COMP 3500 Introduction to Operating Systems
Homework 2 – Processes, Threads, Scheduling

Maximum Points Possible: 100
Individual Assignment

There should be no collaboration among students. A student shouldn't share any project code with any other student. Collaborations among students in any form will be treated as a serious violation of the University's academic integrity code.

Question 1:

Including the initial parent process, how many processes are created by the program shown below.

```
#include <stdio.h>
#include <unistd.h>
int main()
{
int i;
for (i = 0; i < 4; i++)
fork();
return 0;
}
```

Question 2:

Using the program shown below, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid t pid, pid1;
/* fork a child process */
pid = fork();
if (pid < 0) { /* error occurred */
fprintf(stderr, "Fork Failed");
return 1;
}
else if (pid == 0) { /* child process */
pid1 = getpid();
printf("child: pid = %d",pid); /* A */
```

```
printf("child: pid1 = %d",pid1); /* B */
}
else { /* parent process */
pid1 = getpid();
printf("parent: pid = %d",pid); /* C */
printf("parent: pid1 = %d",pid1); /* D */
wait(NULL);
}
return 0;
}
```

Question 3:

Using the program shown below, explain what the output will be at lines X and Y.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
int i;
pid t pid;
pid = fork();
if (pid == 0) {
for (i = 0; i < SIZE; i++) {
nums[i] *= -i;
printf("CHILD: %d ",nums[i]); /* LINE X */
}
}
else if (pid > 0) {
wait(NULL);
for (i = 0; i < SIZE; i++)
printf("PARENT: %d ",nums[i]); /* LINE Y */
}
return 0;
}
```

Question 4:

Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

Question 5:

A variation of the round-robin scheduler is the regressive round-robin scheduler. This scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 50 milliseconds. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 10 milliseconds is added to its time quantum, and its priority level is boosted. (The time quantum for a process can be increased to a maximum of 100 milliseconds.) When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same. What type of process (CPU-bound or I/O-bound) does the regressive round-robin scheduler favor? Explain.

Question 6:

The following processes are being scheduled using a preemptive, roundrobin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an ***idle task*** (which consumes no CPU resources and is identified as P_{idle}). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.
b. What is the turnaround time for each process?
c. What is the waiting time for each process?
d. What is the CPU utilization rate?

Thread Priority Burst Arrival
P_1 40 20 0
P_2 30 25 25
P_3 30 25 30
P_4 35 15 60
P_5 5 10 100
P_6 10 10 105

Question 7:

Consider the following code segment:

a. How many unique processes are created?
b. How many unique threads are created?

```
pid t pid;
pid = fork();
if (pid == 0) {/* child process */
fork();
thread create( . . .);
}
fork();
```

Question 8:

The program shown below uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

```
#include <pthread.h>
#include <stdio.h>
int value = 0;
void *runner(void *param); /* the thread */
int main(int argc, char *argv[])
{
        pid t pid;
        pthread t tid;
        pthread attr t attr;
        pid = fork();
        if (pid == 0) { /* child process */
                pthread attr init(&attr);
                pthread create(&tid,&attr,runner,NULL);
                pthread join(tid,NULL);
                printf("CHILD: value = %d",value); /* LINE C */
        }
        else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PARENT: value = %d",value); /* LINE P */
        }
}
void *runner(void *param) {
        value = 5;
        pthread exit(0);
}
```