Rate Monotonic Analysis (RMA)

郭大維 教授

ktw@csie.ntu.edu.tw

即時及嵌入式系統實驗室

(Real-Time and Embedded System Laboratory)

國立台灣大學資訊工程系

Major References:

An Introduction to Rate Monotonic Analysis
—Tutorial Notes, SEI, CMU*

Distributed Real-Time System Design Using Generalized Rate Monotonic Theory
—Tutorial Notes, Lui Sha, SEI, CMU, 1992.*

*The RMA Project is sponsored by the U.S. Dept. of Defense.

- Can we have a better way to predict the schedulability of a process set more precisely?
- Are we still able to predict the schedulability of a process set if the basic assumptions of rate monotonic scheduling [LL:73] are violated?
  - Rate monotonic priorities
  - Unique priority per unique period
  - Preemptive scheduling
  - Deadlines are coincident with start of period
  - Only periodic tasks
- Do we have an analytical framework for reasoning the timing behavior of a process set or have an engineering basis for designing real-time systems.

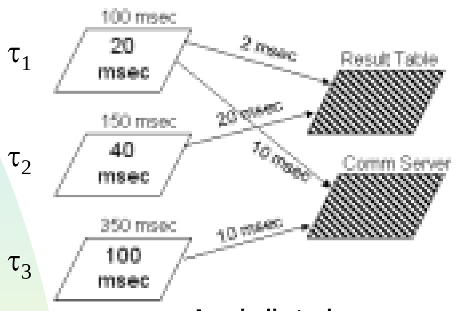
Why are deadlines missed?

- For a given task, consider：
  - the time needed by higher priority tasks
  - the time needed to do this task’s work
  - delays caused by lower priority tasks：priority inversion (blocking)
- To improve the performance of real-time systems：
  - identify and limit sources of priority inversion*

**check previous examples*

A Sample Problem

Periodic tasks **Servers**



Aperiodic tasks

Emergency 50 msec	Event handling task 40 msec
5 msec	2 msec
d:6 msec	Desired respone 4 msec on average

Periodic Requirements

- Periodic task
  - Ready to execute at fixed intervals
  - deadline = the beginning of the next period (to be relaxed later!)
- Rate Monotonic Algorithm
  - Assign higher priorities to tasks with shorter periods (to be relaxed later!)
  - If $U(T) \le n(2^{\frac{1}{n}} - 1)$ for some n-process set T, T is schedulable [KM:91,LL:73].

Periodic Requirements

Task τ_1: $C_1=20$, $P_1=100$, $U_1=0.2$

Task τ_2: $C_2=40$, $P_2=150$, $U_2=0.267$

Task τ_3: $C_3=100$, $P_3=350$, $U_3=0.286$

Total utilization: 75.3% $\le U(3): 3(2^{\frac{1}{3}} - 1) = 77.9\%$

24.7% of the CPU is usable for lower-priority background computation!

ADD more computation!

Periodic Requirements

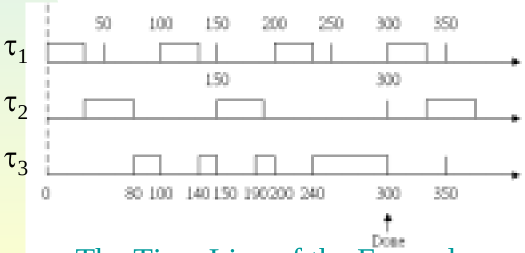
Task τ_1: $C_1=40$, $P_1=100$, $U_1=0.4$

$\Rightarrow$ the utilization factor of the first two tasks:

66.7% < U(2): 82.8%

Total utilization: 95.3% > U(3) !

The test result is inconclusive !



The Time Line of the Example

Improving the Schedulability Bound

- Theorem 2[LSD87]: A set of n independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if and only if

$$\forall i, 1 \le i \le n, \min_{(k,l)\in R_i} \sum_{j=1}^{i} C_j \frac{1}{lP_k} \left\lceil \frac{lP_k}{P_j} \right\rceil \le 1$$

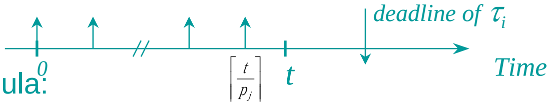
$$R_i = \{(k,l) \mid 1 \le k \le i, l = 1, \Lambda\ , \lfloor P_i / P_k \rfloor\}$$

i.e., $$(\min_{(k,l)\in R_i}(\sum_{j=1}^{i} C_j \left\lceil \frac{lP_k}{P_j} \right\rceil - lP_k) \le 0)$$

Improving the Schedulability Bound - Theorem 2 Revisited

- Rate Monotonic Analysis (RMA) [2]
  - Basic Idea:

    Before time t after the critical instance of process τ_i, a high priority process τ_j may request $c_j\left\lceil \frac{t}{p_j} \right\rceil$ amount of computation time.



  - Formula:

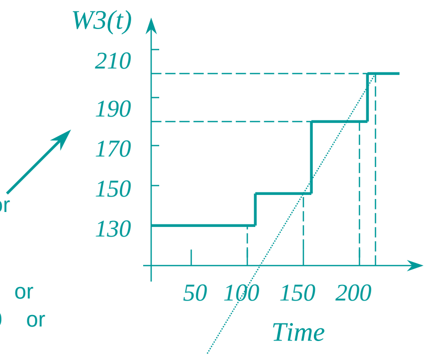
$$W_i(t) = \sum_{j=1}^{i} c_j \left\lceil \frac{t}{p_j} \right\rceil \le t \le d_i \quad \text{for some } t \text{ in } \{kp_j \mid j = 1,\ldots,i;\ k = 1,\ldots,\lceil p_i / p_j \rceil\}$$

  - A sufficient and necessary condition and many extensions...

