**CSci 5103 (Spring 2016)**
**Assignment 1**
**Part A (100 points)**
**Due February 1, 2016**

*This part of the assignment must be done **individually**.*

**Problem 1 (10 points):** Which of the following instructions should **NOT** be allowed in user mode?
a) Disable all interrupts.
b) Read the time-of-the-day clock.
c) Set the time-of-the day clock.
d) Change the base-bound register values.
e) Set a timer.
f) Turn off timer interrupt.
g) Read from kernel memory.
h) Write to kernel memory.
i) Fetch an instruction from kernel memory.
j) Switch from user to monitor mode.

**Problem 2 (20 points):** What are the key differences between traps, system calls, exceptions, and interrupts?

**Problem 3 (21 points):** For each of the system calls, identify at-least 3 conditions that cause it to fail: *fork*, *open*, *exec*, *waitpid*, *read, write*, *unlink,*

**Problem 4 (30 points):** A file whose file descriptor is fd contains the following sequence of bytes:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f (length = 17 including terminating newline char)
The program makes the following sequence of system calls (assume all variables have been initialized properly):

```
1  lseek (fd, 3, SEEK_SET);
2  read (fd, &buffer1, 4);
3  fprintf (stderr, "%s", buffer1);
4  lseek (fd, 3, SEEK_CUR);
5  read (fd, &buffer2, 4);
6  write (2, buffer2, 4);
7  lseek (fd, 10, SEEK_END);
8  write(fd, buffer1, strlen(buffer1));
```

(Please note: version posted earlier had a typo on line 8, it contained strlen(a) instead of strlen(buffer1)

a. What is the output after executing lines 3 and 6?
b. What is the difference between *fprintf()* and *write()*? What does the operating system do for each function?
c. List the contents of the file after line 8 has finished execution. What is the length of the file?

**Problem 5 (10 points):** What are the advantages and disadvantages of implementing threads in user space?

**Problem 6 (9 points)**: What is the difference between hyperthreading and multi-core CPUs?

**CSci 5103 (Spring 2016)**
**Assignment 1**
**Part B (100 points)**
**Due February 1, 2016**

*This part of the assignment must be done **individually**.*

**Objective:**

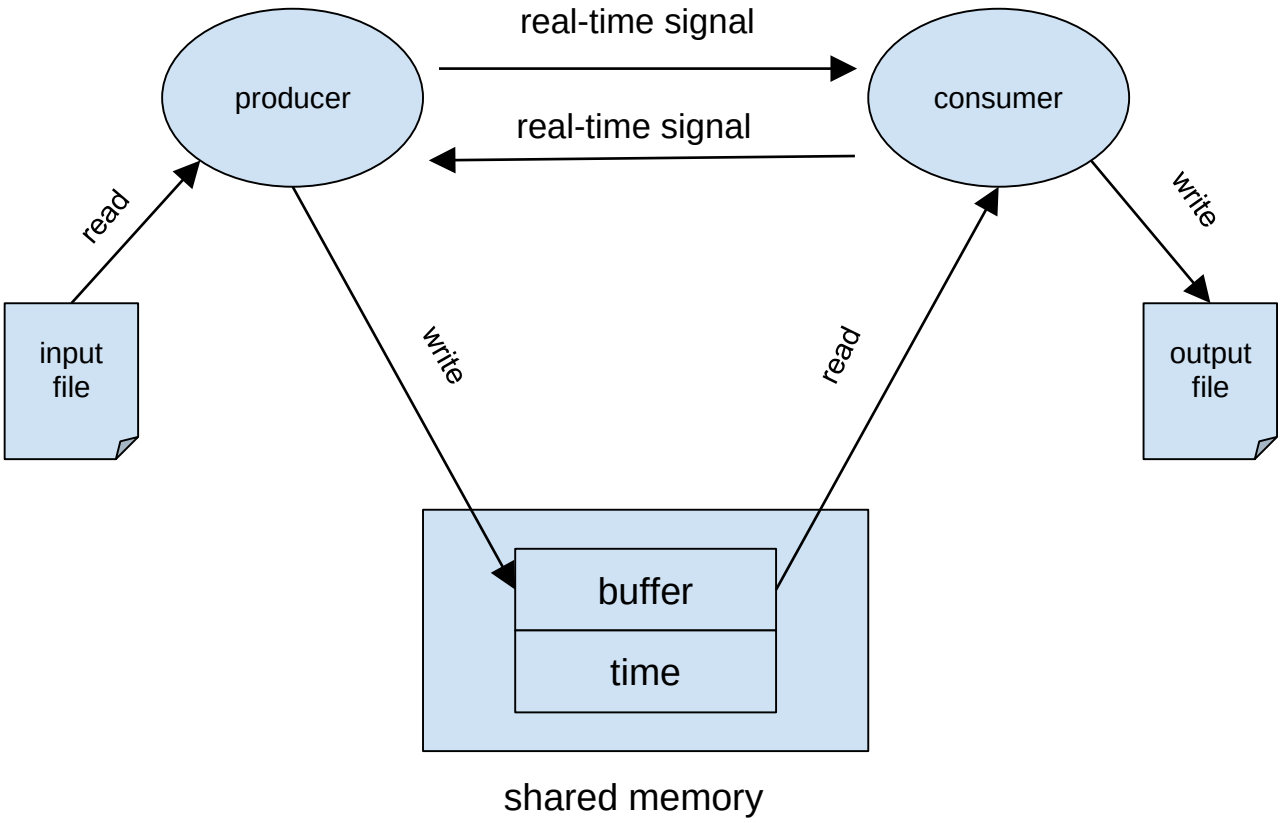
The objective of this assignment is to understand the following topics:

