TASK SCHEDULING

- Task scheduling refers to determining the order in which the various tasks are to be taken up for execution by the operating system.
- Every operating system relies on one or more task schedulers to prepare the schedule of execution of various tasks it needs to run.
- Each task scheduler is characterized by the scheduling algorithm it employs.

Terminologies

- Scheduling Points
  - The points on timeline at which the scheduler makes decisions regarding which task is to be run next.
  - In clock driven scheduler, scheduling points are defined at the time instants marked by interrupts generated by a periodic timer.
  - In event driven scheduler, determined by occurrence of certain events.
- Preemptive Scheduler
  - When a high priority task arrives, suspends any low priority task that may be executing and takes up the higher priority task for execution.

- Utilization
  - The average time for which task executes per unit time interval.
  - For a periodic task T_i, the utilization $u_i = \frac{e_i}{p_i}$ where e_i is the execution time and p_i is the period of T_i.
  - For a set of periodic tasks $\{T_i\}$: the total utilization due to all tasks U=$\sum_{i=1}^{n} \frac{e_i}{p_i}$

- Jitter
  - Deviation of a periodic task from its strict periodic behavior.
  - The arrival time jitter is the deviation of the task from arriving at the precise periodic time of arrival.
  - Completion time jitter is the deviation of the completion of a task from precise periodic points.

Classification of real time task scheduling algorithms (based on scheduling points)

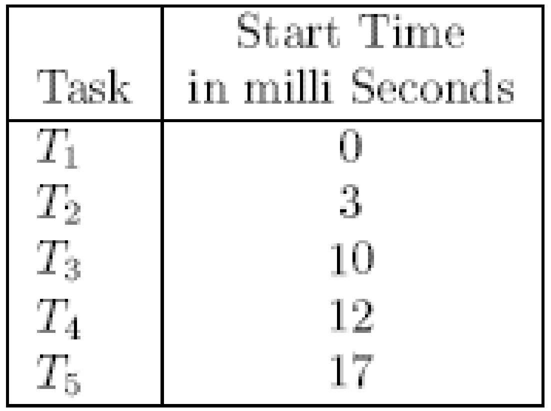
- Clock driven: scheduling points are determined by the interrupts received by a clock
  - Table driven
  - Cyclic
- Event driven: scheduling points are determined by certain events
  - Simple priority based
  - Rate Monotonic Analysis (RMA)
  - Earliest Deadline First (EDF)
- Hybrid: scheduling points are determined using both clock interrupts as well as event occurrences
  - Round Robin

Classification of real time task scheduling algorithms (type of task acceptance test)

- Acceptance test is used to decide whether a newly arrived task would be taken up for scheduling or be rejected.
- Planning based
  - When a task arrives the scheduler first determines whether the task can meet its deadlines, if it is taken up for execution, if not it is rejected.
- Best effort
  - All tasks that arrive are taken up for scheduling and best effort is made to meet its deadlines

Clock Driven Scheduling

- **Clock driven schedulers make their scheduling decisions regarding which task to run next only at the clock interrupt points**
- **Also called as offline schedulers**
- **Table driven scheduling**
  - **They precompute which task would run when and store this schedule in a table at the time the system is designed or configured.**



Task	Start Time in milli Seconds
T_1	0
T_2	3
T_3	10
T_4	12
T_5	17

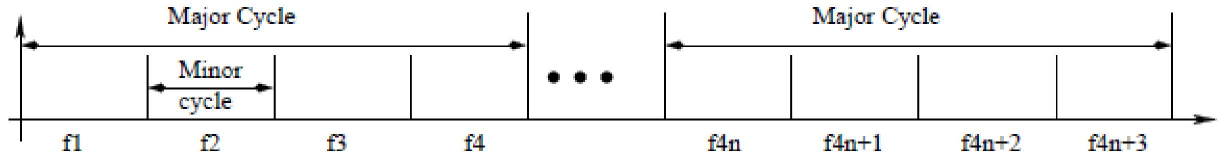
- Freedom to select own schedule for the application programmer. (schedule table)

- For any given task set it is sufficient to store entries only for LCM(p1, p2, p3 ……………pn) duration in the schedule table.

- A major cycle of a set of tasks is an interval of time on the timeline such that in each major cycle, the different tasks recur identically.

