Lab 3 for uC/OS-II: Ceiling Priority Protocol

Prof. Li-Pin Chang
ESSLab@NCTU

Objective

- To implement Ceiling Priority Protocol for ucOS’s mutex locks

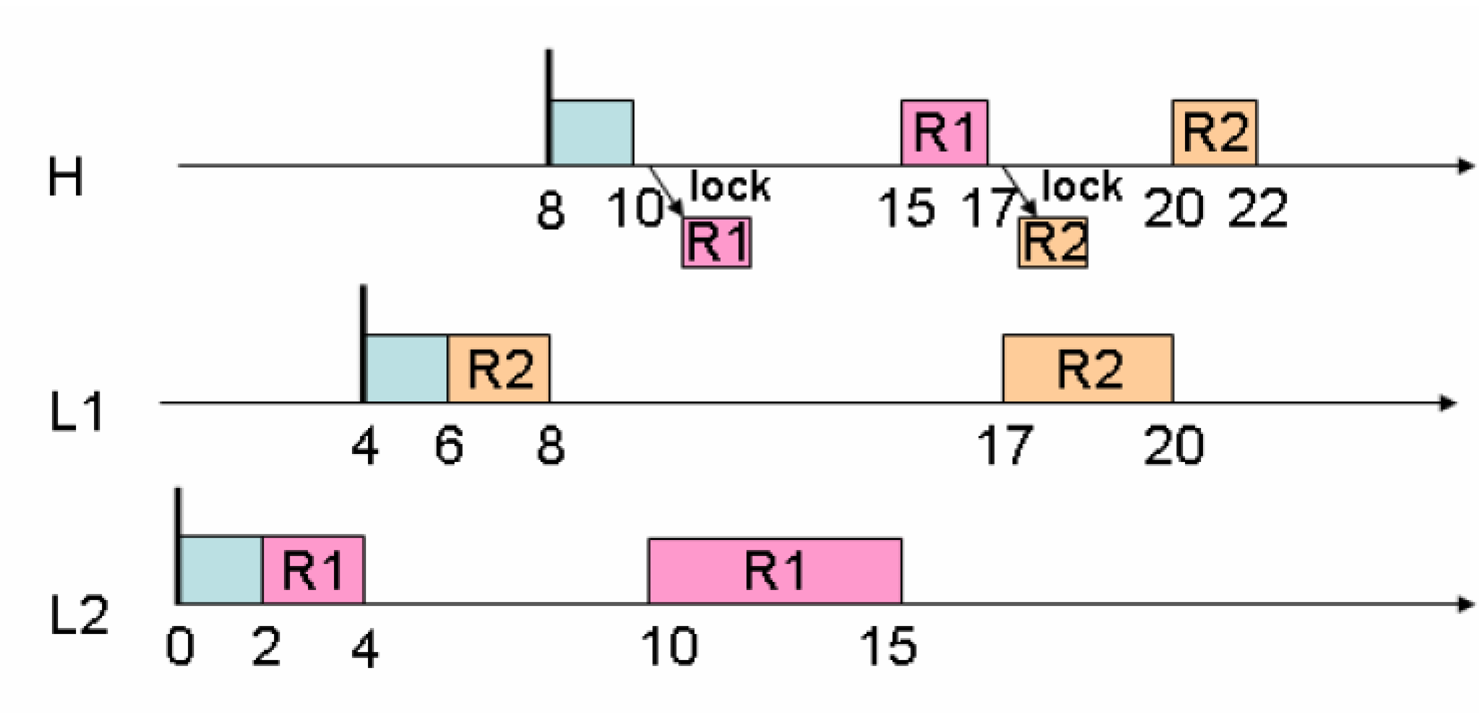
ucOS Mutex Locks

- Mutex lock is assigned to a priority upon creation
  - Its priority is higher than the priorities of all the tasks sharing the mutex lock
  - E.g., the lock's priority will be 2 if T_3 and T_4 share the lock
  - When a task blocks another task via a mutex lock, the task inherits the priority of the lock

