CIS 721 - Real-Time Systems
Lecture 8: Preemption Thresholds

Mitch Neilsen
neilsen@ksu.edu

Outline

- Commonly Used Approaches For Real-Time Scheduling (Ch. 4)
  - Clock-Driven Scheduling (Ch. 5)
  - **Priority-Driven Scheduling**
    - **Periodic Tasks (Ch. 6)**
      - **Priority Assignment**
      - **Preemption Thresholds**
    - Aperiodic and Sporadic Tasks (Ch. 7)

Periodic Task Model

- **Periodic task set:** $\{T_1, \ldots.., T_n\}$, each task consists of a set of **jobs**: $T_i = \{J_{i1}, J_{i2}, \ldots\ldots.\}$
- ϕ_i: p**hase** of task T_i = time when its first job is released
- p_i: p**eriod** of T_i = inter-release time
- e_i or C_i: e**xecution time** of T_i
- u_i: **utilization** of task T_i is given by $u_i = e_i / p_i$
- D_i: (relative) **deadline** of T_i, typically $D_i = p_i$

Preemption Thresholds

- Y. Wang and M. Saksena, “Scheduling Fixed-Priority Tasks with Preemption Threshold”, In Proceedings of the IEEE Intl. Conf. on Real-Time Computing Systems and Applications, Dec. 1999.
- Scheduling with Preemption Thresholds
  - Task Model and Run-Time Model
  - Response Time Analysis
  - Priority and Preemption Threshold Assignment Algorithms
  - Example: ThreadX Real-Time Operating System

Task Model

- Task Set $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$
  - Each task τ_i is characterized by (C_i, T_i, D_i), denoted $\tau_i \sim (C_i, T_i, D_i)$.
  - Each task τ_i is assigned a priority $\pi_i \in \{1,2,..,n\}$
  - and a preemption threshold $\gamma_i \in \{\pi_i, \pi_i + 1,..,n\}$.
- **Notes:**
  - 1 = lowest priority, n = highest priority.
  - π_i = static priority.
  - γ_i = dynamic priority.

Run-Time Model

- Modified fixed-priority, preemptive scheduling.
- When task τ_i is released, it is scheduled using its static priority π_i .
- After task τ_i starts executing, another task τ_j can preempt τ_i only if $\pi_j > \gamma_i \geq \pi_i$.

Extremes

- If $\gamma_i = \pi_i$ for each i, then the result is **preemptive**, priority-based scheduling.
- If γ_i = n (max. priority) for each i, then the result is **non-preemptive**, priority-based scheduling.

Example – assume arbitrary phasing

Task	C_i	T_i	D_i	π_i	**WCRT** Preemptive	**WCRT** Non-Preemptive
τ_1	20	70	50	3	20	**55**
τ_2	20	80	80	2	40	75
τ_3	35	200	100	1	**115**	75

Task	**Priority**	**Preemption Threshold**	**WCRT**
τ_1	3	3	40
τ_2	2	3	75
τ_3	1	2	95