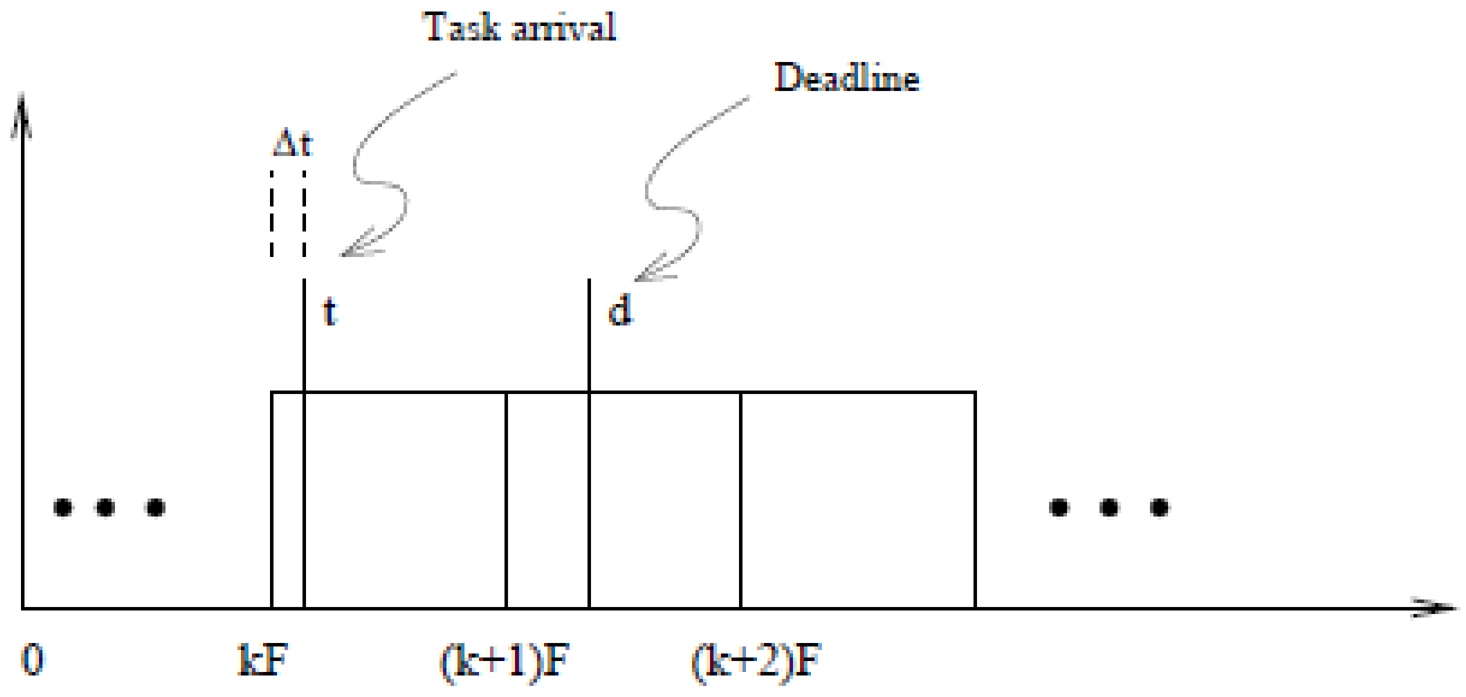
- Cyclic schedulers
  - These are very popular and are being extensively used in industries.
  - Ex: computer controlled air conditioners
  - Cyclic scheduler repeats a precomputed schedule.
  - Major cycle is divided into one or more minor cycles (*frames*)
  - The scheduling points of a cyclic scheduler occur at frame boundaries.
  - The assignment of tasks to frames is stored in a schedule table.