[2] *Sha, "An Intorduction to Rate Monotonic Analysis," tutorial notes, SEI, CMU, 1992*

Improving the Schedulability Bound - Theorem 2 Revisited

- A RMA Example:
  - ✓ T1(20,100), T2(30,150), T3(80, 210), T4(100,400)
  - T1
    - c1 <= 100
  - T2
    - c1 + c2 <= 100 or
    - 2c1 + c2 <= 150
  - T3
    - c1 + c2 + c3 <= 100 or
    - 2c1 + c2 + c3 <= 150 or
    - 2c1 + 2c2 + c3 <= 200 or
    - 3c1 + 2c2 + c3 <= 210
  - T4
    - c1 + c2 + c3 + c4 <= 100 or
    - 2c1 + c2 + c3 + c4 <= 150 or
    - ....



Improving the Schedulability Bound

- Theorem 3[JP86]: For a set of independent, periodic tasks, if each task meets its first deadline, with worst-case task phasings, the deadline will always be met

Completion Time (CT) test：

Let W_i=completion time of task i, W_i may be computed by the following iterative formula：

$$W_i(n+1) = C_i + \sum_{j<i} \left\lceil \frac{W_i(n)}{P_j} \right\rceil C_j, \qquad W_i(0) = 0$$

Task i is schedulable if its completion time is before its deadline. $W_i \le P_i$ (*Take P_i as d_i !!)

Reference:

[LSD87] Lehoczky, Sha, and Ding, "The Rate Monotonic Scheduling Algorithm-Exact Characterization and Average Case Behavior", TR, Dept. of statistics, LMU,1987

[JP86] Joseph and Pandya, "Finding Response Times in a Real-Time System", BCS Computing Journal, vol 29, no.5, 1986, pp390-395

Improving the Schedulability Bound – an example

Task τ_1: C_1=40, P_1=100, U_1=0.4

Task τ_2: C_2=40, P_2=150, U_2=0.267

Task τ_3: C_3=100, P_3=350, U_3=0.286

Apply Theorem 2

$C_1+C_2+C_3 \le P_1$ 40+40+100 > 100

or $2C_1+C_2+C_3 \le P_2$ 80+40+100 > 150

or $2C_1+2C_2+C_3 \le 2P_1$ 80+80+100 > 200

or $3C_1+2C_2+C_3 \le 2P_2$*120+80+100 <= 300 *

or $4C_1+3C_2+C_3 \le P_3$ 160+120+100 > 350

Apply Theorem 3

$$W_3(0) = 0$$

$$W_3(1) = C_3 + \sum_{j<3} \left\lceil \frac{0}{P_j} \right\rceil C_j = C_3 = 100$$

$$W_3(2) = C_3 + \sum_{j<3} \left\lceil \frac{100}{P_j} \right\rceil C_j = 100 + \left\lceil \frac{100}{100} \right\rceil 40 + \left\lceil \frac{100}{150} \right\rceil 40 = 180$$

$$W_3(3) = 100 + \left\lceil \frac{180}{100} \right\rceil 40 + \left\lceil \frac{180}{150} \right\rceil 40 = 260$$

$$W_3(4) = 100 + \left\lceil \frac{260}{100} \right\rceil 40 + \left\lceil \frac{260}{150} \right\rceil 40 = 300$$

$$W_3(5) = 100 + \left\lceil \frac{300}{100} \right\rceil 40 + \left\lceil \frac{300}{150} \right\rceil 40 = 300$$

The computation is converged!

$\Theta\, W_3 = 300 \le P_3 = d_3 = 350 \Rightarrow$ Schedulable !

Summary

- Utilization bound test is simple but conservative.
- Completion time test is more exact but also more complicated! In fact, tests based on Theorems 2 and 3 have a pseudo-polynomial-time time complexity.
- To this point, UB, CT, and Schedulability-Point tests share the same limitations.
  - all tasks are periodic and not interacting with each another.
  - deadlines are always the end of the period.
  - no interrupts.
  - rate monotonic priorities assigned.
  - all tasks are on a single processor.
  - zero context switch overhead. (stack dispelling)
  - tasks do not suspend themselves.

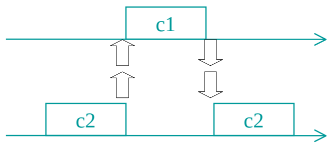
Practical Applications

- Modeling context switching
- Schedulability with priority inversion
- Schedulability with Interrupts
- Schedulability without a rate monotonic priority
- Handling pre-period deadlines

Important Issue:

Identify the sources of blocking & manage them!

Modeling Context Switching



- RMA:

  Pure priority-driven preemptive schedules impose two scheduling actions per task (start of the period & end of the period)

$$U_i = \frac{c_i + 2s}{P_i}$$

Modeling Context Switching - Remark

Can it be more efficient than a cyclic executive?

