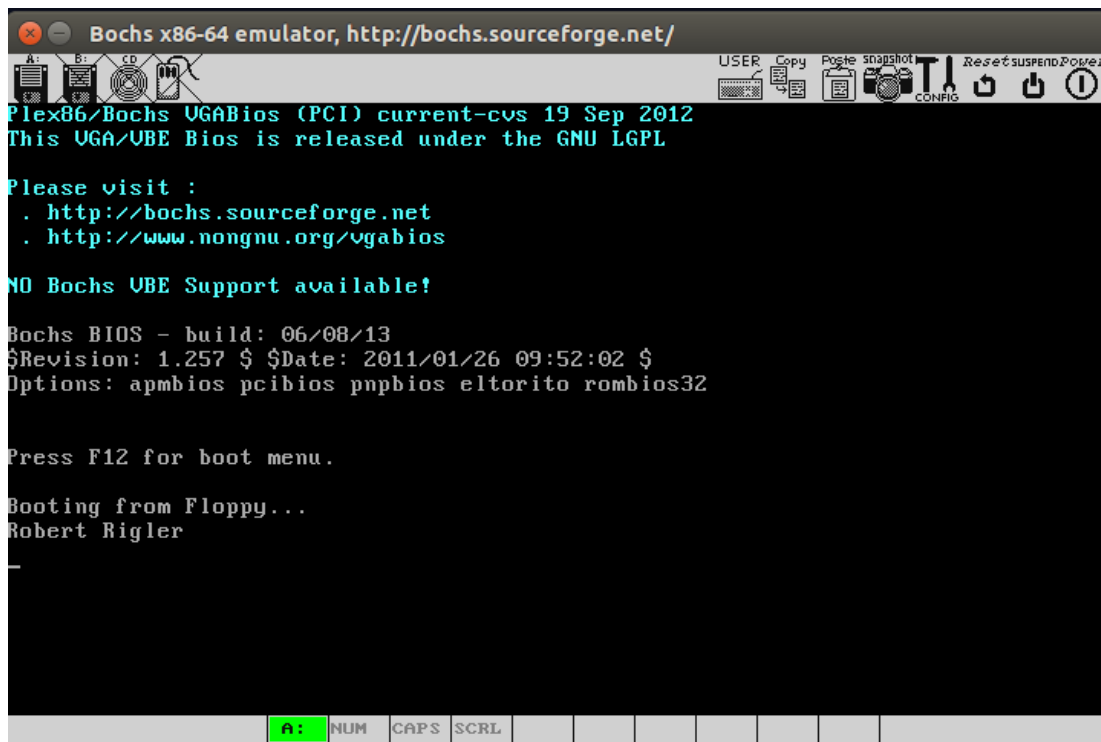
## Item 3 - Bootloader

### 1. Make a Bootloader that displays my name

Listing 3: Bootloader Assembly code

```
1  [BITS 16]
2  [ORG 0X7C00]
3  top:
4      mov ax,0x0000
5      mov ds,ax
6      mov si,HelloWorld
7      call writeString
8      jmp $
9
10 writeString:
11     mov ah,0x0E ;Display Character
12     mov bh,0x00
13     mov bl,0x07
14
15 nextchar:
16     lodsb
17     cmp al,0
18     jz done
19     int 0x10
20     jmp nextchar
21
22 done:
23     ret
24     HelloWorld db 'Robert Rigler',13,10,0
25     times 510-($-$$) db 0
26
27 dw 0xAA55
```

## Evidence of working code



2. Make a Bootloader that displays a triangle of dots

## 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_alignm   : 64
address sizes   : 36 bits physical, 48 bits virtual
power manage    :

