Seminar 4

Solutions

Scheduling

1. The CPU load due to the threads A, B, and C is 1/4+2/5+1/8 = 31/40 = 0.775. It is below the limit given by the scheduling test of Liu and Layland (0.780) so it should be schedulable (ignoring scheduling costs, etc.).

   In order calculate the response time of each thread we first need to determine the priority order. RMS states that priorities should be assigned such that the thread with shortest period (T) should have the highest priority, and so on. We thus get the order A – B – C for the thread priorities. In order to calculate the response times we can either use the recursive formula derived by Joseph and Pandya in their Rate Monotonic Analysis or, since no blocking between the threads are involved, simply draw a scheduling diagram showing what happens at the critical instant when all threads are released simultaneously and see what the response times will be. Here, we choose the latter method, see Figure 1. From any of the two cases in the lower part of the diagram we can deduce that the response times for the threads are A:1, B:3, and C:4 time units which verifies the schedulability of the system.

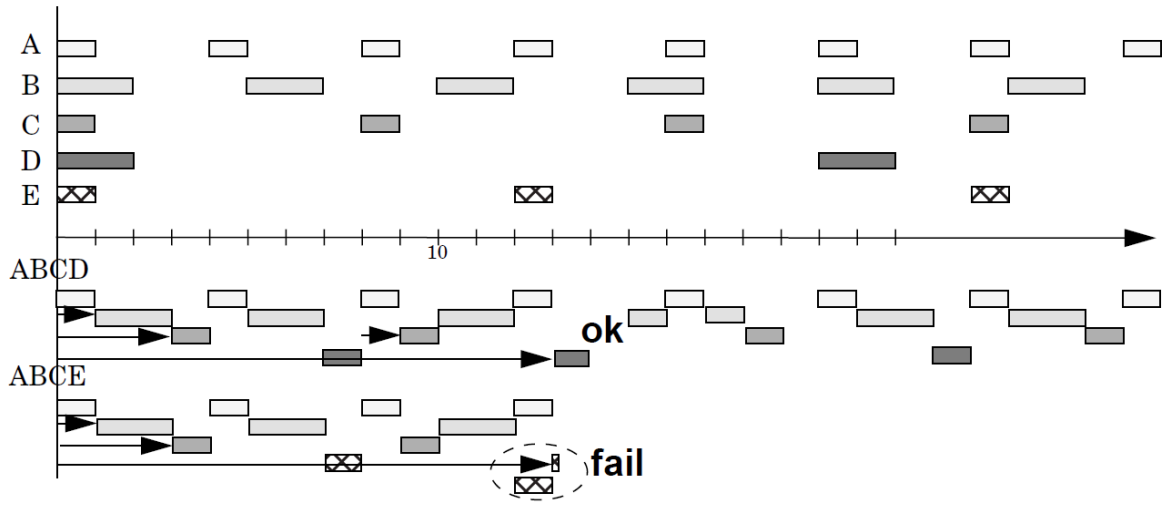

Figure 1: Scheduling diagram for the critical instant thread A to E. The top five lines indicate when the threads would like to run. Below you find the actual scheduling for the thread sets ABCD and ABCE respectively.

2. As the ABCD part of the diagram in Figure 1 shows, the system is schedulable even with the thread D which gets the response time 14 time units (even though the scheduling test of Liu & Layland casts doubt on the schedulability of the system – 1/4+2/5+1/8+2/20=0.875>0.757).
3. The system ABCE is not schedulable because E cannot complete until t=13.2 which is past the deadline. Hence, not schedulable.

