Sample Midterm Solution Sketch:

<u>Question 1: T/F Questions</u>

1. T
2. F
3. T
4. F
5. T
6. F
7. T
8. F
9. T
10. F
11. F
12. T
13. F
14. F
15. F (LWPs are mapped to kernel threads).
16. F
17. F
18. T
19. T
20. F

<u>Question 2:</u> Scheduling

See lecture notes for 2(a) and 2(b). 2(c) is straightforward, similar to HW2 question.

<u>Question 3: Processes, threads, synchronization</u>

Problem 3a: (see lecture notes)

- Critical section problem: N processes that may potentially be executing in parallel and are all competing to use shared data. The problem is how to ensure that when one process is executing in its critical section no other process is allowed to execute in its critical section.
- Mutual Exclusion: If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
- Progress: if no process is executing in its critical section and there exists some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
- Bounded Waiting: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Problem 3b:
Four conditions are necessary/sufficient for a deadlock to occur. Each of the four conditions must hold. By ensuring that at least one of these conditions cannot hold, we can prevent deadlock.

1. Mutual Exclusion: This can lead to deadlock because mutual exclusion ensures that only one process at a time can use the shared resource.
2. Hold and Wait: The situation of having processes holding the resources already allocated to them while they are waiting for the other resources they need for their computation can cause deadlock.
3. No preemption: The situation of having no preemption means that a process cannot be interrupted during its computation, and therefore the resources allocated to it cannot be freed unless the

computation ends. With preemption you could put an end to some critical process execution (e.g., the one holding the resources most needed) and solve a deadlock situation.

4. Circular Wait: It is a special case of hold-and-wait situation. A circular chain of processes exists in which each process waits for one or more resources held by the next process in the chain.

Problem (3c) – No deadlock exists.

Problem (3d)
Solution:

```
var a: array [0..2] of semaphore {initial condition = 0   }
 agent: semaphore {initial condition =   1 }
agent code

repeat
   Set i,j to a value = (P,T), (T,M) or (M,P)
   wait(agent);
   signal(a[i]);
    signal(a[j]);
until false;
smoker code

repeat
   Set r,s to a value =            (P,T), (T,M) or (M,P)
   wait(a[r]);
   wait(a[s]);
   "smoke"
    signal(agent);
until false;
```

Key intuition: There is a specific order in which the items (paper(P), tobacco(T), matches(M)) are placed by the agent and requested by the smokers.

Agent places (P,T) (T, M) or (M,P).
Smoker with matches requests other items in the order (P,T); smoker with paper requests items in the order (T,M); smoker with tobacco requests items in the order (M,P).

Problem 3e: See Pseudo-code for mutual exclusion using test-and-set in lecture notes.