嵌入式系统导论

习题3：资源共享

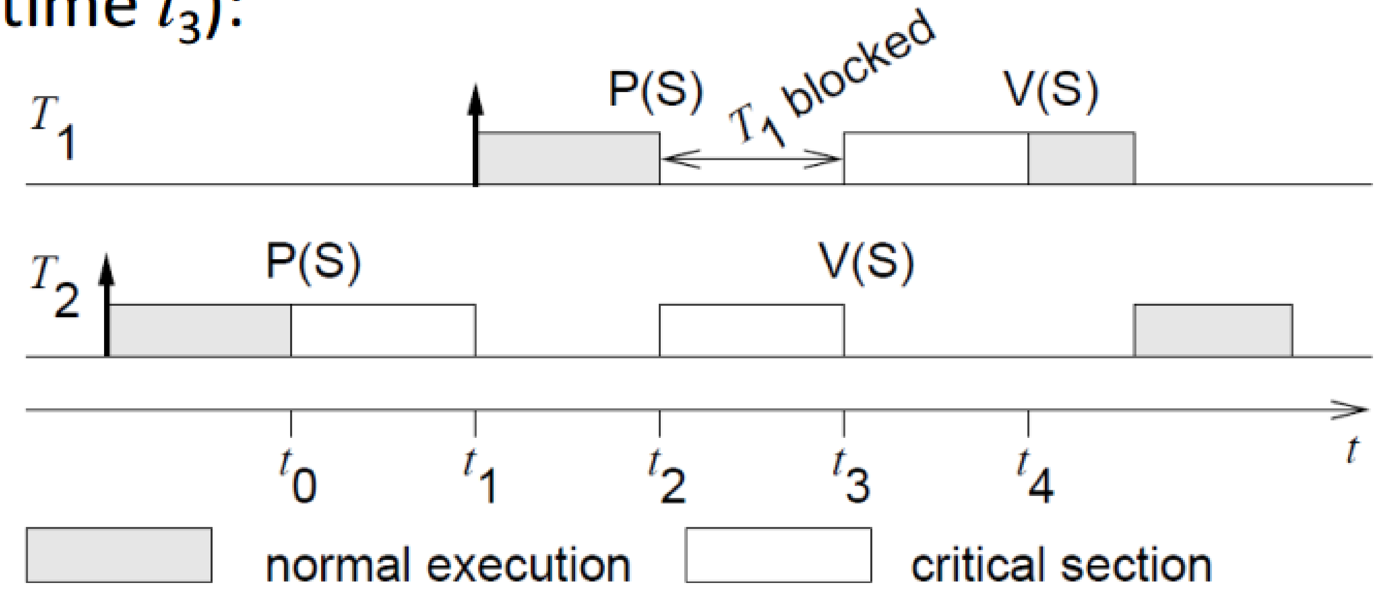
Terms

- Each exclusive resource R_i must be protected by a different semaphore S_i and each critical section operating on a resource must begin with a wait(S_i) primitive and end with a signal(S_i) primitive.

- All tasks blocked on the same resource are kept in a queue associated with the semaphore. When a running task executes a wait on a locked semaphore, it enters a waiting state, until another tasks executes a signal primitive that unlocks the semaphore.

Priority Inversion (1)

