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

Due on Monday, December 15th, 2014

Dr Mark Elshaw

Robert Rigler : 4939377

Contents

Item 1 - Linux Command Line

1. Logfile containing evidence of activities

Item 2 - Assembly Code

1. Right-Angled triangle

The aim of this task was to produce a right-angled triangle of height specified by the user.
The process outline for this task is quite simple:

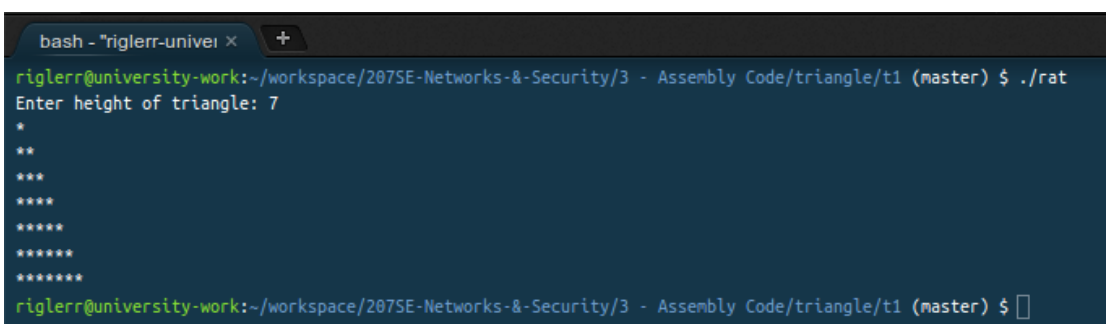
1. Get the user input and assign that value to the ecx register
2. Start the outer loop, then push the outerLoop value to the stack and move the innerLoop value into ecx.
3. Start the InnerLoop and print the '*' character the number of times specified in ecx.
4. After the InnerLoop is finished, start a new line and pop the OuterLoop value off of the stack.
5. Increment the InnerLoop value by 1 (So that one more triangle is draw next iteration) and repeat the Outerloop with the value in the ecx register (which is automatically decremented by one each time the Outerloop is called).

Listing 1: Right angled triangle

```
section .data

Promt db 'Enter height of triangle: '
pLen equ $-Promt
chr db '*'
nl:     db "   ", 0x0a ; varab;e to draw a new line
nl_len  equ $-nl ; length of new line varable

section .bss
num resb 2 ;reserve 2 bytes for the input variable

section .text
global _start
_start:

;Ask user for size of triangle
mov eax,4
mov ebx,1
mov ecx,Promt
mov edx,pLen
int 80h

;store the variable
mov eax,3
mov ebx,0
mov ecx,num
mov edx,2
int 80h

mov ecx, [num] ;derefference input and store in ecx
sub ecx,'0' ;convert from ascii to decimal
xor ch,ch ; clear upper half of ecx

```

```
mov ebx,1

lo: ;outer loop, amount of lines in triangle
    push ecx ; push outer loop count to stack
    mov ecx,ebx ; place inner loop count in the loop counter
        li: ; inner loop, amount of stars in line

            push ecx
            push ebx ;push ecx and ebx to stack so they can be used in drawing
            mov eax,4
            mov ebx,1
            mov ecx,chr
            mov edx,1
            int 80h ;draw star
            pop ebx
            pop ecx ; pop ecx last, so it is has the correct loop counter value
        loop li ; end of inner loop

    push ebx ; push ebx to stack, so to use ebx in starting a new line
    mov eax, 4
    mov ebx, 1
    mov ecx, nl
    mov edx, nl_len
    int 0x80 ;draws a new line

    pop ebx ;pop inner loop count off stack
    inc ebx ;increment inner loop count (to draw 1 more triangle next iteration)
    pop ecx ;pop outler loop count off stack to use as counter for lo
loop lo ;end of outer loop
    int 80h;

mov eax,1
mov ebx,0
int 80h;exit
```

Evidence of Right angled triangle



2. Isosceles Triangle

The process to draw this triangle is similar to the previous task, but an additional loop is needed to print the required number of spaces before drawing the asterisks.