Run-time scheduling certainly appears to have more overhead than the scheduling overhead for a cyclic executive. But there is a hidden overhead in the cyclic case. Task periods must sometimes to be shortened to fit within the cyclic executive structure! Machine utilization increases, but not because more useful work is done!

* Cyclic executive - where all work(tasks) fit into a common period (major frame) and executed non-preemptively.

Priority Inversion

- Delay to a task's execution caused by lower-priority tasks is known as priority inversion.
- Systems potentially contain many sources of priority inversion.

$\Rightarrow$ Identifying, reducing, and modeling priority inversion is central to schedulability analysis.

Priority Inversion

⇒ Sources of Priority Inversion
- ◆Non-rate-monotonic priority assignment (syntactically in RMA ?!!)
- ◆Non-preemptibility
- ◆Interrupts
- ◆Not enough priority levels
- ◆FIFO queues (of course, they are more!)
- ◆Synchronization

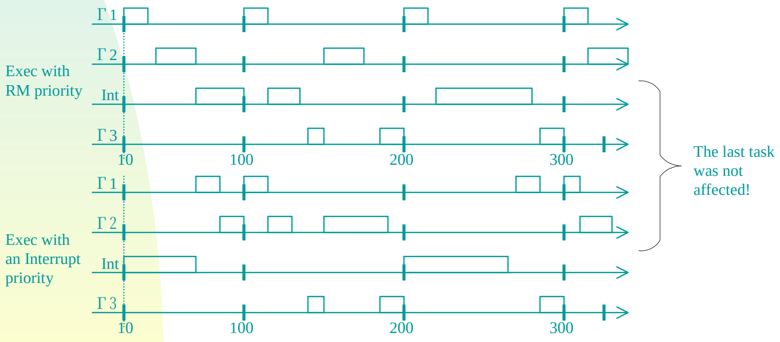
Schedulability with Interrupts

- Interrupt processing can be inconsistent with the RM priority assignment:
  - ◆Execute with a high priority despite long "period"!
  - ◆Delay execution of tasks with higher rate monotonic priorities.
  - ◆Source of priority inversion!

Schedulability with Interrupts

- Example

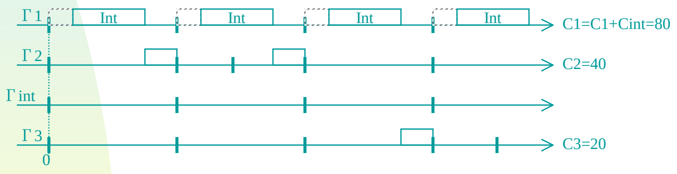
Task Γ1 : C1 =20, P1 =100, U1=0.2
Task Γ2 : C2 =40, P2 =150, U2=0.267
Interrupt : Cint=60, Pint=200, Uint=0.3
Task Γ3 :C3 =20, P3 =350, U3=0.057



Exec with RM priority
Exec with an Interrupt priority
Γ1
Γ2
Int
Γ3
0
100
200
300
The last task was not affected!

Schedulability with Interrupts

- Solution 1
  - Should interrupts be modeled in the same manner as task's context switching (i.e., treated as extra execution time)?



Schedulability with Interrupts

- ◆The completion time for $\Gamma 1$ is correct!
  - =〉 Correct model for $\Gamma 1$.
- ◆ $\Gamma 2$ fails!
  - Θ It is too pessimistic for $\Gamma 2$.
  - $\Gamma 2$ should be affected at most once per period!
- ◆ $\Gamma 3$ fails!
  - $\Gamma 3$ is preempted too frequently!

=〉 Consider each task separately!

Rule of Thumb

- Task schedulability is affected by
  - ◆Preemption effects:
    Potentially many times per period.
  - ◆Execution effects:
    Once per period!
  - ◆Blocking effects:
    at most once per period for each source of blocking

*Blocking effects occur at a lower frequency, and thus each blocking effect can impact the task only once per period.

Schedulability with Interrupts

- Solution 2: Modeling Interrupts with Blocking Time

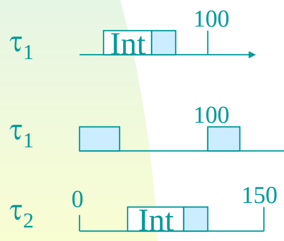
Example:

τ_1: $C_1=20, P_1=100, U_1=0.2$

τ_2: $C_2=40, P_2=150, U_2=0.267$

τ_{int}: $C_{int}=60, P_{int}=200, U_{int}=0.3$

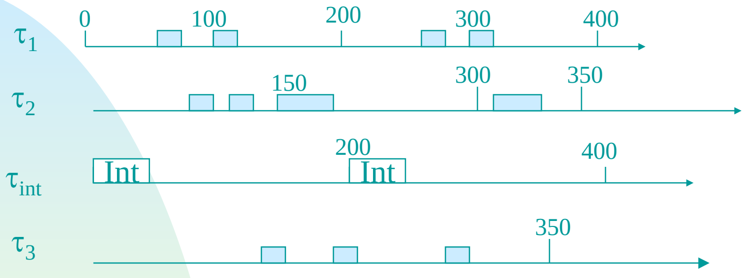
$\tau 3$: $C_3=20, P_3=350, U_3=0.057$



$$\frac{C_1+C_{int}}{P_1}=0.8\le 1.0$$ For τ_1

$$\frac{C_1}{P_1}+\frac{C_2+C_{int}}{P_2}=0.86>U(2)=0.828$$ For τ_2

Schedulability with Interrupts



For τ_3

$$\frac{C_1}{P_1}+\frac{C_2}{P_2}+\frac{C_{int}}{P_{int}}+\frac{C_3}{P_3}=0.824>U(4)=0.756$$