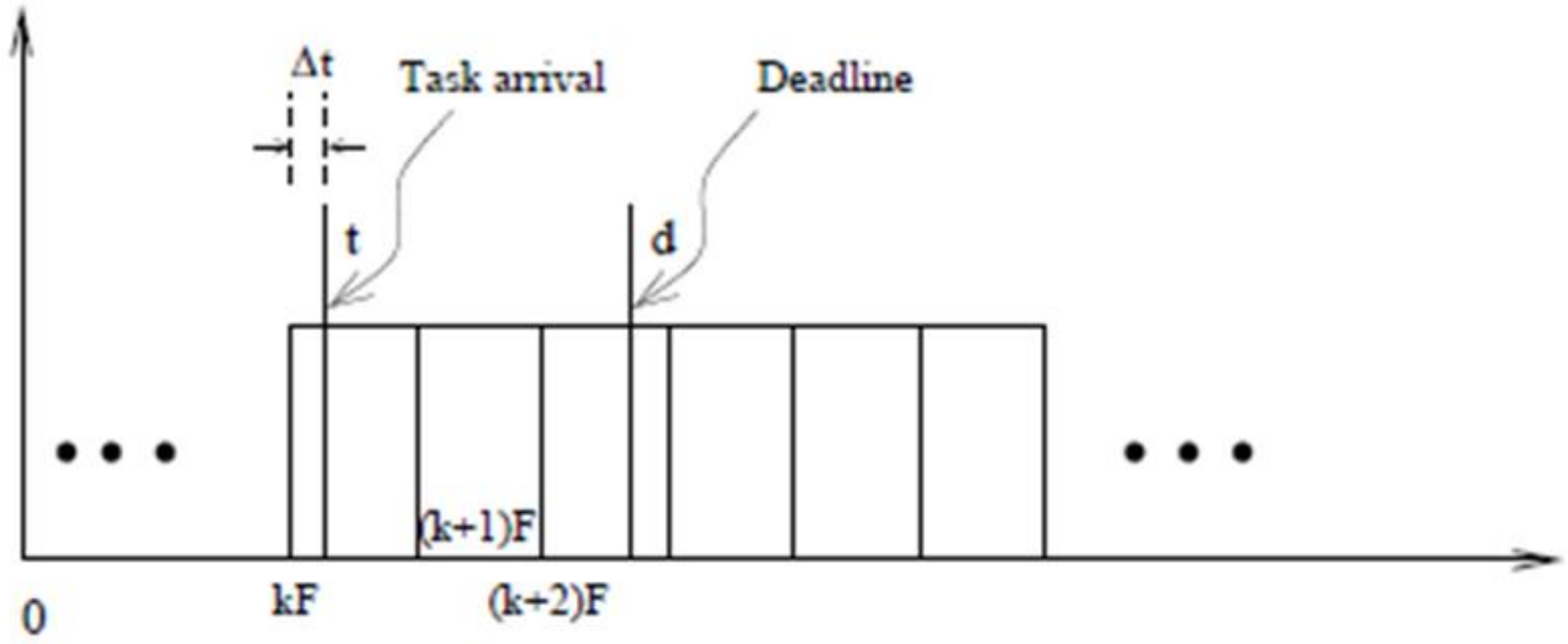
Task Number	Frame Number
T3	F1
T1	F2
T3	F3
T4	F2



- A selected frame should satisfy the following 3 constraints
- Minimum Context Switching
  - A task instance must complete running within its assigned frame.
  - The selected frame size should be larger than the execution time of each task, so that when a task starts at frame boundary, it should be able to complete within the same frame.
  - Max ({e_i}) ≤ F, e_i is the execution time of task T_i and F is the frame size.
- Minimization of Table size
  - To minimize the storage requirement of the schedule table.
  - Cyclic schedulers are used in small embedded applications with very small storage capacity.
  - Major cycle must be an integral multiple of frame size.

- Satisfaction of task deadline
  - Between the arrival of a task and its deadline, there must exist at least one full frame.




Δt
Task arrival
Deadline
t
d
(k+1)F
0
kF
(k+2)F

- Suppose a task arises after Δt time units have passed since the last frame, then assuming that a single frame is sufficient to complete the task, the task can complete before its deadline iff 2F – Δt ≤ d_i
- The worst case scenario for a task to meet its deadline occurs for its instance that has the minimum separation i.e min (Δt) = gcd(F, p_i) from the start of a frame.
- The constraint can be written for every T_i:
- 2F - gcd(F, p_i) ≤ d_i

HYBRID SCHEDULERS

- TIME SLICED ROUND ROBIN SCHEDULING
  - Preemptive scheduling method.
  - In round robin scheduling, the ready tasks are held in a circular queue.
  - Once a task is taken up, it runs for a certain fixed interval of time called its *time slice.*
  - If a task does not complete within its allocated time slice, it is inserted back into the circular queue.
  - A time sliced round robin scheduler is less proficient than table driven or cyclic scheduler for scheduling real time tasks.
  - The scheduler treats all tasks equally and all tasks are assigned identical time slices irrespective of their priority, criticality, or closeness of deadline.