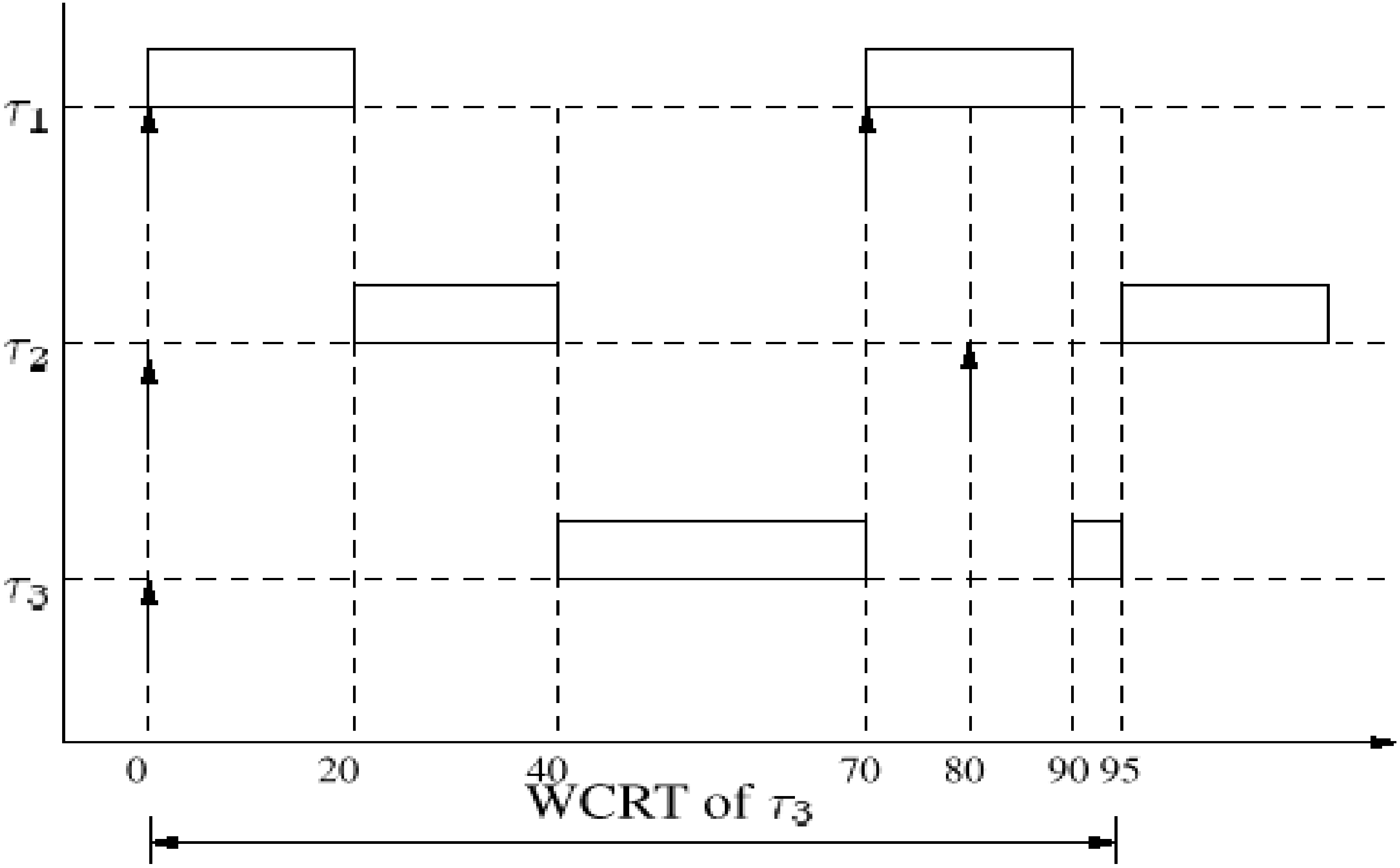


Figure 1. Run-time Behavior with Preemption Threshold

Problem Statement

- Given a task set $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$, determine if there exists an assignment $\{ (\pi_i, \gamma_i) \mid i = 1, 2, \ldots, n \}$ such that Γ is schedulable.
- In other words, determine if there exists an **optimal** assignment of task priorities and preemption thresholds.

Solutions

- **Brute Force** - try all possible assignments of priorities and preemption thresholds.
  - Time Complexity in O(n! n!) => not feasible for large n.
- Use a **Branch and Bound Algorithm** to perform an efficient search for priorities and preemption thresholds.

