Real-Time Scheduling

Scheduling of Reactive Systems

[Some parts of this lecture are based on a real-time systems course of Colin Perkins

http://csperkins.org/teaching/rtes/index.html]

Reminder of Basic Notions

- Jobs are executed on processors and need resources
- Parameters of jobs
  - temporal:
    - release time – r_i
    - execution time – e_i
    - absolute deadline – d_i
    - derived params: relative deadline (D_i), completion time, response time, ...
  - functional:
    - laxity type: hard vs soft
    - preemptability
  - interconnection
    - precedence constraints (independence)
  - resource
    - what resources and when are used by the job
- Tasks = sets of jobs

Scheduling Reactive Systems

We have considered scheduling of individual jobs

From this point on we concentrate on reactive systems

i.e. systems that run for unlimited amount of time

Scheduling Reactive Systems

We have considered scheduling of individual jobs

From this point on we concentrate on reactive systems

i.e. systems that run for unlimited amount of time

Recall that a task is a set of related jobs that jointly provide some system function.

Scheduling Reactive Systems

We have considered scheduling of individual jobs

From this point on we concentrate on reactive systems

i.e. systems that run for unlimited amount of time

Recall that a task is a set of related jobs that jointly provide some system function.

- We consider various types of tasks
  - Periodic
  - Aperiodic
  - Sporadic

Scheduling Reactive Systems

We have considered scheduling of individual jobs

From this point on we concentrate on reactive systems

i.e. systems that run for unlimited amount of time

Recall that a task is a set of related jobs that jointly provide some system function.

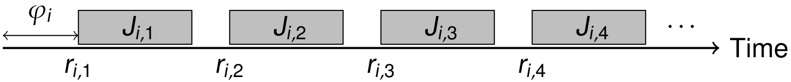
- We consider various types of tasks
  - Periodic
  - Aperiodic
  - Sporadic

- Differ in execution time patterns for jobs in the tasks

- Must be modeled differently
  - Differing scheduling algorithms
  - Differing impact on system performance
  - Differing constraints on scheduling

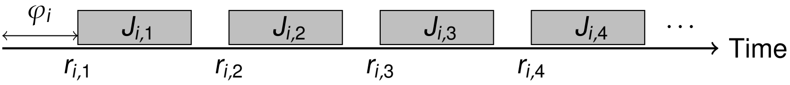
Periodic Tasks

A periodic task T_i is a sequence of jobs $J_{i,1}, J_{i,2}, \ldots J_{i,n}, \ldots$ with the constant differences between release times of consecutive jobs, the constant execution times, and the constant relative deadlines of all jobs.



Periodic Tasks

A periodic task T_i is a sequence of jobs $J_{i,1}, J_{i,2}, \dots J_{i,n}, \dots$ with the constant differences between release times of consecutive jobs, the constant execution times, and the constant relative deadlines of all jobs.



- The *phase* φ_i of a task T_i is *the release time* $r_{i,1}$ *of the first job* $J_{i,1}$ in the task T_i ;
  tasks are *in phase* if their phases are equal
- The *period* p_i of a task T_i is *the length of the constant time interval between release times of consecutive jobs* in T_i
- The *execution time* e_i of a task T_i is *the constant execution time of all jobs* in T_i
- The *relative deadline* D_i is *the constant relative deadline of all jobs* in T_i

Periodic Tasks – Notation

The 4-tuple $T_i = (\varphi_i, p_i, e_i, D_i)$ refers to a periodic task T_i with phase φ_i, period p_i, execution time e_i, and relative deadline D_i

Periodic Tasks – Notation

The 4-tuple $T_i = (\varphi_i, p_i, e_i, D_i)$ refers to a periodic task T_i with phase φ_i, period p_i, execution time e_i, and relative deadline D_i

For example: jobs of $T_1 = (1, 10, 3, 6)$ are

- released at times 1, 11, 21, $\ldots$,
- execute for 3 time units,
- have to be finished in 6 time units (the first by 7, the second by 17, ...)

Periodic Tasks – Notation

The 4-tuple $T_i = (\varphi_i, p_i, e_i, D_i)$ refers to a periodic task T_i with phase φ_i, period p_i, execution time e_i, and relative deadline D_i

For example: jobs of $T_1 = (1, 10, 3, 6)$ are

- released at times 1, 11, 21, $\ldots$,
- execute for 3 time units,
- have to be finished in 6 time units (the first by 7, the second by 17, ...)