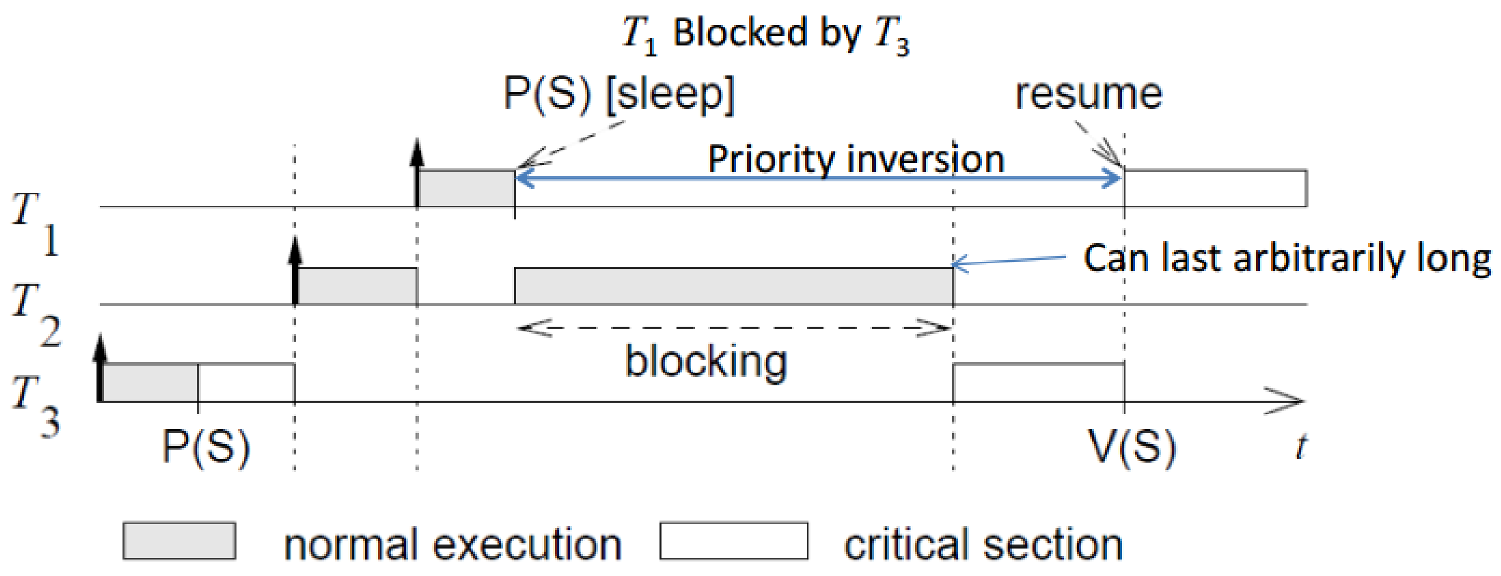
- Priority T_1 assumed to be > than priority of T_2.
- If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (at time t_3):



For 2 tasks: blocking is bounded by the length of the critical section

Priority Inversion (2)

- Blocking with >2 tasks can exceed the length of any critical section
  - Priority of T_1 > priority of T_2 > priority of T_3.
  - T_2 preempts T_3: T_2 can prevent T_3 from releasing the resource.



Resource Access Protocols

- Basic idea: Modify the priority of those tasks that cause blocking. When a task T_i blocks one or more higher priority tasks, it temporarily assumes a higher priority.

- Methods:
  - Priority Inheritance Protocol (PIP), for static priorities
  - Priority Ceiling Protocol (PCP), for static priorities
  - Stack Resource Policy (SRP),
    - For static and dynamic priorities
  - others ...

Priority Inheritance Protocol(PIP)

- Assumptions:
  - n tasks which cooperate through m shared resources; fixed priorities, all critical sections on a resource begin with a wait(S_i) and end with a signal(S_i) operation.

- Basic idea:
  - When a task T_i blocks one or more higher priority tasks, it temporarily assumes (inherits) the highest priority of the blocked tasks.