1. Get the user input for the height of the triangle and place it in the ecx register.
2. Work out the width of the triangle using the formula $2h - 1$ where h is the height specified by the user, and place this value in a variable called height.
3. Start the outerLoop and push its value to the stack.
4. Calculate the number of spaces that need to be drawn using the formula $(width - noOfAsterisks)/2$. The noOfAsterisks is the current value of the innerLoop (which is always an odd number). And place that value in ecx to be used as the control value for the third loop.
5. When dividing the value of $(width - noOfAsterisks)$ by 2, if the value is zero(on the last line), then do not print any spaces.
6. Draw spaces ecx times.
7. Pop innerLoop value of off stack and start innerLoop.
8. Start a new line, add 2 to the value of the innerLoop (The number of stars in each line increases by two each time.
9. Pop the value of the outerLoop off of the stack and start the next iteration.

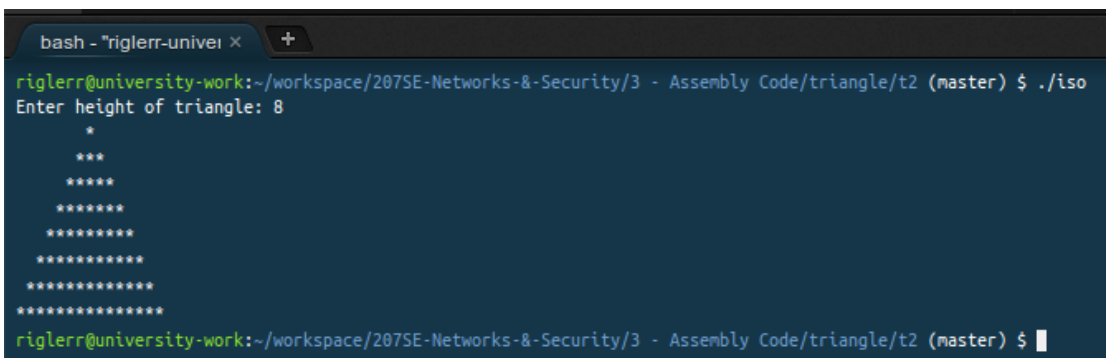
Listing 2: Isosceles Triangle Code

```
section .data

Promt db 'Enter height of triangle: '
pLen equ $-Promt
chr db '*'
nl:     db "   ", 0x0a ; varab;e to draw a new line
nl_len  equ $-nl ; length of new line varable
ns: db " " ; variable to draw a space
ns_l equ $-ns ; length of space variable

section .bss
num resb 2 ;reserve 2 bytes for the input variable
width resb 2

section .text
global _start
_start:

;Ask user for size of triangle
mov eax,4
mov ebx,1
mov ecx,Promt
mov edx,pLen
int 80h

;store the variable
```

```
mov eax,3
mov ebx,0
mov ecx,num
mov edx,2
int 80h

mov ecx, [num] ;derefference input and store in ecx
sub ecx,'0' ;convert from ascii to decimal
xor ch,ch ; clear upper half of ecx

mov ebx,1
mov eax,ecx

    ;this block places w=2n-1 into eax
    push ebx ;push inner loop to stack
    mov ebx,2 ; move 2 into ebx
    mul ebx ; multiply eax by 2
    sub eax,1 ; subtract 1 from eax
    pop ebx
    mov [width],eax ; place width value into width variable

lo: ;outer loop, amount of lines in triangle

    push ecx ; push outer loop count to stack

    ;This block works out the number of spaces to print before drawing
    ;noOfSpace = (width-noOfAsterisks)/2
    mov ecx,[width] ; make loop counter equal to the width of the triangle
    sub ecx,ebx ;subtract number of asterisks
    shr ecx,1 ; shift right, divides ecx by 2^1 (2)
   jz l2 ; jump to  the l2 label if the result of the division was 0

    l3:
    push ebx ; push ebx to stack, so to use ebx in starting a new line
    push ecx
    mov eax, 4
    mov ebx, 1
    mov ecx, ns
    mov edx, ns_l
    int 0x80 ;draws a new line
    pop ecx ;pop inner loop count off stack
    pop ebx

    loop l3

    l2:

    mov ecx,ebx ; place inner loop count in the loop counter
        li: ; inner loop, amount of stars in line

            push ecx
            push ebx ;push ecx and ebx to stack so they can be used in drawing
            mov eax,4
```

```
            mov ebx,1
            mov ecx,chr
            mov edx,1
            int 80h ;draw star
            pop ebx
            pop ecx ; pop ecx last, so it is has the correct loop counter value
        loop li ; end of inner loop

    push ebx ; push ebx to stack, so to use ebx in starting a new line
    mov eax, 4
    mov ebx, 1
    mov ecx, nl
    mov edx, nl_len
    int 0x80 ;draws a new line

    pop ebx ;pop inner loop count off stack
    ; add two to the inner loop counter,
    ;each line always has an odd number of asterisks
    add ebx,2


    pop ecx ;pop outler loop count off stack to use as counter for lo
loop lo ;end of outer loop
    int 80h;

mov eax,1
mov ebx,0
int 80h;exit
```