EVENT DRIVEN SCHEDULING

- A prominent shortcoming of the cyclic schedulers is that it becomes very complex to determine a suitable frame size as well as a feasible schedule when the number of tasks increases.
- In almost every frame some processing time is wasted.
- Event driven schedulers are used in all moderate and large size applications having many tasks
- The scheduling points are determined by task completion and task arrival events.
- This class of schedulers are normally preemptive.

Foreground Background Scheduler

- Real time tasks in an application are run as foreground tasks.
- Among the foreground tasks, at every scheduling point the highest priority task is taken up for scheduling.
- The sporadic, aperiodic and non real time tasks are run as background tasks.
- Background tasks run at the lowest priority.

- Assume there are n foreground tasks which are denoted as :

T_1 , $T_{2,}$,……T_n.
- Let T_B be the only background task.
- Let e_B be the processing time requirement of T_B .
- The completion time (cT_B) for the background task is given by
- $cT_B = \frac{e_B}{1-\sum_{i=1}^{n}\frac{e_i}{p_i}}$
- All the foreground tasks together would result in CPU utilization of $\sum_{i=1}^{n}\frac{e_i}{p_i}$
- The average time available for execution of the background tasks in every unit of time is $1 - \sum_{i=1}^{n}\frac{e_i}{p_i}$

EARLIEST DEADLINE FIRST (EDF) SCHEDULING

- At every scheduling point, the task having the shortest deadline is taken up for scheduling.
- A task set is schedulable under EDF, if and only if it satisfies the condition that the total processor utilization due to the task set is less than 1.
- For a set of periodic real time tasks {T1, T2,T3,.....Tn}, EDF schedulability criteria can be expressed as

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum_{i=1}^{n} u_i \leq 1$$

  - **Ui is the average utilization due to task Ti and n is the total number of tasks in the task set**

The expression is both a necessary and sufficient condition for a set of tasks to be EDF schedulable.

- In practical problems, the period of a task may at times be different from its deadline.
- If p_i > d_i, then each task needs e_i amount of computing time every min(p_i, d_i) duration of time.
- Schedulability test is
- $\sum_{i=1}^{n} \frac{e_i}{\min(p_i,\ d_i)} \leq 1$.

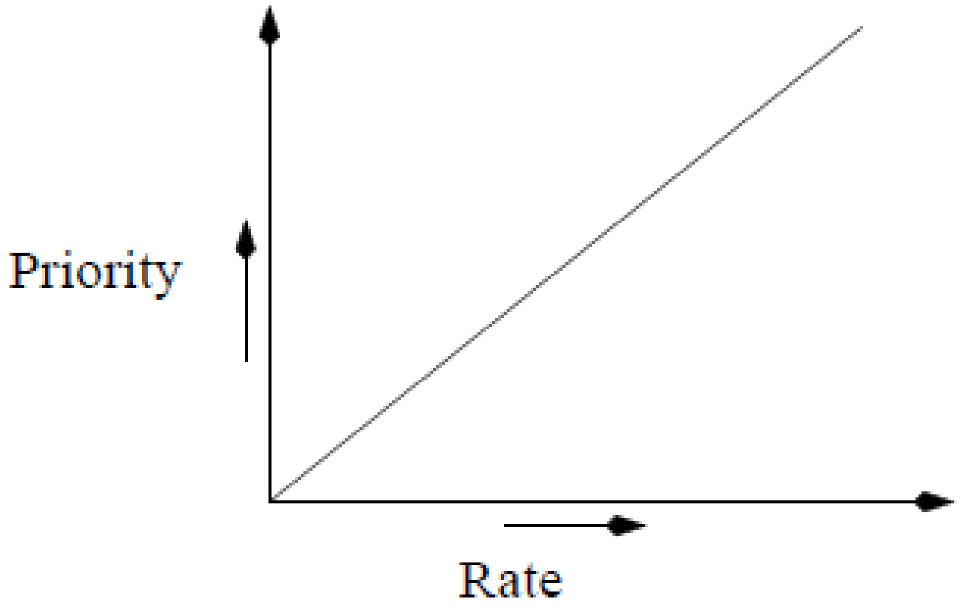
- All ready tasks are maintained in a sorted priority queue
- Tasks are always kept sorted according to the proximity of their deadline
- When a task arrives, a record for it can be inserted into the heap.
- At every scheduling point, the next task to be run can be found at the top of the heap.
- When a task is taken up for scheduling, it needs to be removed from the priority queue. This can be achieved in 0(1) time.