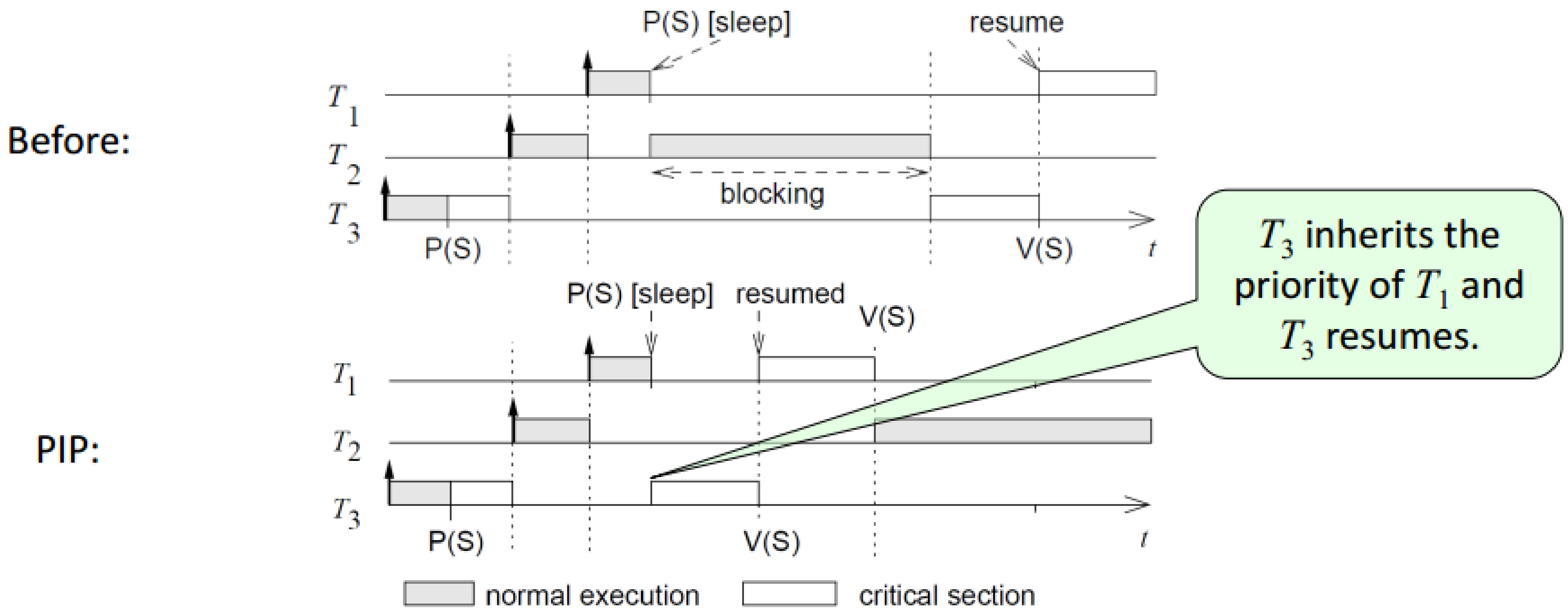
- Terms:
  - We distinguish a fixed nominal priority P_i and an active priority p_i larger or equal to P_i. Tasks T_i, ..., T_n are ordered with respect to nominal priority where T_1 has highest priority. Tasks do not suspend themselves.

PIP: Algorithm

- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- When a task T_i tries to enter a critical section and the resource is blocked by a lower priority task, the task T_i is blocked. Otherwise it enters the critical section.
- When a task T_i is blocked, it transmits its active priority to the task T_k that holds the semaphore. T_k resumes and executes the rest of its critical section with a priority $p_k = p_i$ (it inherits the priority of the highest priority of the tasks blocked by it).
- When T_k exits a critical section, it unlocks the semaphore and the highest priority task blocked on that semaphore is awakened. If no other tasks are blocked by T_k, then p_k is set to P_k, otherwise it is set to the highest priority of the tasks blocked by T_k.
- Priority inheritance is transitive, i.e., if 1 is blocked by 2 and 2 is blocked by 3, then 3 inherits the priority of 1 via 2.

