

# 03\_Non-Preemptive Scheduler

---

## 03\_Non-Preemptive Scheduler

### 1. Preemption vs Non-Preemption

- 1.1. Disadvantages of Preemption
- 1.2. Advantages of Non-preemption
- 1.3. Disadvantages of Non-preemption

### 2. Non-Preemptive (NP) scheduling

- 2.1. Assumptions
- 2.2. Hybrid NP Solutions
  - Preemption Thresholds
  - Deferred Preemptions
  - Task Splitting Approach
- 2.3. Overall Property of NP-Scheduling Policies
- 2.4. Analysis of NP Scheduling Policies
  - Level-i active period
  - Calculate Level-i active period
  - Analysis of Schedulability
  - Special Case

### 3. Hybrid NP Solutions

- 3.1. Preemption Thresholds
  - Feasibility Analysis
  - Threshold Assign
- 3.2. Deferred Preemptions
  - Modes
  - Analysis of DP scheduling
  - Open-Discussions
- 3.3. Fixed Preemption Points
  - Two important parameters
  - Analysis of TS scheduling
  - Special Case

# Final Remarks about Hybrid NP Solutions

- Preemption Thresholds.
  - Easy to specify and use, but it is difficult to predict the number of preemptions, and where in the schedule they will take place.
  - Hence, PTs have **large preemption overhead.**
- Deferred Preemption.
  - They allow bounding the number of preemptions;
  - But it is difficult to **predict where they will take place.**
- Fixed Preemption Points.
  - Allow more control on preemptions that can be selected so as to **minimize overheads,** or/and reduce WCETs.
  - A large final chunk for a task reduces interference from higher priority tasks (improves responsiveness) but creates **more blocking to higher priority tasks.**

## 1. Preemption vs Non-Preemption

	Advantages	Disadvantages
Preemptive Scheduling	<ul style="list-style-type: none"><li>• Fast handling of exceptions/events</li><li>• Increases CPU efficiency</li><li>• Prioritizes critical tasks</li><li>• Might improve schedulability</li></ul>	<ul style="list-style-type: none"><li>• Context switching costs</li><li>• Chains of preemption</li><li>• Cache misses &amp; delays</li><li>• Increases WCET</li><li>• Less predictable WCET</li></ul>
Non-Preemptive Scheduling	<ul style="list-style-type: none"><li>• Reduces context switching</li><li>• Smaller &amp; more predictable WCET</li><li>• Smaller I/O Jitter</li><li>• Better timing predictability</li><li>• Might improve schedulability</li></ul>	<ul style="list-style-type: none"><li>• Introduces blocking times</li><li>• Scheduling anomalies</li><li>• Lower CPU efficiency</li><li>• Smaller responses in events</li></ul>

### 1.1. Disadvantages of Preemption

- introduces **context switching cost**
- One preemption might induce **more preemptions**
- introduces **cache-related preemption delays (CPRD).**
- increase the WCET (Due to the cache-related preemption delays)

## 1.2. Advantages of Non-preemption

- Reduce context-switching overhead
- Simplifies the access to shared resources
- Reduces the required stack size
- Allows achieving small I/O jitter
- in some cases, improves schedulability in fixed-priority systems

## 1.3. Disadvantages of Non-preemption

- Might reduce schedulability due to blocking
  - A task with higher priority needs to wait lower-priority tasks
- Give rise to scheduling anomalies
  - Unexpected and counter-intuitive system behaviour. (e.g. processor speed up)

# 2. Non-Preemptive (NP) scheduling

---

## 2.1. Assumptions

- consider a set of **hard RT periodic tasks**
- Constrained deadline model:

$$D_i \leq T_i, \forall i$$

- Tasks are assigned **fixed priorities** (RM,DM), and we always use

$$P_1 > P_2 > \dots > P_n$$

## 2.2. Hybrid NP Solutions

- **Winnie**, the Teddy Bear is busy reading a book, but several other animals are calling it (for various reasons)! **What can Winnie do?**



- Respond only to important calls (its parents).  
→ **Preemption Thresholds**  
(do not bother unless the caller is very important)
- Time-postpone the answer.  
→ **Deferred preemptions**  
("I will come back in 10 minutes")
- Schedule/program-postpone the answer.  
→ **Fixed preemption points**  
("I will return as soon as I finish the chapter")

## Preemption Thresholds

Only tasks of **very high priority** can preempt low priority tasks.

For each task define a **different minimum priority threshold** that can preempt it.

## Deferred Preemptions

Define for each task **the longest no-preemption time interval** for its execution.

