Embedded Systems Design and Modeling

Chapter 12 Part 2
Scheduling Anomalies

Outline

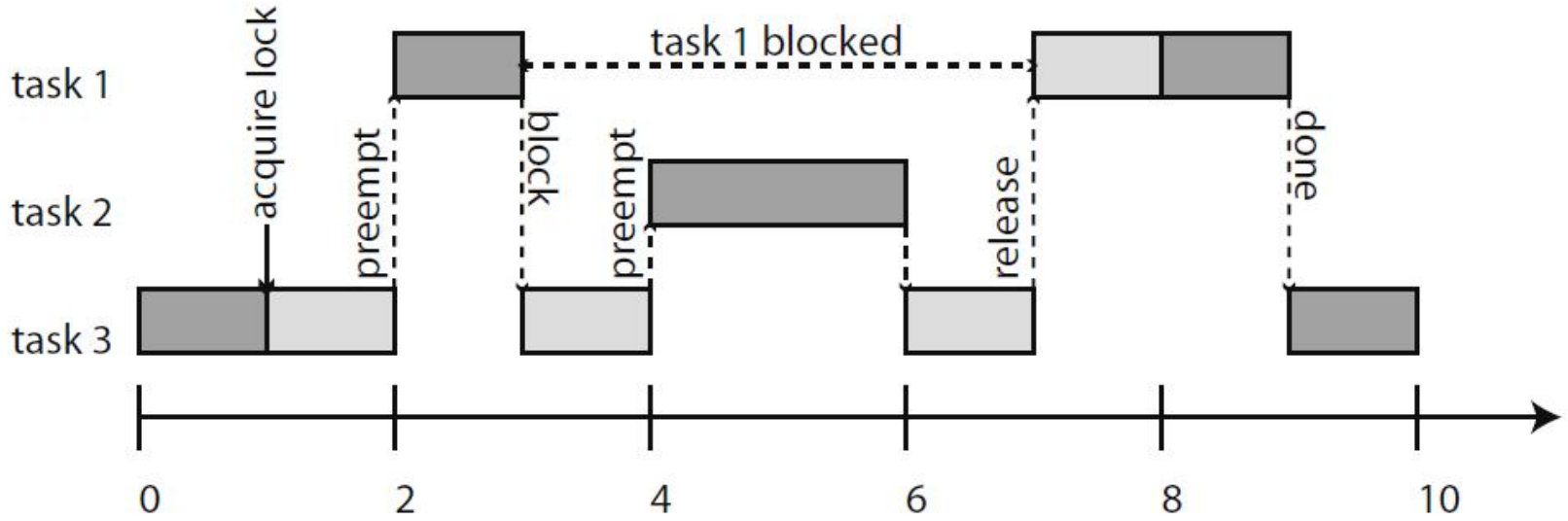
- Scheduling anomalies definition
- Scheduling anomalies cases
  - Caused by mutual exclusion:
    1. Priority inversion
       - Solution: priority inheritance
       - Real case: Mars Pathfinder
    2. Deadlock
       - Solution: priority ceiling protocol
  - Seen in multiprocessor environments:
    3. Richard's anomalies (non-monotonic, brittle)
    4. Again, mutual exclusion issues

Basics

- Scheduling anomalies definition: when a schedule shows unexpected and counterintuitive behaviors under special circumstances
- Often (but not always) caused by mutual exclusion locks:
  - Mutexes are needed to control accesses to shared resources
  - They can also complicate the scheduling and cause anomalies

Anomalies Caused by Mutexes

1. Priority inversion:
   - A high priority task is ready to execute but is blocked by a lower priority task that holds a lock it needs
   - It can be bounded or unbounded
   - Example: task 1 has highest priority, task 3 lowest. Task 3 acquires a lock on a shared resource. It gets preempted by task 1, which then tries to acquire the lock and blocks. Task 2 preempts task 3 at time 4, keeping the higher priority task 1 blocked for a large amount of time.



