

Lec. 7: Real-Time Scheduling

Part 1: Fixed Priority Assignment

Vijay Raghunathan

ECE568/CS590/ECE495/CS490

Spring 2011

Reading List: RM Scheduling

- [Balarin98] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. Scheduling for embedded real-time systems. IEEE Design & Test of Comp., vol. 15, no. 1, 1998.
 - <http://ieeexplore.ieee.org/iel3/54/14269/00655185.pdf?arnumber=655185>
- [Liu73] C.L. Liu, and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, vol. 20, no. 1, 1973.
 - <http://citeseer.ist.psu.edu/liu73scheduling.html>
- [Sha94] L. Sha, R. Rajkumar, S.S. Sathaye. Generalized rate-monotonic scheduling theory: a framework for developing real-time systems. Proc. of the IEEE, vol. 82, no. 1, 1994.
 - <http://ieeexplore.ieee.org/iel1/5/6554/00259427.pdf?arnumber=259427>
- [Bini03] Enrico Bini, Giorgio Buttazzo and Giuseppe Buttazzo, "Rate Monotonic Analysis: The Hyperbolic Bound", IEEE Trans. on Computers, vol. 52, no. 7, 2003.
 - <http://feanor.sssup.it/~giorgio/paps/2003/ieeetc-hb.pdf>
- [Sha90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. IEEE Trans. Computers, vol. 39, no. 9, 1990.
 - <http://beru.univ-brest.fr/~singhoff/cheddar/publications/sha90.pdf>

Modeling for Scheduling Analysis

- Need to specify workload model, resource model, and algorithm model of the system in order to analyze the timing properties
- Workload model: Describes the applications/tasks executed
 - Functional parameters: What does the task do?
 - Temporal parameters: Timing properties/requirements of the task
 - Precedence constraints and dependencies
- Resource model: Describes the resources available to the system
 - Number and type of CPUs, other shared resources
- Algorithm model: Describes how tasks use the available resources
 - Essentially a scheduling algorithm/policy

General Workload Model

- Set Γ of "n" tasks $\tau_1, \tau_2, \dots, \tau_n$ that provide some functionality
- Each task may be invoked multiple times as the system functions
 - Each invocation is referred to as a task instance
 - $\tau_{i,j}$ indicates the j^{th} instance of the i^{th} task
- Time when a task instance arrives (is activated/invoked) is called its release time (denoted by $r_{i,j}$)
- Each task instance has a run-time (denoted by $C_{i,j}$ or $e_{i,j}$)
 - May know range $[C_{i,jmin}, C_{i,jmax}]$
 - Can be estimated or measured by various mechanisms
- Each task instance has a deadline associated with it
 - Each task instance must finish before its deadline, else of no use to user
 - Relative deadline ($D_{i,j}$): Span from release time to when it must complete
 - Absolute deadline ($d_{i,j}$): Absolute wall clock time by which task instance must complete

Simplified Workload Model

5

- Tasks are periodic with constant inter-request intervals
 - Task periods are denoted by T_1, T_2, \dots, T_n
 - Request rate of τ_i is $1/T_i$
- Tasks are independent of each other
 - A task instance doesn't depend on the initiation/completion of other tasks
 - However, task periods may be related
- Execution time for a task is constant and does not vary with time
 - Can be interpreted as the maximum or worst-case execution time (WCET)
 - Denoted by C_1, C_2, \dots, C_n
- Relative deadline of every instance of a task is equal to the task period
 - $D_{i,j} = D_i = T_i$ for all instances $\tau_{i,j}$ of task τ_i
 - Each task instance must finish before the next request for it
 - Eliminates need for buffering to queue tasks
- Other implicit assumptions
 - No task can implicitly suspend itself, e.g., for I/O
 - All tasks are fully preemptible
 - All kernel overheads are zero

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Resource Model

6

- Tasks need to be scheduled on one CPU
 - Referred to as uni-processor scheduling
 - Will relax this restriction later to deal with multi-processor scheduling
- Initially, will assume that there are no shared resources
 - Will relax this later to consider impact of shared resources on deadlines

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Scheduling Algorithm

7

- Set of rules to determine the task to be executed at a particular moment
- One possibility: Preemptive and priority-driven
 - Tasks are assigned priorities
 - Statically or dynamically
 - At any instant, the highest priority task is executed
 - Whenever there is a request for a task that is of higher priority than the one currently being executed, the running task is preempted and the newly requested task is started
- Therefore, scheduling algorithm == method to assign priorities

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Assigning Priorities to Tasks