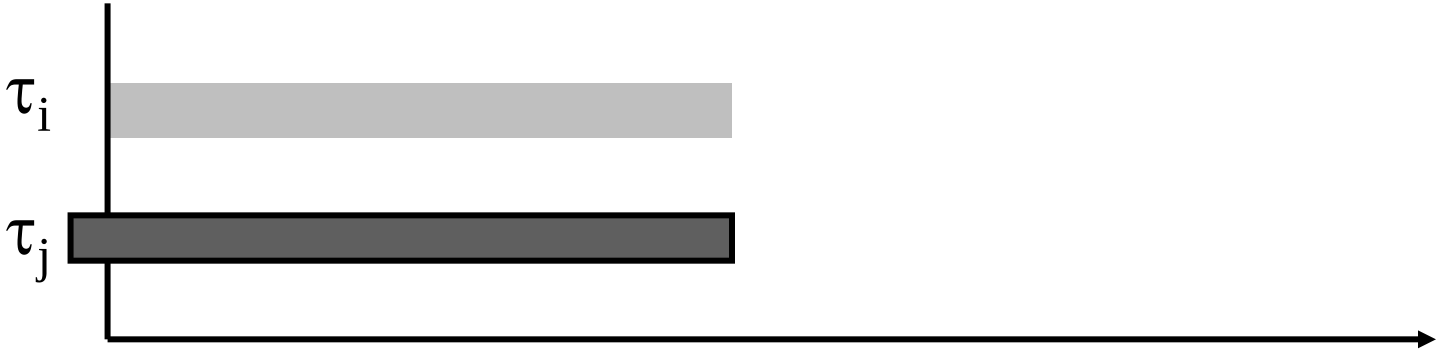
Three Step Process

- **Response Time Analysis**
  - Given assignment { (π_i, γ_i) | i = 1, 2, .. , n }, compute the worst-case response time (R_i or $WCRT_i$) for each task τ_i.
  - A task set Γ is schedulable iff $R_i \leq D_i$ for all i.
- Given a priority assignment { π_i | i = 1, .. , n }, determine a feasible set of **preemption thresholds**, if such a set exists.
- Use a branch and bound algorithm to search for a feasible assignment set of **priorities** (and preemption thresholds).

Response Time Analysis

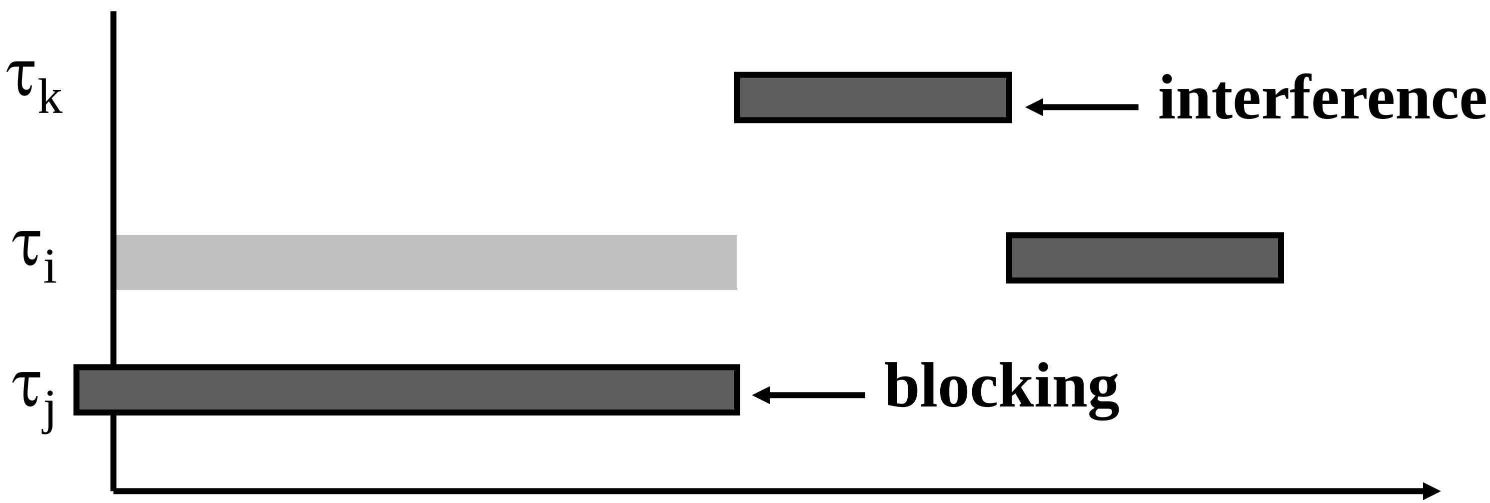
- The **blocking time** of task τ_i is denoted $B(\tau_i)$. Blocking occurs if a lower priority task is running and task τ_i cannot preempt it.

$$B(\tau_i) = \max_j \{ C_j | \gamma_j \geq \pi_i > \pi_j \}$$



Busy Period Analysis

- A **critical instant** occurs when all higher priority tasks arrive at the same time, and the task that contributes to the maximum blocking arrives at the critical instant - ε.



