Homework #2 Solutions

1. Suppose that the following processes arrive for execution at the times indicated. Each process will run with a single burst of CPU activity (i.e., no I/O) which lasts for the listed amount of time.

process	arrival time	CPU burst time	priority
p1	0ms	25ms	3
p2	1ms	9ms	1
p3	20ms	14ms	4
p4	32ms	4ms	2

a. What is the job throughput, average waiting time and average turnaround time for these processes with non-preemptive, FCFS scheduling?

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**					
P2		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**
P3																					R	R	R	R	R	R	R	R	R	R
P4																														

	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
P1																							
P2	**X**	**X**	**X**	**X**																			
P3	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**					
P4			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	

Job throughput: 4 jobs in 52 time units, or **(4/52) jobs per time unit**

Avg waiting time: if wait time = time on ready queue (textbook definition), then this is (0 + 24 + 14 + 16)/4 = **54/4 time units**

if wait time == time on wait queue (my definition), then this is 0 time units.

Avg turnaround time: turnaround time = time of completion of process – time of submission of process (page 128 of textbook)

avg turnaround = ((25-0) + (34-1) + (48-20) + (52-32))/4
= **106/4 time units**

b.With preemptive RR (quantum = 10ms) scheduling? (Different strategies might be used to add a newly submitted process to the ready queue. Explain what strategy you're using.)

Assume new processes go on the TAIL of the ready queue:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	R
P2		R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**											
P3																					R	R	R	R	R	R	R	R	R	**X**
P4																														

	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
P1	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**									
P2																							
P3	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	
P4			R	R	R	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**					

Job throughput: 4 jobs in 51 time units, or **(4/52) jobs per time unit**

Avg waiting time: if wait time = time on ready queue (textbook definition), then this is (19 + 9 + 18 + 12)/4 = **58/4 time units**

if wait time == time on wait queue (my definition), then this is 0 time units.

Avg turnaround time: turnaround time = time of completion of process – time of submission of process (page 128 of textbook)

avg turnaround = ((44-0) + (19-1) + (52-20) + (48-32))/4

= **110/4 time units**

Assume new processes go on the HEAD of the ready queue:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	R
P2		R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**											
P3																					R	R	R	R	R	R	R	R	R	**X**
P4																														

	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
P1	R	R	R	R	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**					
P2																							
P3	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	
P4			R	R	R	R	R	R	R	**X**	**X**	**X**	**X**										

Job throughput: 4 jobs in 51 time units, or **(4/52) jobs per time unit**

Avg waiting time: if wait time = time on ready queue (textbook definition), then this is (23 + 9 + 18 + 7)/4 = **57/4 time units**

if wait time == time on wait queue (my definition), then this is 0 time units.

Avg turnaround time: turnaround time = time of completion of process – time of submission of process (page 128 of textbook)

avg turnaround = ((48-0) + (19-1) + (52-20) + (43-32))/4

= **109/4 time units**

b. With non-preemptive priority scheduling (given the above priorities)?

Assuming higher numbers mean higher priority:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**					
P2		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
P3																					R	R	R	R	R	**X**	**X**	**X**	**X**	**X**
P4																														

	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
P1																							
P2	R	R	R	R	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	
P3	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**														
P4			R	R	R	R	R	R	R	**X**	**X**	**X**	**X**										

Job throughput: 4 jobs in 51 time units, or **(4/52) jobs per time unit**

Avg waiting time: if wait time = time on ready queue (textbook definition), then this is (0 + 42 + 5 + 7)/4 = **54/4 time units**

if wait time == time on wait queue (my definition), then this is 0 time units.

Avg turnaround time: turnaround time = time of completion of process – time of submission of process (page 128 of textbook)

avg turnaround = ((25-0) + (52-1) + (39-20) + (43-32))/4
= **106/4 time units**

Assuming lower numbers mean higher priority:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**					
P2		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**
P3																					R	R	R	R	R	R	R	R	R	R
P4																														

	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
P1																							
P2	**X**	**X**	**X**	**X**																			
P3	R	R	R	R	R	R	R	R	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	
P4			R	R	**X**	**X**	**X**	**X**															

Job throughput: 4 jobs in 51 time units, or **(4/52) jobs per time unit**

Avg waiting time: if wait time = time on ready queue (textbook definition), then this is (0 + 24 + 18 + 2)/4 = **44/4 = 11 time units**

if wait time == time on wait queue (my definition), then this is 0 time units.

Avg turnaround time: turnaround time = time of completion of process – time of submission of process (page 128 of textbook)

avg turnaround = ((25-0) + (34 -1) + (52 -20) + (38-32))/4
= **86/4 time units**

2. Consider the Sleeping-Barber Problem (p202, question 6.7 in the textbook,) with the modification that there are k barbers and k barber chairs in the barber room, intead of just one. Write a program to coordinate the barbers and the customers using Java, C, or pseudo-code. You can use either semaphores or monitors.

