# 17. Advanced Real-Time Concepts

18-349: Embedded Real-Time Systems

**Priya Narasimhan**
Electrical & Computer Engineering
Carnegie Mellon University

http://www.ece.cmu.edu/~ee349

Carnegie Mellon

Electrical & Computer ENGINEERING

## Previous Lecture on Real-Time

- Example real-time systems
  - Simple control systems, multi-rate control systems, hierarchical control systems, signal processing systems
- Terminology
- Priority inversion
- Scheduling algorithms
  - Rate monotonic analysis
- Provides an engineering basis for designing real-time systems

2

## Today's Lecture

- How to decide if a set of tasks is schedulable
  - Utilization bound test
  - RT test
- Synchronization in real-time systems
  - Unbounded priority inversion
  - Basic Priority Inheritance Protocol
  - Priority Ceiling Protocol

3

## Task: {C, T}

- Periodic task
  - Initiated at fixed intervals
  - Must finish before start of next cycle
- Specifying a task $\{C_i, T_i\}$
  - $C_i$ = worst-case compute time (execution time) for task $\tau_i$
  - $T_i$ = period of task $\tau_i$
- Individual task's CPU utilization: $U_i = \frac{C_i}{T_i}$
- Total CPU utilization for a set of tasks

$$U = U_1 + U_2 + \ldots + U_n$$

4

## Schedulability (Recap of Piazza, Lab 4)

- A set of tasks is schedulable if all tasks are guaranteed to meet their deadlines
- Utilization bound (UB) test says that a task set is schedulable if its total utilization is less than a bound called the **Liu & Leyland bound**
  - More complex formulas provide better bounds
  - Application of a theorem proved by Liu and Leyland
    - "*Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*", 1973
    - Must-know, seminal paper in the area of real-time systems

5

## Schedulability: Formal Assumptions

- The utilization bound (UB) test works under a number of assumptions
  - The processor always executes the highest priority task
  - Task priorities are assigned according to rate monotonic policy
  - Tasks do not synchronize with each other
  - Each task's deadline is at the end of its period
  - Tasks do not suspend themselves in the middle of computations
  - Context switches between tasks take zero time
- For a harmonic task set
  - Each task's period is a multiple of all higher-frequency tasks
  - Utilization bound is 1.0 for all task sets

6

## Basic Schedulability: UB Test

- Utilization bound (UB) test: a set of *n* independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1)$$

| | | |
|---|---|---|
| U(1) = 1.0 | U(4) = 0.756 | U(7) = 0.728 |
| U(2) = 0.828 | U(5) = 0.743 | U(8) = 0.724 |
| U(3) = 0.779 | U(6) = 0.734 | U(9) = 0.720 |

- As the number of tasks goes to infinity, the bound approaches ln(2) = 0.693
  - Thus, any number of independent periodic tasks will meet their deadlines if the total system utilization is under 69%

7

## Example Problem: Applying UB Test

| | C | T | U |
|---|---|---|---|
| Task $t_1$: | 20 | 100 | 0.200 |
| Task $t_2$: | 40 | 150 | 0.267 |
| Task $t_3$: | 100 | 350 | 0.286 |

- *Left-hand side*
  - $U_1 + U_2 + U_3$ = total utilization for 3 tasks = .200 + .267 + .286 = .753
- *Right-hand side*
  - U(3) = .779

➢ Apply the UB test: $U_1 + U_2 + U_3 < U(3)$
➢ The periodic tasks in the example are schedulable according to the UB test
➢ Also, 24.7% of the CPU capacity is available for tasks that have no deadline

8

## Drawing a Timeline

- Timelines show one possible execution schedule and provide a graphical view of schedule
- Use the following conventions
  - Arrange tasks in rate monotonic order, highest frequency at the top
  - Assume Liu and Leyland "worst-case" phasing, where all tasks start at time t=0 (*unless otherwise mentioned*)
  - Execution time for t1 is plotted on its line
  - Execution time for t2 is then plotted on its line, accommodating preemption from t1's execution; then this process is repeated for remaining tasks
  - If any task is preempted, its execution time block is divided with a hole in the middle representing the preemption (e.g. t3)

9

## Toward a More Precise Test

- UB test has three possible outcomes:

| | | |
|---|---|---|
| $0 < U \leq U(n)$ | → | Success |
| $U(n) < U < 1.00$ | → | Inconclusive |
| $1.00 < U$ | → | Overload |

- UB test is conservative
  - More precise test can be applied

10

# Response-Time Test (RT Test)

- Theorem
  - For a set of $n$ independent, periodic tasks, if each task meets its *first* deadline, with *worst-case task phasing*, the deadline will *always be met* (again, rate monotonic scheduling is assumed)

- Let $a_n$ = response time of task $i$ where

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j \qquad \text{where } a_0 = \sum_{j=1}^{i} C_j$$

  - Test terminates/converges when $a_{n+1} = a_n$
  - Task $i$ is schedulable if its response time is before its deadline: $a_n < T_i$

