CSCD 340
Lab 2

Your answers will be in the form of a PDF/C Code.

USING LOW LEVEL SYSTEM CALLS

1. Create a C file name **txtToBinary.c**
    a. Opens an input text file (using fopen) by prompting the user for the filename. If the file does not exist or can't be opened the user is reprompted. You may assume all files exist in the current working directory. Name your input file myTextInput.txt
    b. Opens an output file in binary mode (using the open system call) hard code the output filename to myBinary.bin. If the file can't be opened the program will display an error message and exit. You may assume all files exist in the current working directory. NOTE O_CREAT|O_WRONLY
    c. Reads integers from the input file (using fscanf) and writes the integers in binary, to the output file (using the write system call). This will continue until the end of file. The program should exit if any write cannot be performed.

2. Create a C file name **binaryToTxt.c**
    a. Open the binary file myBinary.bin (using the open system call) whose name is hardcoded. If the file cannot be opened display an error message and exit.
    b. Write a function called **convertToText** that uses the read system call, to read from the binary file, and fprintf to write the original numbers to the screen. This will continue until end of file is reached. The program should exit if any read cannot be performed. Don't forget to add a space or carriage return
    c. Write a function called **convert** that uses the read system call, to read from the binary file, and the write system call to write to the screen. This will continue until end of file is reached. The program should exit if any read or write cannot be performed. You will need to reset the file descriptor using the lseek system call at the top of this function.

UNDERSTANDING USER TO KERNEL MODE

3. In class we have been discussing simple system calls in particular as system calls related to "The Universality of I/O". Create a very simple C program named syscalls.c. This program will contain only a main function. This simple program will open a File named test.txt (hard coded file name) it will read one line at a time, print that line to the screen, and then close the file.

   Compile and execute your code one time to ensure it works properly. Use the following text in "test.txt"

   Now is the time for all good
   men to come to the aid of their
   university

   Now execute the command **strace -C -o syscallout.txt ./a.out**. Open the syscallout.txt file and annotate the system calls and what is going on. You don't need to do every line mainly focus on Open, Write, Read and Close if they exist. Note: you are expected to try and explain what is going on not just say "look here is the system call". Pay particular attention to the first few lines in syscallout.txt.

   Name your output your last name, first letter of your first name, syscalls (Example steinerssyscalls.pdf)

4. Create a very simple java program named SysCalls.java. This program will contain only a main method and it will use a Scanner object. Just propagate the Exception by placing throws Exception on the main method header. This simple program will open a File named test.txt (hard coded file name) it will read one line at a time, print that line to the screen, and then close the file.

   Compile and execute your code one time to ensure it works properly. Use the following text in "test.txt"

   Now is the time for all good
   men to come to the aid of their
   university

   Now execute the command **strace -C -o syscalloutjava.txt java SysCalls**. You don't need to comment any of the open, close, read, write for this portion of the lab; however, at the very top of syscalloutjava.txt you must write at least a 3 sentence comparison concerning the syscalls from the Java world compared to the syscalls of the C world. Explain what is java doing that C isn't or vice versa. You must write an additional set of statements explaining why in the Java version there are so many calls to open.

   Name your output your last name, first letter of your first name, syscallsjava (Example steinerssyscallsjava.pdf)

UNDERSTANDING INTERRUPTS

5. Below is the code that handles the signal for SIGFPE and prints "Divide by zero not allowed"

```
int main()
{

    int x;
    signal(SIGFPE,f);
    for(x = -3; x < 3; x++)
            printf("val is: %d\n", 12/x);

    return 0;

}//end main

void f(int signum)
{

    printf("Divide by zero not allowed \n");
    exit(SIGFPE);

}// end f
```

Below is a very simple C program you will named cscd340_lab2sigs.c. You must add the signal handling code for CTRL-C in the printf you will state "Sorry can't CTRL-C". NOTE: you will need to look at man 7 signal for the appropriate signal to override. Do not include an exit in your signal handler.

```
int main()
{
    int x;

    for(x = 0; x < 15; x++)
    {
            printf("val is: %d\n", x);
            sleep(1);
    }// end for

    return 0;
}//end main
```

Compile the code and execute the command **strace –C -f –o sigcallout.txt ./a.out**. As part of the execution try CTRL-C a couple of times. Open the sigcallout.txt file and look at what is going on. You don't need to annotate anything; however, you need to write a 4 sentence comparison between what is going on here as compared to what happen in #3. You must provide an analysis (compare and contrast).

You must also write at least 4 sentence explanation discussing the rt_sigprocmask and the rt_sigaction commands. Note: you are expected to try and explain what is going on.

Name your output your last name, first letter of your first name, sigcalls (Example steinerssigcalls.pdf)

TO TURN IN

A zip containing:

- All PDF files
- Your input files
- Your C files
- Your Java file(s)

You will submit a zip file named your last name first letter of your first name lab2.zip (Example steinerslab2.zip)