(Harmonic base size = 3 U(3)=0.779)

*(3) $\frac{C_{int}}{P_{int}}=0.3\le 1.0$

Apply Theorem 2

For τ_2: $R_2=\{(1,1),(2,1)\}$

$C_1 + C_2 + B_2 = 20 + 40 + C_{int} = 120 > P_1$

$2C_1 + C_2 + B_2 = 40 + 40 + C_{int} = 140 \leqq P_2$

τ_1: $C_1 = 20$, $P_1 = 100$, $U_1 = 0.2$

τ_2: $C_2 = 40$, $P_2 = 150$, $U_2 = 0.267$

τ_{int}: $C_{int} = 60$, $P_{int} = 200$,$U_{int} = 0.3$

Apply Theorem 3

For τ_2:

$\omega_2(0) = 0$

$\omega_2(1) = C_2 + B_2 + \sum_{j<2} \frac{\lceil 0 \rceil}{p_j} C_j$

$= C_2 + B_2 = 100$

$\omega_2(2) = C_2 + B_2 + \sum_{j<2} \frac{\lceil 100 \rceil}{P_j} < j$

$= C_2 + B_2 + C_1 = 120$

$\omega_2(3) = C_2 + B_2 + C_1 \frac{\lceil 120 \rceil}{100}$

$= C_2 + B_2 + 2C_1 = 140$

$\omega_2(4) = C_2 + B_2 + C_1 \frac{\lceil 140 \rceil}{100}$

$=C_2 + B_2 + 2C_1 = 140$

$\omega_2 = 140 < d_2 = P_2 = 150$

Summary

This modeling technique can be used to model:
- Interrupts
- Non-preemptibility
- Non-rate-monotonic priorities
  - -Find priority based on pre-period deadline
  - -Jitter requirement (e.g. high priority I/O-related execution)

Theorem 1:

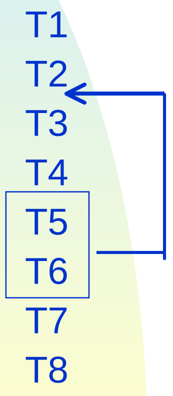
$$\sum_{j<i} \frac{C_j}{P_j} + \frac{C_i + B_i}{P_i} \le U(i)$$

Theorems 2 and 3:

Include B_i in the computation time of τ_i when the schedulability of τ_i *is under consideration!*

Non-Rate-Monotonic Priority Assignment

- Example 1: Preemption/Exec/Blocking Effects



T5-Preemption Effects: C1, C2
T6-Preemption Effects: C1, C2, C5
T7-Preemption Effects: C1, …, C6
(assume no changes over priority order)
T3:
Preemption Effects: C1, C2
Execution Effects: C3
Blocking Effects: C5 + C6

Non-Rate-Monotonic Priority Assignment

- Example 2: Limited Priority Levels

T1 T2
T3 T4
T5 T6
T7 T8

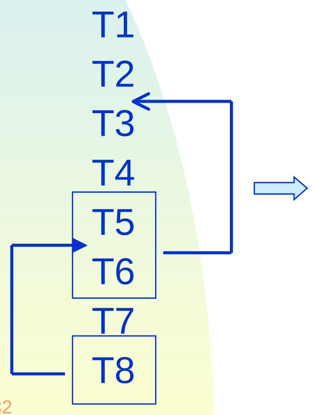
T1:
Preemption: none
Execution: C1
Blocking: C2

T2:
Preemption: C1
Execution: C2
Blocking: none

T3:
Preemption: C1, C2
Execution: C3
Blocking: C4

Non-Rate-Monotonic Priority Assignment

- Example 3: Preemption/Exec/Blocking Effects



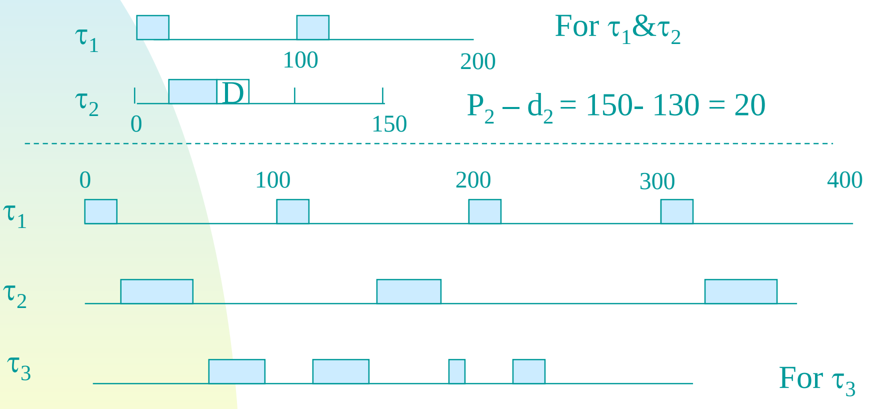
T1
T2
T5
T8
T6
T3
T4
T7

T6:
Preemption: C1, C2, C5
Execution: C6
Blocking: C8

T7:
Preemption: C1-C6
Execution: C7
Blocking: C8

Techniques for Handling Pre-Period Deadlines

- Insert dormant time



Techniques for Handling Pre-Period Deadlines - Insert dormant time

$$\frac{C_1}{P_1} + \frac{C_2 + D_2}{P_2} = \frac{20}{100} + \frac{40 + 20}{150} \le 0.828 = U(2)$$

$$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3} = \frac{20}{100} + \frac{40}{150} + \frac{100}{350} \le 0.779 = U(3)$$

