Solutions to Week 4 Exercises

1. See lecture 4 slide 10.

2. A thread to do keyboard I/O; A thread to update the enemy position; a thread to update the character position; a thread to update the user interface; a main thread to start the other threads and do any other necessary computation.

3. If a web server were not multi-threaded then only one user would be able to connect at once, and others would have to wait until that user had completed all tasks and disconnected before they could connect. This would be very inefficient.

4. An atomic instruction is one whose on a CPU cannot be interrupted part way through. Incrementing a counter is not atomic because it requires loading the value of the counter into the register, then adding to it, then storing it back in memory. Two threads could both execute this and be interleaved thus:
    - load x into register;
    - load x into register;
    - increment x;
    - store x in memory;
    - increment x;
    - store x in memory.

    In this scenario x will only appear to have been incremented once because the second thread loaded its value before the first had stored the result.

5. If two processes attempt to update the array simultaneously then data loss can occur if the process of adding items to the array is not forced to occur under mutual exclusion. Suppose that both threads are going to store something at the next available position in the array:
    - Thread 1: int i = get next available position;
    - Thread 2: int i = get next available position;
    - Thread 1: array[i] = myData;
    - Thread 2: array[i] = myData;
    - Thread 1: increment next available position;
    - Thread 2: increment next available position.

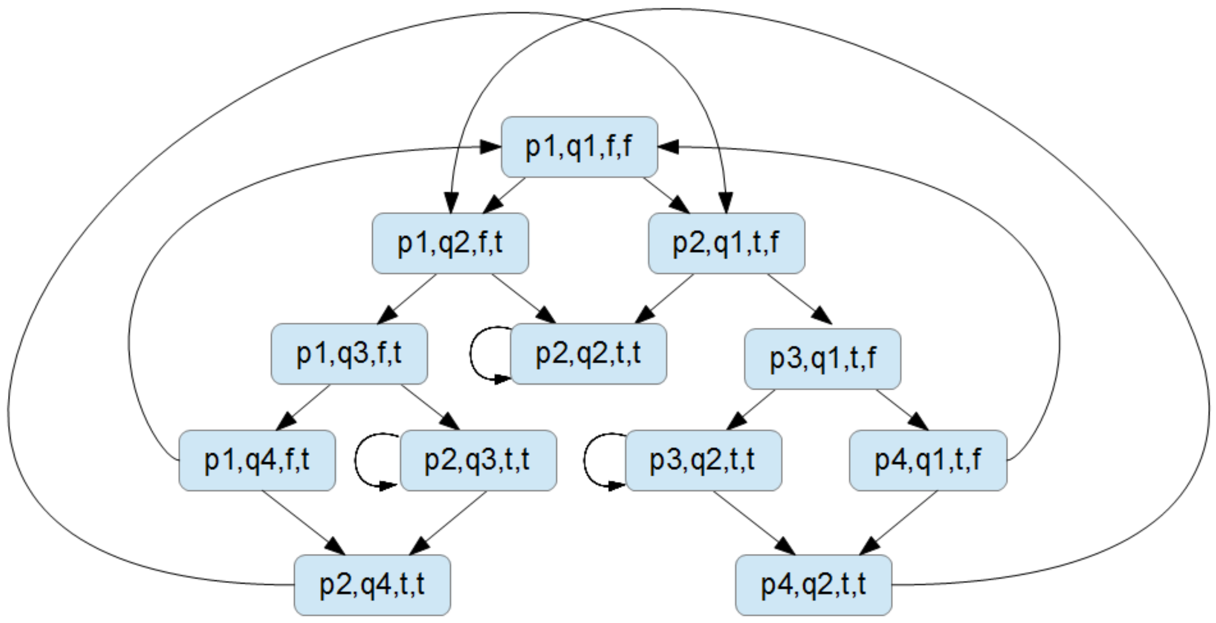
    Now the data from Thread 2 overwrote the data from thread 1, so the data it put in the array was lost.