- Process creation and management on a UNIX system using system functions such as *fork(), exec(), waitpid().*
- Shared memory operations using *shmget(), shmat(), shmdt()*
- Signal handling using *sigemptyset(), sigaddset(), sigprocmask(), sigwaitinfo(), sigqueue()* and other related functions
- Timing operations

**Problem statement:**

In this assignment you will create a region of shared memory between two processes to act as a medium for data transfer. These two processes will communicate in the producer-consumer mode. The producer process will read a file, transfer its contents to the consumer through the shared memory, and the consumer process will store the received data in a new file called **output**. Your program also needs to measure and report the total time taken for data transfer. Your programs must use POSIX real-time signals for communication and synchronization.

The shared memory region must contain the following:
**buffer**: A character array of size **1024 bytes**, used to store the data to be transferred.
**time**: Timestamp indicating the time of insertion



The producer process must read data from a file and write it into the buffer in the shared memory region. It may be necessary to repeat this step multiple times as the file may be much larger than the size of buffer. The producer process must also obtain the current time and store it in the shared memory region. The producer process must then send a real-time signal to the consumer process to indicate that the buffer is full and ready for consumption. The producer process must also send the size of available data in shared memory to the consumer process along with the signal. The producer process must then wait to receive a real-time signal from the consumer process before proceeding. After the entire file has been transferred, the producer process must send a real-time signal to the consumer process along with an accompanying integer value of -1 to indicate completion of data transfer. The producer process must then wait for the consumer process to terminate, report its exit status and then exit gracefully.

The consumer process is responsible for storing the received data in a file called '**output'.** The consumer process must wait to receive a real-time signal from the producer process. The reception of the signal indicates that either data is available for consumption or that data transfer is complete. If the consumer process receives an integer whose value is non-negative, it indicates that the buffer has valid data for consumption. It must read those many bytes of data from the buffer and write it to the output file. It must also measure the total time the buffer contains valid data. It may do so by aggregating the time difference between the current time and the time stored in the buffer. The consumer process must then send a real-time signal to the producer process to indicate that the buffer is empty. If the accompanying integer value is -1, it indicates end of data transfer. At this point, the consumer process must report the total time taken and exit gracefully.

Write a C program named prodcon.c which contains code for both the producer and the consumer processes. When executed, the program must create a shared memory segment and fork a child process. The parent process should act as a producer and the child process should act as the consumer. The input file would be given as a command line argument to the program. Provide a makefile which can compile your program to produce an executable called **prodcon**.

**Usage:**

```
$ prodcon <input_filename>
```

**Recipe:**

The general structure of the producer would be as follows:

```
...
Open the file specified by argv[1] for reading
…
while (not end of file) {
      read a chunk of data into memory
      read the current time
      write the chunk into the shared memory
      write the timestamp into the shared memory
      send a real-time signal to the consumer
      suspend execution until a real-time signal is received
}
// data transfer complete
send a real-time signal to the consumer with an accompanying integer value of -1
wait for the consumer process to terminate
...
exit
```

```
End of producer
```

The general structure of the consumer would be as follows:

```
…
Open the file “output” for writing
…
while(...) {
      suspend execution until a real-time signal is received
      store the integer value sent along with the real-time signal in a variable
            called ‘size’
      if ‘size’ is -1: Close the output file, report the aggregate time taken
            for download and exit gracefully
      read ‘size’ bytes from the shared memory region
      read the current time (A)
      read the timestamp from the shared memory region (B)
      compute the time difference between (A) & (B) and aggregate
      send a real-time signal to the producer process with an accompanying
            integer value of 0
}
End of consumer
```

**Grading criteria:**
Process creation and management - fork/wait/waitpid: **20%**
Shared memory operations: **20%**
Complete data transfer: **25%**
Timing computation and reporting: **10%**
Correct use of real-time signals: **20%**
Comments/Documentation: **5%**

**Note:**
Please ensure that your program can run on the CSELabs UNIX machines.
You will lose significant points for not following instructions, failing to submit a makefile or a readme file.
You will also lose significant points if the output file does not match the input file (Hint: Use the ‘diff’ command to compare your input and output files)

**Submission Instructions:**
Submit a single archive file (.zip/.tar/.rar) containing

- A PDF document for your answers to Part A
- The source files, header files and a Makefile for Part B.
- A README file containing your name, your x500 and the CSELabs machine on which you tested your code.