Disadvantages of PIP

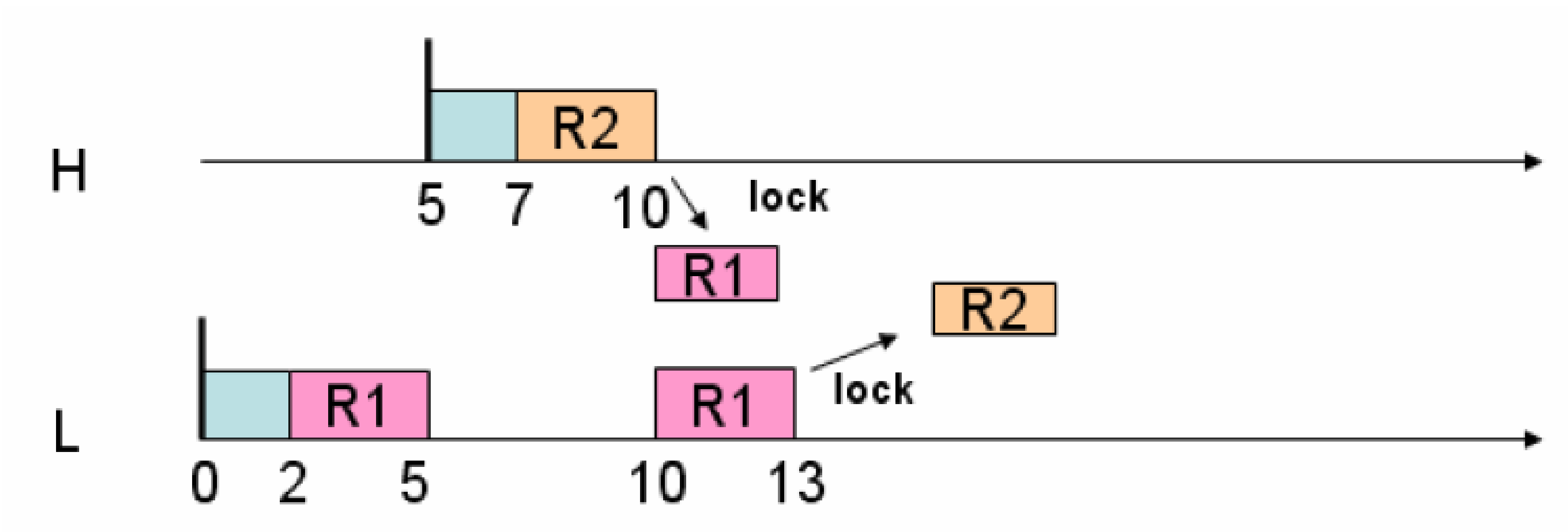
- The “PIP” avoids uncontrolled priority inversion, but it has two disadvantages
  - A high priority task can be blocked multiple times
  - Deadlocks are possible

