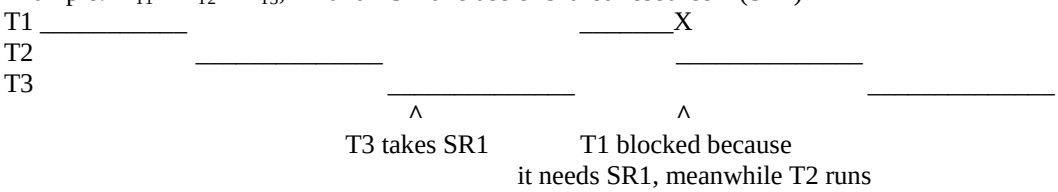
Lecture 19: Priority handling theory

Pre-emption – ability of a high priority task to take control of CPU away from a low priority task
Priority inversion – ability of a low priority task to block a high priority task because of a resource it owns that the high priority task needs
Unbounded priority inversion – occurrence of a high priority task blocked by a low priority task that is itself blocked or starved by another medium priority task\
Example: $P_{T1} > P_{T2} > P_{T3}$; T1 and T3 make use of shared resource 1 (SR1)



Notice that T1 remains block until release of SR1 by T3.
Exercise: show where pre-emption, priority inversion, and unbounded priority inversion are occurring in the above example (hint: they are all occurring at least twice).

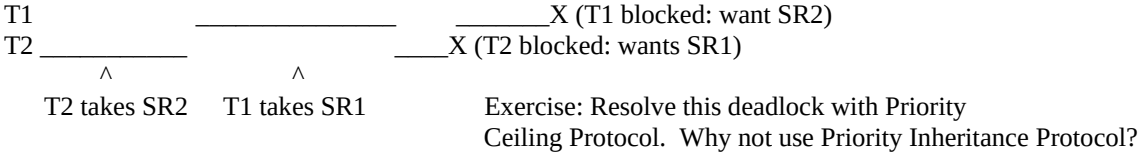
What are possible solutions to the above problem?

-Allow T3 to run at highest priority until SR1 is released. Question: why is this not satisfactory?

-Introduce **Priority Inheritance Protocol** (Sha, Rajkuman, Lehoczky, Carnegie-Mellon U., 1990):
Temporarily raise a lower priority task to the priority of a higher priority task that is currently requesting the same resource that the lower priority task presently owns. The scheduler continues its task time sharing algorithm between multiple tasks. The lower priority task returns to its own priority (or the highest priority of the resources it still owns) once the shared resource it owns is released (called priority disinheritance).
Notes on priority inheritance:

- -Windows NT does not allow this mechanism (ever wonder why your applications seem to hang?)
- -This protocol does NOT avoid deadlock where more than one shared resource is involved
- -Blocking time of a task using m shared resources = sum over m of execution times of all tasks that currently own the m shared resources before release => could be a task's deadline nightmare

-Introduce **Priority Ceiling Protocol:**
A kind of priority inheritance protocol that avoids deadlock by suspending all tasks wanting to own a resource except the task that currently owns the highest priority resource. Resources are assigned priority according to the highest priority task that **may** want (or be locked by) it
Example: $P_{T2} > P_{T1}$ T1 and T2 both make use of shared resources SR1 and SR2



Exercise: Resolve this deadlock with Priority Ceiling Protocol. Why not use Priority Inheritance Protocol?

-Another solution is **Period Transformation**:
Fixed rate-monotomic scheduling is used (the tasks are assigned fixed priority depending on their period; the higher the frequency of a task, the higher its assigned priority). Each task is then divided into smaller running intervals according to T = (e/k, p/k, d/k); where
e = task execution time per period d = task deadline per period
p = task period time k = # time slices that period is divided into by the scheduler.
The idea is that as the period time is sliced into a shorter interval, the task priority is artificially raised so that it runs for e/k at a priority chosen by the scheduler according to the value it uses for k. This way locked shared resources may be freed up sooner. The possibility of deadlock is not precluded however.