Divide Busy Period

- Divide the busy period for τ_i into two parts:
  - the length of time from the critical instant (time 0) to the point when τ_i starts executing its q^{th} job (**$S_i(q)$**).
  - the length of time from the time τ_i starts executing its q^{th} job until it finishes executing its q^{th} job (**$F_i(q)$-$S_i(q)$**).
- Let q = 1, 2, ... , m until we reach q = m s.t. $F_i(m) \leq m\, T_i$ that is, the m^{th} job completes before the next job is released.
- Then,

$$R_i = \max_{q\in\{1,..,m\}} \{F_i(q) - (q-1)T_i\}$$

Worst-Case Start Time ($\mathbf{S_i(q)}$)

$$S_i(q) = B(\tau_i) + (q\text{-}1)C_i + \sum_{\substack{j \in \{1,..,n\} \\ \pi_j > \pi_i}} \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor\right) C_j$$

Worst-Case Finish Time ($\mathbf{F_i(q)}$)

$$F_i(q) = S_i(q) + C_i + \sum_{\substack{j \in \{1,..,n\} \\ \pi_j > \gamma_i}} \left(\left\lceil \frac{F_i(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor \right) \right) C_j$$

$$WCRT(\pi_i, \gamma_i)$$

Algorithm to compute R_i

Input: $C_{1,}, \ldots, C_m, T_1, \ldots, T_m, \pi_1, \ldots, \pi_m, \gamma_1, \ldots, \gamma_m$

Output: $R_1, R_2, \ldots, R_m$

done $=$ FALSE

q $= 1$

while (not done)

  compute S_i(q) and F_i(q)

  if $F_i(\mathrm{q}) \leq \mathrm{q}\, T_i$ then

    done $=$ TRUE

    m $=$ q

  else

    q $=$ q $+ 1$

  end if

end while

$$R_i = \max_{\mathrm{q} \in \{1,..,\mathrm{m}\}} (F_i(\mathrm{q}) - (\mathrm{q} - 1)\, T_i)$$

Example

Task	C_i	T_i	D_i	π_i	**WCRT** Preemptive	**WCRT** Non-Preemptive
τ_1	20	70	50	3	20	**55**
τ_2	20	80	80	2	40	75
τ_3	35	200	100	1	**115**	75

Task	**Priority**	**Preemption Threshold**	**WCRT**
τ_1	3	3	40
τ_2	2	3	75
τ_3	1	2	95

Preemption Threshold Assignment

- **Lemma 5.1:** Changing the preemption threshold of task τ_i from γ_1 to γ_2 may only affect the worst-case response time of task τ_i and those tasks whose priority is between γ_1 and γ_2.
- **Corollary 5.1:** The worst-case response time of task τ_i will not be affected by the preemption threshold assignment of any higher priority task; e.g., any task τ_j with $\pi_j > \pi_i$.
- **This implies that we should assign preemption thresholds from lowest to highest priority.**

Preemption Threshold

- **Theorem 5.1:** Start with a schedulable system with n tasks. If decreasing the value of γ_j does not change the schedulability of task τ_j , then the whole system is still schedulable.
- **Idea:** Keep γ_j as small as possible for each task.
- **Lemma 5.2:** (Quick Test) If setting $\gamma_j = n$ cannot make task τ_j schedulable, then the task set is not schedulable.

Preemptive Scheduling

Preemption Threshold

File

Task	C	T	D	Prio	PT	B	S	F	WCRT
1	5	50	15	9	9	0	0	5	5
2	5	60	25	8	8	0	5	10	10
3	7	80	30	7	7	0	10	17	17
4	7	200	40	6	6	0	17	24	24
5	10	200	50	5	5	0	24	34	34
6	8	200	60	4	4	0	34	42	42
7	12	220	70	3	3	0	42	59	59
8	10	230	70	2	2	0	59	74	74
9	15	240	100	1	1	0	74	96	96

