Outline

- Assumptions
- Fixed-Priority vs. Dynamic-Priority Algorithms
- Maximum Schedulable Utilization
- Optimality of the RM and DM Algorithms
- A Schedulability Test for Fixed-Priority Tasks with Short Response Times
- Schedulability Test for Fixed-Priority Tasks with Arbitrary Response Times
- Sufficient Schedulability Conditions for the RM and DM Algorithms
- *Practical Factors*

Terminologies

- **Blocking**: A (ready) job J_i is blocked when it is prevented from executing by a lower-priority job: The lower priority job executes while J_i waits.
- **Priority Inversion**: We say that a priority inversion occurs whenever a lower-priority job executes while some ready higher-priority job waits.

Outline

- Practical Factors
  1. ***Nonpreemptability***
  2. Self-Suspension
  3. Context Switches
  4. Limited-Priority Levels
  5. Tick Scheduling
  6. Varying Priority in Fixed-Priority Systems
  7. Schedulability Test of Hierarchically Scheduled Periodic Tasks

Reasons for Nonpreemptable

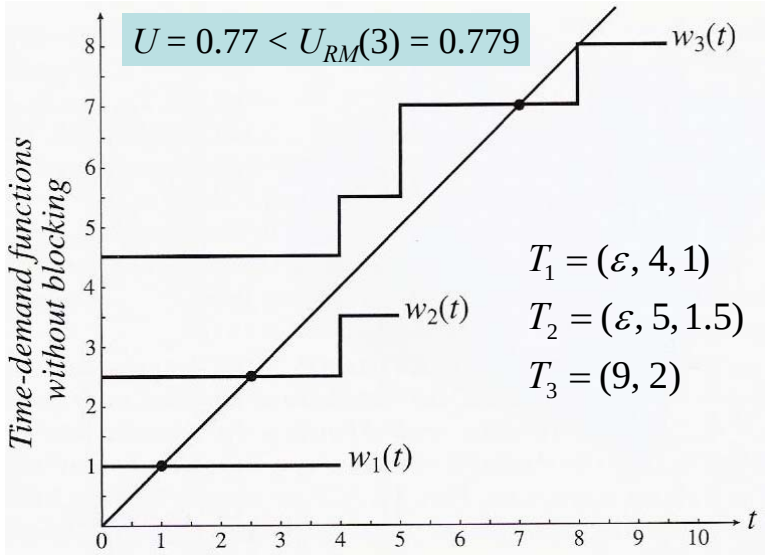
- When a job is using a resource (e.g., a critical section) that must be used in a mutual exclusive manner
  - System calls
- Preemption may be too costly.
  - Disk scheduling

Blocking Time Due to Nonpreemptivity

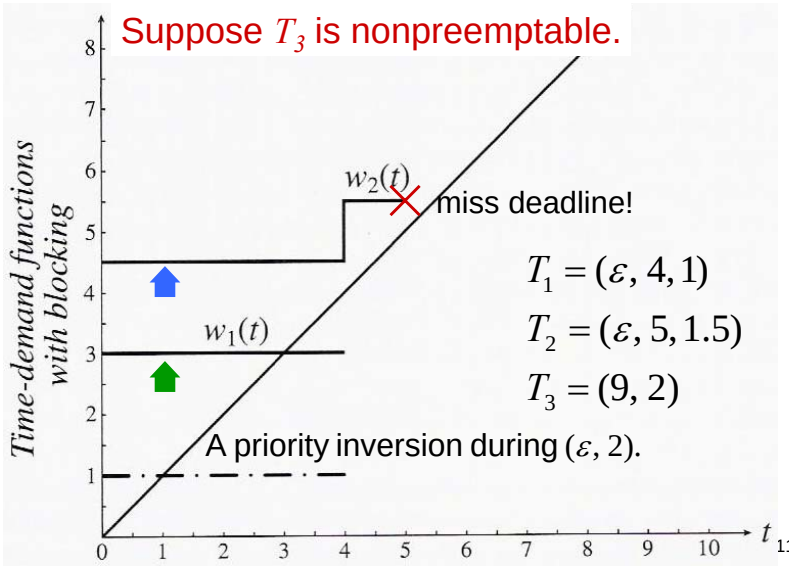
- A higher-priority job that becomes ready when a nonpreemptable lower-priority job is executing is blocked until the nonpreemptable portion of the lower-priority job completes.

➡ When we want to determine whether a task can meet all its deadline, we must consider
  - All the tasks that have higher priorities
  - Nonpreemptable portions of lower-priority tasks

Example (1/2)



Example (2/2)



Blocking Time

- Let $b_i(np)$ denote the longest time for which any job in T_i can be blocked each time it is blocked due to nonpreemptive lower-priority tasks.
- We call $b_i(np)$ its *blocking time per blocking due to nonpreemptivity*.
- We use θ_i ($\theta_i \le e_i$) to denote the maximum execution time of the longest nonpreemptable portion of jobs in the task T_i.

➡ In a fixed-priority system, $b_i(np)$ is given by

$$b_i(np) = \max_{i+1 \le k \le n} \theta_k$$

Effect of Blocking on Schedulability

- The term **blocking time**, denoted by b_i, refers to the maximum total duration for which each job in task T_i may be delayed by both lower-priority tasks and deferred execution of higher-priority tasks.

Time-Demand Function (with Blocking)

- Time-demand function ($D_i \leqq p_i$ for all i)

$$w_i(t) = e_i + b_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k, \quad \text{for } 0 < t \leq \min(D_i, p_i)$$

- Time-demand function ($D_i > p_i$ for some i)

$$w_{i,j}(t) = je_i + b_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k, \quad \text{for } (j-1)p_i < t \leq w_{i,j}(t)$$

Schedulable Utilization (with Blocking)

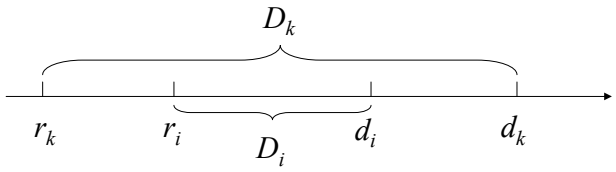
- In a fixed-priority system, the task T_i, for $1 \leq i < n$, is schedulable if

$$\frac{e_1}{p_1} + \frac{e_2}{p_2} + \ldots + \frac{e_i + b_i}{p_i} = U_i + \frac{b_i}{p_i} \leq U_X(i)$$

where U_i is the total utilization of the i highest priority tasks and $U_X(i)$ denotes the appropriate schedulable utilization of the fixed-priority algorithm X used to schedule the tasks.

Blocking Time in Deadline-Driven System

- **Theorem 13.** In a system where jobs are scheduled on the EDF basis, a job J_k with relative deadline D_k can block a job J_i with relative deadline D_i only if D_k is larger than D_i.



References Baker, T. P., "A Stack-Based Resource Allocation Policy for Real-Time Processes," *Proceedings of IEEE Real-Time Systems Symposium*, 1991.

Proof of Theorem 13

- **Proof.** The following observations allow us to conclude that the theorem is true.
  1. In order for J_k to block J_i, J_k must have a lower priority than J_i. Since priorities are assigned on the EDF basis, this implies that the deadline d_k of J_k must be later than the deadline d_i of J_i. In other words, $d_k > d_i$.
  2. In addition, when the higher-priority job J_i is released, the lower-priority job J_k must be executing if J_k is to block J_i. This means that J_k must be released earlier than J_i, or $r_k < r_i$.

  Both inequalities can be true only when $D_k > D_i$.

Schedulability Test for EDF (with Blocking)

- Suppose that we index all periodic tasks according to their relative deadlines; the smaller the relative deadline, the smaller the index.
- A task T_i with a total blocking time b_i is schedulable with other independent periodic tasks on a processor according to the EDF algorithm if

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} + \frac{b_i}{\min(D_i, p_i)} \le 1$$

  The system is schedulable if the condition is met for every $i = 1, 2, \ldots, n$.