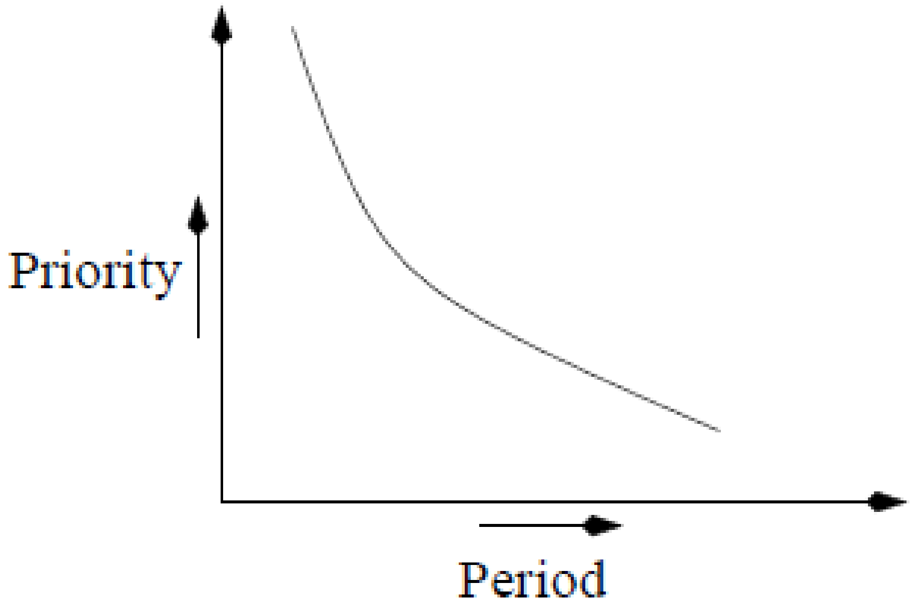
Shortcomings of EDF

- Transient overload problem
  - Occurs when some task takes more time to complete than what was originally planned during the design time
- Resource sharing problem
  - High overheads might have to be incurred
  - Overhead is any combination of excess computation time, memory or other resources required to perform a specific task.
- Efficient Implementation problem
  - Difficult to restrict the number of tasks with distinct deadlines to a reasonable number.

RATE MONOTONIC ANALYSIS (RMA)

- RMA is a static priority algorithm and is extensively used in practical applications.
- RMA assigns priorities to tasks based on their rates of occurrence
- The lower the occurrence rate of a task, the lower is the priority assigned to it.
- Priority of a task is directly proportional to its rate (or, inversely proportional to its period)
- Priority of any task Ti is computed as
  - Priority = $\frac{k}{Pi}$ where Pi is period of task Ti, k is constant