6.
    - For mutual exclusion to be violated we must reach the state (p3,q3,_,_). That is, one process must remain at line 3 whilst the other transitions to line 3. Since the processes are symmetrical we will assume, p is at p3 and show that q cannot transition to q3;
        - If p is at p3 then turn = 1:
            - This is because the only statement that sets turn = 2 is p4, but p2 (await turn = 1) must execute between p4 and p3 (inspection of the program).
            - If turn = 1 then:
                - if q is at q4 it will simply set it to 1 again and continue to q1;
                - if q is at q1 it will continue to q2, turn is unaffected;
                - if q is at q2 (or when it reaches q2) it will have to wait and cannot proceed.

      Therefore mutual exclusion holds.
    - The solution does not satisfy freedom from starvation. If either process terminates in its non-critical section then the other can execute its critical section once. Following that it will set turn such that the terminated process can proceed and it cannot. Since the

terminated process has, in fact, terminated, it will never set turn to allow the executing process to run. An execution trace that leads to starvation of p is: q1 (q terminates), p1,p2,p3,p4.

7. State Diagram: since the state p3,p3,_,_ never arises mutual exclusion is satisfied.



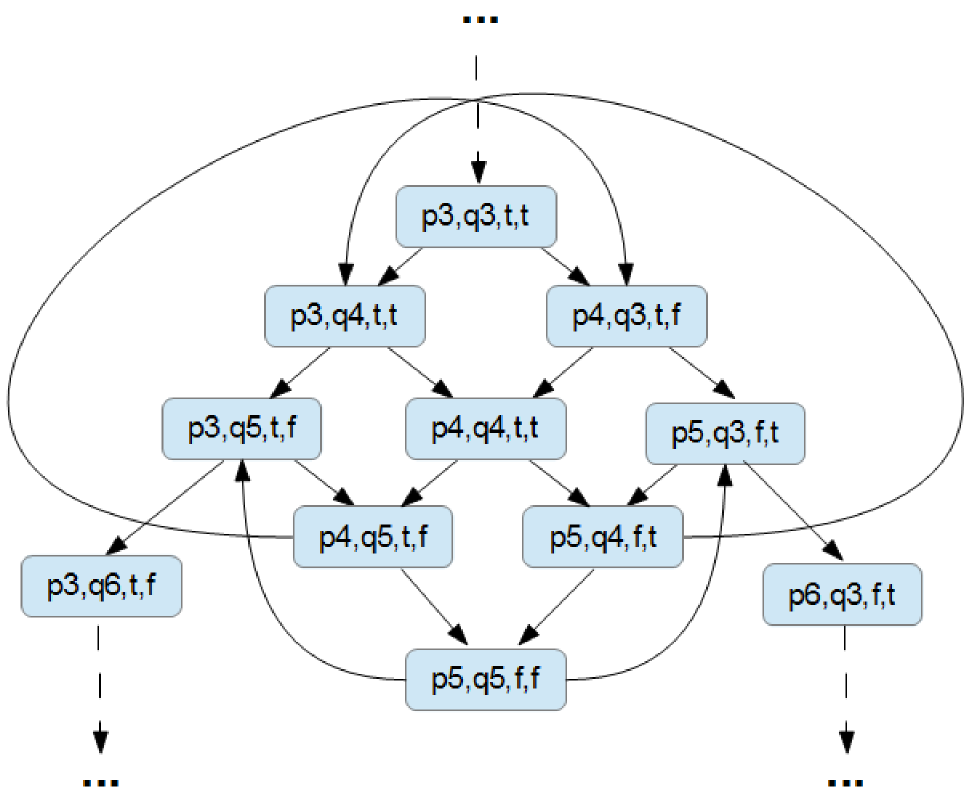
Deadlock can occur: the state p2,q2,t,t is reachable, but there is no path out of it: if the system enters that state then it can never exit.

8.