Default phase of T_i is $\varphi_i = 0$ and default relative deadline is $d_i = p_i$

$T_2 = (10, 3, 6)$ satisfies $\varphi = 0$, $p_i = 10$, $e_i = 3$, $D_i = 6$, i.e. jobs of T_2 are

- released at times 0, 10, 20, $\ldots$,
- execute for 3 time units,
- have to be finished in 6 time units (the first by 6, the second by 16, ...)

Periodic Tasks – Notation

The 4-tuple $T_i = (\varphi_i, p_i, e_i, D_i)$ refers to a periodic task T_i with phase φ_i, period p_i, execution time e_i, and relative deadline D_i

For example: jobs of $T_1 = (1, 10, 3, 6)$ are

- released at times 1, 11, 21, ...,
- execute for 3 time units,
- have to be finished in 6 time units (the first by 7, the second by 17, ...)

Default phase of T_i is $\varphi_i = 0$ and default relative deadline is $d_i = p_i$

$T_2 = (10, 3, 6)$ satisfies $\varphi = 0$, $p_i = 10$, $e_i = 3$, $D_i = 6$, i.e. jobs of T_2 are

- released at times 0, 10, 20, ...,
- execute for 3 time units,
- have to be finished in 6 time units (the first by 6, the second by 16, ...)

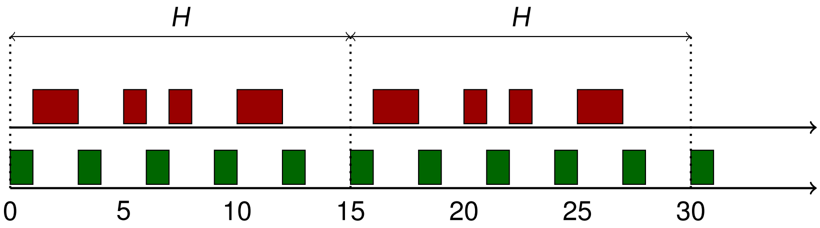
$T_3 = (10, 3)$ satisfies $\varphi = 0$, $p_i = 10$, $e_i = 3$, $D_i = 10$, i.e. jobs of T_3 are

- released at times 0, 10, 20, ...,
- execute for 3 time units,
- have to be finished in 10 time units (the first by 10, the second by 20, ...)

Periodic Tasks – Hyperperiod

The *hyper-period* H of a set of periodic tasks is the least common multiple of their periods

If tasks are in phase, then H is the time instant after which the pattern of job release/execution times starts to repeat



Aperiodic and Sporadic Tasks

- Many real-time systems are required to respond to external events

Aperiodic and Sporadic Tasks

- Many real-time systems are required to respond to external events
- The tasks resulting from such events are *sporadic* and *aperiodic* tasks
  - Sporadic tasks – hard deadlines of jobs
    e.g. autopilot on/off in aircraft

    The usual goal is to decide, whether a newly released job can be feasibly scheduled with the remaining jobs in the system
  - Aperiodic tasks – soft deadlines of jobs
    e.g. sensitivity adjustment of radar surveilance system

    The usual goal is to minimize the average response time
    For rigorous analysis we typically assume that the inter-arrival times between aperiodic jobs are distributed according to a known distribution.

Scheduling – Classification of Algorithms

- Off-line vs Online
  - Off-line – sched. algorithm is executed on the whole task set before activation
  - Online – schedule is updated at runtime every time a new task enters the system

The main division is on

- Clock-Driven
- Priority-Driven

Scheduling – Clock-Driven

- Decisions about what jobs execute when are made at specific time instants
  - these instants are chosen before the system begins execution
  - Usually regularly spaced, implemented using a periodic timer interrupt
  - Scheduler awakes after each interrupt, schedules jobs to execute for the next period, then blocks itself until the next interrupt

    E.g. the helicopter example with the interrupt every 1/180 th of a second

Scheduling – Clock-Driven

- Decisions about what jobs execute when are made at specific time instants
  - these instants are chosen before the system begins execution
  - Usually regularly spaced, implemented using a periodic timer interrupt
  - Scheduler awakes after each interrupt, schedules jobs to execute for the next period, then blocks itself until the next interrupt

    E.g. the helicopter example with the interrupt every 1/180 th of a second
- Typically in clock-driven systems:
  - All parameters of the real-time jobs are fixed and known
  - A schedule of the jobs is computed off-line and is stored for use at runtime; thus scheduling overhead at run-time can be minimized
  - Simple and straight-forward, not flexible

Scheduling – Priority-Driven

- Assign priorities to jobs, based on some algorithm
- Make scheduling decisions based on the priorities, when events such as releases and job completions occur
  - Priority scheduling algorithms are *event-driven*
  - Jobs are placed in one or more queues; at each event, the ready job with the highest priority is executed

(The assignment of jobs to priority queues, along with rules such as whether preemption is allowed, completely defines a priority-driven alg.)