Outline

- Practical Factors
  1. Nonpreemptability
  2. ***Self-Suspension***
  3. Context Switches
  4. Limited-Priority Levels
  5. Tick Scheduling
  6. Varying Priority in Fixed-Priority Systems
  7. Schedulability Test of Hierarchically Scheduled Periodic Tasks

Introduction (1/4)

- While executing, a job may invoke some external operation that is executed on another processor.
- Self-blocking or self-suspension occurs when the job is suspended and waits until such an operation completes before its execution can continue.
- We assume that the maximum amount of time each external operation takes to complete, i.e., the maximum duration of each self-suspension, is known.

Introduction (2/4)

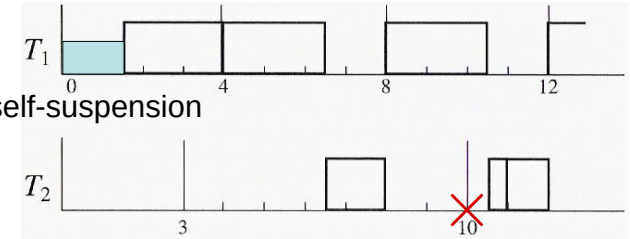
- In a special case, every job in a task T_i self-suspends for x units of time immediately after it is released (e.g., due to input data transmission).
  - The job is ready for execution x time units after it is released.
  - ➡ The time from the instant when the job is ready to its deadline is only $D_i - x$, not D_i.
  - ➡ To determine whether the task T_i is schedulable, we use the shortened deadline $D_i - x$ in the schedulability test; there is no need to modify any of the methods otherwise.

Introduction (3/4)

- A job may self-suspend after its execution has begun, and the amount of time for which jobs in a task self-suspend may differ.
  - ➡ The task no longer behaves as a periodic task.
  - ➡ It may demand more time in some interval than a periodic task.

Introduction (4/4)

T_1 demands 5.5 units of time in (3, 10]!



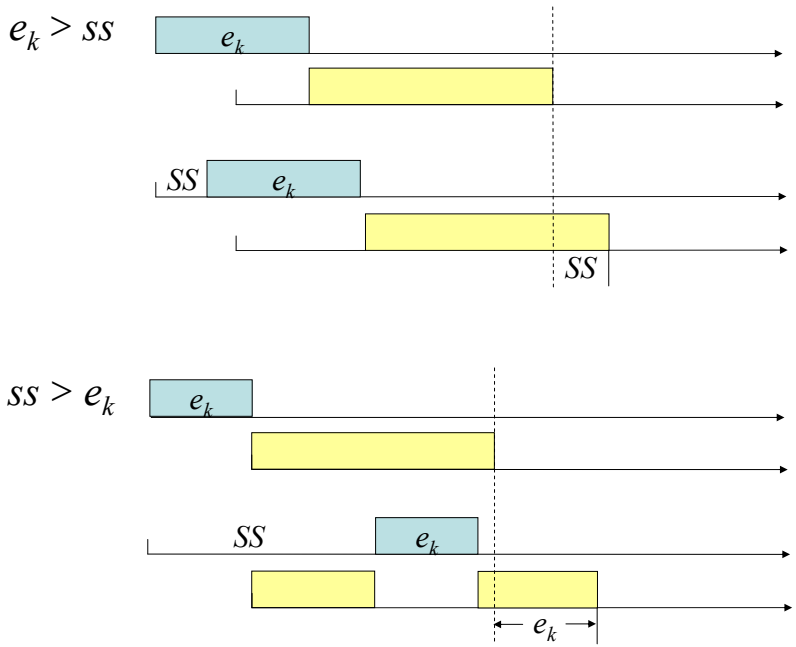
T_1=(4, 2.5), T_2=(3, 7, 2.0)

$J_{2,1}$ misses its deadline!

Self-Suspension (1/2)

- One way to account for the extra delay suffered by a task T_i in a fixed-priority system due to its own self-suspension and the suspension of higher-priority tasks is to treat this delay as a factor of the blocking time of T_i (denoted by $b_i(ss)$).
- We call $b_i(ss)$ the blocking time of T_i due to self-suspension.

$$b_i(ss) = \text{maximum self-suspension time of } T_i + \sum_{k=1}^{i-1} \min(e_k, \text{maximum self-suspension time of } T_k)$$



Self-Suspension (2/2)

- In a system where some tasks are nonpreemptable, the effect of self-suspension is more severe.
- Every time a job suspends itself, it loses the processor. It may be blocked again by a nonpreemptive lower-priority job when it resumes after the suspension.
- If each job in a task T_i can self-suspend for a maximum number K_i times after it starts execution, its total blocking time b_i is given by

$$b_i = b_i(ss) + (K_i + 1)b_i(np)$$

Outline

- Practical Factors
  1. Nonpreemptability
  2. Self-Suspension
  3. ***Context Switches***
  4. Limited-Priority Levels
  5. Tick Scheduling
  6. Varying Priority in Fixed-Priority Systems
  7. Schedulability Test of Hierarchically Scheduled Periodic Tasks

Context Switches

- Include the context-switch time in the execution time of the preempting job.
- Let CS denote the context-switch time of the system: the maximum amount of time the system spends per context switch.
- If each job in any task T_i can self-suspend a maximum of K_i times after its execution starts, we add $2(K_i + 1)CS$ to the execution time e_i.

Outline

- Practical Factors
  1. Nonpreemptability
  2. Self-Suspension
  3. Context Switches
  4. ***Limited-Priority Levels***
  5. Tick Scheduling
  6. Varying Priority in Fixed-Priority Systems
  7. Schedulability Test of Hierarchically Scheduled Periodic Tasks

Limited-Priority Levels

- A real-life system can support only a limited number Ω_s of priority levels.
  - The IEEE 802.5 token ring provides only 8 priority levels
  - Real-time operating systems provide no more than 256 priority levels

➡ Tasks (or jobs) may have non-distinct priorities!

Time Demand Functions of Fixed-Priority Tasks with Nondistinct Priorities (1/2)

- When tasks in a fixed-priority system have nondistinct priorities, a job $J_{i,j}$ of T_i may delayed by a job of an equal priority task T_k.
- Let $\mathbf{T}_E(i)$ denote the subset of tasks, other than T_i, that have the same priority as T_i.
- Let $\mathbf{T}_H(i)$ denote the subset of tasks that have higher priorities than T_i.
- The delay suffered by $J_{i,j}$ due to equal priority tasks is at most equal to the sum of the execution times of all the tasks in $\mathbf{T}_E(i)$.

Time Demand Functions of Fixed-Priority Tasks with Nondistinct Priorities (2/2)

- Time-Demand Function ($D_k \leqq p_k$, for all k)

$$w_i(t) = e_i + b_i + \sum_{T_k \in \mathbf{T}_E(i)} e_k + \sum_{T_k \in \mathbf{T}_H(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k,$$

for $0 < t \leq \min(D_i, p_i)$

- Time-Demand Function ($D_k > p_k$, for some k)

$$w_{i,j}(t) = je_i + b_i + \sum_{T_k \in \mathbf{T}_E(i)} \left(\left\lceil \frac{(j-1)p_i}{p_k} \right\rceil + 1 \right) \cdot e_k + \sum_{T_k \in \mathbf{T}_H(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k,$$

for $(j-1)p_i < t \leq w_{i,j}(t)$.

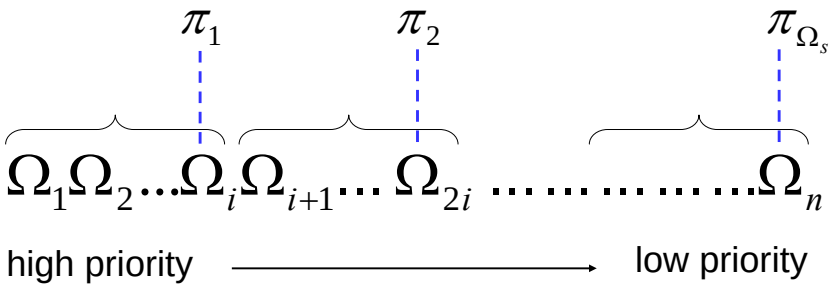
Schedulability Loss of Fixed-Priority Systems

- Whenever the number Ω_n of priorities assigned to tasks (called assigned priorities) by a fixed-priority scheduling algorithm is larger than the number Ω_s of priority levels supported by the system, the Ω_n assigned priorities must be mapped onto Ω_s system priorities.
- The performance of the scheduling algorithm critically depends on this mapping.