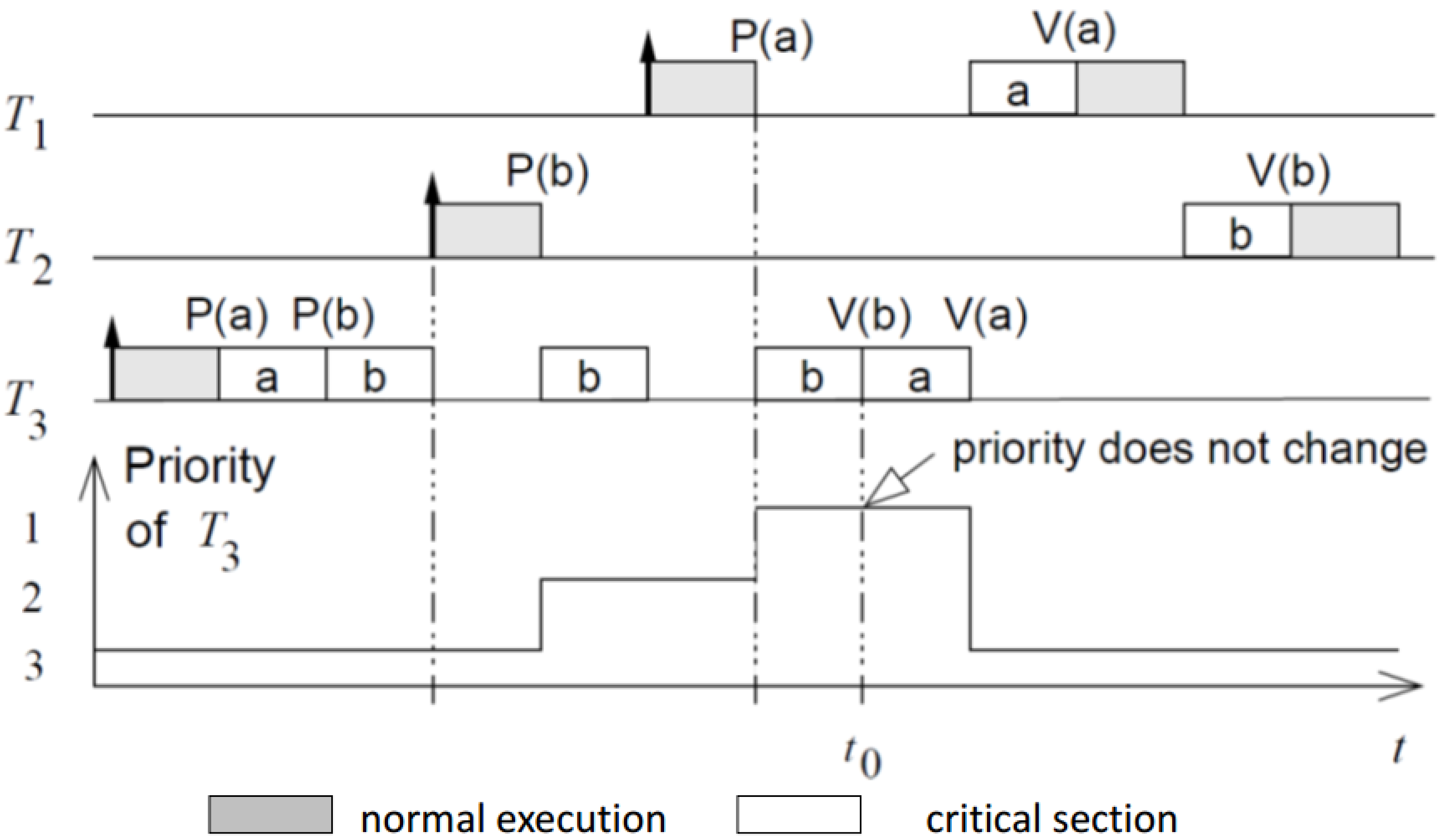
Example:

- How would priority inheritance affect our example with 3 tasks?