Techniques for Handling Pre-Period Deadlines

- Apply critical-zone analysis
  1. Determine the worst-case completion time.

$$\omega_2(0) = 0$$

$$\omega_2(1) = C_2 + \sum_{j<2} C_j \left\lceil \frac{\omega_2(0)}{P_j} \right\rceil = C_2 = 40$$

$$\omega_2(2) = C_2 + C_1 \left\lceil \frac{40}{100} \right\rceil = C_2 + C_1 = 60$$

$$\omega_2(3) = C_2 + C_1 \left\lceil \frac{60}{100} \right\rceil = C_2 + C_1 = 60$$

  2.Compare worst-case completion time with the pre-period deadline.

$$\omega_2 = 60 \le d_2 = 130$$

Another thought about the calculation of worst-case completion!

$$\underbrace{\omega_i(n+1) = C_i}_{\text{exec}} + \underbrace{\sum_{\tau_j \text{ has a priority higher than } \tau_i} C_j \left\lceil \frac{\omega_i(n)}{P_j} \right\rceil}_{\text{preemption}} + \underbrace{B_i}_{\text{blocking caused by ……}}$$

- Incrementally increase priority if needed!!
  1. If the task can not meet a specified pre-period deadline, then raise its priority & try again!
     non-rate-monotonic priority!
  2.Use the same analysis as for interrupts
     Consider the schedulability of all critical tasks!

Measurement Issues

- Common misconception:
  The rate monotonic analysis seems to be useful only if accurate execution times are available!

  In fact, the rate monotonic analysis is forgiving to inaccurate measurement:
  - Importance of accuracy is relative to the length of the period. (deadline)
  - One is more likely to have better estimates for higher frequency tasks.
  - Robust to change!!

Measurement Issues

- The issue of inaccurate execution time is not specific to the rate monotonic analysis approach; it is inherent to hard real-time systems!

  The rate monotonic approach, however, highlights the parameters of importance.
  - higher-priority execution times
  - blocking times,
  - and execution time

Treatment of Synchronization Requirements

Synchronization

$\Downarrow$

Priority Inversion

Identifying, modeling, and reducing sources of priority inversion is central to schedulability analysis!!

Source of Priority Inversion

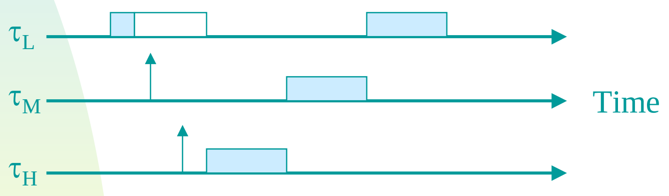
- non-preemptible regions of code
- interrupts
- non-unique priorities for some tasks
- non-rate-monotonic assignment of task priorities
- FIFO of any other non-priority-based queues
- Synchronization and mutual exclusion

Remark: Blocking effects(usually) occur at a lower rate, and thus each blocking effect can impact a task(usually) at most once per period.

Synchronization Protocols

1. No preemption

Critical sections are executed at an "infinitely" high priority!

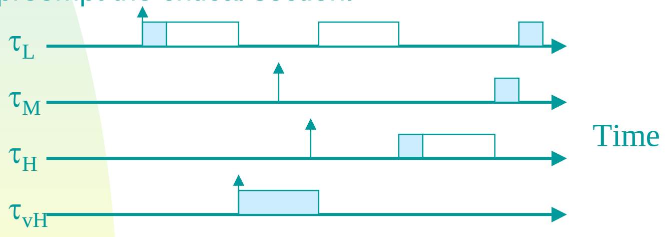


Note that τ_H & τ_M have no intention to enter a critical section!

Synchronization Protocols

2.Highest locker's priority

Execute critical section at the priority of the highest-priority task that may lock the semaphore(/resource); higher-priority tasks may preempt the critical section.

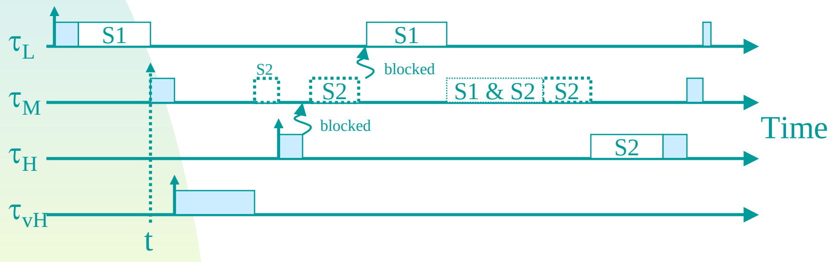


Note that τ_{vH} is no longer blocked by τ_L

Synchronization Protocols

3. Basic Inheritance Protocol (BIP)

Execute the critical section at the priority of the highest-priority task being blocked; higher-priority tasks may preempt the critical section.



