CIS 721 - Real-Time Systems
Lecture 9: Preemption Thresholds

Mitch Neilsen
neilsen@ksu.edu

Daily Humor


I SAW THE CODE FOR YOUR COMPUTER PROGRAM YESTERDAY.
6-8
S. ADAMS E-Mail: SCOTTADAMS@AOL.COM
IT LOOKED EASY. IT'S JUST A BUNCH OF TYPING. AND HALF OF THE WORDS WERE SPELLED WRONG.
© 1994 United Feature Syndicate, Inc.
AND DON'T GET ME STARTED ABOUT YOUR OVER-USE OF COLONS.
THEY REMIND ME OF YOU, SIR.

Outline

- Clock-Driven Scheduling (Ch. 5)
- **Priority-Driven Scheduling**
  - **Periodic Tasks (Ch. 6)**
    - Optimal Priority Assignment
    - Arbitrary Start Times
      - Leung’s Feasibility Test
      - Audsley’s Feasibility Test
    - **Arbitrary Deadlines**
    - **Preemption Thresholds**
  - Aperiodic and Sporadic Tasks (Ch. 7)

Terms and Concepts

- A **task set** $\Gamma = \{\boldsymbol{\tau}_1 , \boldsymbol{\tau}_2 , \ldots , \boldsymbol{\tau}_n \}$ is a collection of related tasks.
- Each **periodic task** τ_i is characterized by:
  - an **execution time** or **run-time** (C_i),
  - a **period** (T_i),
  - a **(relative) deadline** (D_i), and
  - a **phase** or **offset** (φ_i or O_i).

Response Time Analysis

- For each (potentially overlapping) release, a worst-case completion time $w_i(q)$ is defined by:

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

$$w_i^0(q) = C_i + (q-1) \cdot T_i$$

where q is the instance or job number and $w_i(q)$ is the least fixed point of $w^n{}_i(q)$.

- The response time of the q^{th} instance, $R_i(q)$, is given by $R_i(q) = w_i(q) - (q-1)\ T_i$.

Response Time Analysis (cont.)

- Set $q' = min \{ q \mid R_i(q) \leq T_i \}$.
- Then, the **level-i busy period** is [0, $w_i(q')$].
- The **worst-case response time** R_i is given by:

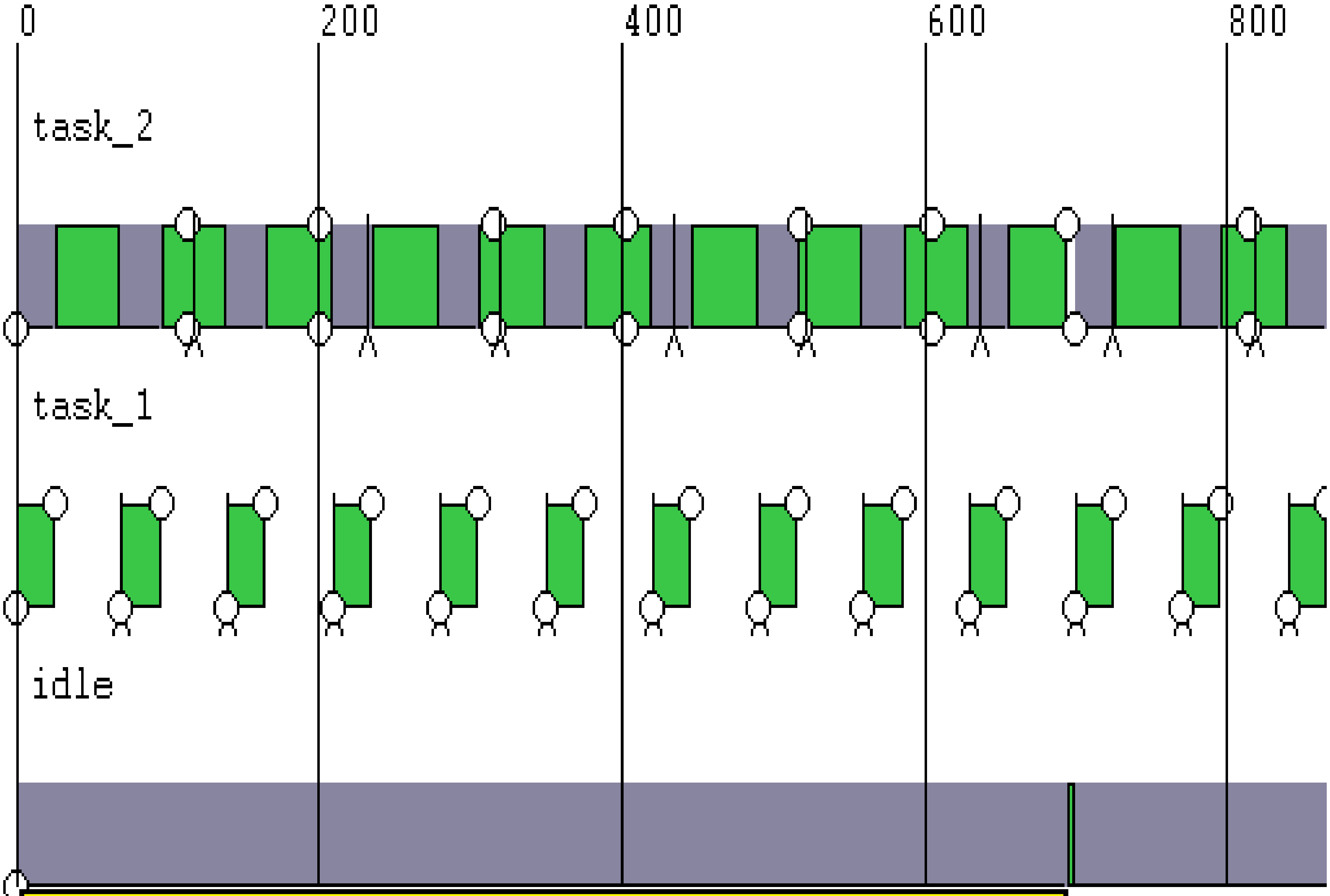
$$R_i = \max_{q=1,2,..,q'} \{R_i(q)\}$$

- If $R_i \leq D_i$ for all i, the system is **schedulable**.

Lehoczky's Example

Task	Period	Run-Time	Phase	Deadline
τ_i	T_i	C_i	φ_i	D_i
τ_1	70	26	0	68
τ_2	100	62	0	118

114, 102, 116, 104, 118, 106, 94 ← task_2 WCRT



Level-i Busy Period for task_2 = [0, 694]

General Time-Driven Analysis

- Check to see if the first job completes before it’s deadline and before the second job in the same task is released.
- If it completes before it’s deadline, but not before the second job is released, then check the second job.
- In general, continue to check all jobs over a level-i busy period; that is, until a deadline is missed or until one job completes before the next one is released.