8

- Static or fixed-priority approach
 - Priorities are assigned to tasks once
 - Every instance of the task has the same priority, determined apriori
- Dynamic approach
 - Priorities of tasks may change from instance to instance
- Mixed approach
 - Some tasks have fixed priorities, others don't

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Deriving An Optimum Fixed Priority Assignment Rule

Critical Instant for a Task

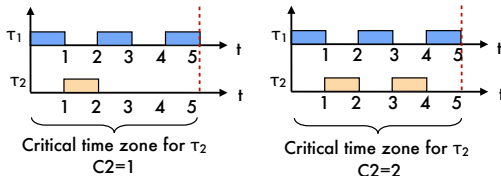
- Overflow is said to occur at time t , if t is the deadline of an unfulfilled request
- A scheduling algorithm is feasible if tasks can be scheduled so that no overflow ever occurs
- Response time of a request of a certain task is the time span between the request and the end of response to that task
- Critical instant for a task = instant at which a request for that task will have the maximum response time
- Critical time zone of a task = time interval between a critical instant and the absolute deadline for that task instance

When does Critical Instant occur?

- **Theorem 1:** A critical instant for any task occurs whenever the task is requested simultaneously with requests of all higher priority tasks
- Can use this theorem to determine whether a given priority assignment will yield a feasible schedule or not
 - If requests for all tasks at their critical instants are fulfilled before their respective absolute deadlines, then the scheduling algorithm is feasible

Example

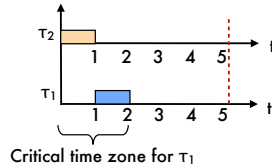
- Consider two tasks τ_1 and τ_2 with $T_1=2$, $T_2=5$, $C_1=1$, $C_2=1$
- Case 1: τ_1 has higher priority than τ_2
 - Priority assignment is feasible
 - Can increase C_2 to 2 and still avoid overflow



Example (contd.)

13

- Case 2: τ_2 has higher priority than τ_1
 - Priority assignment is still feasible
 - But, can't increase beyond $C_1=1, C_2=1$



Case 1 seems to be the better priority assignment for schedulability...
can we formalize this?

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Rate-Monotonic Priority Assignment

14

- Assign priorities according to request rates, independent of execution times
 - Higher priorities for tasks with higher request rates (shorter time periods)
 - For tasks τ_i and τ_j , if $T_i < T_j$, $\text{Priority}(\tau_i) > \text{Priority}(\tau_j)$
- Called Rate-Monotonic (RM) Priority Assignment
 - It is optimal among static priority assignment based scheduling schemes
- **Theorem 2:** No other fixed priority assignment can schedule a task set if RM priority assignment can't schedule it, i.e., if a feasible priority assignment exists, then RM priority assignment is feasible

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Proof of Theorem 2 (RM optimality)

15

- Consider n tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$ ordered in increasing order of time periods (i.e., $T_1 < T_2 < \dots < T_n$)
- Assumption 1: Task set is schedulable with priority assignment $\{\text{Pr}(1), \dots, \text{Pr}(n)\}$ which is not RM
 - Therefore, \exists at least one pair of adjacent tasks, say τ_p and τ_{p+1} , such that $\text{Pr}(p) < \text{Pr}(p+1)$ [higher value is higher priority]
 - Otherwise, assignment becomes RM (violates assumption)
- Assumption 2: Instances of all tasks arrive at $t=0$
 - Therefore, $t=0$ is a critical instant for all tasks. From Theorem 1, we only need to check if first instance of each task completes before deadline

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Proof (contd.)

16

- Swap the priorities of tasks τ_p and τ_{p+1}
 - New priority of τ_p is $\text{Pr}(p+1)$, new priority of τ_{p+1} is $\text{Pr}(p)$
 - Note that $\text{Pr}(p+1) > \text{Pr}(p)$ (by assumption 1)
- Tasks $\{\tau_1, \dots, \tau_{p-1}\}$ should not get affected
 - Since we are only changing lower priority tasks
- Tasks $\{\tau_{p+2}, \dots, \tau_n\}$ should also not get affected
 - Since both τ_p and τ_{p+1} need to be executed (irrespective of the order) before any task in $\{\tau_{p+2}, \dots, \tau_n\}$ gets executed
- Task τ_p should not get affected
 - Since we are only increasing its priority

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Proof (contd.)