Evidence of triangle



Item 3 - Bootloader

This tasks was to create a bootloader using Assembly and boot into this bootloader using bochs. The process involved:

1. Writing and compiling the code.
2. Writing the code to the first 512 bytes of a bootable image.
3. use bochs to run the bootloader.

1. Boot pragma-linux using bochs

I created a Bash script to compile, create and simulate the bootloader.

Listing 3: Bash Script

```
#!/bin/bash


#User propt to enter file name.
echo "Enter name of asm file: "

read asm_name  #name stored in variable

echo "Compiling file."

#appends the filename with the .asm extension
asm_ext="$asm_name"".asm"

#Using nasm to compile the file
nasm $asm_ext

echo "Writing file to disk image."

#Writes the file to the image
dd if=$asm_name bs=512 of=a.img

#starts bochs
bochs
```

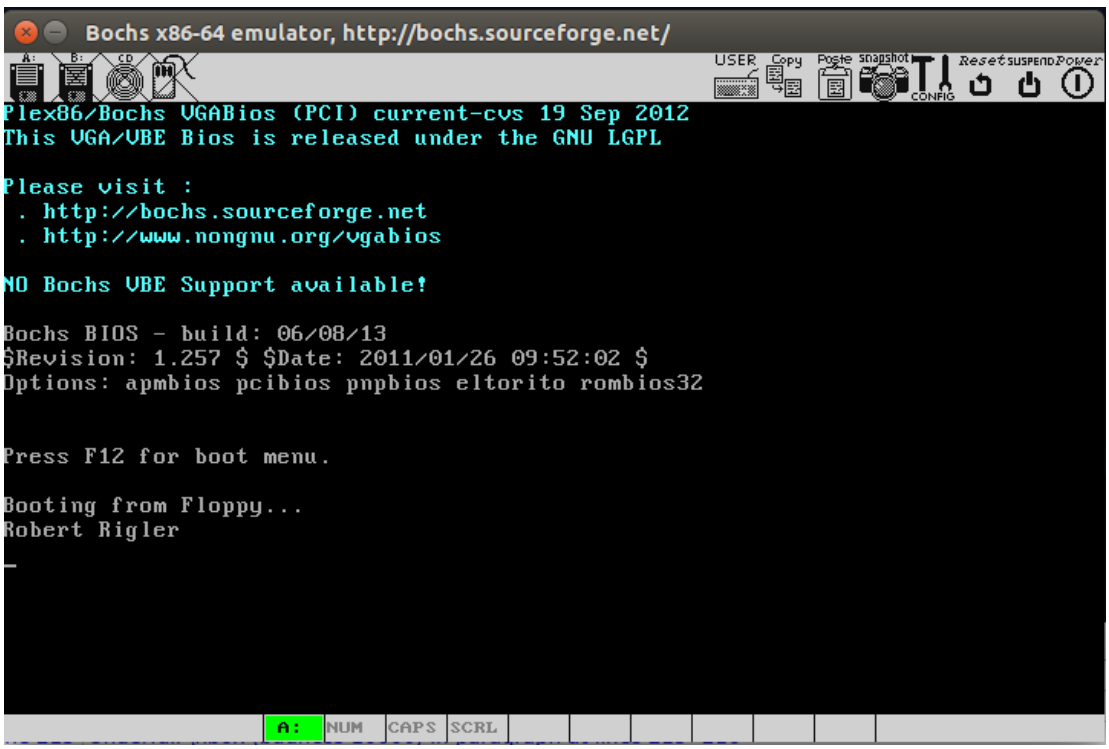
1. Make a Bootloader that displays my name

