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?

Solution [5 points]
To determine when a thread’s memory can be reaped.
3 points for saying what refcounts are (i.e., the number of interested threads) but not what they are used for.

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

Solution [5 points]
To initiate restoring of a user thread’s stack at the end of a user signal handler so that the user resumes execution from the point where it was previously switched out.

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

Solution [5 points]
To kill dead child processes with non-zero refcounts without knowing their pids.
(If the parent knows the pid, it can use Wait (without blocking).)
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.)

**Solution [6 points]**

- W: arrival 0; service 5; departure 5; response 5
- X: arrival 3; service 6; departure 11; response 8
- Y: arrival 5; service 6; departure 17; response 12
- Z: arrival 9; service 4; departure 21; response 12

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

**Solution [9 points]**

Below solution assumes Z joins behind Y at 9. It’s also possible for Y to rejoin behind Z.

---

**Interval** | **Fifo queue at end of interval (job being served at left)**
---|---
0 to 3 | W 2
3+ | W 2, X 6
4+ | X 6, W 1
5+ | X 5, W 1, Y 6
6+ | W 1, Y 6, X 4
7+ | Y 6, X 4 | W departs
9+ | X 4, Y 4, Z 4 (assuming Z joins behind Y) | W departs
11+ | Y 4, Z 4, X 2 | W departs
13+ | Z 4, X 2, Y 2 | W departs
15+ | X 2, Y 2, Z 2 | W departs
17+ | Y 2, Z 2 | W departs
19+ | Z 2 | W departs
21+ | Z 2 | W departs
3. [15 points]
Here is a skeleton of a program that starts threads $t_1, \ldots, t_n$ executing functions $F_1, \ldots, F_n$. 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.

```plaintext
// global variables; initialization
W   // you supply this
    spawn thread $t_1$ executing $F_1$
    spawn thread $t_2$ executing $F_2$
    ....
    spawn thread $t_n$ executing $F_n$

F_i X_i;       // you supply this
F_i Y_i; // you supply this
F_i Z_i;       // you supply this
while true {
    A_i;
    Y_i; // you supply this
    B_i;
    Z_i; // you supply this
}
```

a. At any time at most one thread is in any $B_i$.

$W$: 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 $B_i$.

Solution [5 points]

$W$: Semaphore $s = 4$; $X_i$: <nothing>; $Y_i$: $P(s)$; $Z_i$: $V(s)$;

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 $B_1, B_2, B_1, B_2, \ldots$.

Solution [5 points]

$W$: Semaphore $s1 = 1$; Semaphore $s2 = 0$;

$X_1$: <nothing> $X_2$: <nothing>

$Y_1$: $P(s1)$; $Y_2$: $P(s2)$;

$Z_1$: $V(s2)$; $Z_2$: $V(s1)$;

Other than semaphores, the only atomicity you can assume is atomic reads and writes of integers; e.g., cannot assume that $x++$ is atomic.

Cannot have $t_1$ or $t_2$ skip an execution of $B_1$ or $B_2$; e.g., cannot have $t_1$ execute $A_1$, $A_1$, $B_1$, $A_1$, $\ldots$;

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 $B_2$.

Solution [5 points]

Part b solution also works here.