CSE 451/851 Assignment #1

Assigned: Sept 22, 2023 Due: Oct 6, 2023 11:59 pm

Notes:

- There are **Six** questions in this homework.
- Submit your response to Canvas.
- Latex or Word-based submissions are required for written questions. **Scanned handwritten submissions will not accepted.**
- Please answer each question in detail.

1. Consider the following code:

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

int main()
{
    int i;
    int n = 60;
    for(i=0; i<n; i++){
      fork();}
    printf("hello\n");
    return 0;
}
```

a. **[5 points]** In theory, how many times does the program print hello?
b. **[5 points]** Why might it print a different number of times?
c. **[5 points]** Now, consider the output from the code above, where n=4:

```
[prompt]$ ./a.out
hello
hello
hello
hello
[prompt]$ hello
hello
hello
hello
hello
hello
hello
hello
hello
```

```
hello
hello
hello
```

What is happening in this output? Why is the prompt printed before the other hellos?

d. **[5 points]** Show how you would fix this code to have the expected output where all the hellos come before the prompt. You are only allowed to add some lines. Don't delete or modify any lines.

2. **[10 points] a.** Read the following program and write down the output. We assume the program is run on a computer with a single CPU; when there are multiple processes or threads ready to run, they could run on the CPU in any arbitrary order. So, if there are multiple possibilities in the outputs, provided all; note that, the same information but outputted in different orders are considered as different possibilities.

[5 points] b. How many unique processes are created (including the parent process)?

```
int a;
int main()
{
    int c;
    a=1;
    c=2;
    if(fork() == 0){
        a=10;
        c=20;
        if(fork() == 0){
            printf("a=%d, c=%d\n",a,c);
            c=30;
            exit(0);
        }else{
            wait(NULL);
            a = 40;
            printf("a=%d, c=%d\n",a,c);
            exit(0);
        }
    }
    printf("a=%d, c=%d\n",a,c);
}
```

3. **[10 points]** The following state transition table is a simplified model of process management, with the labels representing transitions between states of READY, RUN, WAITING, and SUSPENDED.

	READY	RUN	WAITING	SUSPENDED
READY	-	1	-	2
RUN	3	-	4	5
WAITING	6	-	-	7
SUSPENDED	8	-		-

Give an example of an event that can cause each of the above transitions. Draw a process state diagram if that helps.

4. a. **[2 points]** What do multiprogramming and time-sharing have in common?
   b. **[4 points]** How are they different?
   c. **[4 points]** Given the two scenarios below, complete the following table

Scenario	Choose Multiprogramming or Time-Sharing	Justify
Multi-user system like the CSE server		
Batch Processing System (does not interface with users)		

5. **[40 points]** Processes can communicate either using shared memory or message passing. In this question you will use POSIX shared memory for a sorting problem. Implement C programs that follow the following instructions:

- **First program** will create a shared memory of size 2048. It will generate 10 random numbers ranging from 1-1000 and write those numbers to the shared memory.
- **Second program** will read numbers from the shared memory sort them in an ascending order and print them.
- **Third program** will read numbers from the shared memory sort them in a descending order, print them and remove the shared memory.
- Create makefile to compile the three programs.
- Submit the three programs (.c files), a read me files and the makefile.

6. **[30 points]** Programing question.

A Unix pipe is a one-way communication channel. A pipe is used to pass a character stream from one process to another. Some of the commonly used examples of pipes are cat <file name> | more, who | wc -l. The pipe(p) (p is declared as int p[2];) command opens the pipe. The command returns two file descriptors p[0] (the read end) and p[1] (the write end). These file descriptors are common to all processes. Therefore, there is a limit to how many pipes can be there at any one time.

Write a program that follows such steps:

A parent process creates two child processes,
child process 1 gets the current date printed, in **MM/DD/YYYY** format, to the terminal then writes it into a pipe,
child process 2 reads that string from the pipe,
child process 2:
converts the date format into **DD/MM/YY**, prints it to the terminal, then it writes the date into another pipe,

Parent process reads the date from the pipe and prints it to the terminal using the format **MM/YY**.

Make sure that the processes are terminated such that **no orphan processes** might be created.

The program should run successfully on **cse-linux** server. Submit the c file to canvas.