17

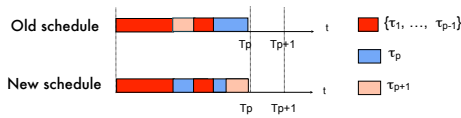
- Consider τ_{p+1} :
- Since original schedule is feasible, in the time interval $[0, T_p]$, exactly one instance of τ_p and τ_{p+1} complete execution along with (possibly multiple) instances of tasks in $\{\tau_1, \dots, \tau_{p-1}\}$
 - Note that τ_{p+1} executes before τ_p
- New schedule is identical, except that τ_p executes before τ_{p+1} (start/end times of higher priority tasks is same)
 - Still, exactly one instance of τ_p and τ_{p+1} complete in $[0, T_p]$. As $T_p < T_{p+1}$, task τ_{p+1} is schedulable

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Proof (contd.)

18



We proved that swapping the priority of two adjacent tasks to make their priorities in accordance with RM does not affect the schedulability (i.e., all tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$ are still schedulable)

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Proof (contd.)

19

- If τ_p and τ_{p+1} are the only such non RM tasks in original schedule, we are done since the new schedule will be RM
- If not, starting from the original schedule, using a sequence of such re-orderings of adjacent task pairs, we can ultimately arrive at an RM schedule (Exactly the same as bubble sort)
- E.g., Four tasks with initial priorities $[3, 1, 4, 2]$ for $[\tau_1, \tau_2, \dots, \tau_n]$

$[3 \ 1 \ 4 \ 2]$ is schedulable



$[3 \ 4 \ 1 \ 2]$ is schedulable



$[4 \ 3 \ 1 \ 2]$ is schedulable



$[4 \ 3 \ 2 \ 1]$ is schedulable

Hence, Theorem 2 is proved.

RM priority assignment

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Processor Utilization Factor

20

- Processor Utilization: fraction of processor time spent in executing the task set (i.e., $1 - \text{fraction of time processor is idle}$)
 - Provides a measure of computational load on CPU due to a task set
 - A task set is definitely not schedulable if its processor utilization is > 1
- For n tasks, $\tau_1, \tau_2, \dots, \tau_n$ the utilization "U" is given by:

$$U = C_1/T_1 + C_2/T_2 + \dots + C_n/T_n$$

- U for a task set Γ can be increased by increasing C_i 's or by decreasing T_i 's as long as tasks continue to satisfy their deadlines at their critical instants
- There exists a minimum value of U below which Γ is schedulable and above which it is not
 - Depends on scheduling algorithm and the task set

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

How Large can U be?

21

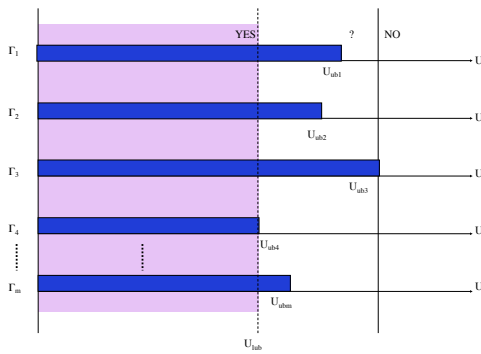
- For a given priority assignment, a task set **fully utilizes** a processor if:
 - The priority assignment is feasible for the task set (i.e., no deadline misses)
 - And, if an increase in the execution time of any task in the task set will make the priority assignment infeasible (i.e., cause a deadline miss)
- The U at which this happens is called the upper bound $U_{ub}(\Gamma, A)$ for a task set Γ under scheduling algorithm A
- The least upper bound of U is the minimum of the U's over all task sets that fully utilize the processor (i.e., $U_{lub}(A) = \min_{\Gamma} [U_{ub}(\Gamma, A)]$)
 - For all task sets whose U is below this bound, there exists a fixed priority assignment which is feasible
 - U above this can be achieved only if task periods T_i 's are suitably related
- $U_{lub}(A)$ is an important characteristic of a scheduling algorithm A as it allows easy verification of the schedulability of a task set
 - Below this bound, a task set is definitely schedulable
 - Above this bound, it may or may not be schedulable

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Least Upper Bound of U

22



Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Utilization Bound for RM

23

- RM priority assignment is optimal; therefore, for a given task set, the U achieved by RM priority assignment is \geq the U for any other priority assignment
- In other words, the least upper bound of U = the infimum of U's for RM priority assignment over all possible T's and all C's for the tasks

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Two Tasks Case

24

- Theorem 3:** For a set of two tasks with fixed priority assignment, the least upper bound to processor utilization factor is $U=2(2^{1/2}-1)$
- Given any task set consisting of only two tasks, if the utilization is less than 0.828, then the task set is schedulable using RM priority assignment

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

General Case

25

- **Theorem 4:** For a set of n tasks with fixed priority assignment, the least upper bound to processor utilization factor is $U = n(2^{1/n} - 1)$
- Equivalently, a set of " n " periodic tasks scheduled by the RM algorithm will always meet deadlines for all task start times if $C_1/T_1 + C_2/T_2 + \dots + C_n/T_n \leq n(2^{1/n} - 1)$

General Case (contd.)

