Problem 1 (7.3)
The CPU scheduler determines which thread to run now and when to preempt for context switching. Deadlock occurs only when the CPU scheduler preempts a thread in specific code lines.

It may contribute to deadlock in this program as the followings:
Situation 1: If thread_one runs first and preempted in between "pthread_mutex_lock(&first_mutex);" and "pthread_mutex_lock(&second_mutex);", thread_two runs and finishes "pthread_mutex_lock(&second_mutex);" then we will occur deadlock.
Situation 2: If thread_two runs first and preempted in between "pthread_mutex_lock(&second_mutex);" and "pthread_mutex_lock(&first_mutex);", thread_one runs and finishes "pthread_mutex_lock(&first_mutex);" then we will also occur deadlock.
In both situations, thread_one acquires first_mutex while thread_two acquires second_mutex. They would wait for the mutex lock of the resource held by the other one in the next code line, which results to both of them occurring starvation. The program satisfies mutual exclusion, hold and wait, circular wait and no preemption. Thus, we can conclude that it is in a deadlock situation.

Problem 2 (7.6)
If deadlock is controlled by the Bankers algorithm in the original system, and the original state is safe. Then, changes a., d., f. can be made safely.

a. If all the other numbers are fixed. By increasing Available, the Work becomes larger which is the upper bound of the original Work. Thus, the original safety algorithm will be satisfied with the new Work.
d. If all the other numbers are fixed. By decreasing Max for one process, the Need of it will also decrease, since "Need = Max – Allocation". Thus, the original safety algorithm will be satisfied with the new Max.
f. If all the other numbers are fixed. By decreasing the number of processes, the Available for the rest processes will increase, which causes the new Work to be the upper bound of the original Work. Thus, the original safety algorithm will be satisfied after decreasing the number of processes.

Problem 3 (7.12)
“Need = Max – Allocation” is done before the algorithm starts.

a.
Work = Available = (0, 3, 0, 1)
step 1: P2 Need = (0, 2, 0, 0) <= Work = (0, 3, 0, 1)
=> Work = Work + Allocation = (0, 3, 0, 1) + (3, 1, 2, 1) = (3, 4, 2, 2)
step 2: P1 Need = (1, 0, 0, 1) <= Work = (3, 4, 2, 2)
=> Work = Work + Allocation = (3, 4, 2, 2) + (2, 2, 1, 0) = (5, 6, 3, 2)
step 3: P3 Need = (4, 1, 0, 2) <= Work = (5, 6, 3, 2)
=> Work = Work + Allocation = (5, 6, 3, 2) + (0, 5, 1, 0) = (5, 11, 4, 2)
step 4: P0 Need = (2, 1, 0, 3) > Work = (5, 11, 4, 2)
P4 Need = (2, 1, 1, 3) > Work = (5, 11, 4, 2)

=> The state is unsafe. Because, whenever P0 or P4 request for the maximum resource they may get, resource D lacks one unit. The request cannot be satisfied and cannot maintain a safe sequence.

b.
Work = Available = (1, 0, 0, 2)
step 1: P1 Need = (1, 0, 0, 1) <= Work = (1, 0, 0, 2)
=> Work = Work + Allocation = (1, 0, 0, 2) + (2, 2, 1, 0) = (3, 2, 1, 2)
step 2: P2 Need = (0, 2, 0, 0) <= Work = (3, 2, 1, 2)
=> Work = Work + Allocation = (3, 2, 1, 2) + (3, 1, 2, 1) = (6, 3, 3, 3)
step 3: P0 Need = (2, 1, 0, 3) <= Work = (6, 3, 3, 3)
=> Work = Work + Allocation = (6, 3, 3, 3) + (3, 0, 1, 4) = (9, 3, 4, 7)
step 4: P3 Need = (4, 1, 0, 2) <= Work = (9, 3, 4, 7)
=> Work = Work + Allocation = (9, 3, 4, 7) + (0, 5, 1, 0) = (9, 8, 5, 7)
step 5: P4 Need = (2, 1, 1, 3) <= Work = (9, 8, 5, 7)
=> Work = Work + Allocation = (9, 8, 5, 7) + (4, 2, 1, 2) = (13, 10, 6, 9)

=> The state is safe. The safe sequence = <P1, P2, P0, P3, P4>.