Notations

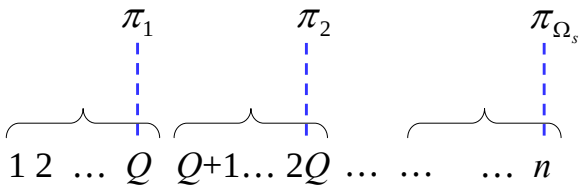
- The assigned priorities: 1, 2, …, Ω_n
- The system priorities: $\pi_1, \pi_2, ..., \pi_{\Omega_s}$, where π_k $(1 \le k \le \Omega_s)$ is a positive integer in the range $[1, \Omega_n]$ and π_j is less than π_k if $j < k$.
- The set $\{\pi_1, \pi_2, ..., \pi_{\Omega_s}\}$ is a priority grid. We map the assigned priorities onto this grid so that all the assigned priorities equal to or higher than π_1 are mapped to π_1 and all assigned priorities in the range $(\pi_{k-1}, \pi_k]$ are mapped to the system priority π_k for $1 < k \le \Omega_s$.

Illustration



Uniform Mapping

- According to this method, the priority grid is uniformly distributed in the range of the assigned priorities.
- Let Q denote the integer $\lfloor \Omega_n / \Omega_s \rfloor$. $\pi_k = kQ$ for $k = 1, 2, ..., \Omega_s - 1$ and π_{Ω_s} is equal to Ω_n.



Example

- Suppose that a system has 10 tasks whose assigned priorities are equal to 1, 2, …, 10. Suppose that a system can support only three priority levels. According to uniform mapping, the system priorities $\pi_1 = ?, \pi_2 = ?,$ and $\pi_3 = ?$

Constant Ratio Mapping

- This method keeps the ratios of $(\pi_{i-1} + 1)/\pi_i$, for $i = 2, 3, ..., \Omega_s$ as equal as possible.

➡ There are more system priority levels at the higher-priority end of the assigned priority range than at the lower-priority end.

Example

- Suppose that a system has 10 tasks whose assigned priorities are equal to 1, 2, …, 10. Suppose that a system can support only three priority levels. According to constant ratio mapping, the system priorities $\pi_1 = ?, \pi_2 = ?$ and $\pi_3 = ?$

Relative Schedulability (1/2)

- Let g denote the minimum of the grid ratios, that is, $g = \min_{2 \le i \le \Omega_s} ((\pi_{i-1} + 1)/\pi_i)$.
- Lehoczky and Sha showed that when
  - Constant ratio mapping is used
  - n is large
  - $D_i = p_i$ for all i

  Schedulable Utilization of the RM algorithm =

$$\begin{cases} \ln(2g) + 1 - g & g > 1/2 \\ g & g \le 1/2 \end{cases}$$

References Lehoczky, J. P., and L. Sha, "Performance of Real-Time Bus Scheduling Algorithms," *ACM Performance Evaluation Review*, Vol. 14, May 1986.

Relative Schedulability (2/2)

- The ratio of this schedulable utilization to $\ln 2$ is the relative schedulability.
- It measures the deterioration in schedulability due to an insufficient number of system priorities.
- For a system containing 100,000 tasks, the relative schedulability is equal to 0.9986 when system priorities is equal to 256.
- ➡ 256 system priority levels are sufficient even for the most complex rate-monotonically scheduled systems.

Outline

- Practical Factors
  1. Nonpreemptability
  2. Self-Suspension
  3. Context Switches
  4. Limited-Priority Levels
  5. ***Tick Scheduling***
  6. Varying Priority in Fixed-Priority Systems
  7. Schedulability Test of Hierarchically Scheduled Periodic Tasks

Terminologies

- A scheduler is event-driven if upon the occurrence of every scheduling event, the scheduler executes immediately.
- A scheduler is time-driven if the execution of the scheduler is triggered by a timer which is set to expire periodically.
  - Scheduling decisions are made at these time instants, called clock interrupts.
  - This method is called tick scheduling or time-based scheduling.

Impacts of Tick Scheduling (1/2)

- A job is ready may not be noticed and acted upon by the scheduler until the next clock interrupt.
- ➡ The delayed action of the scheduler may delay the completion of the job.
- A ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue.
- ➡ The pending (job) queue holds the jobs that have been released or unblocked since the last clock interrupt.

Impacts of Tick Scheduling (2/2)

- When the scheduler executes, it moves the jobs in the pending queue to the ready job queue and places them there in order of their priorities.
- ➡ The time the scheduler takes to move the jobs introduces additional scheduling overhead.

Model of Scheduler

- We can model the scheduler as a periodic task T_0 whose period is p_0:
  - Let p_0 denote the tick size, that is, the length of time between consecutive clock interrupts.
  - Its execution time e_0 is the amount of time the scheduler takes to service the clock interrupt.
  - ➡ This time is spent even when there is no job in the pending job queue.
  - Let CS_0 denote the maximum amount of time that the scheduler takes to move a job from the pending queue to the ready job queue.
  - ➡ This overhead occurs each time a job is placed into the ready queue.

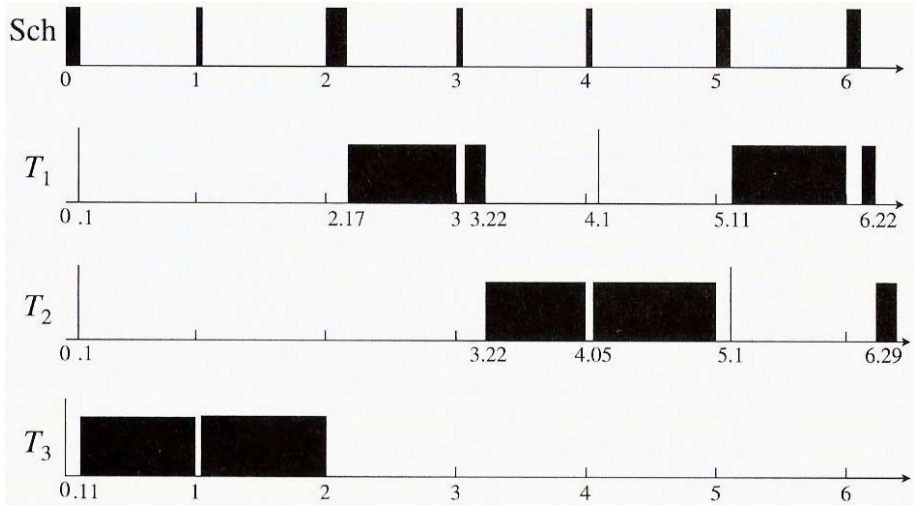
Example (Event-Driven)

- Consider a fixed-priority system consisting of three tasks $T_1 = (0.1, 4, 1)$, $T_2 = (0.1, 5, 1.8)$ and $T_3 = (20, 5)$. The first section of T_3 is nonpreemptable, and the execution time of this section is 1.1. The relative deadlines of the tasks are equal to their respective periods. What are the maximum response times of the tasks?

Example (Time-Driven)

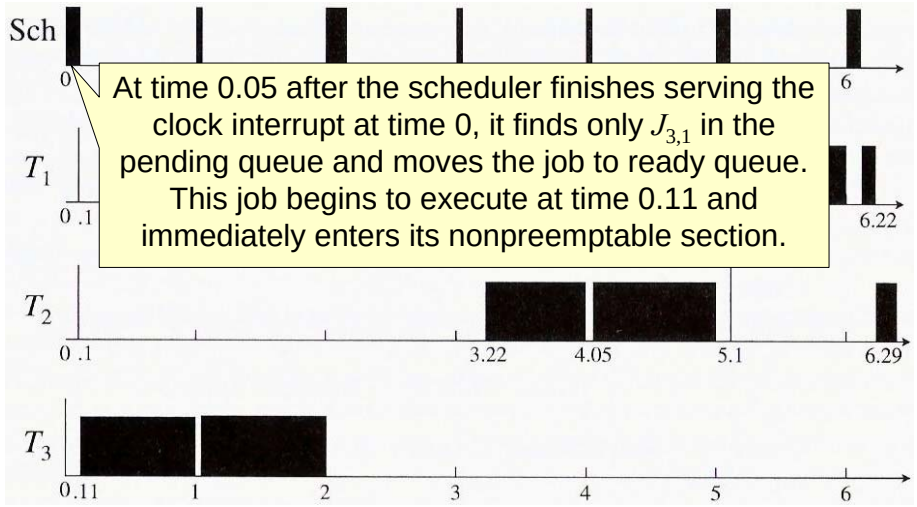
- Now suppose that the scheduler executes only at clock interrupts 0, 1, 2, … (i.e., p_0 is 1). It takes 0.05 unit of time to service a clock interrupt (i.e., e_0 is 0.05) and 0.06 unit of time to move a job from the pending queue to ready queue (i.e., CS_0 is 0.06). What is the schedule?