## Task Splitting Approach

Split the task to **non-preemptable code chunks**;

Preemption can take place only at specified points between those chunks.

## 2.3. Overall Property of NP-Scheduling Policies

- Different Critical Case:
  - Critical instant start in syn with higher priority jobs
- Self-pushing phenomenon
- The largest response time is not in the first execution after the critical instant
  - Response time analysis for Task i must be done for multiple periods

## 2.4. Analysis of NP Scheduling Policies

### Level-i active period

The time interval in which the CPU is **busy executing Task i or other tasks with higher priority**, including also blocking times from lower priority tasks.

Formally, it's a timer interval  $\Delta = [t_s, t_e)$ , such that

$$P_i(t_s) = 0; \quad P_i(t_e) = 0; \quad P_i(t) > 0 \text{ for all } t \text{ in } \Delta$$

$P_i(t)$  is the **Processing load** that is pending at t, from all tasks with priority higher than i, including task i.

- no lower priority task is taken into consideration (because a higher priority pending means after blocked (if exist), no time for lower priority execute)

### Calculate Level-i active period

Estimate the **worst-case blocking time** of Task i from lower-priority jobs.

$$B_i = \max_{j:P_j < P_i} \{C_j - 1\}$$

The busy period is **at least as large as**  $B_i$  plus the WCET of Task i:

$$L_i^{(0)} = B_i + C_i$$

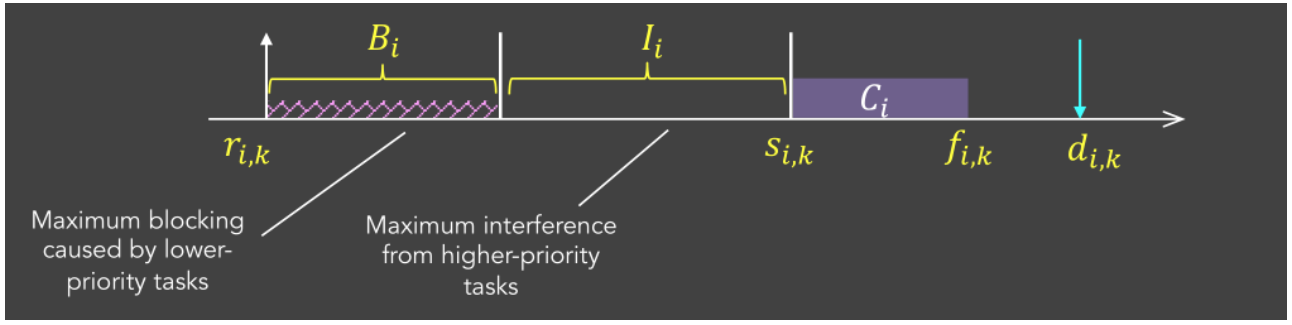
Find how many higher-priority jobs run in this interval

$$L_i^{(s)} = B_i + \sum_{h:P_h \geq P_i} \left\lceil \frac{L_i^{(s-1)}}{T_h} \right\rceil C_h \quad \text{until:} \quad L_i^{(s)} = L_i^{(s-1)}$$

the response time of Task i must be computed for k jobs:

$$k = 1, 2, \dots, K = \left\lceil \frac{L_i}{T_i} \right\rceil$$

# Analysis of Schedulability



$$s_{i,k}^{(0)} = B_i + \sum_{h:P_h > P_i} C_h$$

$$s_{i,k}^{(l)} = B_i + (k-1)C_i + \sum_{h:P_h > P_i} \left( \left\lfloor \frac{s_{i,k}^{(l-1)}}{T_h} \right\rfloor + 1 \right) C_h$$

$B_i$  is the blocking

$(k-1)C_i$  is preceding jobs of the same task

$\sum_{h:P_h > P_i} \left( \left\lfloor \frac{s_{i,k}^{(l-1)}}{T_h} \right\rfloor + 1 \right) C_h$  is the interference from higher priority tasks

$$f_{i,k} = s_{i,k} + C_i$$

$$R_i = \max_{k \in [1, K_i]} \{f_{i,k} - (k-1)T_i\}$$

The set is **schedulable** if and only if

$$R_i \leq D_i, i = 1, 2, \dots, n$$

## Special Case

It is sufficient to **consider only the first job if and only if** the task set is **feasible under preemptive scheduling** and all **deadlines are less (or equal) to the periods**.