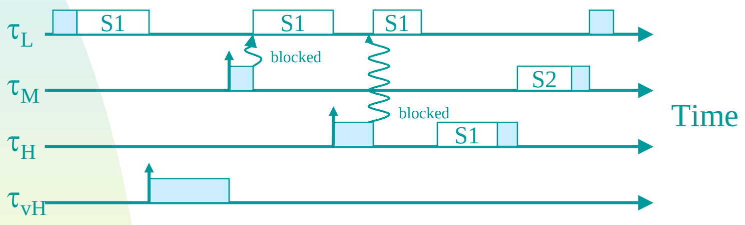
- Note that τ_M is no longer blocked until necessary.

- However, system may be deadlocked or have chained blocking!

Synchronization Protocols

4. Priority Ceiling Protocol

BIP + a "Priority Ceiling" rule about when to grant lock requests (see [Sha 87, 90])



- No deadlock & chained blocking at the cost of reducing the concurrency level of the system.

- Blocked-at-most-once.

Synchronization Protocols

- Schedulability tests for Basic Inheritance Protocol (BIP)

$$\frac{C_1}{P_1}+\frac{B_1}{P_1}\le U(1)\Rightarrow \frac{20}{100}+\frac{20+10}{100}=0.5\le 1.0$$

$$\frac{C_1}{P_1}+\frac{C_2+D_2}{P_2}+\frac{B_2}{P_2}\le U(2)$$

$$\Rightarrow \frac{20}{100}+\frac{40+20}{150}+\frac{10}{150}=0.667\le 0.828$$

$$\frac{C_1}{P_1}+\frac{C_2}{P_2}+\frac{C_3}{P_3}\le U(3)$$

$$\Rightarrow \frac{20}{100}+\frac{40}{150}+\frac{100}{350}=0.753\le 0.779$$

* In PCP, $B_1 = \max(20,10)$

Synchronization Protocols

- Schedulability tests for Non-preemptible Critical Sections

$$\frac{C_1}{P_1}+\frac{B_1}{P_1}=\frac{20}{100}+\frac{20}{100}=0.4\le 1.0$$

$$\frac{C_1}{P_1}+\frac{C_2+D_2}{P_2}+\frac{B_2}{P_2}=\frac{20}{100}+\frac{40+20}{150}+\frac{10}{150}=0.667\le U(2)=0.828$$

$$\frac{C_1}{P_1}+\frac{C_2}{P_2}+\frac{C_3}{P_3}\le U(3)$$

Synchronization Protocols – a comparison

	Bounded Priority Inversion	Blocked at Most Once	Deadlock Avoidance
Nonpreemptible Critical Sections	Yes	Yes[1]	Yes[1]
Highest Locker's Priority	Yes	Yes[1]	Yes[1]
BIP	Yes	No	No
PCP	Yes	Yes[2]	Yes

1Tasks suspending themselves inside critical sections will hand over CPU to (lower-priority) tasks. The later may lock other resources.

2Tasks suspending themselves between critical sections shall not be protected!

3Reasons for task suspensions: I/O

Aperiodic Servers

Aperiodic tasks runs at irregular intervals

Aperiodic deadline

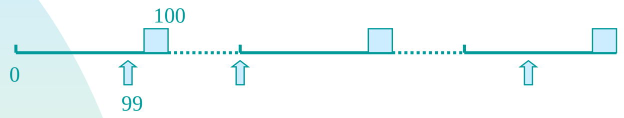
- hard, minimum inter-arrival time.
- Soft, best average response time.

Services such as

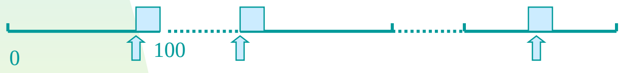
- operator requests
- device interrupts
- …………...

Scheduling Aperiodic Tasks

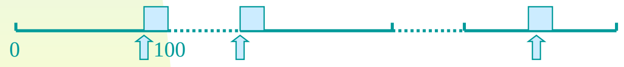
– Polling ~ Average Response Time = 50 units



– Interrupt Handler ~ Average Response Time = 1 unit



– Deferrable Server [Lehoczky87] ~ Average Response Time = 1 unit



- An on-demand service type
- When execution budget is used up, server execution drops to a lower (background) priority until the budgeted execution time is replenished.

Deferrable Server (DS) [Lehoczky87]

- τ_1 as a DS server to preserve CPU bandwidth for a collection of aperiodic tasks.
- τ_1 has an entire period to use its C run-time and gets replenished at the beginning of each of its period.
- Theorem 3 [Lehoczky87]: For τ_1 as the highest priority DS server, and $\tau_2 \ldots \tau_n$ as periodic tasks, the achievable utilization factor

$$U = U_1 + (n-1)\left[\left(\frac{U_1+2}{U_1+1}\right)^{1/(n-1)} - 1\right]$$

when $n \to \infty$

$$U = U1 + \ln\left(\frac{U_1+2}{2U_1+1}\right)$$

U is minimized to 0.6518 when $U_1 = 0.186$.

J.P. Lehoczky, L. Sha, and J.K. Strosnider, “Enhancd Aperiodic Responsiveness in Hard Real-Time Environment,” RTSS’87, pp261-270.

