•Grading: 3 = correct

2 = almost

1 = an attempt

0 = nothing

•Score: Points / Possible

# Homework #3 \_\_\_\_Taylor Whitlock\_\_\_\_\_\_\_\_\_ \_\_01\_

# (38 points) (Name) (Section)

**Chapter 5 – Mutual Exclusion**

**Chapter 6 – Deadlock / Starvation**

|  |  |
| --- | --- |
| Questions: | Answers: |
| 1. (6.1) (4 points) Show how the four conditions of deadlock might apply to the following figure:  Intersection | 1. The different Cars (processes) cannot share the same space/resources (quadrants) – Mutual exclusion  2. Hold and wait – the green bus waits for the silver car in d to finish what its doing.  3. No preemption - No car can forcibly remove another car from its resources  4. Circular Wait - A is waiting on B is waiting on C is waiting on D is waiting on A, so nobody moves. |
| 2. (6.3) (15 points) Consider the following snapshot of a system. (There are no outstanding unsatisfied requests for resources.)  |  |  |  |  | | --- | --- | --- | --- | | Available | | | | | r1 | r2 | r3 | r4 | | 2 | 1 | 0 | 0 |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Current Allocation | | | | Maximum Demand | | | | |  | r1 | r2 | r3 | r4 | r1 | r2 | r3 | r4 | | P1 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | | P2 | 2 | 0 | 0 | 0 | 2 | 7 | 5 | 0 | | P3 | 0 | 0 | 3 | 4 | 6 | 6 | 5 | 6 | | P4 | 2 | 3 | 5 | 4 | 4 | 3 | 5 | 6 | | P5 | 0 | 3 | 3 | 2 | 0 | 6 | 5 | 2 |  a. Compute what each process still might request and display in the columns labeled “still needs”.b. Is this system currently in a safe or unsafe state? Why?c. Is this system currently deadlocked? Why or why not?d. Which processes, if any, are or may become deadlocked?e. If a request from P3 arrives for (0,1,0,0), can that request be safely granted immediately? In what state (deadlock, safe, unsafe) would immediately granting that whole request leave the system? Which processes, if any, are or may become deadlocked if this whole request is granted immediately? | A.  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Still Needs | | | | |  | r1 | r2 | r3 | r4 | | P1 | 0 | 0 | 0 | 0 | | P2 | 0 | 7 | 5 | 0 | | P3 | 6 | 6 | 2 | 2 | | P4 | 2 | 0 | 0 | 2 | | P5 | 0 | 3 | 2 | 0 |   B. The system is safe because every process can finish with the available resources.C. The system is not currently deadlocked, because there is no circular waiting.D. If it finishes in an order other than p1, p4, p5, p2, p3, then p2. P3, and p5 may become deadlockedE. The request can be granted safely, but it would leave the system in an unsafe state. |
| 3. (6.2) (4 points) Apply the deadlock detection algorithm of Section 6.2 to the following data and show the results:Available = | 2 1 0 0 || 2 0 0 1 | | 0 0 1 0 |Request = | 1 0 1 0 | Allocation = | 2 0 0 1 || 2 1 0 0 | | 0 1 2 0 | | Results:| 2 1 0 0 | to | 2 2 2 0 | to| 4 2 2 1 | to| 4 2 3 1 | (final results) |
| 4. (6.3) (6 points) Consider a system with a total of 150 units of memory, allocated to three processes as shown:  |  |  |  | | --- | --- | --- | | Process | Max | Hold | | 1 | 70 | 45 | | 2 | 60 | 40 | | 3 | 60 | 15 |  Apply the banker’s algorithm from Section 6.3 to determine whether it would be safe to grant each of the following requests. If yes, indicate a sequence of terminations that could be guaranteed possible. If no, show the reduction of the resulting allocation table.a. A fourth process arrives, with a maximum memory need of 60 and an initial need of 25 units.b. A fourth process arrives, with a maximum memory need of 60 and an initial need of 35 units. | A. P1 terminates first, then P2, then P3, then P4.B.  |  |  |  |  | | --- | --- | --- | --- | | Process | Max | Hold | Needs | | 1 | 70 | 45 | 25 | | 2 | 60 | 40 | 20 | | 3 | 60 | 15 | 45 | | 4 | 60 | 35 | 25 |  Out of the 150 available units, all but 15 are used, and all processes need more than 15 so it would be deadlocked. |
| 5. (6.5) (9 points) Consider the following ways of handling deadlock:(1) banker’s algorithm,(2) detect deadlock and kill thread, releasing all resources,(3) reserve all resources in advance,(4) restart thread and release all resources if thread needs to wait,(5) resource ordering, and(6) detect deadlock and roll back thread’s actions.a. One criterion to use in evaluating different approaches to deadlock is which approach permits the greatest concurrency. In other words, which approach allows the most threads to make progress without waiting when there is no deadlock. Give a rank order from 1 to 6 for each of the ways of handling deadlock just listed, where 1 allows the greatest degree of concurrency. Comment on your ordering.b. Another criterion is efficiency; in other words, which requires the least processor overhead. Rank order the approaches from 1 to 6, with 1 being the most efficient, assuming that deadlock is a very rare event. Comment on your ordering.c. Does your ordering from (b) change if deadlocks occur frequently?Note: Although the answers are somewhat subjective, your most and least rankings should be correct. In any case, justify your answers. | A. 6 🡪 2 🡪 5 🡪 1 🡪 4 🡪 3Rolling back thread progress is better than restarting it entirely, when detecting but it’s fine to restart only when deadlock is detected. Resource ordering can help just make sure the most important things can be used. Restarting a thread completely when it simply needs to wait hinders progress. The banker’s algorithm can be far less concurrent if there are lots of unsafe zones. Reserving all resources is likely to prevent some threads from even starting.B. 1 🡪 5 🡪 3 🡪 6 🡪 2 🡪 4If deadlock is rare, then there are fewer unsafe zones and the banker’s algorithm will be very efficient. Resource ordering is efficient because it will cut down use on unnecessary things. If deadlock is rare, then advance reservation will be an initial efficiency cost, but then not much after that. Since deadlock is rare, trying to detect it constantly would decrease efficiency, but the thread handling wouldn’t be too bad.C. Yes, if deadlocks occur frequently then it would be more efficient to detect it and handle things that way. |