Scheduling – Priority-Driven

- Assign priorities to jobs, based on some algorithm
- Make scheduling decisions based on the priorities, when events such as releases and job completions occur
  - Priority scheduling algorithms are *event-driven*
  - Jobs are placed in one or more queues; at each event, the ready job with the highest priority is executed

  (The assignment of jobs to priority queues, along with rules such as whether preemption is allowed, completely defines a priority-driven alg.)
- Priority-driven algs. make *locally optimal* scheduling decisions
  - Locally optimal scheduling is often *not* globally optimal
  - Priority-driven algorithms *never* intentionally leave idle processors

Scheduling – Priority-Driven

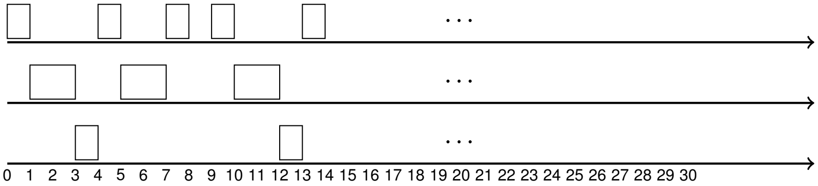
- Assign priorities to jobs, based on some algorithm
- Make scheduling decisions based on the priorities, when events such as releases and job completions occur
  - Priority scheduling algorithms are *event-driven*
  - Jobs are placed in one or more queues; at each event, the ready job with the highest priority is executed

  (The assignment of jobs to priority queues, along with rules such as whether preemption is allowed, completely defines a priority-driven alg.)
- Priority-driven algs. make *locally optimal* scheduling decisions
  - Locally optimal scheduling is often *not* globally optimal
  - Priority-driven algorithms *never* intentionally leave idle processors
- Typically in priority-driven systems:
  - Some parameters do not have to be fixed or known
  - A schedule is computed online; usually results in larger scheduling overhead as opposed to clock-driven scheduling
  - Flexible – easy to add/remove tasks or modify parameters

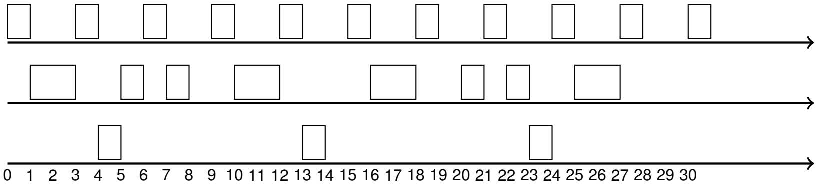
Clock-Driven & Priority-Driven Example

	T_1	T_2	T_3
p_i	3	5	10
e_i	1	2	1

Clock-Driven:



Priority-driven: $T_1 \succ T_2 \succ T_3$



Real-Time Scheduling

Scheduling of Reactive Systems

Priority-Driven Scheduling

Current Assumptions

- Single processor
- Fixed number, *n*, of *independent periodic* tasks
  i.e. there is no dependency relation among jobs
  - Jobs can be preempted at any time and never suspend themselves
  - No aperiodic and sporadic jobs
  - No resource contentions

Current Assumptions

- Single processor
- Fixed number, *n*, of *independent periodic* tasks

  i.e. there is no dependency relation among jobs

  - Jobs can be preempted at any time and never suspend themselves
  - No aperiodic and sporadic jobs
  - No resource contentions

Moreover, unless otherwise stated, we assume that

- **Scheduling decisions take place precisely at**
  - release of a job
  - completion of a job

  (and nowhere else)
- Context switch overhead is negligibly small

  i.e. assumed to be zero
- There is an unlimited number of priority levels

Fixed-Priority vs Dynamic-Priority Algorithms

A priority-driven scheduler is on-line
i.e. it does not precompute a schedule of the tasks

- It assigns priorities to jobs after they are released and places the jobs in a ready job queue in the priority order

  with the highest priority jobs at the head of the queue

- At each scheduling decision time, the scheduler updates the ready job queue and then schedules and executes the job at the head of the queue

  i.e. one of the jobs with the highest priority

Fixed-Priority vs Dynamic-Priority Algorithms

A priority-driven scheduler is on-line
i.e. it does not precompute a schedule of the tasks

- It assigns priorities to jobs after they are released and places the jobs in a ready job queue in the priority order

  with the highest priority jobs at the head of the queue

- At each scheduling decision time, the scheduler updates the ready job queue and then schedules and executes the job at the head of the queue

  i.e. one of the jobs with the highest priority

Fixed-priority = *all jobs in a task* are assigned the same priority

Dynamic-priority = jobs in a task may be assigned different priorities

Note: In our case, a priority assigned to a job does not change. There are *job-level dynamic priority* algorithms that vary priorities of individual jobs – we won't consider such algorithms.