Real Example: Mars Pathfinder

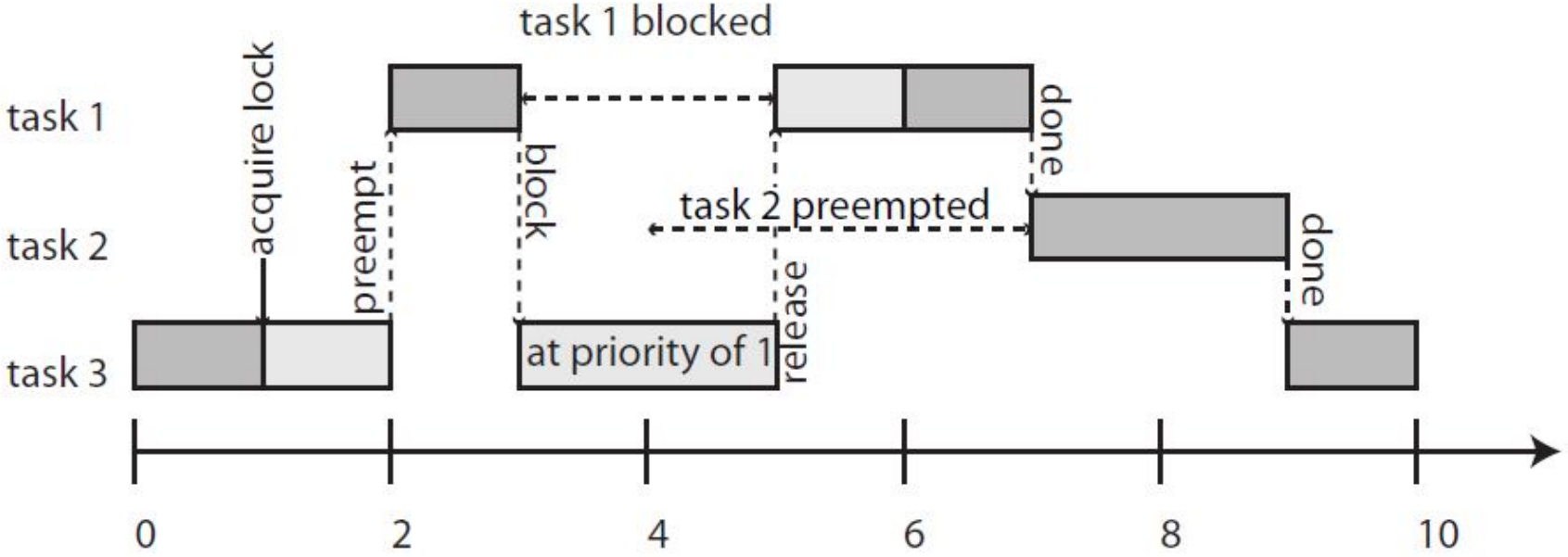
- The Mars Rover Pathfinder landed on Mars on July 4th, 1997.
- After a few days the Pathfinder began missing deadlines, losing data, and self-resets.
- The problem was diagnosed on the ground as priority inversion.
- Two tasks were critical for controlling communication on Pathfinder's communication bus: the scheduler task (bc_sched) and the distribution task (bc_dist).
- Each of these tasks checked every 125ms to be sure that the other had run successfully.

Pathfinder Story (Continued)

- bc_dist was blocked by a much lower priority meteorological science task (ASI/MET).
- ASI/MET was preempted by several medium priority processes such as accelerometers and radar altimeters.
- bc_sched started and discovered that bc_dist had not completed. Under these circumstances, bc_sched reacted by reinitializing the lander's hardware and software and terminating all ground command activities.
- NASA and WindRiver reproduced the failure on Earth and discovered the priority inversion.

