NTNU | Norwegian University of Science and Technology

Operating Systems

Theoretical Exercise 6: Solutions

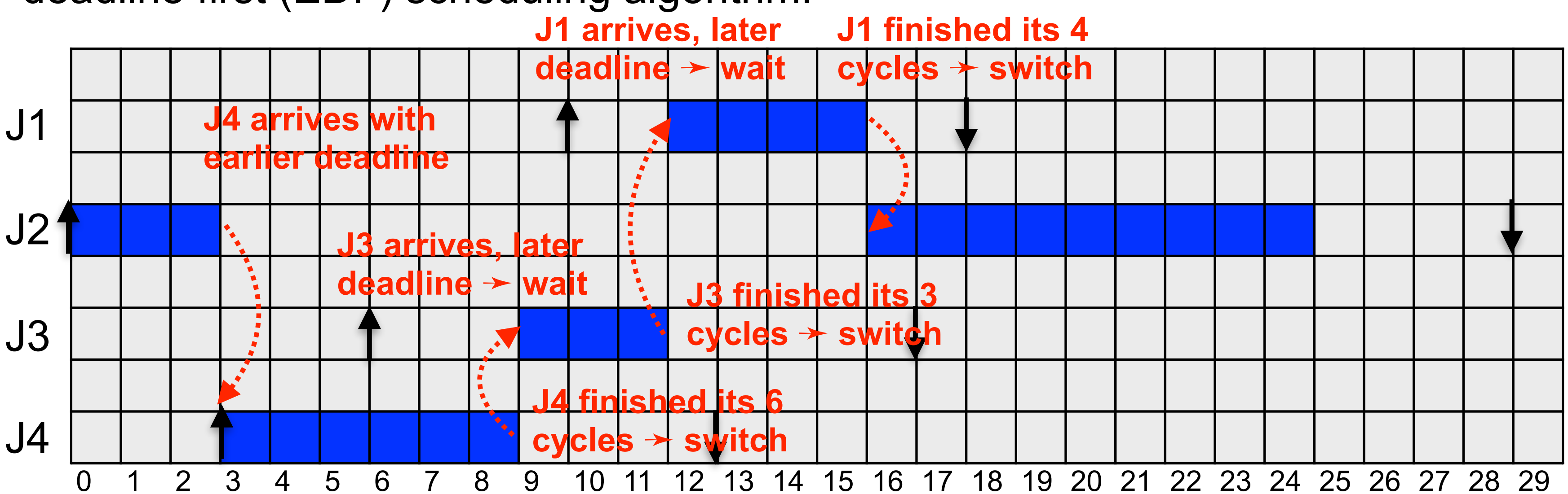
Michael Engel

6.1 EDF scheduling

Suppose that we have a set of four jobs. Release times r_i, deadlines Di, and execution times C_i are as follows: *(Deadlines are given as **absolute** time!)*

- J1: r1=10, D1=18, C1=4
- J2: r2=0, D2=28, C2=12
- J3: r3=6, D3=17, C3=3
- J4: r4=3, D4=13, C4=6

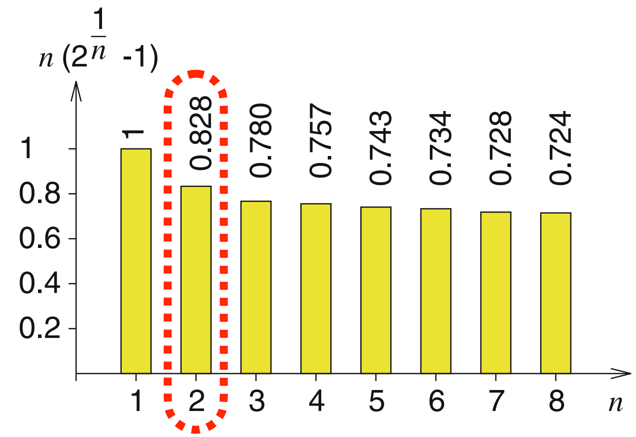
Generate a graphical representation of schedules for this job set, using the earliest deadline first (EDF) scheduling algorithm.



6.2 Rate-monotonic scheduling

Suppose that we have a system comprising two tasks. Task 1 has a period of 5 and an execution time of 2. The second task has a period of 7 and an execution time of 4. Let the deadlines be equal to the periods. Assume that we are using rate monotonic scheduling (RMS).

a. Could any of the two tasks miss its deadline, due to a too high processor utilization?
b. Compute this utilization, and compare it to a bound which would guarantee schedulability!