- **Mutual Exclusion: satisfied.** For mutual exclusion to be violated we would require both processes to be at p3. That is, one must be at line 6 and the other must transition to line 6. Since the processes are symmetrical let's consider the case where p is at p6 and q must transition to q6:
  - If p is at p6 then want p = true:
    - The only process to change wantp is p. Since p is at p6 the last statement executed to assign a value to wantp was p5: wantp = true.
  - If q is at q7,q1, or q2 it will proceed until it reaches q3 and no change will be made to wantp.
  - If q is at q3 (or reaches it from earlier) then it will remain in the while loop q3-q7, altering the value of wantq, but it cannot exit this loop as q cannot change wantp, only p can do that once it has left p6.

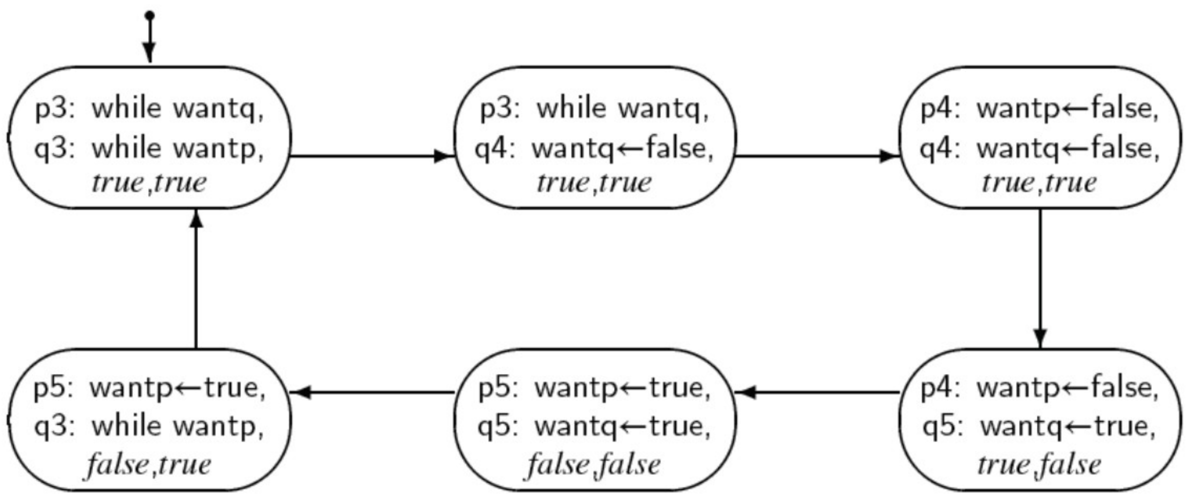
  Therefore mutual exclusion holds.

- **Freedom from Deadlock: satisfied.** For deadlock to occur both processes must be executing the while loop continuously without either being able to escape. The following is an extract of the state diagram for this region: we know that when p is at p3 wantp must be true, since p3 follows execution of either p2 or p5 (both of which are wantp = true) and q does not change wantp. Similarly at q3 wantq = true, therefore we need only consider this in the state diagram (no need for p3,q3,f,t or p3,q3,t,f). The dotted lines and … represent joins to the remainder of the state diagram.



Observe that from every state in the state diagram it is possible to reach a state where either p or q enter their critical section (p3,q6.t.f or p6,q3,t,f). Therefore there is no deadlock in the system (for deadlock to occur there would need to exist a state from which it was not possible to reach one of these.

- **Freedom from Starvation:** not satisfied. Continued execution of the following cycle leads to starvation:


p3: while wantq,
q3: while wantp,
true,true
p3: while wantq,
q4: wantq←false,
true,true
p4: wantp←false,
q4: wantq←false,
true,true
p4: wantp←false,
q5: wantq←true,
true,false
p5: wantp←true,
q5: wantq←true,
false,false
p5: wantp←true,
q3: while wantp,
false,true