3 problems. 45 points total. Closed book, closed notes, no electronic devices

1. [15 points] This question concerns the projects.

a. What are refcounts used for?

b. In project 2 what is the purpose of the trampoline function?

c. Why do we need WaitNoPID if there is already a Wait function?

2. [15 points] Jobs W, X, Y, Z have the following arrival times and service durations (in seconds):

- W: arrival time 0; service duration 5. (So if no other job arrives, W leaves at time 5.)
- X: arrival time 3; service duration 6.
- Y: arrival time 5.; service duration 6.
- Z: arrival time 9; service duration 4.

a. Assuming fifo scheduling, obtain the departure time and response time of each job. (The response time of a job is the time it stays in the system.)

b. Repeat part a assuming fifo queueing with round-robin scheduling using quantum of 2 seconds.

3. [15 points]

Here is a skeleton of a program that starts threads t_1, …, t_n executing functions F_1, …, F_N. Each part below states a synchronization constraint. Fill in **W, X_i, Y_i, Z_i** to satisfy the constraint. The only synchronization construct you can use are semaphores. No busy waiting. Elegance and brevity count. The solution to part a is given below to illustrate.

```
// global variables; initialization
W     // you supply this
spawn thread t1 executing F1 ;
spawn thread t2 executing F2 ;
  ....
spawn thread tN executing FN ;
```

```
Fi
  Xi;       // you supply this
  while true {
     Ai;
     Yi;     // you supply this
     Bi;
     Zi;     // you supply this
}
```

a. At any time at most one thread is in any B_i.
   Solution: **W:** Semaphore s = 1; **X_i:** <nothing>; **Y_i:** P(s); **Z_i:** V(s);

b. At any time at most 4 threads are in any B_i.

c. Assume there are only two threads, t_1 and t_2. Assume that B_1 and B_2 are atomically executed by the hardware. Ensure that the executions of B_1 and B_2 alternate, starting with B_1. That is, in any evolution of the program, the subsequence of executions of B_1 and B_2 has the form B_1, B_2, B_1, B_2, ….

d. Repeat part c but now allow B_1 and B_2 to be code chunks that are not atomically executed by the hardware. Ensure also that there is no overlap in the executions of B_1 and B_2.