$$s_{i,1} = S_i = B_i + \sum_{h:P_h > P_i} \left( \left\lfloor \frac{S_i}{T_h} \right\rfloor + 1 \right) C_h$$

$$R_i = S_i + C_i$$

## 3. Hybrid NP Solutions

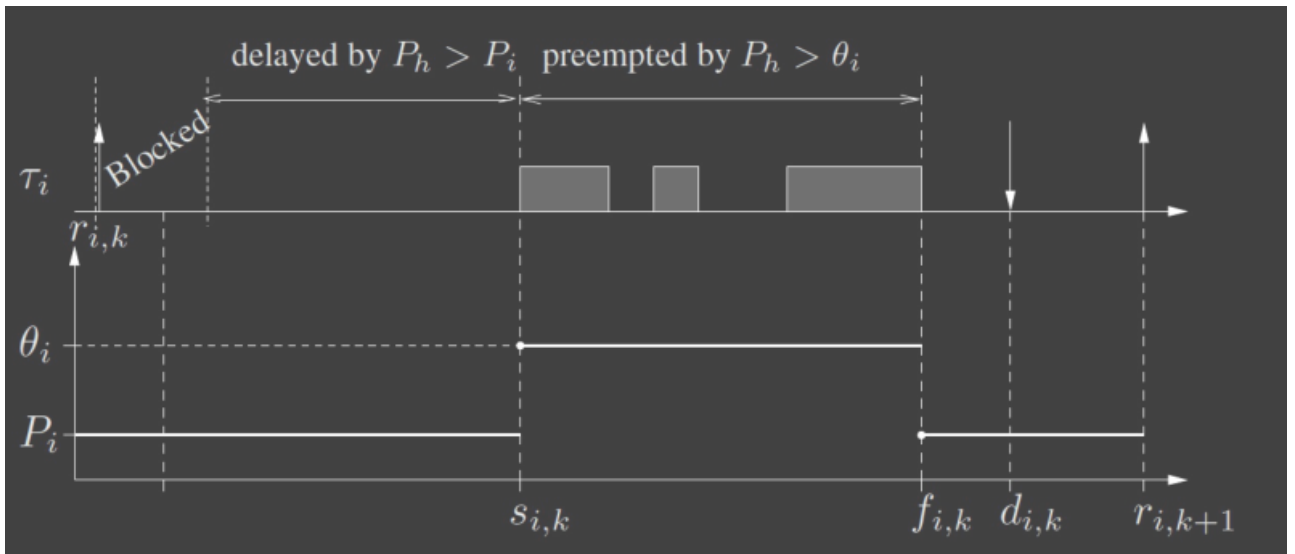
### 3.1. Preemption Thresholds

Each task  $i$  has two priorities:

- **Nominal Priority**  $P_i$  : used as long as the task is in the queue. Based on this value it **preempts (or not) other** tasks.
- **Threshold Priority**  $\theta_i$  : used when the task runs. Only tasks with priority higher than this threshold can preempt task  $i$ .

In general  $\theta_i > P_i$  (personal understanding)

#### Feasibility Analysis



Blocking time

$$B_i = \max_{j: P_j < P_i} \{C_j - 1 \mid P_j < P_i \leq \theta_j\}$$

Level-i active period and K

$$L_i^{(s)} = B_i + \sum_{h: P_h \geq P_i} \left\lceil \frac{L_i^{(s-1)}}{T_h} \right\rceil C_h$$

$$K = \left\lceil \frac{L_i}{T_i} \right\rceil$$

Start time of job k

$$s_{i,k}^{(l)} = B_i + (k-1)C_i + \sum_{h: P_h > P_i} \left( \left\lceil \frac{s_{i,k}^{(l-1)}}{T_h} \right\rceil + 1 \right) C_h$$

Finish time Analysis

$$f_{i,k}^{(0)} = s_{i,k} + C_i$$

$$f_{i,k}^{(l)} = s_{i,k} + C_i + \sum_{h: P_h > \theta_i} \left( \left\lceil \frac{f_{i,k}^{(l-1)}}{T_h} \right\rceil - \left( \left\lfloor \frac{S_{i,k}}{T_h} \right\rfloor + 1 \right) \right) C_h^F$$

Feasibility Check

$$R_i = \max_{k \in [1, K_i]} \{f_{i,k} - (k-1)T_i\}$$

$$R_i \leq D_i, i = 1, 2, \dots, n$$

Notes:

For equation 13,14,15, use  $P_h \geq P_i$  is because, these equations consider the situation that our task is not running. So when all in the queue, they are sorted by P not  $\theta$

## Threshold Assign

Try from  $\theta_i = P_i$ , if not feasible, add one.