Effect of Tick Scheduling



T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

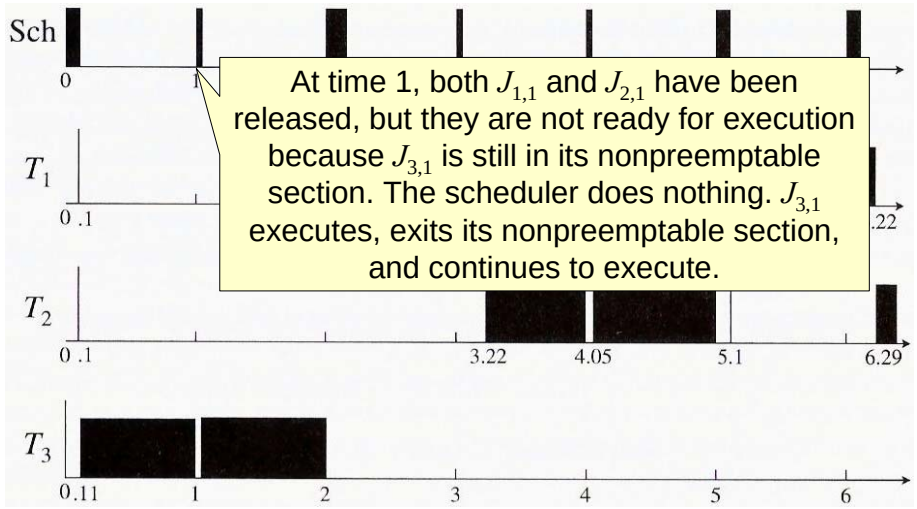
Effect of Tick Scheduling



At time 0.05 after the scheduler finishes serving the clock interrupt at time 0, it finds only $J_{3,1}$ in the pending queue and moves the job to ready queue. This job begins to execute at time 0.11 and immediately enters its nonpreemptable section.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

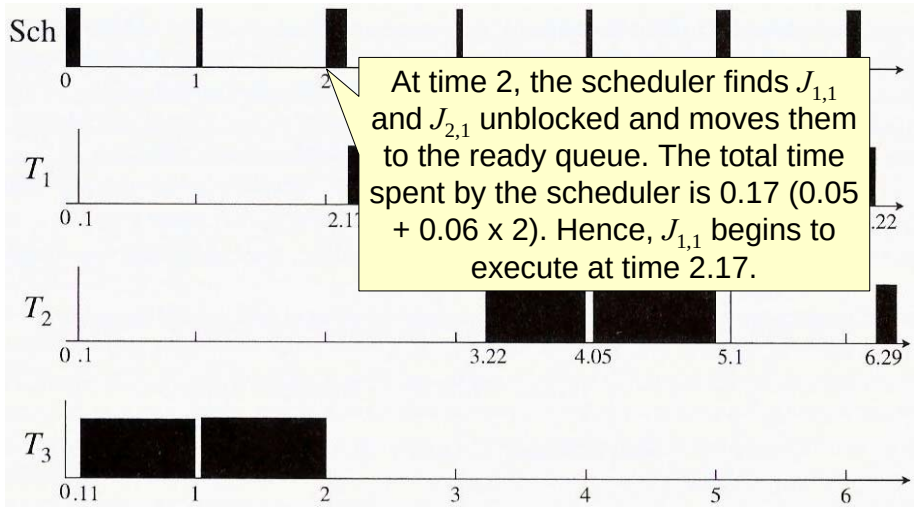
Effect of Tick Scheduling



At time 1, both $J_{1,1}$ and $J_{2,1}$ have been released, but they are not ready for execution because $J_{3,1}$ is still in its nonpreemptable section. The scheduler does nothing. $J_{3,1}$ executes, exits its nonpreemptable section, and continues to execute.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

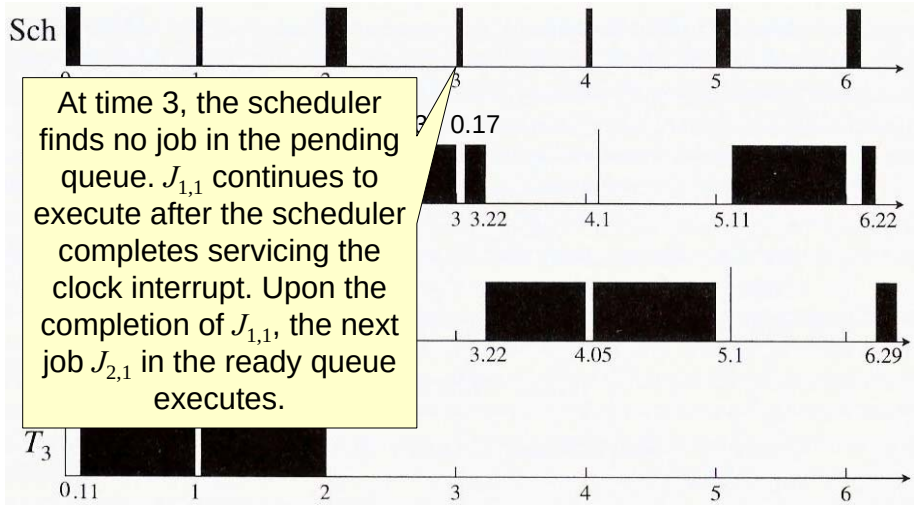
Effect of Tick Scheduling



At time 2, the scheduler finds $J_{1,1}$ and $J_{2,1}$ unblocked and moves them to the ready queue. The total time spent by the scheduler is 0.17 (0.05 + 0.06 x 2). Hence, $J_{1,1}$ begins to execute at time 2.17.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

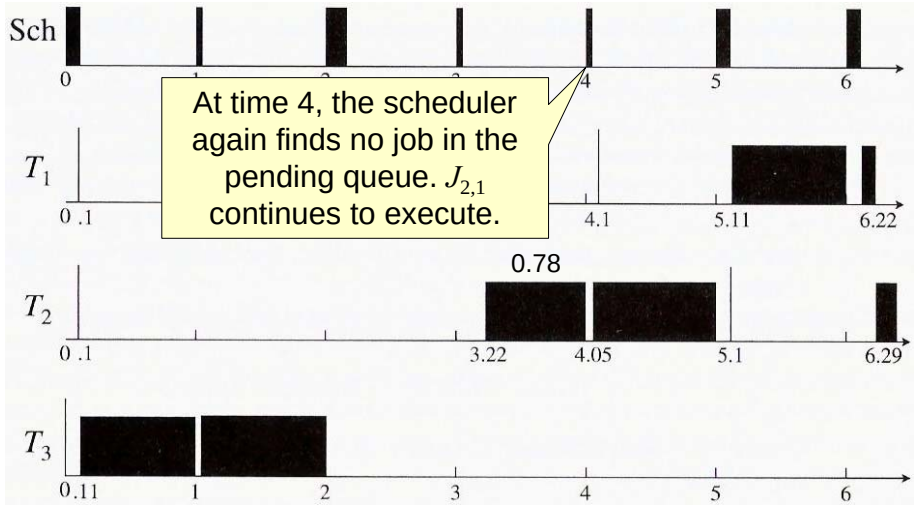
Effect of Tick Scheduling



At time 3, the scheduler finds no job in the pending queue. $J_{1,1}$ continues to execute after the scheduler completes servicing the clock interrupt. Upon the completion of $J_{1,1}$, the next job $J_{2,1}$ in the ready queue executes.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