26

- As $n \rightarrow \infty$, the U rapidly converges to $\ln 2 = 0.69$
- However, note that this is just the least upper bound
 - A task set with larger U may still be schedulable
 - e.g., if $(T_n \% T_i) = 0$ for $i=1,2,\dots,n-1$, then $U=1$
- How to check if a specific task set with n tasks is schedulable?
 - If $U \leq n(2^{1/n}-1)$ then it is schedulable
 - Otherwise, need to use Theorem 1!
- Two ways in which this analysis is useful
 - For a fixed CPU, will a set of tasks work or not? How much background load can you throw in without affecting feasibility of tasks?
 - During CPU design, you can decide how slow/fast a CPU you need

Theorem 1 Recalled

27

- **Theorem 1:** A critical instant for any task occurs whenever the task is requested simultaneously with requests of all higher priority tasks
- Can use this to determine whether a given priority assignment will yield a feasible scheduling algorithm
 - If requests for all tasks at their critical instants are fulfilled before their respective deadlines, then the scheduling algorithm is feasible
- Applicable to any static priority scheme... not just RM

Example #1

28

- Task τ_1 : $C_1 = 20$; $T_1 = 100$; $D_1 = 100$
Task τ_2 : $C_2 = 30$; $T_2 = 145$; $D_2 = 145$
Is this task set schedulable?
- $U = 20/100 + 30/145 = 0.41 \leq 2(2^{1/2}-1) = 0.828$
- Yes!

Example #2

29

- Task τ_1 : $C_1 = 20$; $T_1 = 100$; $D_1 = 100$
Task τ_2 : $C_2 = 30$; $T_2 = 145$; $D_2 = 145$
Task τ_3 : $C_3 = 68$; $T_3 = 150$; $D_3 = 150$
Is this task set schedulable?

- $U = 20/100 + 30/145 + 68/150 = 0.86 > 3(2^{1/3} - 1) = 0.779$

- Can't say! Need to apply Theorem 1

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Hyperbolic Bound for RM

30

- Feasibility analysis of RM can also be performed using a different approach, called the Hyperbolic Bound approach
 - E. Bini, G. Buttazzo, and G. Buttazzo, "Rate Monotonic Analysis: The Hyperbolic Bound", IEEE Transactions on Computers, Vol. 52, No. 7, pp. 933-942, July 2003.
 - <http://feanor.sssup.it/~giorgio/paps/2003/ieeetc-hb.pdf>

- **Theorem:** A set Γ of n periodic tasks with processor utilization U_i for the i -th task is schedulable with RM priority assignment if:

$$\prod_{i=1}^n (U_i + 1) \leq 2$$

- Same complexity as Liu and Layland test but less pessimistic
 - Shows that the bound is tight: best possible bound using individual task utilization factors

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Example #2 revisited

31

- The utilization based test is only a sufficient condition. Can we obtain a stronger test (a necessary and sufficient condition) for schedulability?

- Task τ_1 : $C_1 = 20$; $T_1 = 100$; $D_1 = 100$
Task τ_2 : $C_2 = 30$; $T_2 = 145$; $D_2 = 145$
Task τ_3 : $C_3 = 68$; $T_3 = 150$; $D_3 = 150$

- Consider the critical instant of τ_3 , the lowest priority task
 - τ_1 and τ_2 must execute at least once before τ_3 can begin executing
 - Therefore, completion time of τ_3 is $\geq C_1 + C_2 + C_3 = 20 + 68 + 30 = 118$
 - However, τ_1 is initiated one additional time in $(0, 118)$
 - Taking this into consideration, completion time of $\tau_3 = 2C_1 + C_2 + C_3 = 2 \cdot 20 + 68 + 30 = 138$
- Since $138 < D_3 = 150$, the task set is schedulable

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Response Time Analysis for RM

32

- For the highest priority task, worst case response time R is its own computation time C
 - $R = C$
- Other lower priority tasks suffer interference from higher priority processes
 - $R_i = C_i + I_i$
 - I_i is the interference in the interval $[t, t + R_i]$

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Response Time Analysis (contd.)

23

- Consider task i , and a higher priority task j
- Interference from task j during R_i :
 - # of releases of task $j = \lceil R_i/T_j \rceil$
 - Each release will consume C_j units of processor
 - Total interference from task $j = \lceil R_i/T_j \rceil * C_j$
- Let $hp(i)$ be the set of tasks with priorities higher than that of task i
- Total interference to task i from all tasks during R_i :

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Response Time Analysis (contd.)