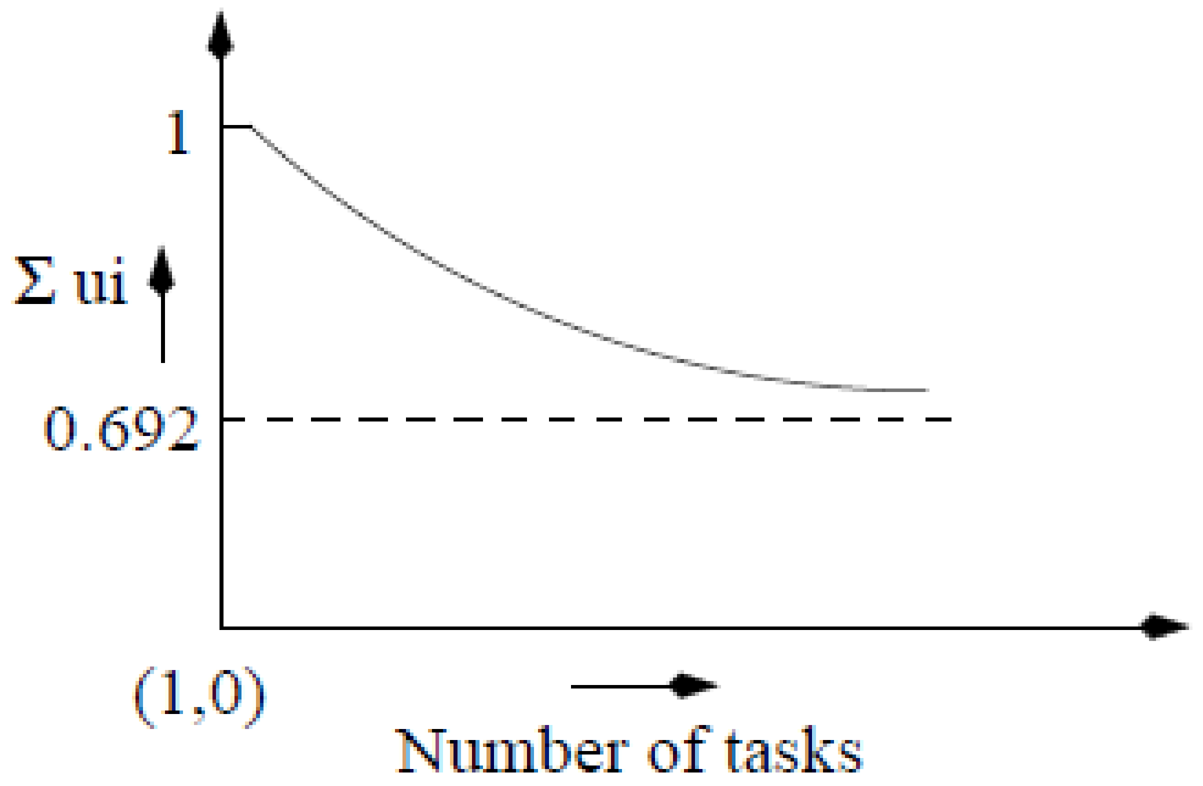

Priority
Rate


Priority
Period

Schedulability test for RMA

- Necessary Condition
- A set of periodic real-time tasks would not be RMA schedulable unless they satisfy the following necessary condition:
  - $\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum_{i=1}^{n} u_i \leq 1$
- This test expresses the fact that the total CPU utilization due to all the tasks in the task set should be less than 1.
- Sufficient Condition (Liu and Layland’s condition)
- A set of n real-time periodic tasks are schedulable under RMA, if
  - $\sum_{i=1}^{n} u_i \leq n(2^{\frac{1}{n}} - 1)$

Achievable utilization with the number of tasks under RMA



Disadvantages

- Difficult to support aperiodic and sporadic tasks under RMA.
- Not optimal when task periods and deadlines differ.