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-fasteoi	acpi
12:	17919	140293	11353	10149	IO-APIC-edge	i8042
16:	156	160	13	16	IO-APIC-fasteoi	ehci_h
18:	2	0	0	0	IO-APIC-fasteoi	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 4: Bash Script

```
1 #!/bin/bash
2
3 # DISPLAYS A MENU
4
5 while true;
6 do
7 echo "1. Display information about the CPU. "
8 echo "2. Display the interrupts system. "
9 echo "3. Display a process PID for a process on the system and its status. "
10 echo "4. exit. "
11
12 read input_variable
13 #STORES THE INPUT INTO A VARIABLE CALLED "input_variable"
14
15 echo "Your choice was $input_variable"
16
17 #CASE STATEMENT TO DIFFERENTIATE OUTPUT RESPECTIVE TO THE USER'S CHOICE.
18
19 case "$input_variable" in
20
21 1) #Display CPU info
22    echo Displaying CPU information
23    grep model /proc/cpuinfo
24    ;;
25 2) #Display the interrupts info
26    echo Displaying interrupts
27    cat /proc/interrupts
28    ;;
29 3) #Display the PID and its status.
30    echo Enter PID
31    read input2
32    ps -p "$input2"
33    ;;
34 4) #stop the script
35    break
36    ;;
37 #END CASE STATEMENT
38 esac
39 #END OF WHILE LOOP
40 done
41
42 echo "Exiting"
```

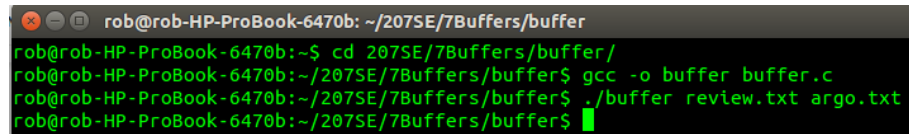
## Item 5 - Buffer tutorial

### 1. Commented version of the provided code

Listing 5: Commented Buffer Code

```
1 #include <fcntl.h>
2 #include <stdlib.h>
3 #include <unistd.h> //Define header files
4 #include <stdio.h>
5
6 #define BUF_SIZE 500 //Define Buffer size as 500.
7 #define OUTPUT_MODE 0700 //Define file permission.
8
9 int main(int argc, char *argv[])
10 {
11     int in_fd, out_fd;
12     int rd_size = 1, wr_size;
13     char buf[BUF_SIZE]; //Declare buffer.
14
15     if (argc != 3)
16         exit(1);
17
18     in_fd = open(argv[1], O_RDONLY); //Open input file.
19     if (in_fd < 0)
20         exit(2);
21
22     out_fd = creat(argv[2], OUTPUT_MODE); //Create output file.
23     if (out_fd < 0)
24         exit(3);
25
26     while (rd_size > 0) {
27
28         rd_size = read(in_fd, buf, BUF_SIZE); // Continuously read from input file
29                                             //into buffer.
30         if (rd_size < 0)
31             exit(4);
32
33         wr_size = write(out_fd, buf, rd_size); // Continuously write from buffer into
34                                             //the output file.
35         if (wr_size <= 0) {
36             close(in_fd); //Close both of the files.
37             close(out_fd);
38             exit(5);
39         }
40     }
41 }
```

## 2. Evidence of compiled code

A terminal window with a dark 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 6: Adpated Code

```
1 #include <fcntl.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 //Define header files
6
7 #define BUF_SIZE 500 //Define Buffer size as 500.
8 #define OUTPUT_MODE 0700 //Define file permission.
9
10 int main(int argc, char *argv[])
11 {
12     int in_fd, out_fd;
13     int buf_count=0,rd_count=0;
14     int rd_size = 1, wr_size;
15     char buf[BUF_SIZE]; //Declare buffer.
16
17     if (argc != 3)
18         exit(1);
19
20     in_fd = open(argv[1], O_RDONLY); //Open input file.
21     if (in_fd < 0)
22         exit(2);
23
24     out_fd = creat(argv[2], OUTPUT_MODE); //Create output file.
25     if (out_fd < 0)
26         exit(3);
27
28     while (rd_size > 0) {
29
30         rd_size = read(in_fd, buf, BUF_SIZE); //Continuously read from input file
31                                             //into buffer.
32         rd_count+= rd_size; //Adds rd_size to the total read count
33         if(rd_size > 0)
34             buf_count +=1; //Counts the number of times the buffer
35                           //is filled (only if rd_size is > 0
36         exit(4);
37
38         wr_size = write(out_fd, buf, rd_size); //Continuously write from buffer into
39                                             //output file.
40         if (wr_size<=0){
41             close(in_fd); //Close input file.
42             close(out_fd); //Close output file
43
44
45             printf("Number of characters read total: %d\n",rd_count -1 );
46             //Prints how many Characters were read.
47         printf("Number of times the buffer was filled: %d\n",buf_count);
48             //Prints how many times the buffer was filled
49     }
```