Fixed-priority Algorithms – Rate Monotonic

Best known fixed-priority algorithm is *rate monotonic (RM)* scheduling that assigns priorities to tasks based on their periods

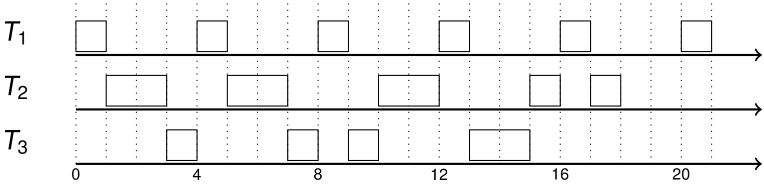
- The shorter the period, the higher the priority
- The *rate* is the inverse of the period, so jobs with higher rate have higher priority

RM is very widely studied and used

Example 1

$T_1 = (4, 1)$, $T_2 = (5, 2)$, $T_3 = (20, 5)$
with rates $1/4$, $1/5$, $1/20$, respectively

The priorities: $T_1 \succ T_2 \succ T_3$



Fixed-priority Algorithms – Deadline Monotonic

The *deadline monotonic (DM)* algorithm assigns priorities to tasks based on their *relative deadlines*

- the shorter the deadline, the higher the priority

Fixed-priority Algorithms – Deadline Monotonic

The *deadline monotonic (DM)* algorithm assigns priorities to tasks based on their *relative deadlines*

- the shorter the deadline, the higher the priority

Observation: When relative deadline of every task matches its period, then RM and DM give the same results

Proposition 1

When the relative deadlines are arbitrary DM can sometimes produce a feasible schedule in cases where RM cannot.

Proof.

Consider e.g. $T_1 = (3, 1, 1)$ and $T_2 = (2, 1)$. □

Dynamic-priority Algorithms – EDF

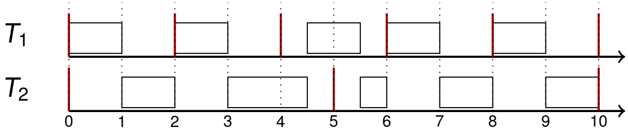
Earliest Deadline First (EDF) assigns priorities to jobs based on their *current* absolute deadlines

- At the time of a scheduling decision, the job queue is ordered by the earliest deadline
  the earlier the deadline, the larger the priority

We focus on EDF in this course!

EDF – Example

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



Note that the processor is 100% “utilized”, not surprising :-)

Other Dynamic-priority Algorithms - LST

Least Slack Time (LST): The job queue is ordered by least slack time.

The *slack time* of a job J_i at time t is equal to $d_i - t - x$ where x is the remaining computation time of J_i at time t

There is also a strict LST which reassigns priorities to jobs whenever their slacks change relative to each other – difficult to implement

This algorithm does not satisfy our assumptions!

Summary of Priority-Driven Algorithms

We consider:

Dynamic-priority:

- **EDF** = at the time of a scheduling decision, the job queue is ordered by the earliest deadline

Fixed-priority:

- **RM** = assigns priorities to tasks based on their periods
- **DM** = assigns priorities to tasks based on their relative deadlines

(In all cases, ties are broken arbitrarily.)

Summary of Priority-Driven Algorithms

We consider:
Dynamic-priority:

- **EDF** = at the time of a scheduling decision, the job queue is ordered by the earliest deadline

Fixed-priority:

- **RM** = assigns priorities to tasks based on their periods
- **DM** = assigns priorities to tasks based on their relative deadlines

(In all cases, ties are broken arbitrarily.)

We consider the following questions:

- Are the algorithms optimal?
- How to efficiently (or even online) test for schedulability?

Summary of Priority-Driven Algorithms

We consider:
Dynamic-priority:

- **EDF** = at the time of a scheduling decision, the job queue is ordered by the earliest deadline

Fixed-priority:

- **RM** = assigns priorities to tasks based on their periods
- **DM** = assigns priorities to tasks based on their relative deadlines

(In all cases, ties are broken arbitrarily.)

We consider the following questions:

- Are the algorithms optimal?
- How to efficiently (or even online) test for schedulability?