Compute

Non-Preemptive Scheduling

Preemption Threshold

File

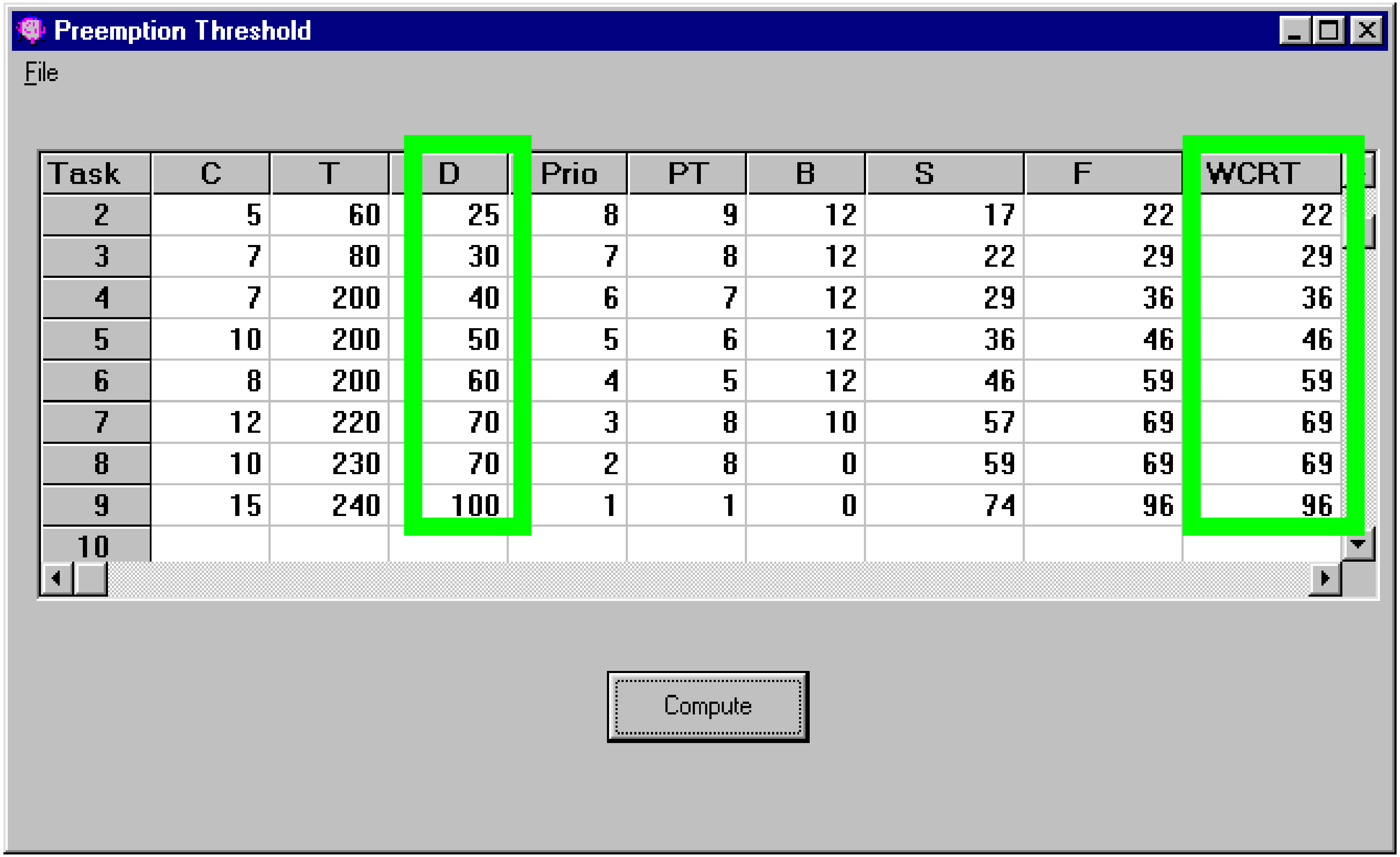
Task	C	T	D	Prio	PT	B	S	F	WCRT
1	5	50	15	9	9	15	15	20	20
2	5	60	25	8	9	15	20	25	25
3	7	80	30	7	9	15	25	32	32
4	7	200	40	6	9	15	32	39	39
5	10	200	50	5	9	15	39	49	49
6	8	200	60	4	9	15	49	57	57
7	12	220	70	3	9	15	67	79	79
8	10	230	70	2	9	15	79	89	89
9	15	240	100	1	9	0	74	89	89

Compute

Algorithm: Assign Preemption Thresholds

```
// Assumes that task priorities are already known
(1)  for (i := 1 to n)
(2)      γ_i = π_i
         // Calculate worst-case response time of τ_i
(3)      R_i = WCRT(τ_i, γ_i) ;
(4)      while (R_i > D_i) do     // while not schedulable
(5)          γ_i++ ; // increase threshold
(6)          if γ_i > n then
(7)              return FAIL; // system not schedulable.
(8)          endif
(9)          R_i = WCRT(τ_i, γ_i) ;
(10)     end
(11) end
(12) return SUCCESS
```