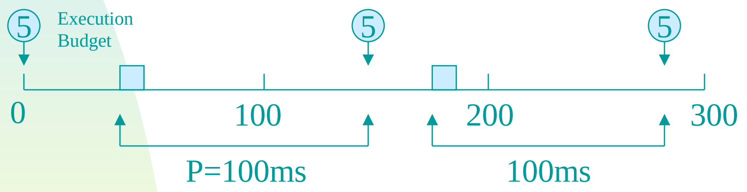
Remarks:

- Deferrable Servers:
  - –fixed execution budget
  - –replenishment interval
  - –priority adjusted to meet requirements.

    Note that the response time performance improves as the replenishment rate decreases because the execution budget increases and more services can be provided.
- Polling:
  - –From the scheduling point of view, polling converts the servicing of aperiodic events into an "equivalent" periodic tasks.
  - –Not an on-demand service type.
  - –As the rate of polling increases, the response time for polling approach improves.
- The rationale behind aperiodic servers:
  - –No "system" benefit to finish periodic work early !

Sporadic Server [Sprunt et. al. 90]

- Modeled as periodic tasks
  - ◆fixed execution budget(c)
  - ◆replenishment interval (p)



Replenishment occurs one "period" after the start of usage !

- ◆Priority adjusted to meet requirements.

Sporadic Server [Sprunt et. al. 90]

- A sporadic server differs from a deferrable server in its replenishment policy ：
  - A 100 msec deferrable server replenishes its execution budget every 100 msec , no matter when the execution budget is used.
- The affect of a sporadic server on lower priority tasks is no worse than a periodic task with the same period and execution time.

Sporadic Server [Sprunt et. al. 90]

- A sporadic server (SS) under the fixed-priority scheduling framework.
- Terms:
  - Ps: the priority at which the system is executing.
  - Pi: one priority level in the system.
  - Active: A priority level Pj is active is Pj <= Ps.
  - Idle: not active
  - RTi: replenishment time for SS executing at priority level Pi

Brinkley Sprunt, "Aperiodic Task Scheduling for Real-Time Systems," Ph.D. Thesis
Dept. of Electrical and Computer Engineering, Carnegie Mellon University, August 1990.

Sporadic Server [Sprunt et. al. 90]

- Rules:
  - Replenishment Time RTi
    - If SS has execution time available, and Pi becomes active at time t, then RTi = t + Pi.
    - If SS's execution time is exhausted, and SS's execution time becomes non-zero (is replenished) at time t and Pi is active, then RTi = t + Pi.
  - Replenishment Amount
    - Determined when Pi becomes idle or SS's execution time has been exhausted.
    - The amount is equal to the amount of server execution time consumed since the last time at which the status of Pi changes from idle to active.

Sporadic Server [Sprunt et. al. 90]

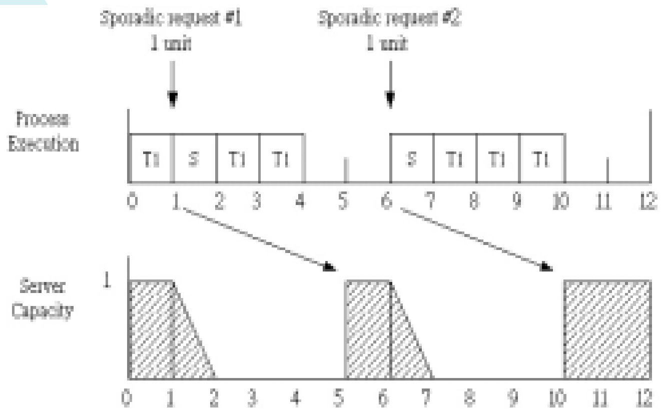
Important theorems:

- Theorem 1 [Sprunt90:p28]:Given a real-time system composed of soft-real-time aperiodic tasks and hard real-time periodic tasks, let the soft real-time aperiodic tasks be serviced by a polling server that starts at full capacity and executes at the priority level of the highest priority periodic task. If the polling server is replaced with a sporadic server having the same period, execution time, and priority, the sporadic server will provide high-priority aperiodic service at times earlier than or equal to the times the polling server would provide high-priority aperiodic service.

Sporadic Server [Sprunt et. al. 90]

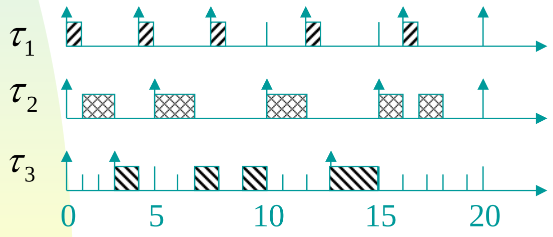
- Theorem 5 [Sprunt90:p34]: A periodic task set that is schedulable with a periodic task Ti, is also schedulable if Ti is replaced by a sporadic server with the same period and execution time.
- Schedulability analysis of sporadic servers is equivalent to periodic tasks! -> overcome the penalty paid by the deferrable servers!
- Remark
  - In terms of server size, the sporadic server approach is better than the deferrable server approach.
  - Although the sporadic server approach claims low implementation overhead, it seems to be a little bit higher than the deferrable server approach.
  - If aperiodic services are requested very heavily , the differences between DS and SS will diminish.