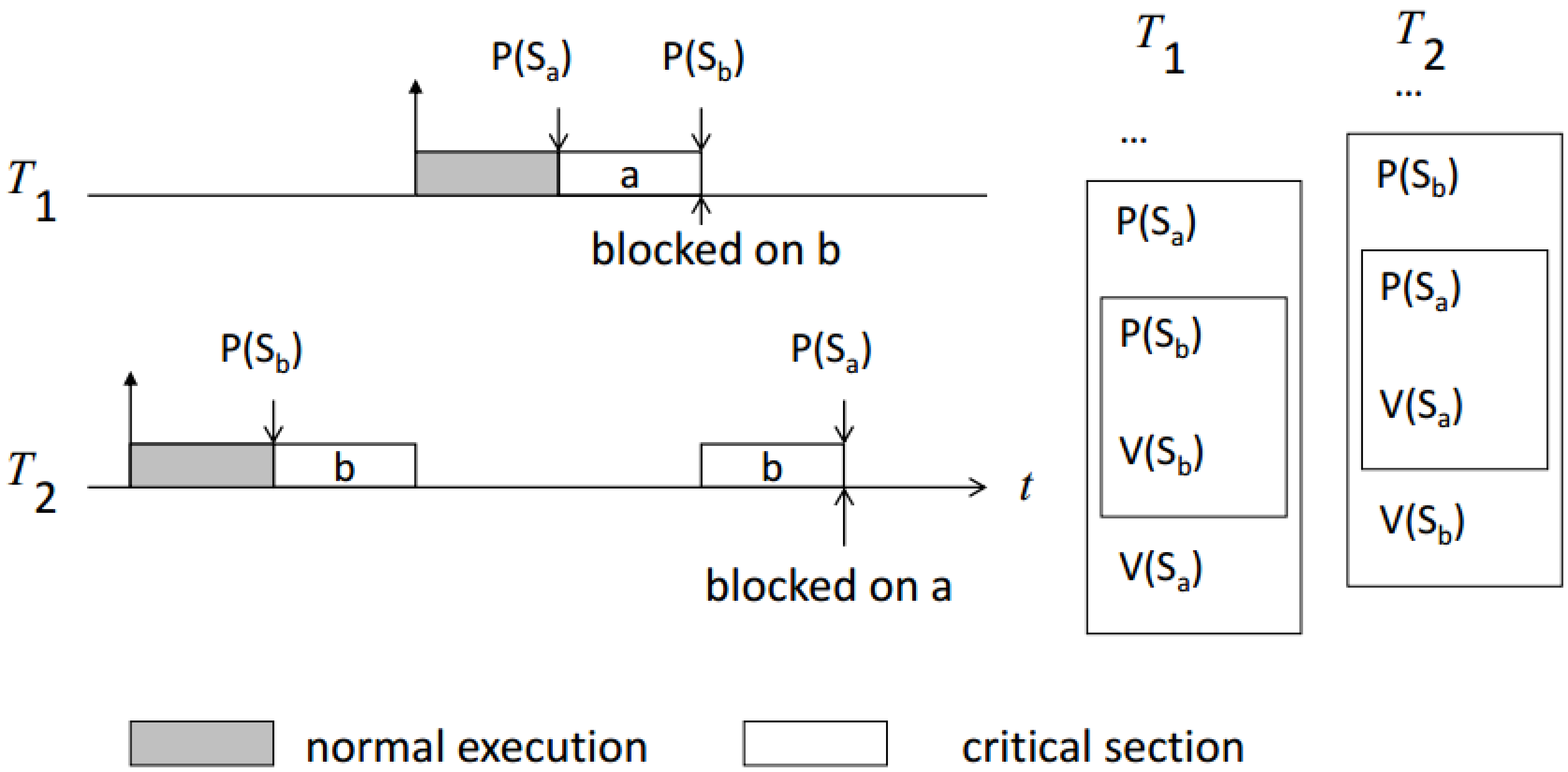
- Direct Blocking: higher-priority task tries to acquire a resource held by a lower-priority task
- Push-through Blocking: medium-priority task is blocked by a lower-priority. Task that has inherited a higher priority from a task it directly blocks

Example:



P(a)
V(a)
T1
a
P(b)
V(b)
T2
b
P(a) P(b)
V(b) V(a)
T3
a
b
b
b
a
priority does not change
Priority of T3
1
2
3
t0
t
normal execution
critical section

Deadlock is Possible



- Problem exists also when no priority inheritance is used

PIP--->PCP

- The Priority Inheritance Protocol (PIP)
  - does not prevent deadlocks
  - can lead to chained blocking
    - (Several lower priority tasks can block a higher priority task)
  - and has inherent static priorities of tasks

→The Priority Ceiling Protocol (PCP)
  - avoids multiple blocking
  - guarantees that, once a task has entered a critical section, it cannot be blocked by lower priority tasks until its completion.

Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

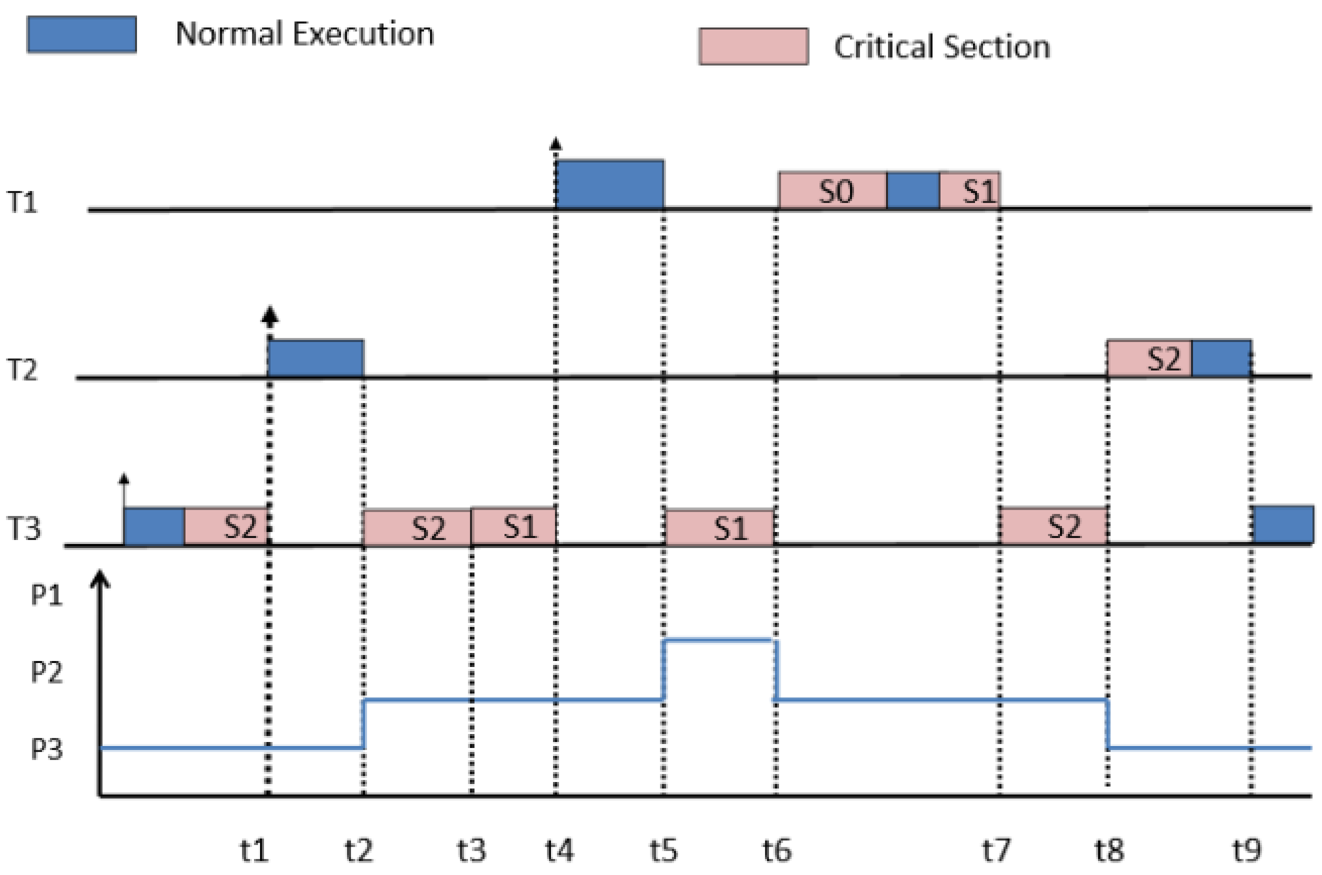
PCP

- A task is not allowed to enter a critical section if there are already locked semaphores which could block it eventually
- Hence, once a task enters a critical section, it can not be blocked by lower priority tasks until its completion.
- This is achieved by assigning priority ceiling.
- Each semaphore S_k is assigned a priority ceiling $C(S_k)$. It is the priority of the highest priority task that can lock S_k.
  This is a static value.

A job j executing $P(S)$ is granted access to S if the priority of j is strictly higher than the ceiling of any semaphore locked by a job other than j. Otherwise, j becomes blocked and S is not allocated to j.

.pdf

Example：



C(S0=P1) C(S1=P1) C(S2=P2)

Source: http://www.ida.liu.se/ unmbo/RTS_CUGS_files/Lecture3.pdf

作业

- 11月30日晚12:00之前提交EX1到ftp相应文件夹