CSCE 313 # QUIZ 3 Spring 2021

Total Points 100, Due Monday 3/29/21 11:59 pm

Q1 [20 pts]: Schedule the following workload in a single CPU-single core system and then compute Average Response Time (ART) under preemptive Shortest Remaining Time First (SRTF).

Process	Arrival Time	Service Duration
P1	1	7
P2	2	3
P3	2	5
P4	3	1
P5	7	2

Q2 [15 pts]: Consider the program below and answer the following questions with proper explanation.

A. What is the output? How much time does the program take to run?[10 points]
B. What is the output with line 5 commented? How much time will it take now? [10 points]

```
1 void signal_handler (int signo){
2   printf ("Got SIGUSR1\n");
3 }
4 int main (){
5   signal (SIGUSR1, signal_handler);//comment out fr b)
6   int pid = fork ();
7   if (pid == 0){// child process
8       for (int i=0; i<5; i++){
9          kill(getppid(), SIGUSR1);
10         sleep (1);
11      }
12  }else{ // parent process
13      wait(0);
14  }
15}
```

Q3 [25 pts]: Assume the following processes A, B, C are loaded in memory of a system that uses multiprogramming(MP). These processes have 13, 7, and 11 instructions respectively. Also assume that the dispatcher lives at address 100 in memory and spans 4 instructions (i.e., 100-103). The following table shows only instruction addresses in the memory with I/O requests labelled, along with the duration of these I/O operations in terms of CPU instructions. Although I/O operations do not take CPU instructions, the duration means that the I/O operations will finish by the time the corresponding number of CPU instructions execute. Please draw a trace of these 3 processes running together in the CPU using Shortest Remaining Time First (SRTF) with preemption and no timer (i.e., we are not running a timer that is used by time sharing or round-robin). Recall that for SRTF, you only need to decide based on the next CPU burst, not the entire

remaining time of the process. You can skip the first invocation of the dispatcher to decide the first process to run in the CPU. You can assume that all I/O operations are for different devices and hence they are independent, i.e., they can proceed simultaneously.

Process A	Process B	Process C
5000	8000	12000
5001	8001	12001(I/O, takes 3 ins.)
5002	8002	12002
5003	8003 (I/O, takes **7 ins.**)	12003
5004 (I/O, takes **6 ins**.)	8004	12004
5005 (I/O, takes **4 ins**.)	8005	12005
5006	8006	12006(I/O, takes 1 ins.)
5007		12007
5008(I/O, takes **5 ins.**)		12008(I/O, takes 2 ins.)
5009		12009
5010		12010
5011		
5012		

Q4 [20 pts]: Do you think the following program can possibly produce the output: **"2 children are done"**. If No, why not? If Yes, why? Please explain your answer.

```
int num_children_done = 0;
void my_handler (int sig){
     cout<<++num_children_done<<" children done"<<endl;
}
int main (){
    // install handler
    signal (SIGCHLD, my_handler);
    // create 5 child procs,
    for (int i=1; i<=5; i++){
        int pid = fork ();
        if (pid == 0){
            sleep (5);//all die simultaneously
            return 0;
        }
    }
    while (true)
       sleep (1);
}
```

Q5 [20 pts]: The following program counts the number of primes in a range of numbers accurately given enough time. Now, since this program takes a long time, the user may get impatient and press Ctrl+C to kill it. The default behavior for that is immediate termination of the program without any output. Your task is to change this program by giving it a "best-effort" ability, meaning that it will print out the count of however many

prime numbers it could find before receiving the SIGINT signal generated by the Ctrl+C. The count will obviously be inaccurate; however, it is the best given the limited time. In summary, you will write a signal handler for SIGINT that will: a) print the partial count and b) terminate the program after that. You are not allowed to use multiple threads, or other processes.

```
int num_primes = 0; // holds # of primes
void count_primes (int start, int end){
      // check each number in range [start, end] for primality
      for (int num = start; num <= end; ++num) {
            int i;
            for (i = 2; (i <= num) && (num % i != 0); ++i)
                  ;
            if (i == num)  // if no divisor found, it is a prime
                  ++num_primes;
      }
}
int main (int ac, char** av){
      count_primes (1, 1000000); // count all primes <= 1 million
      cout <<"Found "<<num_primes<<" prime numbers in the range"<<endl;
}
```