1) In a system with threads, is there one stack per thread or one stack per process ? Explain.

2) Consider a computer that does not have a TEST AND SET LOCK instruction but does have an instruction to swap the contents of a register and a memory word in a single indivisible action. Can that be used to write a routine **enter_region** such as the one found on page 63 of the text ? If so, explain how it works.

3) Can the Wait and Signal definitions defined on Slide 35 of Processes be used without disabling and enabling interrupts ? If not, what advantage(s) do Wait and Signal provide over the alternative; allowing the user to disable and enable interrupts directly around critical sections ? Assume the Wait and Signal routines are implemented as system calls, as they really are in protected mode OSs such as UNIX. Are mutual exclusion, progress and bounded waiting satisfied using semaphores with FIFO wait queues ?

4) Draw diagrams showing how the ready queue (R_QUEUE) and semaphore queue (S_QUEUE) change over time given three processes which share a critical section using Wait and Signal calls on a semaphore. The processes actions are described below. Assume all three processes are on the ready queue initially. Also assume that the process scheduled to run is removed from all queues and is pointed to be a global variable called CURRENT.
   1) Process P1 runs and calls Wait. Since the semaphore value was initialized to 1, it immediately enters its critical section.
   2) While in the critical section, the timer fires and P1 is preempted.
   3) P2 runs and calls Wait.
   4) P3 runs and calls Wait.
   5) P1 runs and calls Signal.
   Assume that a processes waiting on a semaphore has priority over a process that is leaving the critical section (calling Signal).

5) Suppose you were programming a multi-threaded process in a system that does not provide Wait and Signal system calls. Suppose you wanted to run two threads which updated a shared buffer, therefore you need to provide mutual exclusion. Additionally, suppose it is important that each of these processes have a fair chance to update the buffer since each of them do something different to the buffer. Can you use Peterson's solution and Wait and Signal routines to provide this functionality ? If so, show how you would do this. Are mutual exclusion, progress and bounded waiting satisfied if you use FIFO wait queues in your implementation of Wait and Signal ?

6) Consider the procedure put_forks on page 78 of the dining philosophers solution. Suppose that the variable **state[i]** was set to THINKING *after* the two calls to **test**, rather than *before*. How would this change affect the solution for the case of three philosophers ? For 100 philosophers ?

7) Measurements of a certain system have shown that the average process runs for a time

T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following.

a) $Q = \infty$
b) $Q > T$
c) $S < Q < T$
d) $Q = S$
e) Q nearly 0

8) Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

a) Round robin
b) Priority scheduling.
c) First-come, first-served (run in order 10, 6, 2, 4, 8)
d) Shortest job first.

For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d) assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

9&10) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

a) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling. (5pts)
b) What is the turnaround time of each process for each of the scheduling algorithms in part a ? (5pts)
c) What is the waiting time of each process for each of the scheduling algorithms in part a ? (5pts)
d) Which of the schedules in part a results in the minimal average waiting time (over all processes)? (5pts)