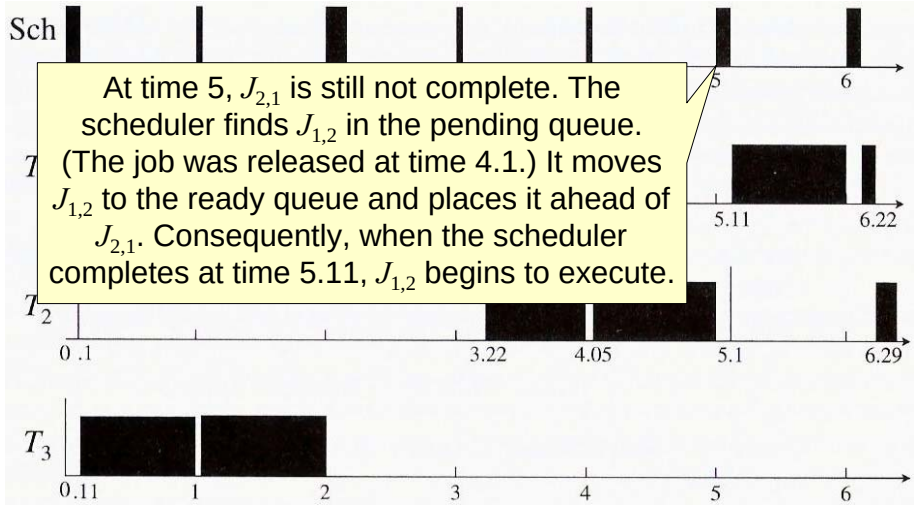
Effect of Tick Scheduling



At time 4, the scheduler again finds no job in the pending queue. $J_{2,1}$ continues to execute.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

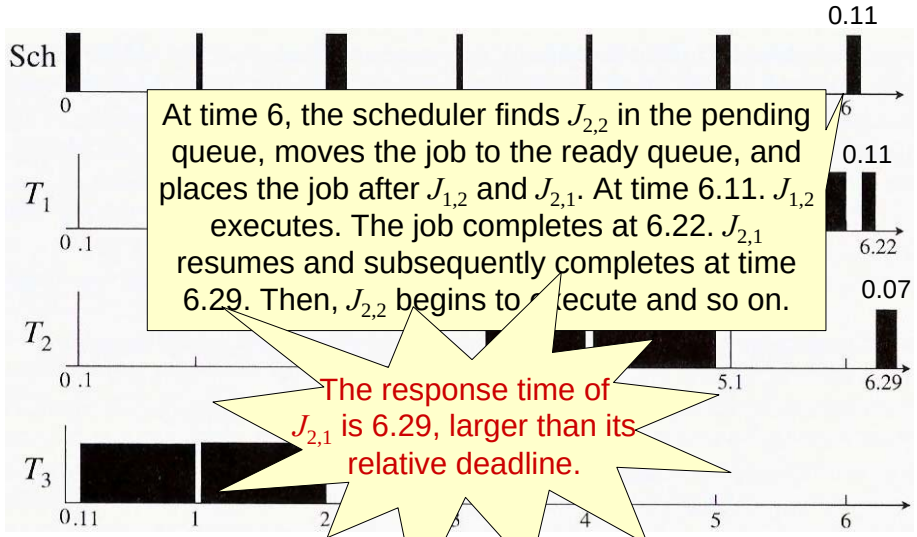
Effect of Tick Scheduling



At time 5, $J_{2,1}$ is still not complete. The scheduler finds $J_{1,2}$ in the pending queue. (The job was released at time 4.1.) It moves $J_{1,2}$ to the ready queue and places it ahead of $J_{2,1}$. Consequently, when the scheduler completes at time 5.11, $J_{1,2}$ begins to execute.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5); p_0 =1, e_0 = 0.05. and CS_0 = 0.06

Effect of Tick Scheduling



At time 6, the scheduler finds $J_{2,2}$ in the pending queue, moves the job to the ready queue, and places the job after $J_{1,2}$ and $J_{2,1}$. At time 6.11. $J_{1,2}$ executes. The job completes at 6.22. $J_{2,1}$ resumes and subsequently completes at time 6.29. Then, $J_{2,2}$ begins to execute and so on.

T_1 = (0.1, 4, 1), T_2 = (0.1, 5, 1.8), T_3 = (20, 5), p_0 =1, e_0 = 0.05. and CS_0 = 0.06

Fixed-Priority Systems (1/2)

- We can take into account the additional time demand introduced by tick scheduling in a fixed-priority system by using the following modified task parameters in the computation of the time-demand function of task T_i:
  1. Include the task $T_0 = (p_0, e_0)$ in the set of higher-priority tasks;
  2. Add $(K_k + 1)CS_0$ to the execution time e_k of every higher-priority task T_k (i.e., for $k = 1, 2, \ldots, i$), where K_k is the number of times T_k may self-suspend;
  3. For every lower-priority task T_k, $k = i + 1, \ldots, n$, add a task (p_k, CS_0) in the set of higher-priority tasks;

Fixed-Priority Systems (2/2)

4. Make the blocking time $b_i(np)$ due to nonpreemptability of T_i equal to

$$\left(\left\lceil \max_{i+1 \le k \le n} \theta_k / p_0 \right\rceil + 1\right) p_0$$

where θ_k is the maximum execution time of nonpreemptable sections of the lower-priority task T_k.

A job in T_i may suffer up to p_0 units of delay waiting in the pending queue each time when it becomes ready, and we can treat this delay as a factor of its blocking time.

Apply to Previous Example (1/2)

- $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 5)$; $p_0 = 1$, $e_0 = 0.05$, and $CS_0 = 0.06$
- We compute the time-demand function of T_2 as if the system has the following five tasks:
  - $T_0 = (1, 0.05)$ (Rule 1)
  - $T_{0'} = (20, 0.06)$ (Rule 3)
  - $T_1 = (4, 1.06)$ (Rule 2)
  - $T_2 = (5, 1.86)$ (Rule 2)
  - $T_3 = (20, 5)$
- T_0 and $T_{0'}$ have higher priorities than the other tasks.

Apply to Previous Example (2/2)

- $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 5)$; $p_0 = 1$, $e_0 = 0.05$, and $CS_0 = 0.06$, nonpreemptable section of T_3 is 1.1.
- What is the maximum possible response time of T_1, T_2, and T_3?

$$b_i(np) = \left(\left\lceil \max_{i+1 \le k \le n} \theta_k / p_0 \right\rceil + 1\right) p_0$$

$$w_i(t) = e_i + b_i + \sum_{\pi_k > \pi_i} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k, \quad \text{for } 0 < t \le \min(D_i, p_i)$$

Conclusion of Tick Scheduling

- The major source of inaccuracy is the seemingly loose bound on the blocking time. (But this bound cannot be tightened in general.)
- By choosing a smaller p_0, we can reduce the extra blocking time introduced by tick scheduling at the expense of higher overhead for servicing clock interrupt.

Dynamic-Priority Systems

- We can take into account of the effect of tick scheduling in a dynamic-priority system in a similar way. We modify the task parameters as follows:
  1. Add the task $T_0 = (p_0, e_0)$.
  2. Add $(K_k + 1)CS_0$ to the execution time e_k of every task T_k for $k = 1, 2, \ldots, n$.
  3. Make the blocking time $b_i(np)$ due to nonpreemptability of T_i equal to
  $$\left(\left\lceil \max_{i+1 \le k \le n} \theta_k / p_0 \right\rceil + 1\right) p_0$$
  where θ_k is the maximum execution time of nonpreemptable sections of a task T_k whose relative deadline is larger than the relative deadline of T_i. (Hint)

Outline

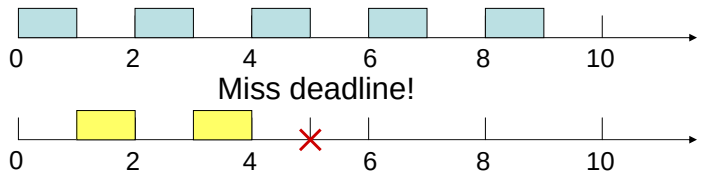
- Practical Factors
  1. Nonpreemptability
  2. Self-Suspension
  3. Context Switches
  4. Limited-Priority Levels
  5. Tick Scheduling
  6. ***Varying Priority in Fixed-Priority Systems***
  7. Schedulability Test of Hierarchically Scheduled Periodic Tasks

Varying Priority

- In general, each job in a task T_i may have more than one segment and the segments may have different priorities.
  - When tasks contend for resources, we sometimes raise the priority of a job segment during which the job holds some nonpreemptable resource in order to speed up the release of the resource.
  - Sometimes, raising the priority of a job segment is a way to make the job, and hence the task containing it, schedulable.