Here's a solution that uses semaphores:

```
// shared data
Semaphore waiting_room_mutex = 1;
Semaphore barber_room_mutex = 1;

Semaphore barber_chair_free = k;
Semaphore sleepy_barbers = 0;

Semaphore barber_chairs[k] = {0, 0, 0, …};
int       barber_chair_states[k] = {0, 0, 0, …};

int       num_waiting_chairs_free = N;


boolean customer_entry( ) {

   // try to make it into waiting room
   wait(waiting_room_mutex);
   if (num_waiting_chairs_free == 0) {
      signal(waiting_room_mutex);
      return false;
   }
   num_waiting_chairs_free--;    // grabbed a chair
   signal(waiting_room_mutex);

   // now, wait until there is a barber chair free
   wait(barber_chair_free);

   // a barber chair is free, so release waiting room chair
   wait(waiting_room_mutex);
   wait(barber_room_mutex);
   num_waiting_chairs_free++;
   signal(waiting_room_mutex);

   // now grab a barber chair
   int mychair;
   for (int I=0; I<k; I++) {
     if (barber_chair_states[I] == 0) {  // 0 = empty chair
        mychair = I;
        break;
```

```
      }
    }
    barber_chair_states[mychair] = 1;  // 1 = haircut needed
    signal(barber_room_mutex);

    // now wake up barber, and sleep until haircut done
    signal(sleepy_barbers);
    wait(barber_chairs[mychair]);

    // great! haircut is done, let’s leave.  barber
    // has taken care of the barber_chair_states array.
    signal(barber_chair_free);
    return true;
}

void barber_enters() {
  while(1) {
    // wait for a customer
    wait(sleepy_barbers);

    // find the customer
    wait(barber_room_mutex);
    int mychair;
    for (int I=0; I<k; I++) {
      if (barber_chair_states[I] == 1) {
        mychair = I;
        break;
      }
    }
    barber_chair_states[mychair] = 2;  // 2 = cutting hair
    signal(barber_room_mutex);

    // CUT HAIR HERE
    cut_hair(mychair);

    // now wake up customer
    wait(barber_room_mutex);
    barber_chair_states[mychair] = 0;  // 0 = empty chair
    signal(barber_chair[mychair]);
    signal(barber_room_mutex);

    // all done, we’ll loop and sleep again
  }
}
```

3. Consider the following C++-style pseudo-code. We have a producer thread and a consumer thread running concurrently. Both threads have access to the shared data. Will they function correctly (i.e. each produced item will be consumed)? Why or why not?

No, they will NOT function correctly. If the consumer thread gets context switched out after line 46 and before line 48, then the item on the stack might be overwritten by the producer thread (line 27).

4. Use Java, C, or pseudo-code to implement:

   - monitors using semaphores
   - semaphores using monitors

   Your solution may *not* busy-wait.

```
monitors using semaphores:

   - the answer, for the most part, is in section 6.7 of the text.
     here's some brief pseudocode to fill in the blanks.

     Semaphore mutex = 1, next = 0;
     int next_count = 0;

     For each external procedure F:

     wait(mutex);
      ...
     body of F;
     ...
     if (next_count > 0)
         signal(next);
     else
         signal(mutex);


     For each condition x

     int x_count = 0;
     semaphore x_sem = 0;

     ////////////////////////////
     // x.wait
     x_count = x_count + 1;
     if (next_count > 0)
         signal(next);
     else
         signal(mutex);
     wait(x_sem);
     x_count = x_count - 1;
```

```
    ////////////////////////////
    // x.signal
    if (x_count > 0) {
        next_count = next_count + 1;
        signal(x_sem);
        wait(next);
        next_count = next_count - 1;
    }
```

```
- semaphores using monitors:

    class Semaphore : public Monitor
    {
    protected:
        int count;
        condition cond;

    public:
        Semaphore(int initial) {
            count = initial;
        }

        void wait() {
            count = count - 1;
            while (count < 0) {
                cond.wait();
            }
        }

        void signal() {
            count = count + 1;
            cond.signal();
        }
    };
```

5. Use Java, C, or pseudo-code to implement:

    - semaphores using locks
    - locks using semaphores

   Is it possible to produce a solution that doesn't busywait?

Answer: (sort of):

It is trivial to build solutions that busy wait (I won't bother writing any out here).

It is extremely hard to build solutions that don't busy wait. Essentially, you end up having to build a miniature thread scheduler, complete with waiting queues. We graded liberally on this bonus question.