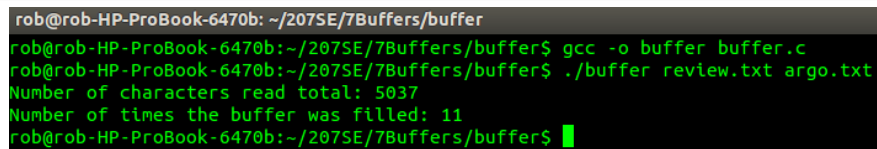
```
50 | exit(5);  
51 | }  
52 | }  
53 | }
```

Firstly I created two variables to hold the Buffer count (*buf\_count*) and the character count(*rd\_count*) (*line 14*).

Then to accumulate the total numbers of characters read I added the value of *rd\_size* to the *rd\_count* variable (*line 32*) each time the text was read into the buffer.

To count the number of times the buffer was filled, each time *rd\_size* was filled and its value above 0, *buf\_count* is incremented by 1(*line 34*).

### 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 7: Adapted code

```
1 #include <fcntl.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 //Define header files
6
7 #define BUF_SIZE 500 //Define Buffer size as 500.
8 #define OUTPUT_MODE 0700 //Define file permission.
9
10 int main(int argc, char *argv[])
11 {
12     int in_fd, in0_fd; // Create integers to hold file handles.
13     int rd_size = 1; // Create integer to hold the amount of bytes in the buffer.
14     char buf[BUF_SIZE]; //Declare 1st buffer.
15     char buf0[BUF_SIZE]; //Declare 2nd buffer.
16
17     if (argc != 3)
18         exit(1);
19
20
21
22     in_fd = open(argv[1], O_RDONLY); //Open 1st file.
23     if (in_fd < 0)
24         exit(2);
25
26     in0_fd = open(argv[2], O_RDONLY); //Open 2nd file.
27     if (in0_fd < 0)
28         exit(3);
29
30     while (rd_size > 0) {
31
32         int i;
33         rd_size = read(in_fd, buf, BUF_SIZE); // Read From 1st file into 1st buffer
34
35         if (rd_size < 0)
36             exit(4);
37         rd_size = read(in0_fd, buf0, rd_size); //Read from 2nd file into 2nd buffer
38
39         for (i = 0; i < BUF_SIZE; i++){//Loop through the contents of each buffer.
40
41             if (buf[i] == buf0[i])// If buffer contents are equal, go to next buffer element.
42                 continue;
43             else { //If buffer contents are not the same,
44                 //close the files and display a message
45                 //and exit the program.
46
47                 close(in_fd); //Close input file.
48                 close(in0_fd); //Close output file
49                 printf("Files are not the same. \n");
50                 exit(5);
```

```
51  
52         } //end else  
53     } //end for  
54 } //end while  
55  
56 printf("Files are the same \n");// Display this message if the files are the same  
57  
58 }//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` function

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 (line 12) in the `cache_reader.h` file. This variable is used in the `Refill()` method (line 15)(*cache\_reader file*). 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 the buffer was refilled, was calculated by dividing the number of bytes read from the text by the size of the buffer.

Listing 8: `cache_example.c`