Lehoczky's test

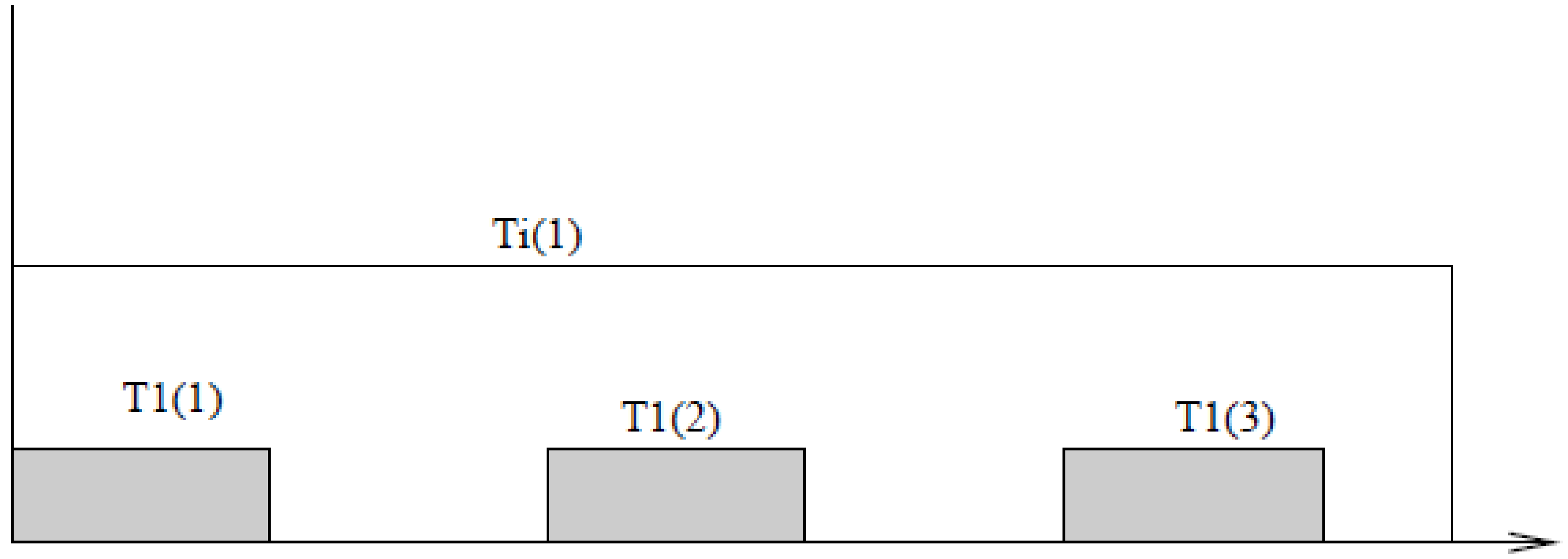


Figure 14: Instances of T_1 Over a Single Instance of T_i

- Let us now determine the exact number of times that T_1 occurs within a single instance of T_i. Given by [$\frac{p_i}{p_1}$]
- Since T_1's execution time is e_1, then the total execution time required due to task T_1 before the deadline of T_i is [$\frac{p_i}{p_1}$]*e_1.
- This expression can easily be generalized to consider the execution times all tasks having higher priority than T_i $(i.e\ T_1, T_2, ... T_{i-1})$.
- Therefore, the time for which T_i will have to wait due to all its higher priority tasks can be expressed as:

$$\sum_{k=1}^{i-1} \lceil \frac{p_i}{p_k} \rceil \times e_k$$

- So, the task T_i would meet its first deadline, iff

$$e_i + \sum_{k=1}^{i-1} \lceil \frac{p_i}{p_k} \rceil \times e_k \leq p_i$$

- Assuming that the task periods equal their respective deadlines, i.e $p_i = d_i$.
- If $p_i < d_i$, the expression need modification as follows.

$$e_i + \sum_{k=1}^{i-1} \lceil \frac{d_i}{p_k} \rceil \times e_k \leq d_i$$

Self Suspension

- In event driven scheduling, the scheduling points are defined by task completion, task arrival, and self-suspension events.
- Let the worst case self suspension time of a task T_i is b_i.
- Let the delay that the task T_i might incur due to its own self suspension and the self suspension of all higher priority tasks be bt_i.
- Then bt_i can be expresses as:

$$bt_i = b_i + \sum_{k=1}^{i-1} min(e_k, b_k)$$

- Considering the effect of self suspension on task completion time, the Lehoczky criterion would be generalized as:

$$e_i + bt_i + \sum_{k=1}^{i-1} \lceil \frac{p_i}{p_k} \rceil * e_k \leq p_i$$

Deadline Monotonic Algorithm (DMA)

- DMA is essentially a variant of RMA and assigns priorities to tasks based on their deadlines, rather than assigning priorities based on task periods as done in RMA.
- DMA assigns higher priorities to tasks with shorter deadlines.
- When the relative deadline of every task is proportional to its period, RMA and DMA produce identical solutions.
- When the relative deadlines are arbitrary, DMA is more proficient than RMA.

Coping with limited priority levels

- Sometimes the number of real time tasks in the application exceed the number of distinct priority levels supported by the underlying operating system.
- Limitations on memory size.
- The number of priority values typically varies from 8 to 256 in commercial operating systems.
- When there are more real time tasks in an application than the number of priority levels available from the OS, more than one task would have to be made to share a single priority value among themselves.