Preemption Thresholds



Task	C	T	D	Prio	PT	B	S	F	WCRT
2	5	60	25	8	9	12	17	22	22
3	7	80	30	7	8	12	22	29	29
4	7	200	40	6	7	12	29	36	36
5	10	200	50	5	6	12	36	46	46
6	8	200	60	4	5	12	46	59	59
7	12	220	70	3	8	10	57	69	69
8	10	230	70	2	8	0	59	69	69
9	15	240	100	1	1	0	74	96	96
10									

Finding Optimal Assignment

- **Problem:** How to find an optimal assignment of task priorities and preemption thresholds.
- **Solution:**
  - Arrange tasks into two sets -- unsorted (remaining higher priority tasks) and sorted (lower priority tasks).
  - Recursively add tasks from unsorted list to sorted list based on “lateness” heuristic.
  - Tasks are added in priority order, from lowest to highest priority.

Priorities and Preemption Thresholds

- Search ($\{\tau_1,\ldots,\tau_n\}$, 1)

remaining tasks

next priority to be assigned

- Initially determine if the task set is schedulable using preemptive priority-based scheduling without preemption thresholds (e.g., can priorities be assigned using RM?).

“Lateness” Heuristic

- From the unsorted list, select the task with the smallest lateness, and add it to the sorted list.
- Since the task selected has the smallest “lateness” (delay), it should need a lower priority and smaller preemption threshold, leaving more time for higher priority tasks.

Greedy Assignment Algorithm

Algorithm: GreedyAssignment(RemainingTasks, nextPriority)

```
    /* Terminating Condition */
    if (RemainingTasks == NULL) then
        /* Call the algorithm in Figure 1 for optimal preemption threshold assignment */
        if (AssignThresholds() == SUCCESS) then return SUCCESS
        else return FAIL
        endif
    endif
```

Greedy Assignment Algorithm (cont.)

/ Assign Heuristic Value to Each Task */*

foreach τ_k in RemainingTasks **do**
    $\pi_k :=$ nextPriority ;   */* tentative assignment */*
    $\mathcal{R}_k :=$ WCRT(τ_k);   */* compute response time */*
    if $\mathcal{R}_k \geq \mathcal{D}_k$ **then** $h\_val_k := \mathcal{D}_k - \mathcal{R}_k$
    else $h\_val_k :=$ GetBlockingLimit(τ_k) ;
    endif
    $\pi_k := n$;   */* reset, to allow computing heuristic value for other tasks */*
end

Get Blocking Limit Function

Input : $\boldsymbol{\tau_k}$, $\boldsymbol{D_k}$

Output : Blocking limit of $\boldsymbol{\tau_k}$

$R_k = WCRT(\tau_k)$
$Max = D_k - R_k$
$Limit = 0$
For $B(\tau_k) = 1$ **to** Max
    $R_k = WCRT(\tau_k, B(\tau_k))$
    If $R_k > D_k$ **Then** *Break*
    Else $Limit = B(\tau_k)$
End For
Return *Limit*

Greedy Assignment Algorithm (cont.)

/ Select the task with the largest heuristic value next */*

τ_k := max_heuristic_val(RemainingTasks) ;

π_k := nextPriority ; */* final priority assignment */*

/ Recursively Assign Priorities to Remaining Tasks */*

if GreedyAssignment(RemainingTasks $-\tau_k$,nextPriority+1) == SUCCESS **then**

    return SUCCESS ;

return FAIL ;

Note

- There are cases when this heuristic algorithm is not able to find a feasible assignment, even though a non-preemptive priority assignment algorithm is able to find a solution.
- Thus, we could apply a non-preemptive assignment algorithm first, before using this heuristic algorithm.

Depth-First Search

- Perform a depth-first search to find an optimal priority assignment.
- When a leaf is reached, call AssignThresholds( ) to see if an optimal preemption threshold assignment exists.

Summary

- To Do:
  - Read Ch. 5-7 + Wang and Saksena's paper on scheduling with preemption thresholds.
  - Homework #1, Homework #2.