```
1  #include "cache_reader.h"
2
3  //Simple file display to show how easy it is to use the cached reader functions
4
5  int main(){
6      char c;
7      int refill_count=0;
8      int byte_count=0;
9      //Open a file
10     cr_file* f = cr_open("text",20);
11
12     //While there are useful bytes coming from it
13     while((c=cr_read_byte(f))!=EOF){
14         //Print them
15         printf("%c",c);
16
17     }
18
19
20     //Then close the file
21     printf("\nByte Count: %d",f->byte_tot);
22     // Displaying the total number of bytes read.
23
24     printf("\nRefill Count: %d\n",f->byte_tot/f->bufferlength);
25     //Displaying the total number of times the buffer was filled.
26     //(No_of_bytes / buffersize).
27     cr_close(f);
28
29     //And finish
30     return 0;
31 }
```

Listing 9: cache\_reader.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //The internals of this struct aren't important
5 //from the user's point of view
6 typedef struct{
7     FILE* file;           //File being read
8     int bufferlength;     //Fixed buffer length
9     int usedbuffer;       //Current point in the buffer
10    char* buffer;         //A pointer to a piece of memory
11                        // same length as "bufferlength"
12    int byte_tot;         //Integer to store the total amount of bytes that were read
13                        //from the file.
14 } cr_file;
15
16
17 //Open a file with a given size of buffer to cache with
18 cr_file* cr_open(char* filename, int buffersize);
19
20
21 //Close an open file
22 void cr_close(cr_file* f);
23
24 //Read a byte. Will return EOF if empty.
25 char cr_read_byte(cr_file* f);
26
27
28
29 //-----
30
31 //Refill an empty buffer. Not intended for users
32 int refill(cr_file* buff);
```

Listing 10: cache\_reader.c

```
1 #include "cache_reader.h"
2
3 int refill(cr_file* buff){
4     //Refills a buffer
5     //Only works when completely used buffer
6     if(buff->usedbuffer!=buff->bufferlength)
7         return 0;
8     else{
9         buff->usedbuffer=0;
10        int len=fread(buff->buffer, sizeof(char), buff->bufferlength, buff->file);
11        //If we didn't fill the buffer, fill up with EOF
12        if(len<buff->bufferlength)
13            for(int i=len;i<buff->bufferlength;i++)
14                buff->buffer[i]=EOF; //Accessing like an array!
15        buff->byte_tot +=len; //Adding len to the byte total.
16        return len;
17    }
18
19 }
20
21 void cr_close(cr_file* f){
22     free(f->buffer);
23     fclose(f->file);
24 }
25
26
27 cr_file* cr_open(char * filename, int buffersize){
28
29     //Info on malloc
30     //http://www.space.unibe.ch/comp_doc/c_manual/C/FUNCTIONS/malloc.html
31     FILE* f;
32     if ((f = fopen(filename, "r")) == NULL){
33         fprintf(stderr, "Cannot open %s\n", filename);
34         return 0;
35     }
36
37     cr_file* a=(cr_file*)malloc(sizeof(cr_file));
38     a->file=f;
39     a->bufferlength=buffersize;
40     a->usedbuffer=buffersize; //Start off with no characters,
41                             // so refill will work as expected
42     a->buffer=(char*)malloc(sizeof(char)*buffersize);
43     a->byte_tot =0;
44     refill(a);
45     return a;
46 }
47
48
49
50
51
52
```

```
53
54
55 //-----
56 char cr_read_byte(cr_file* f){
57
58     char btoRet; // byte to hold the character to return.
59     if (f->usedbuffer >= f->bufferlength){ // if the buffer is all used, refill()
60         printf(" \n "); // starts a new line very time the buffer needs to be refilled.
61         refill(f);
62
63     }
64     else{ // If buffer hasn't been fully used, return the chracter and increase
65         // the usedBuffer position by 1.
66
67         btoRet = f->buffer[f->usedbuffer]; //Place next character in the
68             //btoRet variable.
69         f->usedbuffer +=1; //Move the buffer position up by 1.
70         return btoRet; //return the varibale.
71
72     }
73
74 }
```



## Item 7 - Kernel

### 1. Description of the commands for loading and unloading Linux kernel modules.

- lsmod - Lists all of the available loaded kernel modules.
- modinfo - Displays information about a particular kernel module.
- insmod - Install and load a kernel module
- rmmod - Unload a module