Priority assignment to tasks

- Uniform scheme
- Arithmetic scheme
- Geometric scheme
- Logarithmic scheme

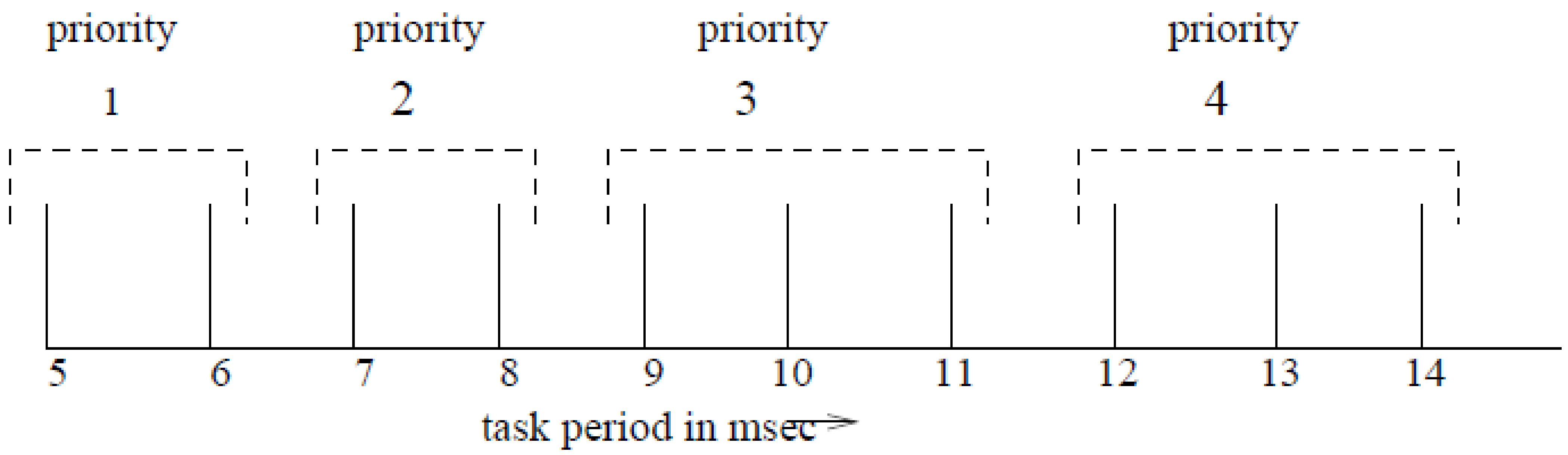
Uniform scheme

- Uniform division of tasks among the available priority levels can easily be achieved when the number of priority levels squarely divides the number of tasks to be scheduled.
- If uniform division is not possible, then more tasks should be made to share the lower priority levels.
- If there are N number of tasks and n priority levels then N/n number of tasks are assigned to each level and the rest of the tasks are distributed among the lower priority levels.

Example

In a certain application being developed, there are 10 periodic real-time tasks T_1, T_2…….. T_{10} whose periods are: 5,6,……14 milli secs respectively. These tasks are to be scheduled using RMA. However, only 4 priority levels are supported by the underlying operating system. In this operating system, the lower the priority value, the higher is the priority of the task. Assign suitable priority levels to tasks using the uniform assignment scheme for scheduling the tasks using RMA.

Priority Level 1: T_1, T_2.
Priority Level 2: T_3, T_4
Priority Level 3: T_5, T_6, T_7
Priority Level 4: T_8, T_9, T_{10}



Arithmetic scheme

- ‘r’ tasks having the shortest periods are assigned to the highest priority level, 2r tasks are assigned the next highest priority level and so on.
- Let N be the total number of tasks.
- Then N=r+2r+3r+4r+…….nr, where n is the total number of priority levels.

Geometric scheme

- If r tasks having the shortest periods are assigned the highest priority, then the next kr^2 tasks are assigned the immediately lower priority and so on.
- If N is the total number of tasks and n is the total number of priority levels then $N = r + kr^2$ + kr^3 +……. kr^n

Logarithmic scheme

- The shorter period (higher priority)tasks should be allotted distinct priority levels as much as possible.
- Many lower priority tasks can be clubbed together at the same priority levels without causing any problem to the schedulability of the high priority tasks
- For priority allocation, the range of task periods are divided into a sequence of logarithmic intervals.
- The tasks can then be assigned to priority levels based on the logarithmic interval they belong to.

- In this scheme, if p_{max} is the maximum period among the tasks and p_{min} is the minimum period among the tasks, then r is calculated as

$$r = \left(\frac{p_{max}}{p_{min}}\right)^{\frac{1}{n}}$$

where n is the total number of priority levels.

- Taks with periods up to r are assigned to the highest priority, tasks with periods in the range $r\ to\ r^2$ are assigned to next highest priority level, tasks with periods in the range r^2 to r^3 are assigned to the next highest level and so on.

Scheduling

- Turn Around Time (TAT)
  - Time period between completion and arrival i.e total time spent by process in the system (CT-AT)
- Completion Time (CT)
  - The time at which process is getting completed.
- Response Time (RT)
  - Once a process gets into the system how much time does it require to get CPU for first time.