Listing 4: Bootloader Assembly code

```
[BITS 16]
[ORG 0X7C00]
top:
    mov ax,0x0000
    mov ds,ax
    mov si,HelloWorld
    call writeString
    jmp $

```

```
writeString:
    mov ah,0x0E ;Display Character
    mov bh,0x00
    mov bl,0x07

nextchar:
  lodsb
  cmp al,0
  jz done
  int 0x10
  jmp nextchar

done:
  ret
  HelloWorld db 'Robert Rigler',13,10,0
  times 510-($-$$) db 0

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

Command used : *cat /proc/cpuinfo*

```
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 lm 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_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

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

Command used : *cat /proc/interrupts*

```
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
t0
  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
cd:usb1, ehci_hcd:usb2
 18:          2          0          0          0   IO-APIC-fasteoi   firewi
re_ohci, mmc0
 23:          0          0          0          0   IO-APIC-edge      lis3lv
02d
 40:          0          0          0          0   PCI-MSI-edge      PCIe P
ME
 41:          0          0          0          0   PCI-MSI-edge      PCIe P
ME, pciehp
 42:          0          0          0          0   PCI-MSI-edge      PCIe P
ME
 43:          0          0          0          0   PCI-MSI-edge      PCIe P
ME
 44:          0          0          0          0   PCI-MSI-edge      xhci_h
cd
 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      iwlwif
i
 49:       4755      31559       2541       2296   PCI-MSI-edge      i915
 50:       1179         80         17         57   PCI-MSI-edge      snd_hd
a_intel
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
nterrupts
IWI:       2279       1883       1820       1826   IRQ work interrupts
RTR:          2          0          0          0   APIC ICR read retries
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.

Command used : *grep model /proc/cpuinfo*

```
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.

Command used : *cap /proc/cmdline*

```
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.

Command used : *awk* $'\{print\ \$3, \$4\}'$ */proc/diskstats* | *grep sda*

```
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 5: Bash Script

```bash
#!/bin/bash

 # DISPLAYS A MENU

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. "
echo "4. exit. "

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

echo "Your choice was $input_variable"

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

case "$input_variable" in

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

echo "Exiting"
```

Item 5 - Buffer tutorial

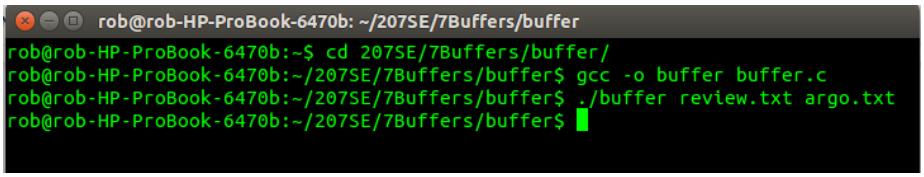
This task involved using buffers, specifically using buffers in term of reading an writing data from a file.