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

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

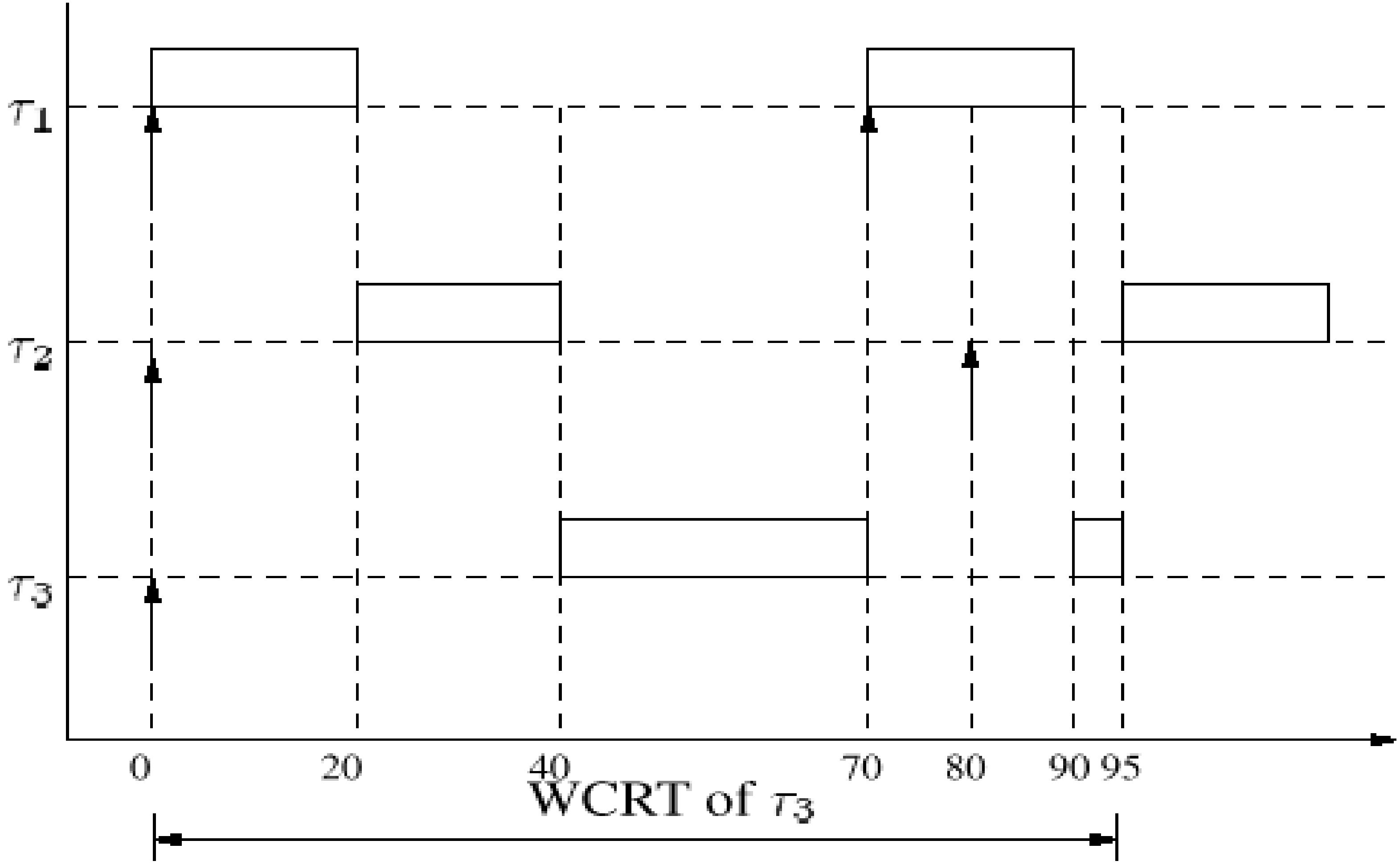


Figure 1. Run-time Behavior with Preemption Threshold

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 $\{ (\pi_i, \gamma_i) \mid i = 1, 2, .., n \}$, 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 $\{ \pi_i \mid i = 1, .., n \}$, 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).