- The algorithm is **optimal**: If there is a feasible schedule, it will find it.

### Algorithm: Assign Minimum Preemption Thresholds

**Input:** A task set  $\mathcal{T}$  with  $\{C_i, T_i, D_i, P_i\}, \forall \tau_i \in \mathcal{T}$

**Output:** Task set feasibility and  $\theta_i, \forall \tau_i \in \mathcal{T}$

// Assumes tasks are ordered by decreasing priorities

$$P_1 > P_2 > \dots > P_n$$

(1) **begin**

(2)     **for** ( $i := n$  **to** 1) **do**     // from the lowest priority task

(3)          $\theta_i := P_i$ ;

(4)         Compute  $R_i$  by Equation (8.15);

$$R_i = \max_{k \in [1, K_i]} \{f_{i,k} - (k-1)T_i\}$$

(5)         **while** ( $R_i > D_i$ ) **do**     // while not schedulable

(6)              $\theta_i := \theta_i + 1$ ;     // increase threshold

(7)             **if** ( $\theta_i > P_1$ ) **then**     // system infeasible

We reduce the tasks that can preempt Task i

(8)             **return** (INFEASIBLE);

(9)         **end**

(10)         Compute  $R_i$  by Equation (8.15);

(11)         **end**

(12)     **end**

(13)     **return** (FEASIBLE);

(14) **end**

The algorithm is optimal: If there is a feasible schedule, it will find it.



## 3.2. Deferred Preemptions

we define for each task an interval of time  $q_i$  during which it cannot be preempted

### Modes

#### Floating Model

The non-preemption interval is explicitly defined in the code by the programmer.

#### Activation-triggered

The non-preemption interval is **triggered by the arrival of a higher priority task** -- which normally would preempt the current job -- and is enforced for  $q_i$  time slots

## Analysis of DP scheduling

### Blocking

Each Task  $i$  can be blocked by the non-preemptive subjobs of any lower-priority task.

- If floating model:

$$B_i = \max_{j:P_j < P_i} \{q_j - 1\}$$

- if activation-triggered model:

$$B_i = \max_{j:P_j < P_i} \{q_j\}$$

Others are the same with **NP scheduling**

## Open-Discussions

Given a set of periodic tasks that is feasible under preemptive scheduling, what is the longest non-preemptive interval  $Q_i$  for each task  $i$ , so that it can continue to execute for  $Q_i$  time in non-preemptive mode, without violating the schedulability of the original task set?

### 3.3. Fixed Preemption Points

Each task  $i$  is split into  $m_i$  **chunks**.

- During those chunks (or, subjobs), the task cannot be preempted;
- Preemption can only happen in between these subjobs.

Assuming known WCET for each chunk, we can calculate:

$$C_i = \sum_{k=1}^{m_i} q_{ik}$$

Two important parameters

$$q_i^{\max} = \max_{k \in [1, m_i]} \{q_{ik}\}$$

Affects the feasibility of higher priority tasks (those that this subjob can preempt)

$$q_i^{\text{last}} = q_{i, m_i}$$

Affects the response time of Task  $i$

### Analysis of TS scheduling

Blocking time:

$$B_i = \max_{j: P_j < P_i} \{q_j^{\max} - 1\} \quad q_i^{\text{last}} = q_{i, m_i}$$

Level- $i$  active period

$$L_i^{(0)} = B_i + C_i$$

$$L_i^{(s)} = B_i + \sum_{n: P_h \geq P_i} \left\lceil \frac{L_i^{(s-1)}}{T_h} \right\rceil C_h$$

$$k = 1, \dots, K = \left\lceil \frac{L_i}{T_i} \right\rceil$$

the starting times of the **last subjob of each job of each task** are:

$$s_{i,k}^{(0)} = B_i + C_i - q_i^{last} + \sum_{h: P_h > P_i} C_h$$

$$s_{i,k}^{(l)} = B_i + kC_i - q_i^{last} + \sum_{n: P_h > P_i} \left( \left\lceil \frac{s_{i,k}^{(l-1)}}{T_h} \right\rceil + 1 \right) C_h$$

Finish time and response time

$$f_{i,k} = s_{i,k} + q_i^{last}$$

$$R_i = \max_{k \in [1, K_i]} \{f_{i,k} - (k-1)T_i\}$$

Feasibility Check

$$R_i \leq D_i, \quad i = 1, 2, \dots, n$$

## Special Case

If the task set is feasible under preemptive execution, then the analysis can be done by using only the first job of each task