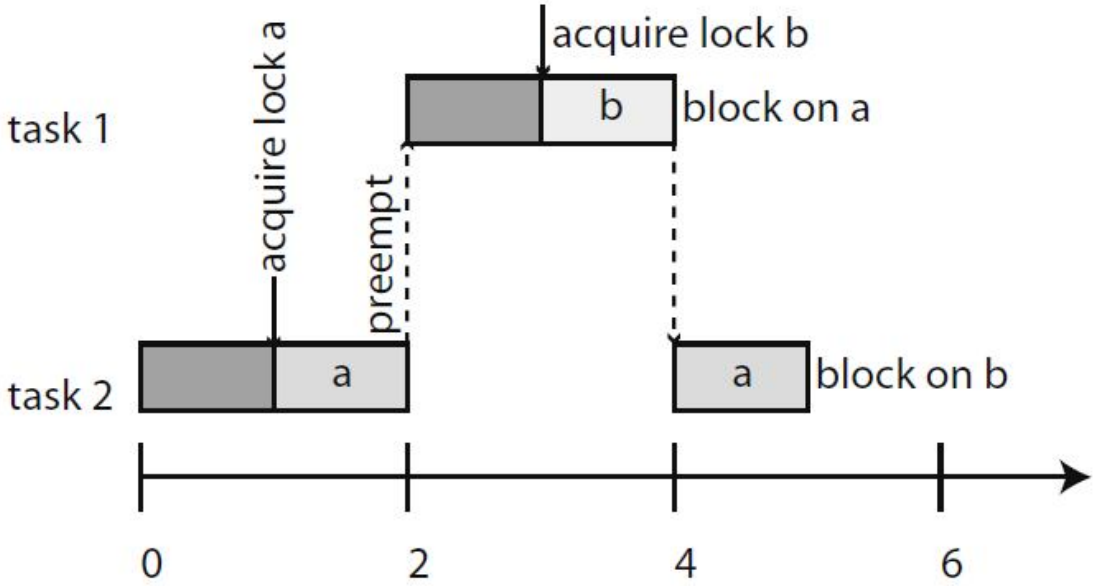
Priority Inversion Solution

- Priority inheritance: when a task blocks the execution of another higher priority task, it executes at the highest priority of all of the tasks it blocks.
- Example: Task 1 has highest priority, task 3 lowest. Task 3 acquires a lock on a shared object, entering a critical section. It gets preempted by task 1, which then tries to acquire the lock and blocks. Task 3 inherits the priority of task 1, preventing preemption by task 2.



Anomalies Caused by Mutexes

2. Deadlock:
   - The lower priority task starts first and acquires lock a, then gets preempted by the higher priority task, which acquires lock b and then blocks trying to acquire lock a. The lower priority task then blocks trying to acquire lock b, and no further progress is possible.