24

- This leads to:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Smallest R_i that satisfies the above equation will be the worst case response time
- Fixed point equation: can be solved iteratively

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Algorithm

25

```
for i in 1..N loop -- for each process in turn
  n := 0
  w_i^n := C_i
  loop
    calculate new w_i^{n+1} from Equation
    if w_i^{n+1} = w_i^n then
      R_i := w_i^n
      exit {value found}
    end if
    if w_i^{n+1} > T_i then
      exit {value not found}
    end if
    n := n + 1
  end loop
end loop
```

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

What is wrong with this test?

26

- Can you use this test to check for schedulability?

$$C_i + I_i \leq T_i$$
$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil C_j$$

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

RM Schedulability

27

- Consider tasks $\tau_1, \tau_2, \dots, \tau_n$ in decreasing order of priority
- For task τ_i to be schedulable, a necessary and sufficient condition is that we can find some t in $[0, T_i]$ satisfying the condition

$$t = \lceil t/T_1 \rceil C_1 + \lceil t/T_2 \rceil C_2 + \dots + \lceil t/T_{i-1} \rceil C_{i-1} + C_i$$

- But do we need to check at exhaustively for all values of t in $[0, T_i]$?

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

RM Schedulability (contd.)

28

- Observation: RHS of the equation jumps only at multiples of T_1, T_2, \dots, T_{i-1}
- It is therefore sufficient to check if the inequality is satisfied for some t in $[0, T_i]$ that is a multiple of one or more of T_1, T_2, \dots, T_{i-1}

$$t \geq \lceil t/T_1 \rceil C_1 + \lceil t/T_2 \rceil C_2 + \dots + \lceil t/T_{i-1} \rceil C_{i-1} + C_i$$

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

RM Schedulability (contd.)

29

- Notation
 - $W_i(t) = \sum_{j=1..i} C_j \lceil t/T_j \rceil$
 - $L_i(t) = W_i(t)/t$
 - $L_i = \min_{0 \leq t \leq T_i} L_i(t)$
 - $L = \max\{L_i\}$
- General sufficient and necessary condition:
 - Task τ_i can be scheduled iff $L_i \leq 1$
- Practically, we only need to compute $W_i(t)$ at all times
 - $\alpha_i = \{kT_j \mid j=1, \dots, i; k=1, \dots, \lfloor T_i/T_j \rfloor\}$
 - These are the times at which tasks are released
 - $W_i(t)$ is constant at other times
- Practical RM schedulability conditions:
 - If $\min_{i \in \alpha_i} W_i(t)/t \leq 1$, task τ_i is schedulable
 - If $\max_{i \in \{1, \dots, n\}} \{\min_{t \in \alpha_i} W_i(t)/t\} \leq 1$, then the entire set is schedulable

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Example

40

- Task set:
 - $\tau_1: T_1=100, C_1=20$
 - $\tau_2: T_2=150, C_2=30$
 - $\tau_3: T_3=210, C_3=80$
 - $\tau_4: T_4=400, C_4=100$
- Then:
 - $\alpha_1 = \{100\}$
 - $\alpha_2 = \{100, 150\}$
 - $\alpha_3 = \{100, 150, 200, 210\}$
 - $\alpha_4 = \{100, 150, 200, 210, 300, 400\}$
- Plots of $W_i(t)$: task τ_i is RM-schedulable iff any part of the plot of $W_i(t)$ falls on or below the $W_i(t)=t$ line.

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Deadline Monotonic Assignment

- Relax the $D_i = T_i$ constraint to now consider $C_i \leq D_i \leq T_i$
 - Priority of a task is inversely proportional to its relative deadline
 - $D_i < D_j \Rightarrow P_i > P_j$
 - **DM is optimal**; Can schedule any task set that any other static priority assignment can
 - Example: RM fails but DM succeeds for the following task set
- | | Period
T | Deadline
D | Comp
Time, C | Priority
P | Response
Time, R |
|--------|---------------|-----------------|-------------------|-----------------|-----------------------|
| Task_1 | 20 | 5 | 3 | 4 | 3 |
| Task_2 | 15 | 7 | 3 | 3 | 6 |
| Task_3 | 10 | 10 | 4 | 2 | 10 |
| Task_4 | 20 | 20 | 3 | 1 | 20 |
- **Schedulability Analysis**: One approach is to reduce task periods to relative deadlines
 - $C_1/D_1 + C_2/D_2 + \dots + C_n/D_n \leq n(2^{1/n}-1)$
 - However, this is very pessimistic
 - A better approach is to do critical instant (response time) analysis

Can one do better?

- Yes... by using dynamic priority assignment
- In fact, there is a scheme for dynamic priority assignment for which the least upper bound on the processor utilization is 1
- More later...