A Sporadic Server Example



Aperiodic Task Processing – A Compasion

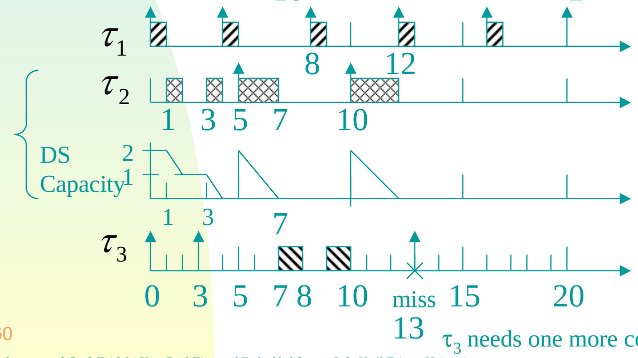
- A task set example
  - Periodic task1：C1=1， P1=4 ▨
  - Periodic task2/ deferrable server/ sporadic server：C2=2， P2=5 ▨
  - Periodic task3：C3=3， P3=10 ▨
- Periodic task executions



Aperiodic Task Processing – A Compasion

- deferrable server execution
  - Aperiodic Instant / Request Time

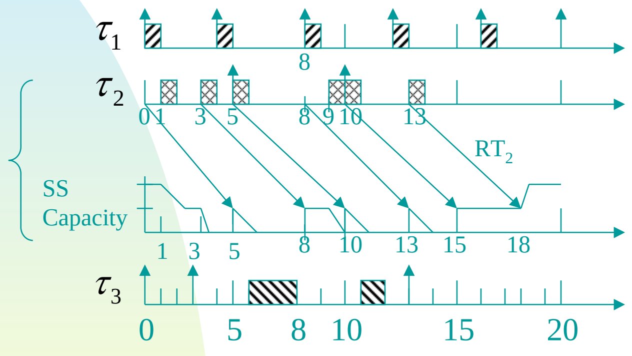
Aperiodic Instant	Request Time
1	1
3	1
5	2
10	2



τ_3 needs one more computation time.

Aperiodic Task Processing – A Compasion

- sporadic server execution



◆ Instant	RTime
1	1
3	1
5	2
10	2

- Note that the 3rd and 4th requests response times are delayed by 3 & 2 time units in this case, respectively.

Sample Problem: aperiodic tasks

- Emergency events
  - ◆5 ms of work
  - ◆arrives every 50ms, worst case
  - ◆hard deadline 6ms after arrival
  - ⇨ Emergency Server (ES)
    - ●a sporadic server
      - C_i = 5 ms
      - P_i = 50 ms (replenishment interval)

Sample Problem: aperiodic tasks

- routine event
  - 2 ms of work on average
  - arrives every 40ms on average
  - desired average response of 4ms after arrival.

⇨ Routine Server (RS)

- a sporadic server
  - $C_i = 10$
  - $P_i = 100$ ms (replenishment interval)

Sample Problem: aperiodic tasks

- How to derive it ?!

  Using M/M/1 queuing approximation

$$\text{Response Time}_{\text{avg}} = \frac{ServiceTime_{avg}}{(1-\frac{Workload_{avg}}{Capacity})} \quad \text{Workload}_{\text{avg}} = \frac{ServiceTime_{avg}}{InterArrivalTime_{avg}}$$

$$\text{Capacity} = \frac{\mathrm{Re}\,sponseTime_{avg} * ServiceTime_{avg}}{IntervalArrivalTime_{avg}(\mathrm{Re}\,sponseTime_{avg} - ServiceTime_{avg})}$$

$$\text{Capacity} = \frac{4*2}{40(4-2)} = 0.1 = \frac{Budget}{Interval}$$

But what replenishment interval we should pick up ?? 50ms , 80ms , 100ms ,200ms ,. . .

Take 100ms and make RS's priority > any periodic τ_i's priority.

Sample Problem: all tasks

Now we have the following sample problem: (BIP)

	C	P	D	B	U
ES	5	50			0.1
RS	10	100			0.1
τ_1	20	100		30	0.2
τ_2	40	150	20	10	0.267 = 40/150
τ_3	100	350			0.286 = 100/350

Harmoically related! (ES, RS, τ_1)

Sample Problem: all tasks

- UB Test
  - ○ τ_1 $$\frac{(2*5)+10+20}{100}+\frac{30}{100}=0.7<1.00$$ ✓
  - ? τ_2 $$\frac{(2*5)+10+20}{100}+\frac{40+20+10}{150}=0.867>U(2)=0.828$$
  - ? τ_3 $$\frac{(2*5)+10+20}{100}+\frac{40}{150}+\frac{100}{350}=0.953>U(3)=0.779$$
    - If PCP is used B1=max(20,10)=20 instead of 30.

Sample Problem: all tasks

- CT Test for τ_2

$$W_2(1) = B_2 + C_2 + \left\lceil \frac{W_2(0)}{P_1} \right\rceil C_1^*$$

$$= 10 + 40 + \left\lceil \frac{0}{100} \right\rceil 40 = 50$$

$$W_2(2) = 10 + 40 + \left\lceil \frac{50}{100} \right\rceil 40 = 90$$

$$W_3(2) = 10 + 40 + \left\lceil \frac{90}{100} \right\rceil 40 = 90 \Rightarrow \mathit{Done}$$

Since

$$W_2 = 90 \le P_2 - D_2 = 150 - 20 = 130$$

Summary

- Rate monotonic analysis offers a general framework for considering timing issues through the life cycle ~ early detection and minimization of priority inversion.
- Implementation:
  - schedulability analysis
  - facilitates separation of concerns
- Testing
  - identification of bottlenecks
  - discovering of timing errors
- Post-deployment
  - easy to understand effects of changes.