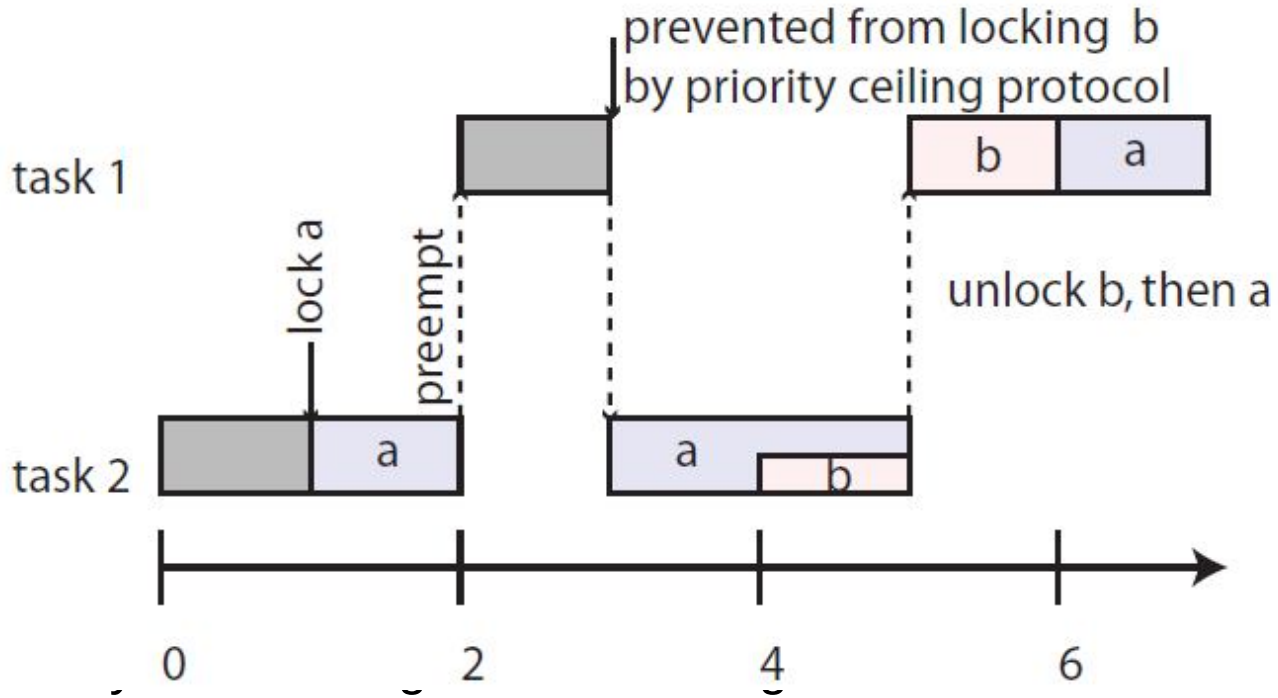


Deadlock Solution

- Priority Ceiling Protocol: Every lock or semaphore is assigned a priority ceiling equal to the priority of the highest-priority task that can potentially lock it.
- A task can acquire a lock only if the task's priority is strictly higher than the priority ceilings of all locks currently held by other tasks.
- This prevents deadlocks by blocking a task to acquire a lock held by other tasks.
- There are extensions supporting dynamic priorities and dynamic creations of locks.

Priority Ceiling Protocol Example

- Locks a and b have priority ceilings equal to the priority of task 1. At time 3, task 1 attempts to lock b, but it cannot because task 2 currently holds lock a, which has priority ceiling equal to the priority of task 1.



Partial Summary

- To successfully share resources, a system needs two properties:
  - Freedom from mutual deadlock
  - Freedom from unbounded priority inversion
    - Is bounded priority inversion acceptable?
- The combination of priority inheritance protocol and the priority ceiling protocol guarantee the above properties.

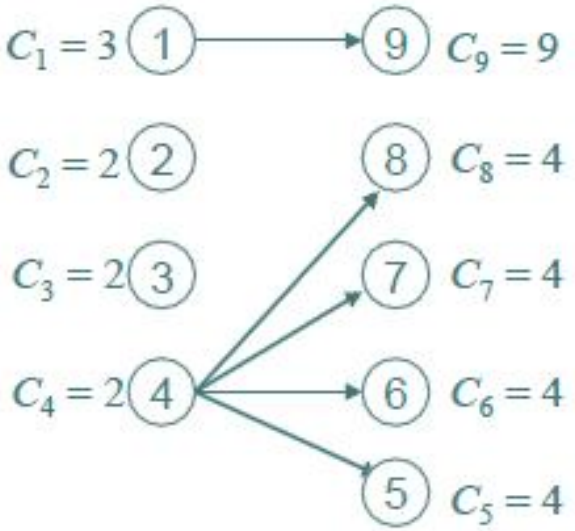
Anomalies in Multiprocessors

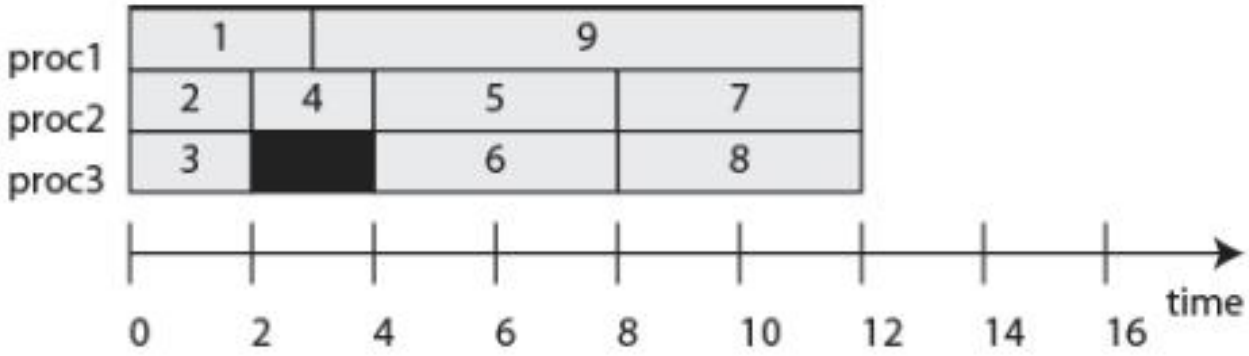
3. Known as Richard's anomalies

- Theorem: If a task set with fixed priorities, execution times, and precedence constraints is scheduled according to priorities on a fixed number of processors, then increasing the number of processors, reducing execution times, or weakening precedence constraints may not improve the schedule length and may even make it longer.