Task Synchronization

- So far, we considered independent tasks
- In reality, tasks do interact: semaphores, locks, monitors, rendezvous, etc.
 - shared data, use of non-preemptable resources
- Jeopardizes systems ability to meet timing constraints
 - e.g., may lead to an indefinite period of “priority inversion” where a high priority task is prevented from executing by a low priority task

Priority Inversion Example

- Let τ_1 and τ_3 be two tasks that share a resource (protected by semaphore S), with τ_1 having a higher priority. Let τ_2 be an intermediate priority task that does not share any resource with either. Consider the following sequence of actions:
- τ_3 gets activated, obtains a lock on the semaphore S, and starts using the shared resource
- τ_1 becomes ready to run and preempts τ_3 . While executing, τ_1 tries to use the shared resource by trying to lock S. But S is already locked and therefore τ_1 is blocked
- Now, τ_2 becomes ready to run. Since only τ_2 and τ_3 are ready to run, τ_2 preempts τ_3 .

Priority Inversion Example (contd.)

45


- What would we prefer?
 - τ_1 , being the highest priority task, should be blocked no longer than the time τ_3 takes to complete its critical section
- But, in reality, the duration of blocking is unpredictable
 - τ_3 can remain preempted until τ_2 (and any other pending intermediate priority tasks) are completed
- The duration of priority inversion becomes a function of the task execution times, and is not bounded by the duration of critical sections

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Just another theoretical problem?

46

- Recall the Mars Pathfinder from 1997?
 - Unconventional landing - bouncing onto Martian surface with airbags
 - Deploying the Sojourner rover: First roving probe on another planet
 - Gathering and transmitting voluminous data, including panoramic pictures that were such a hit: http://en.wikipedia.org/wiki/Mars_Pathfinder
 - Used VxWorks real-time kernel (preemptive, static-priority scheduling)
- But...
 - A few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data
 - Reported in the press as "software glitches" and "the computer was trying to do too many things at once"



Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

What really happened on Mars?

47

- The failure was a priority inversion failure!
- A high priority task **bc_dist** was blocked by a much lower priority task **ASI/MET** which had grabbed a shared resource and was then preempted by a medium priority communications task
- The high priority **bc_dist** task didn't finish in time
- An even higher priority scheduling task, **bc_sched**, periodically creates transactions for the next bus cycle
- **bc_sched** checks whether **bc_dist** finished execution (hard deadline), and if not, resets the system

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Software Timeline

48

```
|< ----- .125 seconds ----->|
|<*****|                |*****|  |**>|
|< -bus active ->|<- bc_dist active ->|  bc_sched active
|<- bus active ->|                |<->|
-----|-----|-----|-----|-----|
t1      t2      t3      t4  t5  t1
```

The *** are periods when tasks other than the ones listed are executing. Yes, there is some idle time.

t1 - bus hardware starts via hardware control on the 8 Hz boundary. The transactions for the this cycle had been set up by the previous execution of the bc_sched task.
t2 - 1553 traffic is complete and the bc_dist task is awakened.
t3 - bc_dist task has completed all of the data distribution
t4 - bc_sched task is awakened to setup transactions for the next cycle
t5 - bc_sched activity is complete

Embedded Systems (Spring 2011) Lecture 07: Real-Time Scheduling (part 1)

Why was it not caught before launch?

- The problem only manifested itself when **ASI/MET** data was being collected and intermediate tasks were heavily loaded
- Before launch, testing was limited to the "best case" high data rates and science activities
 - Data rates on Mars were higher than anticipated; the amount of science activities proportionally greater served to aggravate the problem.
 - Did not expect or test the "better than we could have imagined" case
- Did see the problem before launch but could not get it to repeat when they tried to track it down
 - Neither reproducible or explainable
 - Attributed to "hardware glitches"
 - Lower priority - focus was on the entry and landing software

What saved the day?

- How did they find the problem?
 - Trace/log facility + a replica on earth
- How did they fix it?
 - Changed the creation flags for the semaphore so as to enable "priority inheritance"
 - VxWorks supplies global configuration variables for parameters, such as the "options" parameter for the semMCreate used by the select service
 - Turns out that the Pathfinder code was such that this global change worked with minimal performance impact
 - Spacecraft code was patched: sent "diff"
 - Custom software on the spacecraft (with a whole bunch of validation) modified the onboard copy

Diagnosing the Problem

- Diagnosing the problem as a black box would have been impossible
- Only detailed traces of actual system behavior enabled the faulty execution sequence to be captured and identified
- Engineer's initial analysis that "the data bus task executes very frequently and is time-critical – we shouldn't spend the extra time in it to perform priority inheritance" was exactly wrong
- See http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/ for a description of how things were diagnosed and fixed