Scenario 1: Multiple blocking in ucOS2 PIP



Task H is in turn blocked by task L1 and task L2

Scenario 2: Deadlock in uCOS-2 PIP

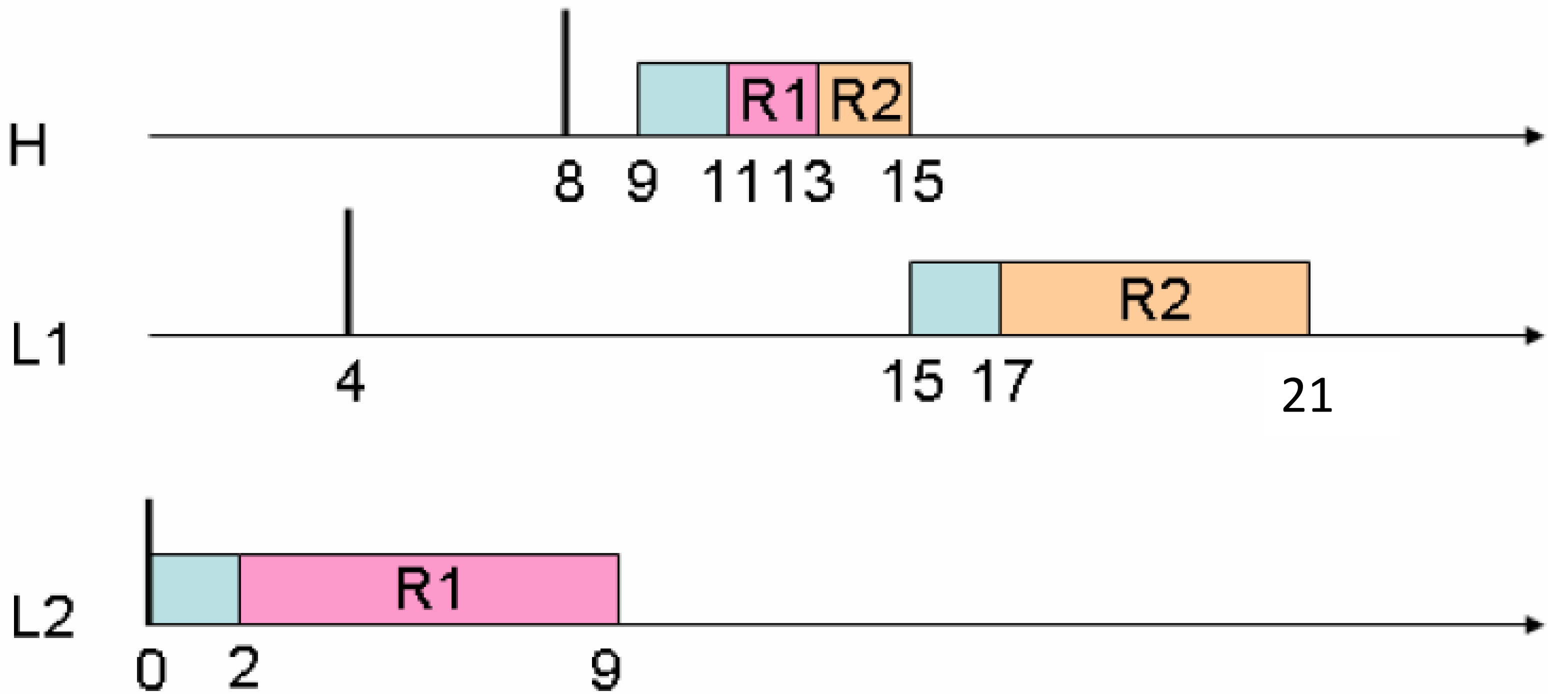


A deadlock

Ceiling Priority Protocol

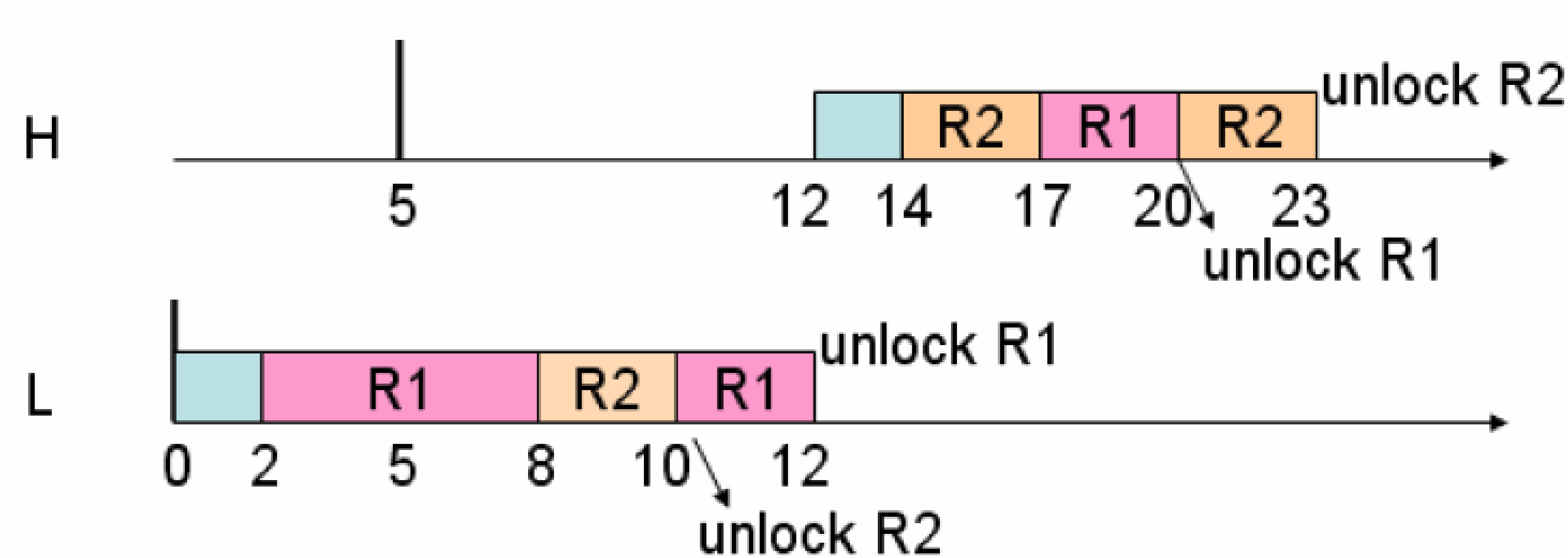
- Highest Locker’s Priority Protocol
- The priority of the mutex lock is higher than the priorities of all the tasks sharing the lock
- Differently, when a task acquires a mutex lock, the task’s priority boosts to the priority of the lock