- This test must be **repeated for every task $t_i$ if required**
  - The value of $i$ will change depending upon the task you are looking at
- Stop the test once current iteration yields a value of $a_{n+1}$ beyond the deadline for that task (else, you may never terminate).
- The square parentheses represent a 'ceiling function'

# Example: Applying RT Test – I

- Is the following task set schedulable? Assume T=D as before
  - Note that this is the same as the previous task set, except that $C_1$ is now 40s

| | C | T | U |
|---|---|---|---|
| ✓Task $\tau_1$: | 40 | 100 | 0.4 |
| ✓Task $\tau_2$: | 40 | 150 | 0.267 |
| ? Task $\tau_3$: | 100 | 350 | 0.286 |

- Utilization of first two tasks: 0.667 < U(2) = 0.828
  - First two tasks are schedulable by UB test

- Utilization of all three tasks: 0.953 > U(3) = 0.779
  - UB test is inconclusive
  - Need to apply RT test to the third task

12

## Example: Applying RT Test – II

- Use RT test to determine if $\tau_3$ (i.e., i = 3) meets its first deadline
- Compute the response time iterations, i.e., $a_0$, $a_1$, .....
- Wait for the test to converge and then compare with the deadline $T_3$

$$a_0 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 40 + 40 + 100 = 180$$

$$a_1 = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_0}{T_j} \right\rceil C_j = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0}{T_j} \right\rceil C_j$$

$$= 100 + \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) = 100 + 80 + 80 = 260$$

13

## Example: Applying the RT Test – III

$$a_2 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_1}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) = 300$$

$$a_3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_2}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

$a_3 = a_2 = 300$ Done! Test has converged

- Now, compare with deadline
- Task $\tau_3$ is schedulable using RT test

$$\boldsymbol{a_2 = 300 < T_3 = 350}$$

14

## Underlying Assumptions

- UB and RT tests share the same limitations/assumptions

- All tasks run on a single processor
- All tasks are periodic and noninteracting
- Deadlines are always at the end of the period
- There are no interrupts
- Rate-monotonic priorities are assigned
- There is zero context-switch overhead
- Tasks do not suspend themselves

15

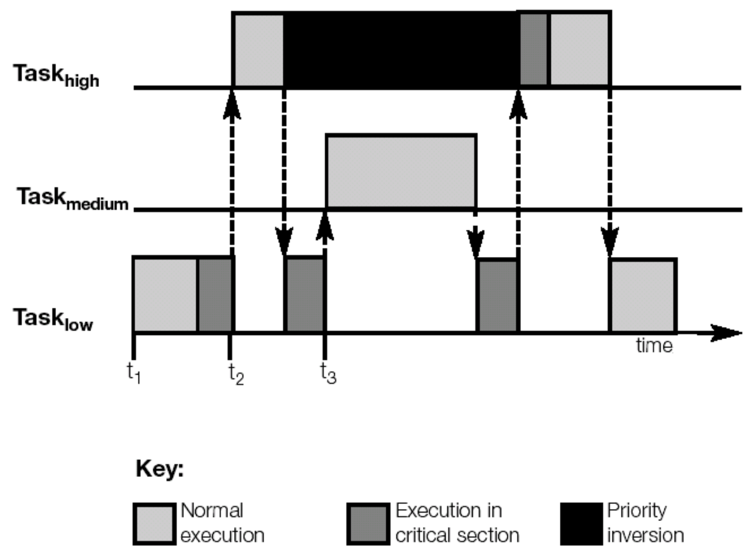
## RECAP: Priority Inversion

- Delay to a task's execution caused by interference from lower priority tasks is known as priority inversion
- Priority inversion is modeled by *blocking time*
- Identifying and evaluating the effect of sources of priority inversion is important in schedulability analysis

- Sources of priority inversion
  - Synchronization and mutual exclusion
  - Non-preemptable regions of code
  - FIFO (first-in-first-out) queues
  - Anything else?

16

## RECAP: Priority Inversion




17

## RECAP: Priority Inversion

- Recall that task schedulability is affected by
  - Preemption: two types of preemption
    - Can occur several times per period
    - Can only occur once per period
  - Execution: once per period
  - Blocking: at most once per period for each source
- The schedulability formulas are modified to add a “blocking” or “priority inversion” term to account for inversion effects.

18

## UB Test with Blocking

$$f_i = \sum_{j \in H_n} \frac{C_j}{T_j} + \frac{1}{T_i} \sum_{k \in H_1} C_k + \frac{C_i}{T_i} + \frac{B_i}{T_i}$$

- $H_n$ Preemption (can hit *n* times)
- $H_1$ Preemption (can hit once)
- Execution
- Blocking

You will not be tested on this formula
*[And there was much rejoicing in the classroom]*

19

## RT Test with Blocking

- Blocking is also included in the RT test:

$$a_{n+1} = B_i + C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j$$

$$\text{where } a_0 = B_i + \sum_{j=1}^{i} C_j$$

- Perform test as before, adding in blocking effect.