### 2. List of the loaded modules

This list was generated by using the lsmod command

```
rob@rob-HP-ProBook-6470b: ~
rob@rob-HP-ProBook-6470b:~$ lsmod
Module                  Size  Used by
ctr                     13049  1
ccm                     17773  1
nvram                   14411  0
btusb                   32412  0
uvcvideo                80885  0
videobuf2_vmalloc      13216  1 uvcvideo
videobuf2_memops       13362  1 videobuf2_vmalloc
videobuf2_core         40664  1 uvcvideo
videodev               134688  2 uvcvideo,videobuf2_core
snd_hda_codec_hdmi     46368  1
snd_hda_codec_idt      54762  1
hp_wmi                  14062  0
sparse_keymap          13948  1 hp_wmi
intel_rapl              18773  0
x86_pkg_temp_thermal   14205  0
intel_powerclamp       14705  0
coretemp               13435  0
kvm                     455835  0
crct10dif_pclmul       14289  0
crc32_pclmul           13113  0
ghash_clmulni_intel    13216  0
arc4                    12608  2
aesni_intel            55624  2
aes_x86_64             17131  1 aesni_intel
lrw                     13286  1 aesni_intel
gf128mul               14951  1 lrw
glue_helper            13990  1 aesni_intel
ablk_helper            13597  1 aesni_intel
cryptd                 20359  3 ghash_clmulni_intel,aesni_intel,ablk_helper
iwlvm                  232285  0
mac80211               630653  1 iwlvm
joydev                  17381  0
serio_raw              13462  0
iwlwifi                169932  1 iwlvm
rfcomm                  69160  8
cfg80211               484040  3 iwlwifi,mac80211,iwlvm
snd_hda_intel          56451  3
snd_hda_codec          192906  3 snd_hda_codec_hdmi,snd_hda_codec_idt,snd_hda_intel
lpc_ich                 21080  0
```

```

rob@rob-HP-ProBook-6470b: ~
snd_page_alloc      18710  2  snd_pcm,snd_hda_intel
snd_seq_midi        13324  0
i915                784207  4
snd_seq_midi_event  14899  1  snd_seq_midi
wmi                 19177  1  hp_wmi
snd_rawmidi         30144  1  snd_seq_midi
snd_seq             61560  2  snd_seq_midi_event,snd_seq_midi
drm_kms_helper      55071  1  i915
drm                 303102  5  i915,drm_kms_helper
snd_seq_device      14497  3  snd_seq,snd_rawmidi,snd_seq_midi
i2c_algo_bit       13413  1  i915
tpm_infineon        17372  0
snd_timer           29482  2  snd_pcm,snd_seq
snd                 69322  17  snd_hwdep,snd_timer,snd_hda_codec_hdmi
eq_device,snd_seq_midi
video               19476  1  i915
hp_accel            26012  0
lis3lv02d           20156  1  hp_accel
soundcore           12680  1  snd
mei_me              18627  0
hp_wireless         12637  0
input_polldev       13896  1  lis3lv02d
mei                 82276  1  mei_me
mac_hid             13205  0
binfmt_misc         17468  1
parport_pc          32701  1
ppdev               17671  0
lp                  17759  0
parport             42348  3  lp,ppdev,parport_pc
psmouse             106714  0
firewire_ohci       40409  0
ahci                 25819  3
e1000e              254433  0
sdhci_pci            23172  0
libahci             32716  1  ahci
sdhci                43015  1  sdhci_pci
firewire_core       68769  1  firewire_ohci
ptp                  18933  1  e1000e
crc_itu_t           12707  1  firewire_core
pps_core            19382  1  ptp
rob@rob-HP-ProBook-6470b:~$ git pull

```

### 3. Description of four loaded modules

My selected modules are ccm, nvram, btusb, uvcvideo

- **ccm**

The 'ccm' module is the Clock Control Module, it controls the hardware clocks on the motherboard which consists of two crystal oscillators and a control chip. The 'ccm' ensures that the proper time is kept, so that any devices connected to the clock are using the correct time.

- **nvram**

The 'nvram' module is the Non-Volatile memory module. This driver allows the user to access the contents of the RAM in real time.

- **btusb**

The 'btusb' module is a generic Bluetooth USB driver. This driver controls the USB Bluetooth devices connected to the computer.

- **uvcvideo**

The 'uvcvideo' module is a UVC (USB Video Class) driver which controls webcams that are compliant to the UVC specification. It makes sure that the device is compatible with the video streaming functionality on the Universal-Serial-Bus.