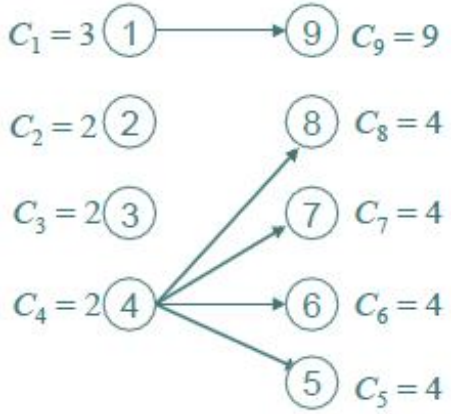
Richard's Anomalies

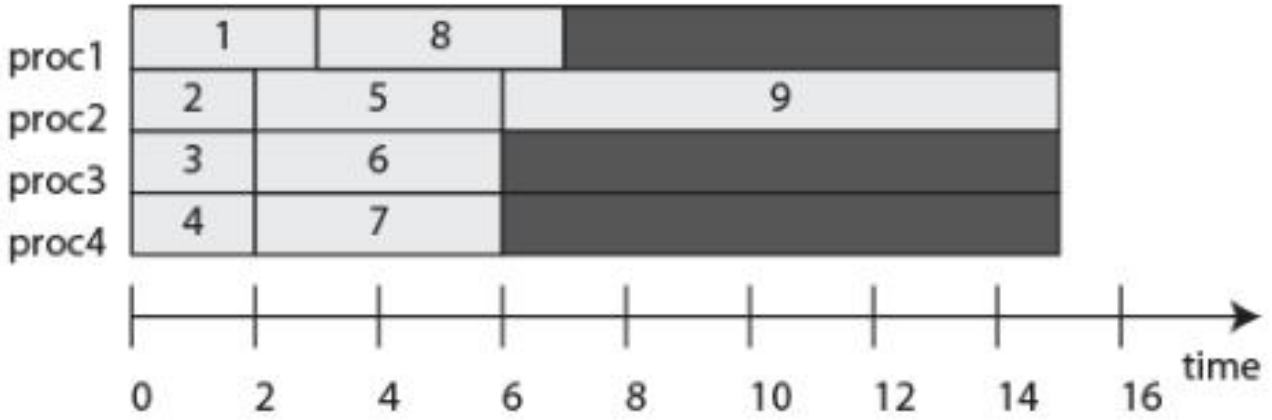
- Consider 9 tasks with the following precedence graph and execution times.
- Assume lower numbered tasks have higher priority than higher numbered tasks.
- The priority-based three-processor schedule:





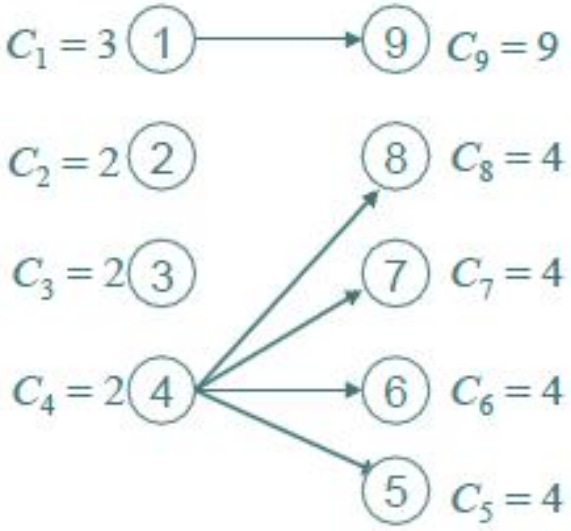
Adding One More Processor

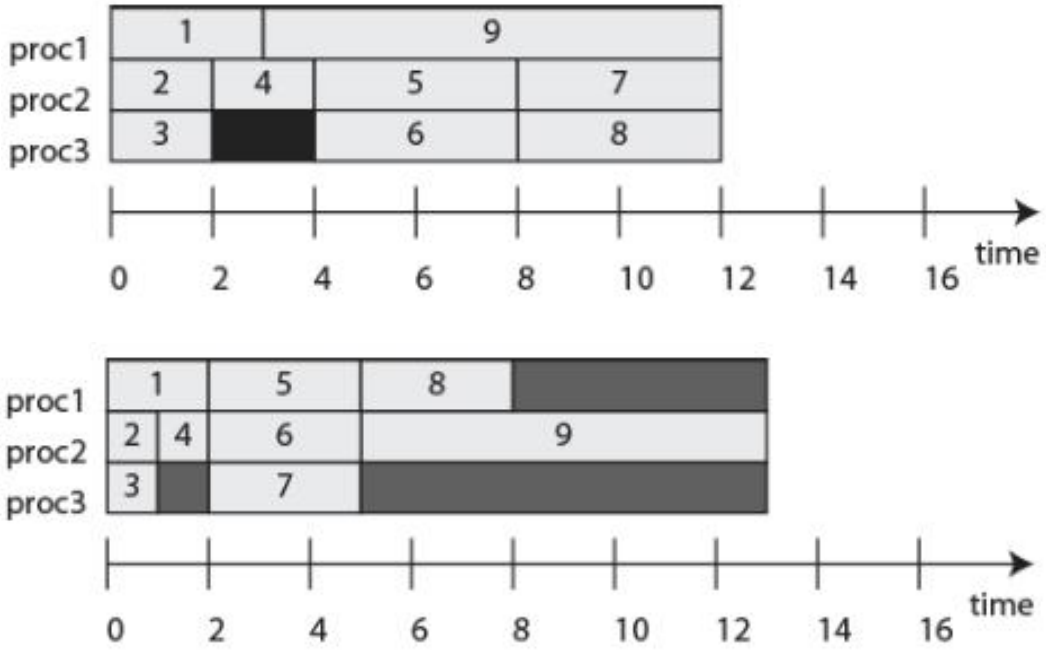




- The four-processor schedule takes longer!
- Priority-based scheduling is greedy. A smarter scheduler for this example could hold off scheduling 5, 6, or 7, leaving a processor idle for one time unit.
- But if tasks arrive only after their predecessor completes, then greedy scheduling may be the only practical option.