You will not be tested on this formula
*[And there was way more rejoicing and spontaneous applause]*

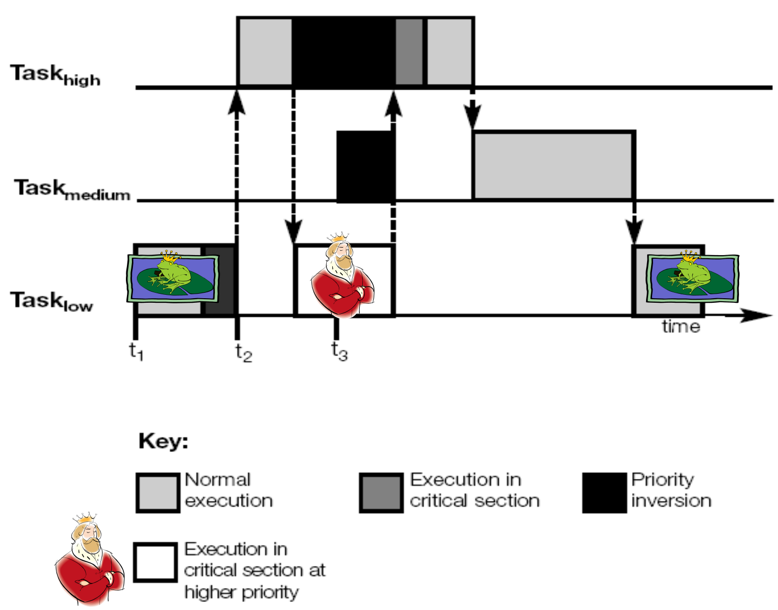
20

# Synchronization Protocols

- No preemption
  - Simplest of all
  - Let the locking task run to completion and unlock
  - Noone else gets to run during that time
- Basic priority inheritance
  - Highest locker priority protocol (you need to implement this in lab4)
- Priority ceiling
- Each protocol prevents **unbounded** priority inversion
  - You cannot avoid priority inversion, but you can put a time bound on how long it will take



# Basic Priority Inheritance Protocol



## Can Deadlocks Occur in Priority Inheritance?

- Task T1 wants to lock L1 and then L2
- Task T2 wants to lock L2 and then L1
- Task T2 has a higher priority than T1

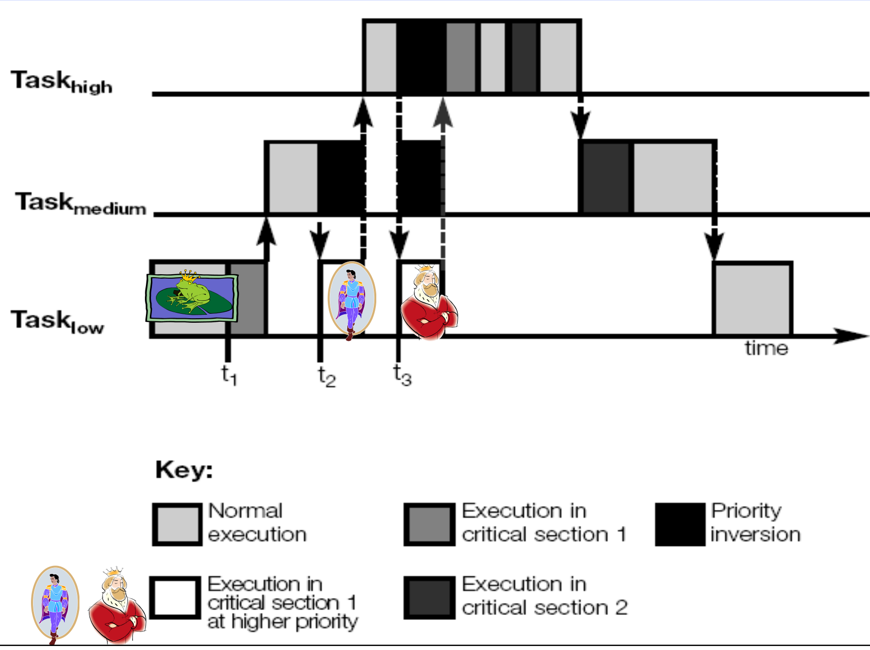
- Suppose that T1 runs first, locks L1 and is then preempted by T2
- Now, T2 runs, locks L2 and wants L1

- According to priority inheritance protocol, T1 will be elevated to T2's priority, and will start to run, but will soon want L2
  - L2 has been previously locked by task T2
  - T2 cannot release L2 because it is blocked, waiting for L1

- Both tasks are deadlocked!
- Remember how we work around this?

23

## Priority Ceiling Protocol



## Summary

- Utilization bound test
- RT test
- Handling priority inversion
  - Basic inheritance
  - Priority ceiling