Necessary RMS condition:

For a single processor and for n tasks, the accumulated utilization U_{sum} does not exceed the following bound ($T_i = D_i$ for RMS!)

$$U_{sum} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) = 2/5 + 4/7 = 0.4 + 0.571 = 0.971 > 0.828$$

=> schedulability cannot be guaranteed, a task could miss its deadline!

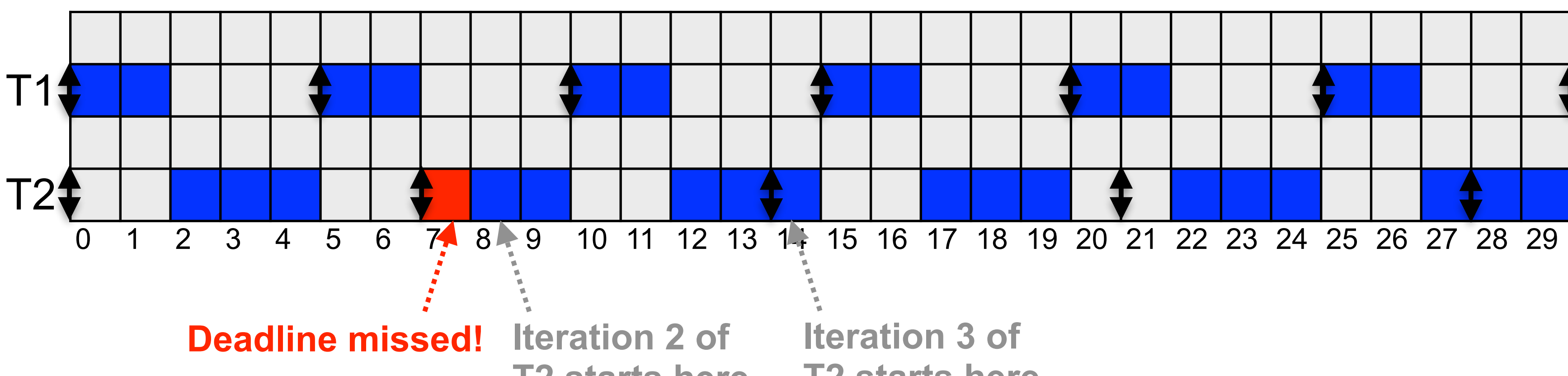
6.2 Rate-monotonic scheduling

Suppose that we have a system comprising two tasks. Task 1 has a period of 5 and an execution time of 2. The second task has a period of 7 and an execution time of 4. Let the deadlines be equal to the periods. Assume that we are using rate monotonic scheduling (RMS).

c. Generate a graphical representation of the resulting schedule! Suppose that tasks will always run to their completion, even if they missed their deadline.

For RMS, the priority of tasks is a monotonically decreasing function of their period

=> a task with lower priority is preempted when a task with higher priority arrives: T1 has the higher priority and can preempt T2



6.3 Priority inversion

Let A, B, and C be three tasks with priorities A=1 (highest), B=3, C=5 (lowest). Tasks A and C use a shared resource (e.g. shared memory) protected by a semaphore. The execution of the tasks is shown in fig. 1:

Tasks
are activated
(once only)
at these times:

A: t = 2T

B: t = 4T

C: t = 0T

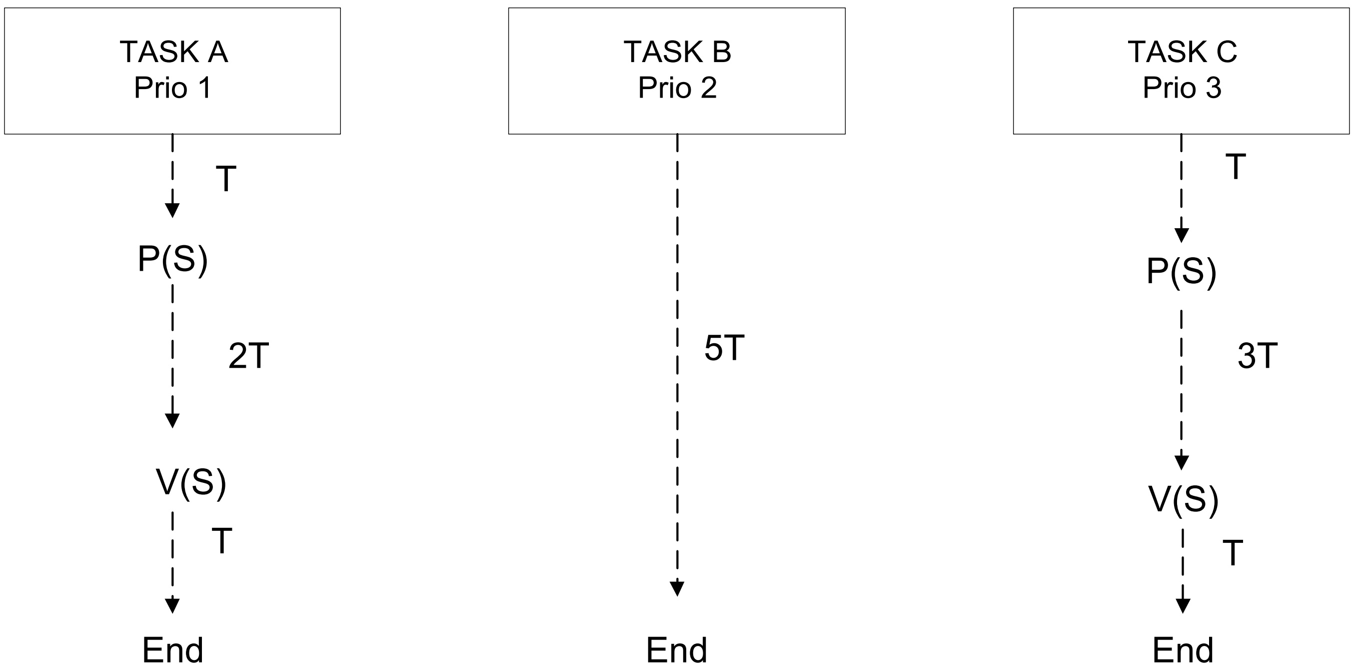


Figure 1: Execution of the tasks

6.3 Priority inversion

Priorities A=1 (highest), B=3, C=5 (lowest).
Tasks A and C use a shared resource:

Tasks are activated at

A: t = 2T   B: t = 4T   C: t = 0T

a. Use fig. 2 to visualize the desired execution sequence for the time interval 0 ≤ t ≤ 14T. Desired = without blocking due to priority inversion!