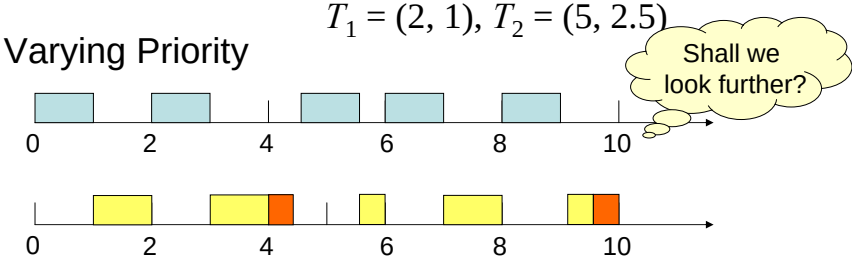
Example

Fixed Priority



$T_1 = (2, 1), T_2 = (5, 2.5)$

Varying Priority

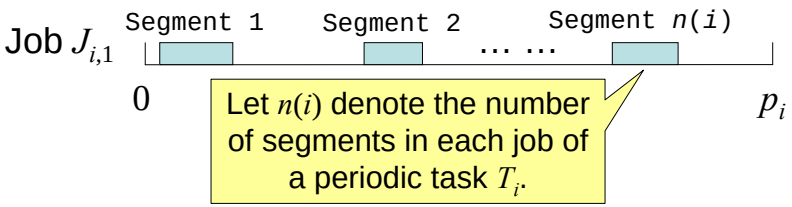


Assumptions

- Jobs do not suspend themselves.
- Every job can be preempted at any time.
- Context-switch overhead is negligible.
- Jobs in each task are scheduled on a FIFO basis.
- Before a job completes, the subsequent jobs in the same task do not compete for processor time with jobs in other tasks.

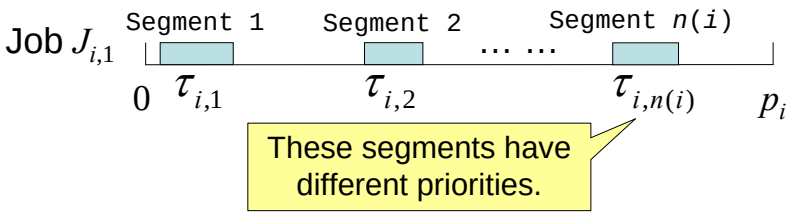
Subtasks (1/4)

Task T_i

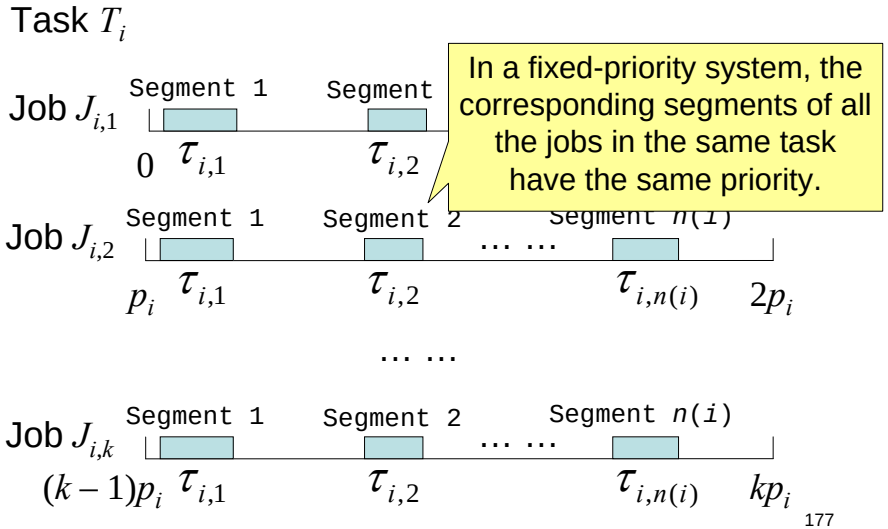


Subtasks (2/4)

Task T_i



Subtasks (3/4)



Subtasks (4/4)

- It is convenient to think of each such task T_i as composed of $n(i)$ subtasks $T_{i,1}$, $T_{i,2}$, …, $T_{i,n(i)}$.
- A job in $T_{i,k}$ is the k-th segment of a job in T_i.
- The j-th job in $T_{i,k}$ is ready when the j-th job in $T_{i,k-1}$ completes, for all $1 < k \leq n(i)$.
- The maximum execution times of the subtasks are $e_{i,1}$, $e_{i,2}$, …, $e_{i,n(i)}$, and the sum of these execution times is e_i.
- The relative deadlines of the subtasks of T_i are $D_{i,1}$, $D_{i,2}$, …, $D_{i,n(i)}$, and $D_{i,1} \leq D_{i,2} \leq ... \leq D_{i,n(i)} = D_i$

Canonical Form (1/2)

- A task is in canonical form if every later subtask has a higher priority than its immediate predecessor subtask.
- A task that is not in canonical form can be transformed into canonical form as follows:
  1. Starting with the last subtask.
  2. Lower the priority of each immediate predecessor to the priority of the immediate successor if the given priority of the immediate predecessor is higher.
  3. Otherwise, leave the priority unchanged.

Canonical Form (2/2)

  4. Repeat this process until the priority of the first subtask is determined.
  5. If the priorities of adjacent subtasks become the same, combine these subtasks into one subtask and decrement the number of subtasks accordingly.
- **Example**: Suppose that the task T_i has four subtasks with priorities 4, 1, 3, and 2, respectively. Transform it into canonical form.

Interference Block

- An interference block of a subtask $T_{i,k}$ is a chain of one or more contiguous subtasks $T_{l,x}$, $T_{l,x+1}$, …, $T_{l,y}$ in another task T_l, for some $l \neq i$ and $y \geq x$, that have the following properties.

  (1) $\pi_{l,x}, \pi_{l,x+1}, ..., \pi_{l,y} \geq \pi_{i,k}$

  (2) $\pi_{l,x-1}$ (if any) $< \pi_{i,k}$

  (3) $\pi_{l,y+1}$ (if any) $< \pi_{i,k}$

Extended General Time-Demand Analysis

- Transforming the Target Task
- Identifying the Interference Blocks
- Computing the Length of Level-$\pi_{i,1}$ Busy Interval
- Maximum Response Time of $T_{i,1}$
- Maximum Response Times of Later Subtasks
- Maximum Response Time W_i
- The Procedure
- An Example

Transforming the Target Task

- We first transform the target task into canonical form if the task is not already in the form.
- The maximum response time of the transformed target task is no less than the maximum response time of the given task. (sufficient test)
- Hereafter, by subtasks $T_{i,1}$, $T_{i,2}$, …, $T_{i,n(i)}$, we mean the subtasks of the transformed target task.

Identifying the Interference Blocks

- Let h_l denote the number of interference blocks in T_l and $E_{l,x}$ denote the sum of the execution times of all the subtasks in the x-th interference block in T_l.
- The values of h_l and $E_{l,x}$ are different for different target subtasks.

Computing the Length of Busy Interval (1/6)

- With respect to $T_{i,1}$, we partition the other tasks in the system into the following disjoint subsets according to the priorities of their subtasks:
  - $\mathbf{L}(1)$: A task is in this subset if it contains no interference block of the target subtask $T_{i,1}$.
  - $\mathbf{H}(1)$: A task is in this subset if all its subtasks form a single interference block of $T_{i,1}$.
  - $\mathbf{H/L}(1)$: A task is in this subset if it contains at least one interference block of $T_{i,1}$, and either its first subtask or its last subtask or both are not in an interference block.
  - $\mathbf{HLH}(1)$: A task is in this subset if its first and last subtasks are in two different interference blocks.