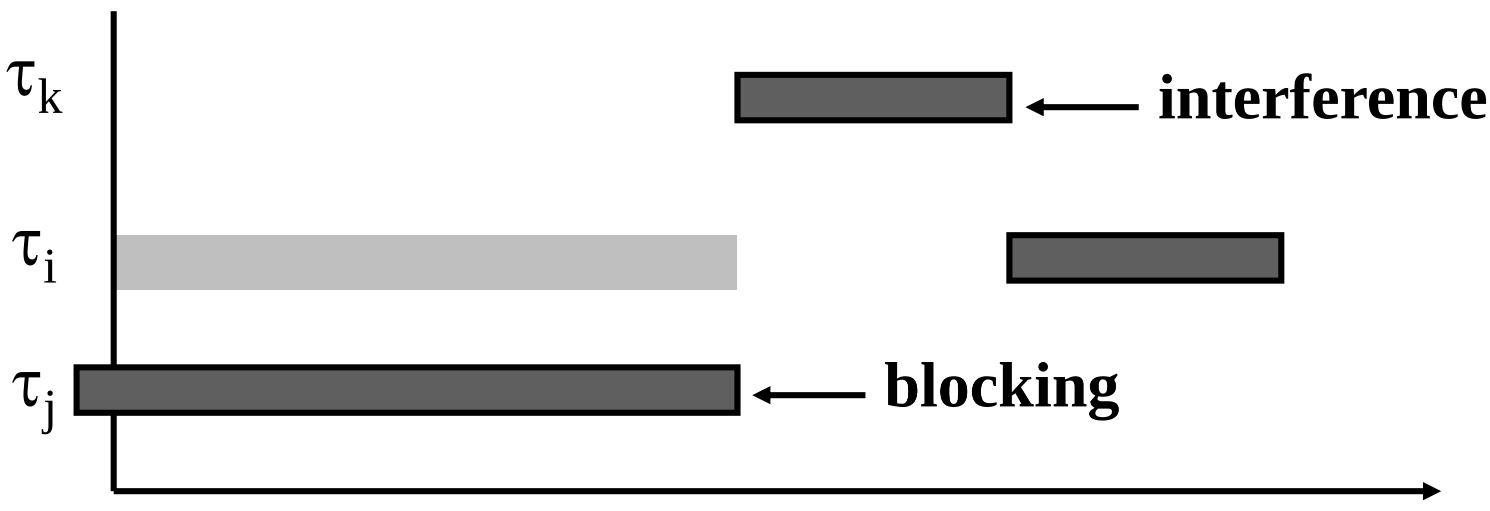
Response Time Analysis

- 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 ($\mathbf{S_i(q)}$).
  - the length of time from the time τ_i starts executing its q^{th} job until it finishes executing its q^{th} job ($\mathbf{F_i(q)\text{-}S_i(q)}$).
- 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(q) ≤ q T_i then
        done = TRUE
        m = q
    else
        q = q + 1
    end if
end while
```

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

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

for (i := 1 to n)
    $\gamma_i = \pi_i$
    // Calculate worst-case response time of τ_i
    $\mathcal{R}_i$ = WCRT(τ_i,γ_i) ;
    while ($\mathcal{R}_i > D_i$) **do**   *// while not schedulable*
        γ_i++ ; *// increase threshold*
        if $\gamma_i > n$ **then**
            return FAIL; *// system not schedulable.*
        endif
        $\mathcal{R}_i$ = WCRT(τ_i,γ_i) ;
    end
end
return SUCCESS

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

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	10	57	74	74
8	10	230	70	2	3	0	59	74	74
9	15	240	100	1	1	0	74	96	96

Compute

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	10	44	57	57
7	12	220	70	3	3	10	57	74	74
8	10	230	70	2	4	0	59	74	74
9	15	240	100	1	1	0	74	96	96

Compute

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	10	34	44	44
6	8	200	60	4	4	10	44	57	57
7	12	220	70	3	3	10	57	74	74
8	10	230	70	2	5	0	59	74	74
9	15	240	100	1	1	0	74	96	96

Compute

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	10	27	34	34
5	10	200	50	5	5	10	34	44	44
6	8	200	60	4	4	10	44	57	57
7	12	220	70	3	3	10	57	74	74
8	10	230	70	2	6	0	59	74	74
9	15	240	100	1	1	0	74	96	96

Compute

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	10	20	27	27
4	7	200	40	6	6	10	27	34	34
5	10	200	50	5	5	10	34	44	44
6	8	200	60	4	4	10	44	57	57
7	12	220	70	3	3	10	57	74	74
8	10	230	70	2	7	0	59	74	74
9	15	240	100	1	1	0	74	96	96

Compute

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	10	15	20	20
3	7	80	30	7	7	10	20	27	27
4	7	200	40	6	6	10	27	34	34
5	10	200	50	5	5	10	34	44	44
6	8	200	60	4	4	10	44	57	57
7	12	220	70	3	3	10	57	74	74
8	10	230	70	2	8	0	59	69	69
9	15	240	100	1	1	0	74	96	96

Compute

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	10	15	20	20
3	7	80	30	7	7	12	22	29	29
4	7	200	40	6	6	12	29	36	36
5	10	200	50	5	5	12	36	46	46
6	8	200	60	4	4	12	46	59	59
7	12	220	70	3	7	10	57	74	74
8	10	230	70	2	8	0	59	69	69
9	15	240	100	1	1	0	74	96	96

Compute

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	12	17	22	22
3	7	80	30	7	7	12	22	29	29
4	7	200	40	6	6	12	29	36	36
5	10	200	50	5	5	12	36	46	46
6	8	200	60	4	4	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

Compute

Preemption Thresholds

Preemption Threshold

File

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									

Compute

Preemption Thresholds – ThreadX RTOS

- Response Time Analysis to computer WCRT given Priorities and Preemption Thresholds
- Algorithm to optimally assign Preemption Thresholds given Priority Assignment
- **Algorithm to Assign Priorities**

Finding Priority Assignment

- **Problem:** How to find an optimal assignment of task priorities and preemption thresholds.
- **Solution Proposed in Paper:**
  - 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 */
(1)     if (RemainingTasks == NULL) then
                /* Call the algorithm in Figure 1 for optimal preemption threshold assignment */
(2)             if (AssignThresholds() == SUCCESS) then return SUCCESS
(3)             else return FAIL
(4)             endif
(5)     endif
```

Greedy Assignment Algorithm (cont.)

/ Assign Heuristic Value to Each Task */*

foreach τ_k in RemainingTasks **do**

  π_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 try a non-preemptive assignment algorithm first, before using this heuristic algorithm (or use a better 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; if not, continue searching.

Summary

- Preemption thresholds provide a way of generalizing both preemptive and non-preemptive scheduling in a single framework.
- Read Y. Wang and M. Saksena's paper on preemption thresholds.