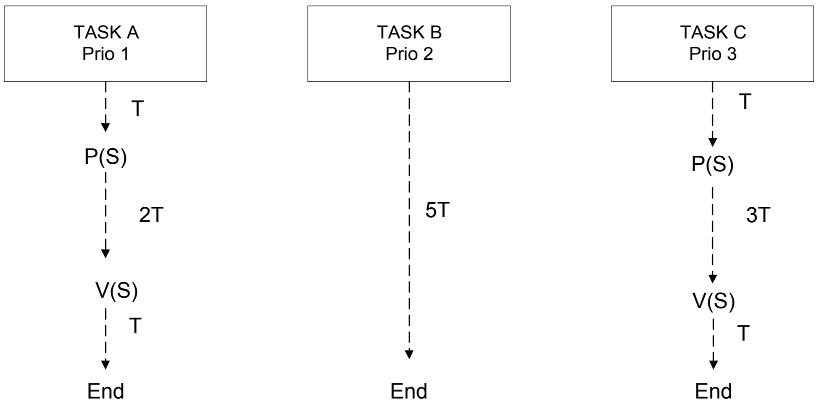


Figure 1: Execution of the tasks

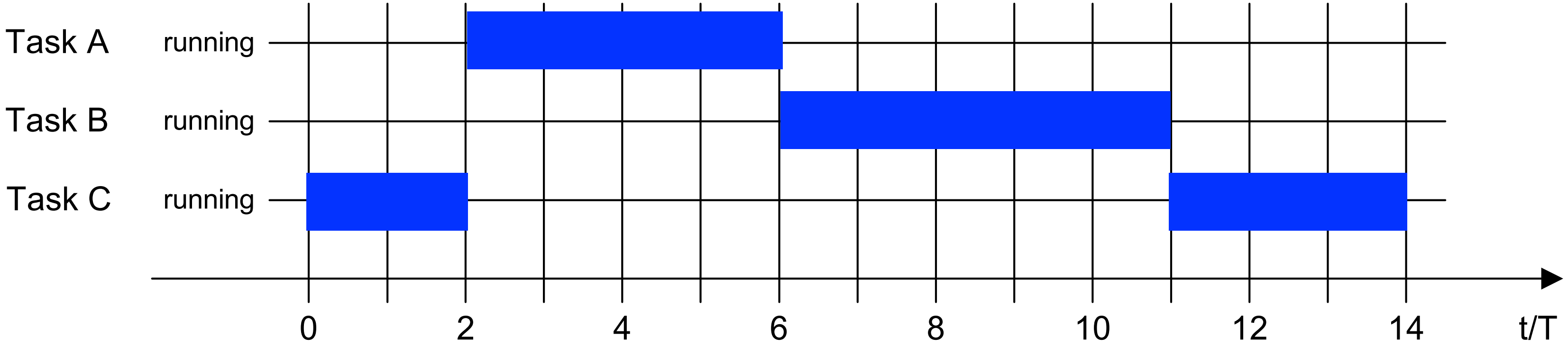


Figure 2: Desired execution sequence

6.3 Priority inversion

Priorities A=1 (highest), B=3, C=5 (lowest).
Tasks A and C use a shared resource:

Tasks are activated at

A: t = 2T   B: t = 4T   C: t = 0T

b. Use fig. 3 to visualize the actual execution sequence with the according tasks states of the three tasks for the time interval 0 ≤ t ≤ 14T.

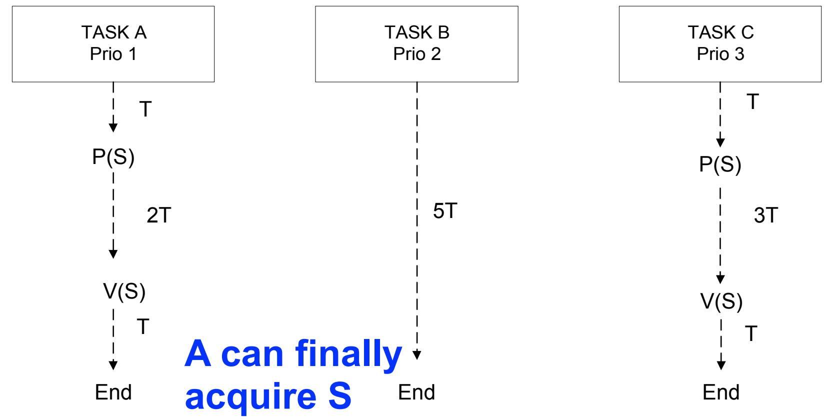
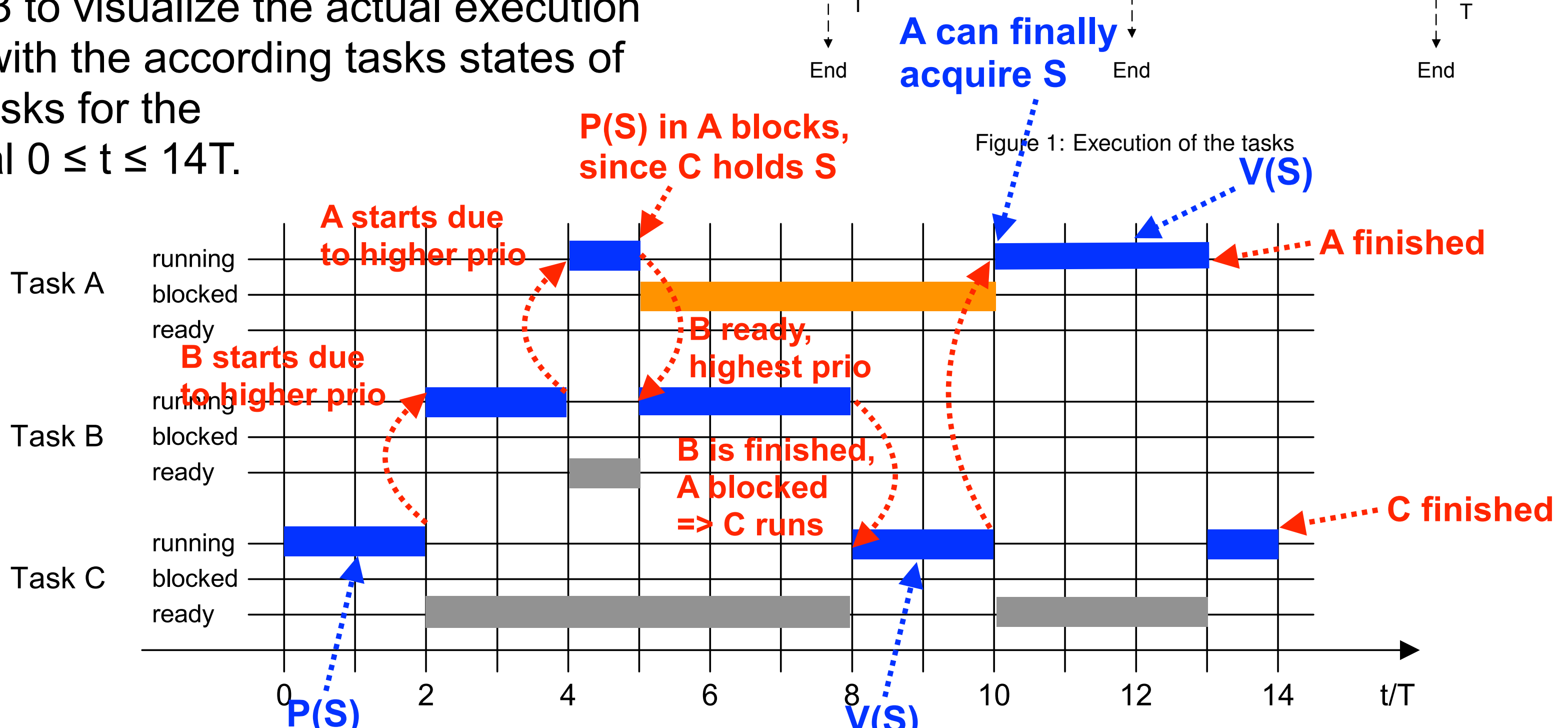