S1 CPP: Removing Multiple Blockings



The result of applying CPP

S2 CPP: Avoiding Deadlocks



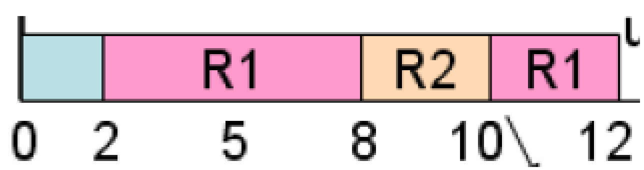
Deadlocks are avoided by using CPP

Implementation

- Reuse your code of Lab 1 (do not use EDF)
- Modify the two functions
  - OSMutexPend
    - Boost the locker’s priority to the mutex priority
  - OSMutexPost
    - Restore the original priority of the locker
- Do not use OSTaskChangePrio to boost tasks’ priorities
  - It calls OS_Sched() and results in unexpected behaviors

Implementation

- All tasks should add proper OSTimeDly() at their beginning to emulate their arrival times
- Emulate durations of CPU execution and resource use with your code from Lab 0
  - 2 ticks → lock R1→ 6 ticks → lock R2 → 2 ticks → unlock R2 → 2 ticks → unlock R1



Output

- Similar to those in prior labs, but add lock/unlock events
- Output the results of using CPP for Scenarios 1 and 2

Output Example of S1

Priority initialization:

R1: 1

R2: 2

Task1: 3

Task2: 4

Task3: 5

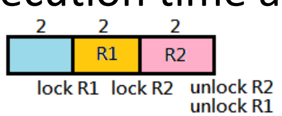
Task arrival time:

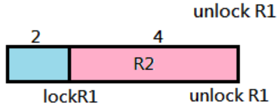
Task1: 8

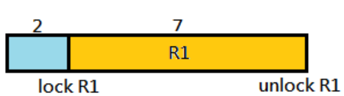
Task2: 4

Task3: 0

Task execution time and resource used:

Task1: 

Task2: 

Task3: 

20	lock	R1	(Prio=5 changes	to=1)
90	unlock	R1	(Prio=1 changes	to=5)
90	complete		5	3
110	lock	R1	(Prio=3 changes	to=1)
130	lock	R2	(Prio=1 changes	to=1)
150	unlock	R2	(Prio=1 changes	to=1)
150	unlock	R1	(Prio=1 changes	to=3)
150	complete		3	4
170	lock	R2	(Prio=4 changes	to=2)
210	unlock	R2	(Prio=2 changes	to=4)
210	complete		4	19