Computing the Length of Busy Interval (2/6)

- Suppose the priority of $T_{i,1}$ is 3:

$\mathbf{L}(1)$	{2, 1, 3}
	{3, 1, 2, 4, 5}
$\mathbf{H}(1)$	{5, 1, 2}
$\mathbf{H/L}(1)$	{6, 7, 8}
$\mathbf{HLH}(1)$	{1, 2, 7, 1}
	{8, 1, 5}

Computing the Length of Busy Interval (3/6)

- Let t_0 denote the beginning of a level-$\pi_{i,1}$ busy interval.
- Since no job of tasks in $\mathbf{L}(1)$ can execute after t_0, we can ignore this subset of tasks.
- Every task T_l in $\mathbf{H}(1)$ may repeatedly preempt the subtask $T_{i,1}$ and may demand $\lceil t / p_l \rceil \cdot e_l$ units of processor time in the interval $(t_0, t_0 + t]$ for any $t \geq 0$.
- At most one interference block of each task T_l in the subset $\mathbf{H/L}(1)$ can execute in the interval $(t_0, t_0 + t]$.

Computing the Length of Busy Interval (4/6)

- If the x-th $(1 \leq x \leq h_l - 1)$ interference block in T_l in the subset $\mathbf{HLH}(1)$ is ready at time t_0, the amount of processor time demanded by the task in $(t_0, t_0 + t]$ is no more than $\max_{1 \leq x \leq h_l - 1} E_{l,x}$.
- If the last interference block is ready at time t_0, the amount of processor time demand by T_l can be as large as $E_{l,h_l} + E_{l,1}$.

Computing the Length of Busy Interval (5/6)

- The maximum amount of processor time demanded by all the tasks in the $\mathbf{H/L}(1)$ and $\mathbf{HLH}(1)$ is equal to

$$a(1) = \sum_{T_l \in \mathbf{H/L}(1)} \max_{1 \le x \le h_l} E_{l,x} + \sum_{T_l \in \mathbf{HLH}(1)} \max\left(\max_{2 \le x \le h_l - 1} E_{l,x}, E_{l,1} + E_{l,h_l} \right)$$

- Let B_i denote the maximum length of a level-$\pi_{i,1}$ busy interval. B_i is the minimum solution of the equation

$$t = b_i + a(1) + \lceil t / p_i \rceil \cdot e_i + \sum_{T_l \in \mathbf{H}(1)} \lceil t / p_l \rceil \cdot e_l$$

where b_i is the blocking time of T_i.

Computing the Length of Busy Interval (6/6)

- We can solve above equation iterative starting from the initial value $b_i + e_i + a(1)$.
- Let N_i denote the number of jobs in each busy interval.

$$N_i = \lceil B_i / p_i \rceil$$

Maximum Response Time of $T_{i,1}$

- The time-demand function $w_{i,1;j}(t)$ of the j-th job of $T_{i,1}$ in a level-$\pi_{i,1}$ busy interval is given by

$$w_{i,1;j}(t) = b_i + e_{i,1} + (j-1)e_i + a(1) + \sum_{T_l \in \mathbf{H}(1)} \left\lceil \frac{t}{p_l} \right\rceil \cdot e_l$$

- We use $f_{i,k;j}$ to denote the latest completion time of the j-th job in the subtask $T_{i,k}$, for $k = 1, 2, \ldots, n(i)$, in a level-$\pi_{i,1}$ busy interval.
- $f_{i,1;j}$ is the minimum solution of the equation

$$t = w_{i,1;j}(t).$$

Maximum Response Times of Later Subtasks (1/4)

- After finding the time-demand function $w_{i,k-1;j}(t)$ and the latest completion time $f_{i,k-1;j}$ of the j-th job in the subtask $T_{i,k-1}$ for each $k > 1$, we then find the time-demand function $w_{i,k;j}(t)$ and the latest completion time $f_{i,k;j}$ of the j-th job of $T_{i,k}$.
- Because $T_{i,k}$ has a higher priority than $T_{i,k-1}$, some tasks in $\mathbf{H}(k-1)$ may contain subtasks with lower priorities than the current target subtask $T_{i,k}$.

➡ These tasks cannot repeatedly preempt $T_{i,k}$.

Maximum Response Times of Later Subtask

How about $\mathbf{H/L}(k-1)$ and $\mathbf{HLH}(k-1)$?

- We further partition the subset $\mathbf{H}(k-1)$ into the following disjoint subsets:
  - $\mathbf{H}(k)$: A task is in this subset if all its subtasks form a single interference block of $T_{i,k}$.
  - $\mathbf{H/L}(k)$: A task is in this subset if it contains at least one interference block of $T_{i,k}$, and either its first subtask or its last subtask or both are not in an interference block.
  - $\mathbf{HLH}(k)$: A task is in this subset if its first and last subtasks are in two different interference blocks of the subtask $T_{i,k}$.
  - $\mathbf{L}(k)$: A task is in this subset if all its subtasks have priorities lower than $\pi_{i,k}$.

Maximum Response Times of Later Subtasks (3/4)

- The total processor time demanded by all the tasks in the subsets $\mathbf{H/L}(k)$ and $\mathbf{HLH}(k)$ after the completion time $f_{i,k-1;j}$ of the intermediate predecessor job is given by

$$a(k) = \sum_{T_l \in \mathbf{H/L}(k)} \max_{1 \le x \le h_l} E_{l,x} + \sum_{T_l \in \mathbf{HLH}(k)} \max\left(\max_{2 \le x \le h_l - 1} E_{l,x}, E_{l,1} + E_{l,h_l} \right)$$

where h_l and $E_{l,x}$ are parameters of interference blocks of $T_{i,k}$.

Maximum Response Times of Later Subtasks (4/4)

- The time-demand function $w_{i,k;j}(t)$ of the j-th job in the target subtask $T_{i,k}$ is given by

$$w_{i,k;j}(t) = b_i + (j-1)e_i \qquad J_{i,1;j}, J_{i,2;j}, \ldots, J_{i,k;j}$$
$$+ \sum_{x=1}^{k} \left[e_{i,x} + a(x) + \sum_{T_l \in \mathbf{H}(x-1) - \mathbf{H}(x)} \left\lceil \frac{f_{i,x-1;j}}{p_l} \right\rceil \cdot e_l \right] + \sum_{T_l \in \mathbf{H}(k)} \left\lceil \frac{t}{p_l} \right\rceil \cdot e_l$$

where the completion time $f_{i,0;j}$ of j-th job of an nonexisting subtask $T_{i,0}$ is 0 and $\mathbf{H}(0)$ is $\mathbf{T}$.

To make the equation general.

Maximum Response Time W_i

- The maximum response time W_i of the target task is given by

$$W_i = \max_{1 \le j \le N_i} (f_{i,n(i);j} - (j-1)p_i)$$

maximum number of jobs of T_i in the busy interval

The Procedure (1/2)

1. Transform the target task T_i into canonical form. After the transformation, its subtasks $T_{i,k}$ for $k = 1, 2, \ldots, n(i)$ have increasing priorities.
2. Compute the length B_i of the longest level- $\pi_{i,1}$ busy interval and maximum number N_i jobs of T_i in the busy interval.

The Procedure (2/2)

3. For $j = 1, 2, \ldots, N_i$, compute the latest completion time of $f_{i,n(i);j}$ of the j-th job in T_i. This is done by computing the latest completion time $f_{i,k;j}$ of the j-th job in each subtasks $T_{i,k}$ of T_i, for $k = 1, 2, \ldots, n(i)$, starting from the first subtask in precedence order until the last subtask.
4. Compute an upper bound W_i of the maximum response time of T_i.