Figure 1: Execution of the tasks



Whoops, messed this up – wrong I/O times & start times of A and B!

…3: Actual execution sequence

6.3 Priority inversion

Priorities A=1 (highest), B=3, C=5 (lowest).
Tasks A and C use a shared resource:

Tasks are activated at

A: t = 2T   B: t = 4T   C: t = 0T

b. Use fig. 3 to visualize the actual execution sequence with the according tasks states of the three tasks for the time interval 0 ≤ t ≤ 14T.

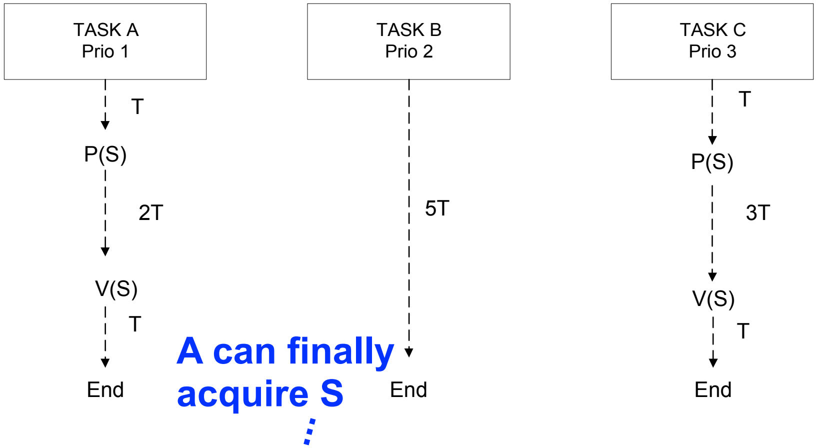
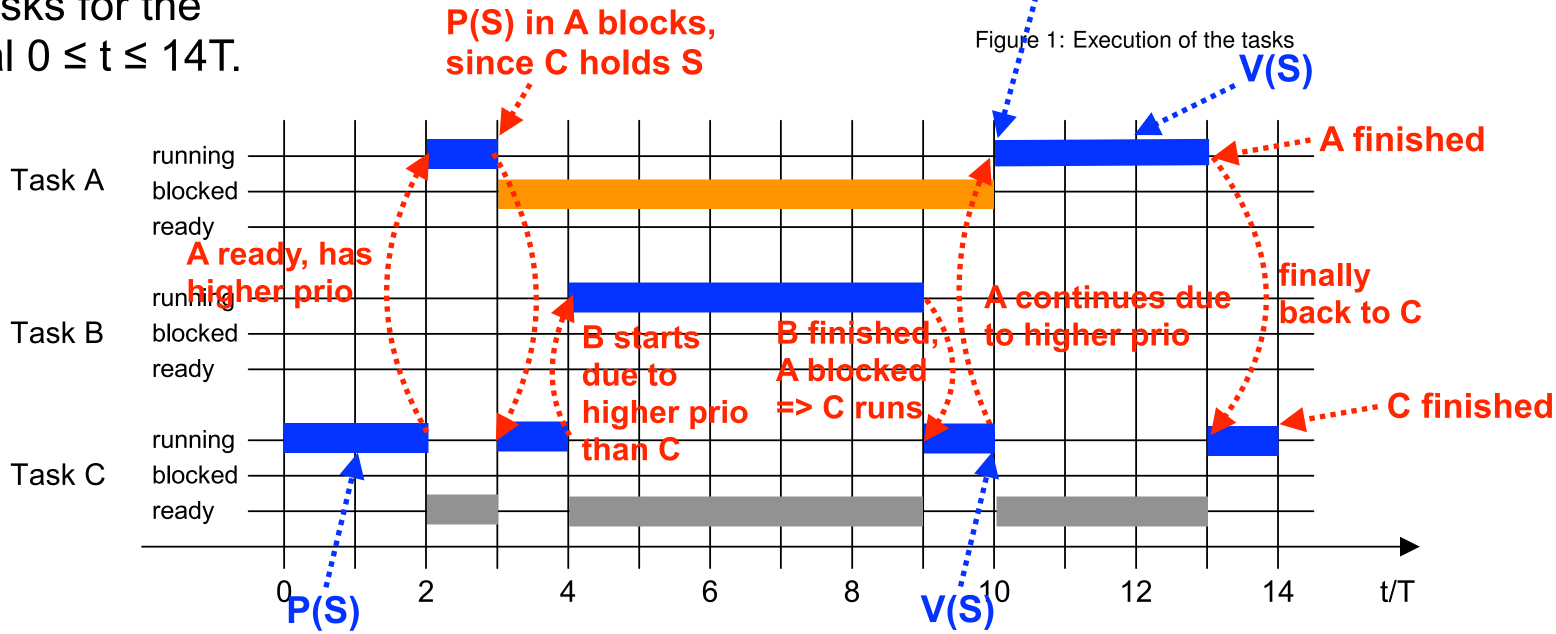


Figure 1: Execution of the tasks



This is the correct version!

Figure 3: Actual execution sequence

6.3 Priority inversion

Priorities A=1 (highest), B=3, C=5 (lowest).
Tasks A and C use a shared resource

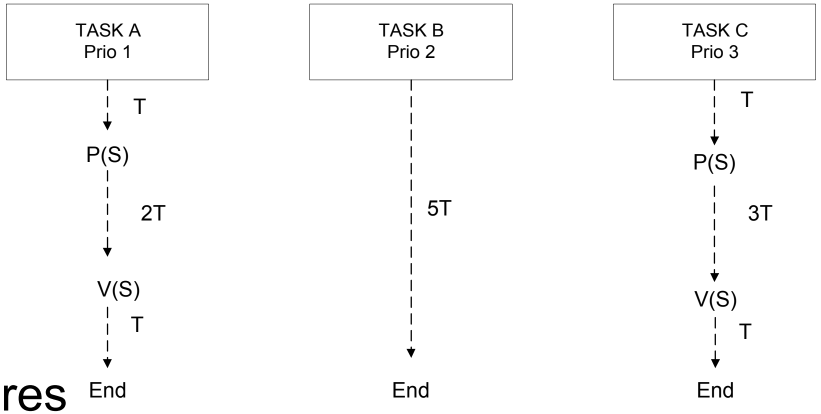


Figure 1: Execution of the tasks

c. Assume there are additional tasks with priorities between those of A and C which do not access the shared resource (no semaphores used). These tasks' individual priorities and execution times are not known. How long would the high priority task A be delayed in the worst case, then?

Additional tasks with priorities between A and C can delay the execution of task C further => the absolute point in time at which C releases semaphore S is delayed

Accordingly, A could be delayed for an arbitrary amount of time