Scheduling with blocking

1. In order to compute the response times in a system with blocking between the threads we need to apply Generalized Rate Monotonic Analysis and the more general version of the formula derived by Joseph and Pandya. To do so we first need to determine the maximum blocking times (the blocking factor – B) for each thread. If we assume the threads call at most one monitor method at a time (no nested calls) we get:
   Thread A: max([M1:b+M2.t], [M1:c]) = [M1:b+M2.t] =1.0+0.5 = 1.5. The maximum occurs when B is in monitor M1 (calling b) at the same time C is blocking M2 (calling t). Since A has the highest priority it can only suffer from direct blocking.
   Thread B: max([M2:t indirect via A,C], [M1:c]) + [M3:y] = 0.5+2.5 = 2.5. B can thus be blocked by C either directly or indirectly if it inherits the priority of A. Here, it turns out that the blocking time will be the same either way, however.
   Thread C: [M3:y indirectly via B,D] = 2.0. C can thus be delayed by D if it temporarily inherits the priority of B, which can happen for at most 2.0 time units.
   Thread D: There are no threads with a lower priority that can delay it, so the blocking time is trivially 0.
   By applying the recursive formula from the Generalized Rate Monotonic Analysis and using the threads maximum execution time as a starting point (the response time must be at least as long as the maximum execution time) we get:

   $R_A = C_A+B_A = 1+1.5 = 2.5.$ (no need to continue iterating)

   $R_B = C_B+B_B+ceil(R_B/T_A)*C_A = 2+2.5+ceil(2/4)*1 = 5.5$
   $R_B = 2+2.5+ceil(5.5/4)*1 = 6.5$
   $R_B = 2+2.5+ceil(6.5/4)*1 = 6.5$ (stable – no further iteration necessary)

   $R_C = C_C+B_C+ceil(R_C/T_A)*C_A+ceil(R_C/T_B)*C_B = 1+2+ceil(1/4)*1+ceil(1/5)*2 = 6$
   $R_C = 1+2+ceil(6/4)*1+ceil(6/5)*2 = 9$
   $R_C = 1+2+ceil(9/4)*1+ceil(9/5)*2 = 10$
   $R_C = 1+2+ceil(10/4)*1+ceil(10/5)*2 = 10$ (stable)

$R_D = C_D+B_D+ceil(R_D/T_A)*C_A+ceil(R_D/T_B)*C_B+ceil(R_D/T_C)*C_C =$
$2+0+ceil(2/4)*1+ceil(2/5)*2+ceil(2/8)*1 = 6$
$R_D = 2+0+ceil(6/4)*1+ceil(6/5)*2+ceil(6/8)*1 = 9$
$R_D = 2+0+ceil(9/4)*1+ceil(9/5)*2+ceil(9/8)*1 = 11$
$R_D = 2+0+ceil(11/4)*1+ceil(11/5)*2+ceil(11/8)*1 = 13$
$R_D = 2+0+ceil(13/4)*1+ceil(13/5)*2+ceil(13/8)*1 = 14$
$R_D = 2+0+ceil(14/4)*1+ceil(14/5)*2+ceil(14/8)*1 = 14$ (stable)

The response times are thus: A: 2.5, B:6.5, C:10, and D: 14 time units.

Note that some of the ceiling terms are right on the limit to increase by one. There are thus no margins at all for compensating for any execution time being longer than specified which makes the calculations not very robust.

Regarding the schedulability of the system we see that neither thread B nor thread C will meet their deadlines in the presence of the blocking in question.

2. After the rewriting of D and M3, the worst-case blocking time for B will decrease to 0.5 + 0.5 = 1.0. The response time will then be 2.0+1.0+1.0=4.0 which means that B will meet its deadline. It is, however, quite sensitive since any additional computing time of A or B would cause A to be run once more before B completes and the response time would be slightly more than the deadline of 5. So even if the deadline in theory is met, having no margins means that we are very sensitive to errors in the estimated execution times. Anyway, B is formally OK, but what about C which did not meet its deadline either? With M3 rewritten we obtain the maximum blocking time of 0.5 (M3:yk indirect via B,D). This gives the response time (using the iteration formula as earlier) $R_C = 7.5 < 8$. Since D as the lowest priority thread will never be blocked waiting for resources, all deadlines are met.

3. To decrease the maximum blocking time for A one could merge the monitors M1 and M2, either by rewriting them or by using the priority ceiling or immediate inheritance protocols. That gives the response time 2 exactly. Another alternative is to rewrite M1 and B in such a way that the blocking time of 1.0 gets smaller. It is not clear from the facts given if that can be done in this application.