Reducing Execution Times By 1

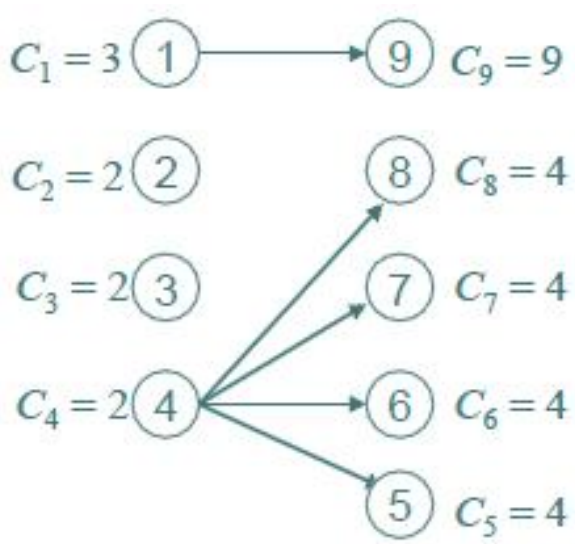


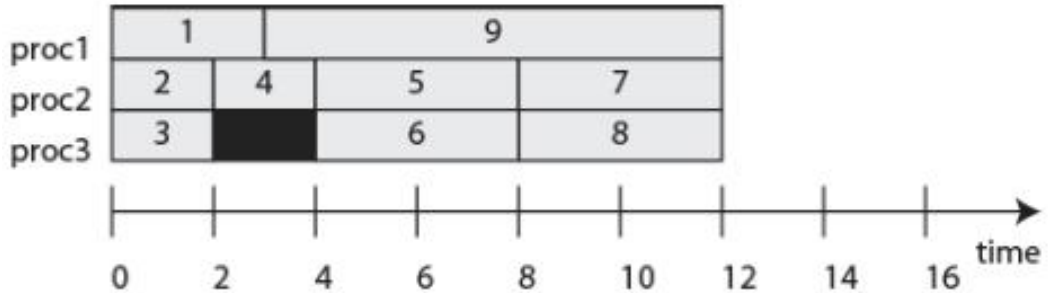


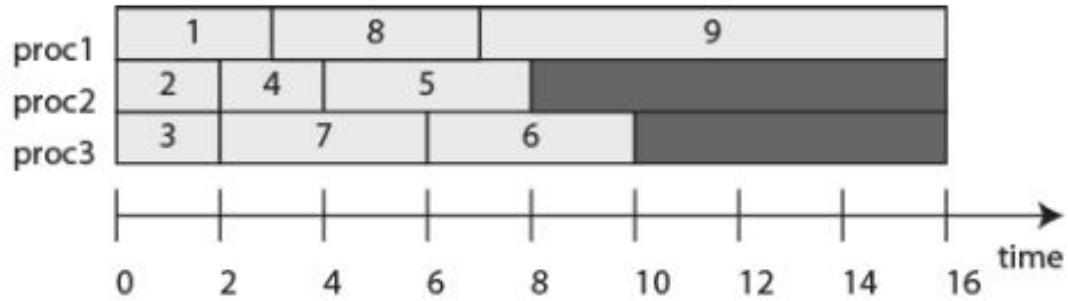
- Reducing the computation times by 1 also results in a longer execution time!
- Again, this is caused by the greedy approach due to dynamic scheduling.

Weakening Precedence Constraints

- Removing the precedence constraints (4,8) and (4,7):



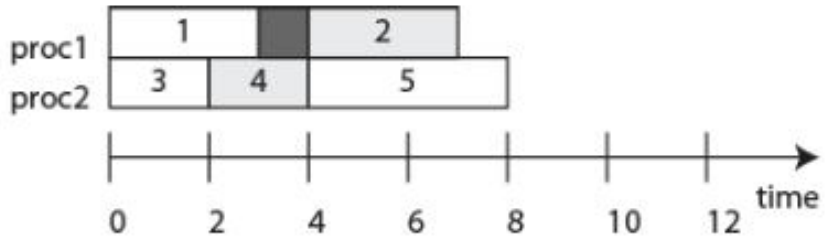


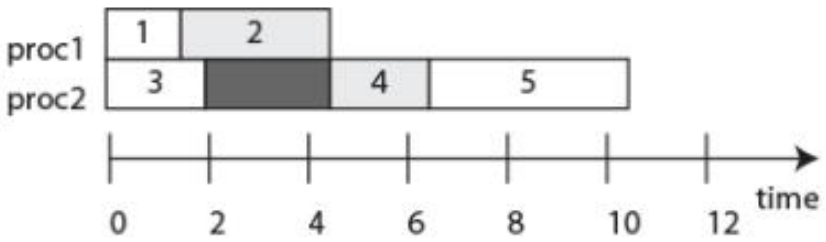


- Weakening precedence constraints can also result in a longer schedule!

Anomalies in Multiprocessors

4. Anomalies caused by mutexes:
   - Assume tasks 2 and 4 share the same resource in exclusive mode, and tasks are statically allocated to processors. Then if the execution time of task 1 is reduced, the schedule length increases.





Conclusions

- In general, all scheduling algorithms suffer from possible anomalies.
- Timing behavior under all known task scheduling strategies is brittle:
  - Small changes can have big and unexpected consequences.
- And is non-monotonic:
  - Improvements in performance at a local level can result in degradations in performance at a global level,
- Since execution times are hard to predict, anomalies can result in system failures.
- Chapter 12 homework: 1 thru 5 for 1403/3/8