Example (1/8)

Subtask	p_i	D_i	$e_{i,k}$	$\pi_{i,k}$
$T_{1,1}$	7	7	2.5	1
$T_{2,1}$	10	15	0.9	5
$T_{2,2}$	10	15	1.6	2
$T_{3,1}$	12	20	0.6	2
$T_{3,2}$	12	20	0.1	6
$T_{3,3}$	12	20	0.7	2
$T_{3,4}$	12	20	0.1	6
$T_{3,5}$	12	20	0.5	3
$T_{4,1}$	15	15	1.0	2
$T_{4,2}$	15	15	0.5	1
$T_{4,3}$	15	15	1.5	4

Example (2/8)

- Compute the maximum response time W_2 of the task T_2 in the fixed-priority system whose parameters are listed in the previous slide. The blocking time of all the tasks is 0.

- T_2 has two subtasks, $T_{2,1}$ and $T_{2,2}$.
  - The task is already in canonical form
  - We proceed to compute the maximum length B_2 of a level-5 busy interval

Example (3/8)

- Comparing with $\pi_{2,1} = 5$, we divide the other three tasks into subsets.
  - $\mathbf{L}(1)$
  - $\mathbf{H/L}(1)$
  - $\mathbf{H}(1)$
  - $\mathbf{HLH}(1)$
- $a(1)$ = ?

Example (4/8)

- To compute B_2 and N_2:

$$\begin{cases} t = b_i + a(1) + \lceil t / p_i \rceil \cdot e_i + \sum_{T_l \in \mathbf{H}(1)} \lceil t / p_l \rceil \cdot e_l \\ N_i = \lceil B_i / p_i \rceil \end{cases}$$

Example (5/8)

- To find the maximum completion time $f_{2,1;1}$ of the first job in $T_{2,1}$ in a level-5 busy interval:

$$w_{i,1;j}(t) = b_i + e_{i,1} + (j-1)e_i + a(1) + \sum_{T_l \in \mathbf{H}(1)} \left\lceil \frac{t}{p_l} \right\rceil \cdot e_l$$

Example (6/8)

- Examine the subset $\mathbf{H}(1)$ and partition it into subsets:

Example (6/8)

- Examine the subset $\mathbf{H}(1)$ and partition it into subsets:

- Compute the first job of $T_{2,2}$ in a level-5 busy interval:

$$\begin{cases} a(k) = \sum_{T_l \in \mathbf{H/L}(k)} \max_{1 \le x \le h_l} E_{l,x} + \sum_{T_l \in \mathbf{HLH}(k)} \max\left(\max_{2 \le x \le h_l - 1} E_{l,x}, E_{l,1} + E_{l,h_l} \right) \\ w_{i,k;j}(t) = b_i + (j-1)e_i + \sum_{x=1}^{k} \left[e_{i,x} + a(x) + \sum_{T_l \in \mathbf{H}(x-1) - \mathbf{H}(x)} \left\lceil \frac{f_{i,x-1;j}}{p_l} \right\rceil \cdot e_l \right] + \sum_{T_l \in \mathbf{H}(k)} \left\lceil \frac{t}{p_l} \right\rceil \cdot e_l \end{cases}$$

Example (7/8)

- Compute the maximum completion time $f_{2,1;2}$ of the second job in $T_{2,1}$:

- Compute the maximum completion time $f_{2,2;2}$ of the second job in $T_{2,2}$:

Example (8/8)

- The upper bound $f_{2,2;2} = 23.6$ is a loose one. Since the first level-5 busy interval ends at time 19.6, $f_{2,2;2}$ is surely no greater than 19.6. Why?

Example (8/8)

- The upper bound $f_{2,2;2} = 23.6$ is a loose one. Since the first level-5 busy interval ends at time 19.6, $f_{2,2;2}$ is surely no greater than 19.6. Why?
  - The above expression of $w_{2,2;2}(t)$ tells us that the source of inaccuracy is the inclusion of $a(2) = 1.5$, which is the execution time of the interference block $\{T_{4,1}, T_{4,2}\}$ in $\mathbf{H/L}(2)$.
  - In this case, this time is already included in the term $3 \times \lceil 18/15 \rceil$ and is therefore counted twice.
- The term $a(x)$ must be included to ensure the correctness of the bound, e.g., $f_{2,1;2} = 15$.

Outline

- Practical Factors
  1. Nonpreemptability
  2. Self-Suspension
  3. Context Switches
  4. Limited-Priority Levels
  5. Tick Scheduling
  6. Varying Priority in Fixed-Priority Systems
  7. ***Schedulability Test of Hierarchically Scheduled Periodic Tasks***

Hierarchical Scheduling Schemes

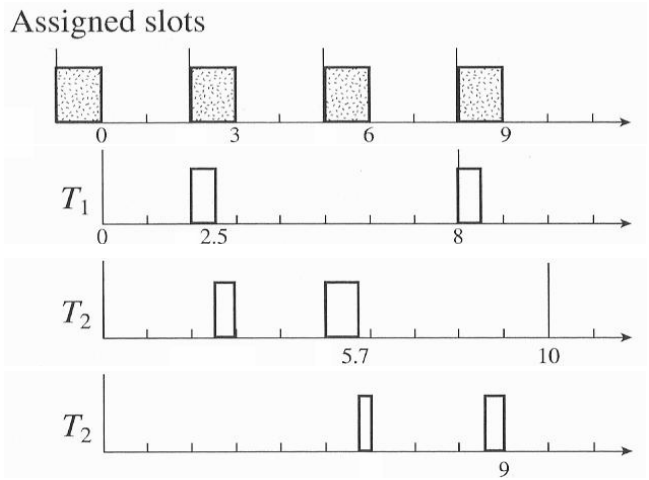
- The scheduling scheme may be hierarchical.
- The scheduler may use one approach to schedule individual subsystems on the processor and use another approach to divide the time given to each subsystem among the tasks in the subsystem.
- Two common hierarchical scheduling schemes:

	Priority-Driven / Round-Robin	Fixed-Time Partitioning
Subsystems	Priority-Driven	Cyclic
Tasks	Round Robin	Some Algorithms

Fixed-Time Partitioning Systems

- The cyclic scheduler at the low level partitions the time into intervals of a fixed length *RL*.
  - Each interval is a round.
  - The round length *RL* must be smaller than the minimum period and the minimum relative deadline of all tasks in the system.
- We consider a subsystem that has n independent, preemptable periodic tasks and is given a slot of length τ in each round by the cyclic scheduler.

Example Illustrating the Fixed-Time Partition Scheme



$T_1 = (\phi_1, 8, 0.5), T_2 = (\phi_2, 10, 1.2), T_3 = (\phi_3, 14, 0.8)$

Schedulability of Fixed-Priority Tasks (1/3)

- The time-demand function: (pp.142)

$$w_i(t) = e_i + b_i + \sum_{T_k \in \mathbf{T}_E(i)} e_k + \sum_{T_k \in \mathbf{T}_H(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k,$$

for $0 < t \le \min(D_i, p_i)$

$$w_{i,j}(t) = je_i + b_i + \sum_{T_k \in \mathbf{T}_E(i)} \left(\left\lceil \frac{(j-1)p_i}{p_k} \right\rceil + 1 \right) \cdot e_k + \sum_{T_k \in \mathbf{T}_H(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k,$$

for $(j-1)p_i < t \le w_{i,j}(t)$.

Schedulability of Fixed-Priority Tasks (2/3)

- The supply function:

$$supply(t) \le \left\lfloor \frac{t}{RL} \right\rfloor \times \tau + \max\left(0, \tau - RL + t - \left\lfloor \frac{t}{RL} \right\rfloor \times RL\right)$$

Schedulability of Fixed-Priority Tasks (3/3)

$T_1 = (\phi_1, 8, 0.5), T_2 = (\phi_2, 10, 1.2), T_3 = (\phi_3, 14, 0.8); RL = 3, \tau = 1$

- To find the maximum possible response time of task T_3, we solve for the minimum value of t satisfying the equation

$$0.8 + 0.5 \times \left\lceil \frac{t}{8} \right\rceil + 1.2 \times \left\lceil \frac{t}{10} \right\rceil \le \left\lfloor \frac{t}{3} \right\rfloor + \max\left(0, 1 - 3 + t - 3 \times \left\lfloor \frac{t}{3} \right\rfloor\right)$$

$t^{(1)} = 2.5$

$t^{(2)} = \left\lceil t^{(1)} / \tau \right\rceil \times RL = 9.0$