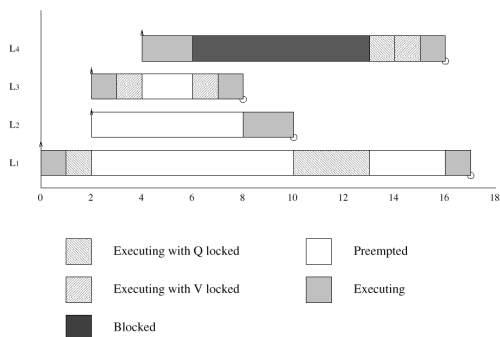
Process Interactions and Blocking

- Priority inversions
- Blocking
- Priority inheritance

Process	Priority	Execution Seq	Release Time
L_4	4	EEQVE	4
L_3	3	EVVE	2
L_2	2	EE	2
L_1	1	EQQQQE	0

Example: Priority Inversion

53



Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Priority Inheritance

54

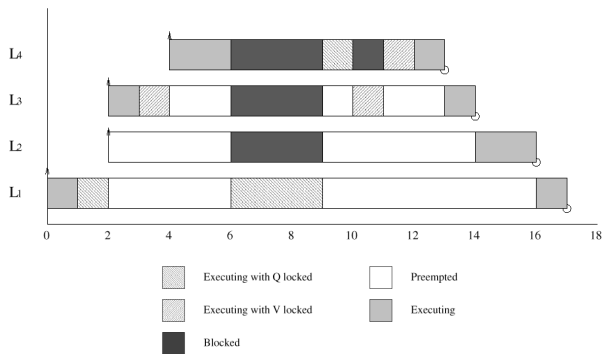
- Simple method for eliminating priority inversion problems
- Basic Idea:** If a high priority task H gets blocked while trying to lock a semaphore that has already been locked by a low priority task L, then L temporarily *inherits* the priority of H while it holds the lock to the semaphore
 - The moment L releases the semaphore lock, its priority drops back down
- Any intermediate priority task, I, will not preempt L because L will now be executing with a higher priority while holding the lock

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Example: Priority Inheritance

55



Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Response Time Calculations

56

- $R = C + B + I$
 - solve by forming recurrence relation
- With priority inheritance:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$B_i = \sum_{k=1}^K usage(k, i) CS(k)$$

Embedded Systems (Spring 2011)

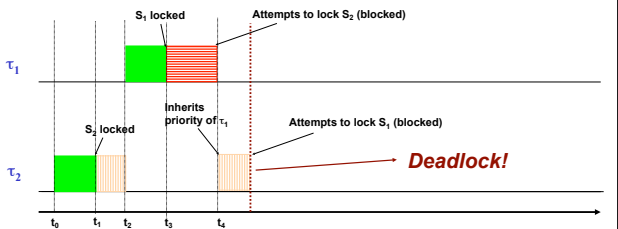
Lecture 07: Real-Time Scheduling (part 1)

Response Time Calculations

- Where usage is a 0/1 function:
 - $usage(k, i) = 1$ if resource k is used by at least 1 process with priority $< i$, and at least one process with a priority greater or equal to i .
 - $= 0$ otherwise
- $CS(k)$ is the computational cost of executing the critical section associated with resource k

Priority Inheritance Can Lead to Deadlock

- Two tasks τ_1 and τ_2 with two shared data structures protected by binary semaphores S_1 and S_2 .
 - τ_1 : { ... Lock(S_1) ... Lock(S_2) ... Unlock(S_2) ... Unlock(S_1) ... }
 - τ_2 : { ... Lock(S_2) ... Lock(S_1) ... Unlock(S_1) ... Unlock(S_2) ... }
- Assume τ_1 has higher priority than τ_2



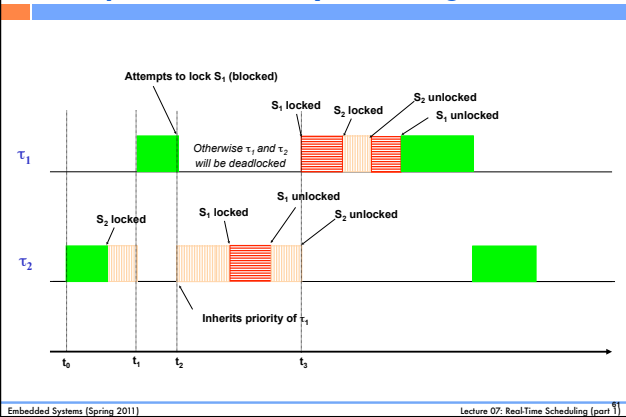
Priority Ceiling Protocols

- Basic idea:
 - Priority ceiling of a binary semaphore S is the highest priority of all tasks that may lock S
 - When a task τ attempts to lock a semaphore, it will be blocked unless its priority is $>$ than the priority ceiling of all semaphores currently locked by tasks other than τ
 - If task τ is unable to enter its critical section for this reason, the task that holds the lock on its semaphore with the highest priority ceiling is
 - Said to be blocking τ
 - Hence, inherits the priority of τ

