Practice Problems #1
Processes, Threads and Scheduling

These problems are meant to help you prepare for exams. You do not need to turn them in. There will be no solutions posted, but you are welcome to check your solutions with us or with your colleagues.

1. Most round-robin schedulers use a fixed size quantum. Give an argument in favor of a small quantum. Now give an argument in favor of a large quantum. Compare and contrast the types of systems and jobs to which the arguments apply. Are there any for which both are reasonable?

2. In a system with threads, is there normally one stack per thread or one stack per process? Explain.

3. What is the fundamental difference between a process and a thread?

4. Round-robin schedulers normally maintain a list of all runnable processes, with each process occurring exactly once in the list. What would happen if a process occurred twice in the list? Can you think of any reason for allowing this?

5. Assume a system with priority scheduling in which user processes run with the lowest priority and system processes run with the highest priority. Lowest priority has round-robin scheduling. Is it possible for processes in the lowest class to starve? Explain your answer.

6. Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time? (Your answer will depend on X).

7. Assume that 5 processes arrive at the ready queue at the times shown below. The estimated next burst times are also shown. Assume that an interrupt occurs at every arrival time.
   a. What is the waiting time for each process if **preemptive "shortest remaining time first"** scheduling is used?
   b. What is the waiting time for each process if **non-preemptive "shortest job first"** scheduling is used?

Process	Arrival Time	Burst Time
P1	**0**	**7**
P2	**1**	**1**
P3	**3**	**2**
P4	**4**	**4**
P5	**5**	**3**

8. The following program contains no syntax errors. As it executes, it will create one or more processes. Show how processes are created and what is printed to the screen when this program executes.

```
main(){
int pid1, pid2, pid3;
int counter;

pid1 = 10; pid2 = 20; pid3 = 30; counter = 1;

pid1 = fork();

printf("%d, %d, %d, %d \n", pid1, pid2, pid3, counter);

pid2 = fork();

printf("%d, %d, %d, %d \n", pid1, pid2, pid3, counter);
while (counter < 2)
     {
     pid3 = fork();
     printf("%d, %d, %d, %d \n", pid1, pid2, pid3,
counter);
     counter = counter +1;
} }
```

9. The following program contains no syntax errors. As it executes, it will create one or more processes. Show how processes are created and what is printed to the screen when this program executes.

```
#include <stdio.h>
main(){
int a, x, y, n;
a=-5;
n=1;
x=-15;
y=-20;
a=fork();
if(a==0)
     y=fork();

while (n<3){
     if(y==0)
          x=fork();
     n++;
     }
printf("x=%d y=%d a=%d\n", x, y, a);}
```

10. The following program will create some processes as it executes. Call this executable program “parent”. Draw a hierarchical chart describing which processes this parent process creates and which processes its children create. Next to each process name include the values of variables pid1, pid2, pid3, and counter right after each process is created. For “parent” include the values right before it creates its first child. You can assume that the nonzero values returned by fork() are as follows: 50, 150, 250, 350, 450, 550, etc.

```
#include <sys/types.h>
#include <unistd.h>
main()
{
     int pid1, pid2, pid3;
     int counter;
     pid1 = 10; pid2 = 20; pid3 = 30; counter = 1;
     pid1 = fork();
     pid2 = fork();
     while (counter < 2)
     {
          pid3 = fork();
          counter = counter +1;
     }
}
```

11. The following program contains no syntax errors. As it executes it will create one or more processes. As a result, one or more statements will be printed to the screen. Simulate the execution of this program and answer the questions below.
    a. Show how processes are created to the right of the source code.
    b. What is the total number of processes created including the parent process?
    c. What will this program print on the screen when it executes?

```
#include <stdio.h>
main(){
   int x, y, count;
   x = 20; y = 30; count = 1;
   while (count < 3){
         if ( x != 0){
               x = fork();
               y = y + 20;}
               }
         else {
               x = fork();
               y = y + 35;}
   printf("\n y = %d", y);
   count = count + 1;}}
```

12. Assume that 4 processes arrive at the ready queue at the times shown below. The estimated next burst times are also shown. Assume that an interrupt occurs at every arrival time. Show below how you would assign the CPU to these processes to minimize the average waiting time. If needed, consider the time slice is 9.

Process	Arrival Time	Burst Time
P1	**0**	**2**
P2	**1**	**2**
P3	**3**	**5**
P4	**4**	**3**

13. Assume that 5 processes arrive at the ready queue at the times shown below. The estimated next burst times are also shown. Assume that an interrupt occurs at every arrival time and that the time slice is 10. Preemptive "shortest remaining time first" scheduling is used to assign the CPU to these processes.
    a. Draw a graph showing how the CPU will be assigned to these processes to satisfy their burst times and for how long.
    b. Determine the **waiting time** for each process for their entire time of execution.

Process	Arrival Time	Burst Time
P1	**0**	**8**
P2	**1**	**2**
P3	**5**	**4**
P4	**6**	**2**
P5	**9**	**1**

14. Individual CPU and I/O requirements for 3 processes are shown below using a generic unit of time. If these processes execute in a time sharing environment where the CPU scheduling algorithm used by the OS is Round Robin with a time slice of 3 time units, you are to describe below how the CPU will be assigned to each process and for how long until all 3 processes have finished execution. You can assume that P1 gets to the ready queue just before P2, and P2 just before P3. You can assume that interrupts will be generated when a process needs I/O, when an I/O is finished, and when the time slice expires. Also assume that the time slice is reset with every interrupt and that the I/O for the processes are different so that there is no I/O queue. Assume also that an interrupt from a completed I/O for process "X" will place process "X" in the ready queue BEHIND the process that was just interrupted.