1. Commented version of the provided code

Listing 6: Commented Buffer Code

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

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

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

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

  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.
  if (out_fd < 0)
     exit(3);

  while (rd_size > 0) {

     rd_size = read(in_fd, buf, BUF_SIZE); // Continuously read from input file
                                            //into buffer.
     if (rd_size <0)
     exit(4);

     wr_size = write(out_fd, buf, rd_size); // Continuously write from buffer into
                                              //the output file.
     if (wr_size<=0){
        close(in_fd);                       //Close both of the files.
  close(out_fd);
exit(5);
}
  }
}
```

2. Evidence of compiled code



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

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

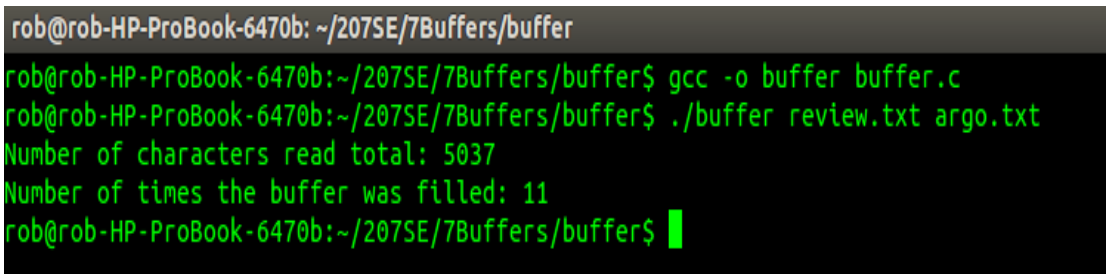
Listing 7: Adpated Code

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

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

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

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

  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.
  if (out_fd < 0)
     exit(3);

  while (rd_size > 0) {

     rd_size = read(in_fd, buf, BUF_SIZE);//Continuously read from input file
                                          //into buffer.
    rd_count+= rd_size;  //Adds rd_size to the total read count
      if(rd_size > 0)
      buf_count +=1;//Counts the number of times the buffer
                    //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.
     if (wr_size<=0){
        close(in_fd);                            //Close input file.
  close(out_fd);                                 //Close output file


    printf("Number of characters read total: %d\n",rd_count );
    //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);
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



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 8: Adapted code

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

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

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.
  char buf0[BUF_SIZE]; //Declare 2nd buffer.

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



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

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

  while (rd_size > 0) {

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

     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.

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

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

```

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

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 comaprison 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 9: cache_example.c

```c
#include "cache_reader.h"

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

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

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

  }


  //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);
//Displaying the total number of times the buffer was filled.
//(No_of_bytes / buffersize).
  cr_close(f);

  //And finish
  return 0;
}
```

Listing 10: cache_reader.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 usedbuffer;    //Current point in the buffer
  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.
} cr_file;


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


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

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



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

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

Listing 11: cache_reader.c

```c
#include "cache_reader.h"

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

}

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


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

  //Info on malloc
  //http://www.space.unibe.ch/comp_doc/c_manual/C/FUNCTIONS/malloc.html
  FILE* f;
  if ((f = fopen(filename, "r")) == NULL){
    fprintf(stderr, "Cannot open %s\n", filename);
    return 0;
  }

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

```c


//-----------------------------------------------------------------
char cr_read_byte(cr_file* f){

  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);

  }
    else{ // If buffer hasn't been fully used, return the chracter and increase
     // the usedBuffer position by 1.

       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.

    }

}
```

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
iwldvm                232285  0
mac80211              630653  1 iwldvm
joydev                 17381  0
serio_raw              13462  0
iwlwifi               169932  1 iwldvm
rfcomm                 69160  8
cfg80211              484040  3 iwlwifi,mac80211,iwldvm
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.

Item 8 - Kernel part II

This task involves changing the code in cache_reader.h and cache_reader.c to use the kernel system calls open(), read(), and close() instead of the c alternatives fopen(), fread(), and fclose().