Example of Priority Ceiling Protocol

- Two tasks τ_1 and τ_2 with two shared data structures protected by binary semaphores S_1 and S_2 .
 - τ_1 : { ... Lock(S_1) ... Lock(S_2) ... Unlock(S_2) ... Unlock(S_1) ... }
 - τ_2 : { ... Lock(S_2) ... Lock(S_1) ... Unlock(S_1) ... Unlock(S_2) ... }
- Assume τ_1 has higher priority than τ_2
- Note: priority ceilings of both S_1 and $S_2 =$ priority of τ_1

Example of Priority Ceiling Protocol



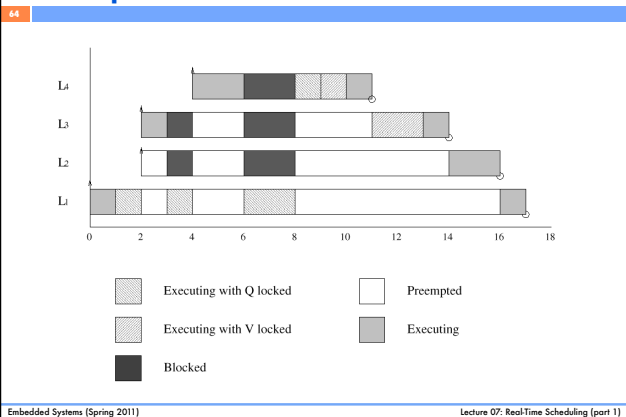
Priority Ceiling Protocols (contd.)

- Two forms
 - Original ceiling priority protocol (OCPP)
 - Immediate ceiling priority protocol (ICPP)
- On a single processor system
 - A high priority process can be blocked at most once during its execution by lower priority processes
 - Deadlocks are prevented
 - Transitive blocking is prevented
 - Mutual exclusive access to resources is ensured (by the protocol itself)

OCPP

- Each process has a static default priority assigned (perhaps by the deadline monotonic scheme)
 - Each resource has a static ceiling value defined, this is the maximum priority of the processes that use it
 - A process has a dynamic priority that is the maximum of its own static priority and any it inherits due to it blocking higher priority processes
 - A process can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself).
- $$B_i = \max_{k=1}^K usage(k, i) CS(k)$$

Example of OCPP



ICPP

65

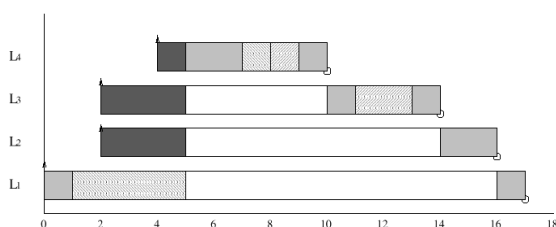
- Each process has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the processes that use it
- A process has a dynamic priority that is the maximum of its own static priority and the ceiling values of any resources it has locked.

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Example of ICPP

66



Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

OCPP vs. ICPP

67

- Worst case behavior identical from a scheduling point of view
- ICPP is easier to implement than the original (OCPP) as blocking relationships need not be monitored
- ICPP leads to less context switches as blocking is prior to first execution
- ICPP requires more priority movements as this happens with all resource usages; OCPP only changes priority if an actual block has occurred.

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)

Schedulability Impact of Task Synchronization

68

- Let B_i be the duration in which τ_i is blocked by lower priority tasks
- The effect of this blocking can be modeled as if τ_i 's utilization were increased by an amount B_i/T_i
- The effect of having a deadline D_i before the end of the period T_i can also be modeled as if the task were blocked for $E_i = (T_i - D_i)$ by lower priority tasks
 - As if utilization increased by E_i/T_i
- Theorem: A set of n periodic tasks scheduled by RM algorithm will always meet its deadlines if:

$$i, 1 \leq i \leq n, \frac{C_1}{T_1} + \frac{C_2}{T_2} + \dots + \frac{C_i + B_i + E_i}{T_i} \leq i(2^{1/i} - 1)$$

Embedded Systems (Spring 2011)

Lecture 07: Real-Time Scheduling (part 1)