To measure abilities of scheduling algorithms and to get fast online tests of schedulability we use a notion of **utilization**

Utilization

- *Utilization u_i of a periodic task T_i* with period p_i and execution time e_i is defined by $u_i := e_i/p_i$

  u_i is the fraction of time a periodic task with period p_i and execution time e_i keeps a processor busy

Utilization

- *Utilization u_i of a periodic task T_i* with period p_i and execution time e_i is defined by $u_i := e_i/p_i$

  u_i is the fraction of time a periodic task with period p_i and execution time e_i keeps a processor busy
- *Total utilization $U^{\mathcal{T}}$ of a set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$* is defined as the sum of utilizations of all tasks of $\mathcal{T}$, i.e. by

$$U^{\mathcal{T}} := \sum_{i=1}^{n} u_i$$

Utilization

- *Utilization u_i of a periodic task T_i* with period p_i and execution time e_i is defined by $u_i := e_i/p_i$

  u_i is the fraction of time a periodic task with period p_i and execution time e_i keeps a processor busy
- *Total utilization $U^{\mathcal{T}}$ of a set of tasks $\mathcal{T} = \{T_1, \dots, T_n\}$* is defined as the sum of utilizations of all tasks of $\mathcal{T}$, i.e. by

$$U^{\mathcal{T}} := \sum_{i=1}^{n} u_i$$

- U is a *schedulable utilization* of an algorithm ALG if all sets of tasks $\mathcal{T}$ satisfying $U^{\mathcal{T}} \leq U$ are schedulable by ALG.

  Maximum schedulable utilization U_{ALG} of an algorithm ALG is the *supremum of schedulable utilizations of ALG*.
  - If $U^{\mathcal{T}} < U_{ALG}$, then $\mathcal{T}$ is schedulable by ALG.
  - If $U > U_{ALG}$, then there is $\mathcal{T}$ with $U^{\mathcal{T}} \leq U$ that is not schedulable by ALG.

Utilization – Example

- $T_1 = (2, 1)$ then $u_1 = \frac{1}{2}$

Utilization – Example

- $T_1 = (2, 1)$ then $u_1 = \frac{1}{2}$
- $T_1 = (11, 5, 2, 4)$ then $u_1 = \frac{2}{5}$
  (i.e., the phase and deadline do not play any role)

Utilization – Example

- $T_1 = (2, 1)$ then $u_1 = \frac{1}{2}$
- $T_1 = (11, 5, 2, 4)$ then $u_1 = \frac{2}{5}$
  (i.e., the phase and deadline do not play any role)
- $\mathcal{T} = \{T_1, T_2, T_3\}$ where $T_1 = (2, 1), T_2 = (6, 1), T_3 = (8, 3)$ then

$$U^{\mathcal{T}} = \frac{1}{2} + \frac{1}{6} + \frac{3}{8} = \frac{25}{24}$$

Real-Time Scheduling

Priority-Driven Scheduling

Dynamic-Priority

Optimality of EDF

Theorem 2
Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a set of independent, preemptable periodic tasks with $D_i \geq p_i$ for $i = 1, \ldots, n$. The following statements are equivalent:

1. *$\mathcal{T}$ can be feasibly scheduled on one processor*
2. *$U^{\mathcal{T}} \leq 1$*
3. *$\mathcal{T}$ is schedulable using EDF*

(i.e., in particular, $U_{EDF} = 1$)

Proof.

1.⇒2. We prove that $U^{\mathcal{T}} > 1$ implies that $\mathcal{T}$ is not schedulable

2.⇒3. We prove that if EDF fails to feasibly schedule, then $U^{\mathcal{T}} > 1$

3.⇒1. Trivial

□

Proof of 1.$\Rightarrow$2.

Assume that $U^{\mathcal{T}} = \sum_{i=1}^{N} \frac{e_i}{p_i} > 1$.

Proof of 1.⇒2.

Assume that $U^{\mathcal{T}} = \sum_{i=1}^{N} \frac{e_i}{p_i} > 1$.

Consider a time instant $t > \max_i \varphi_i$

(i.e. a time when all tasks are already "running")

Proof of 1.⇒2.

Assume that $U^{\mathcal{T}} = \sum_{i=1}^{N} \frac{e_i}{p_i} > 1$.

Consider a time instant $t > \max_i \varphi_i$
(i.e. a time when all tasks are already "running")

Observe that the number of jobs of T_i that are released in the time interval $[0, t]$ is $\left\lceil \frac{t-\varphi_i}{p_i} \right\rceil$. Thus a single processor needs $\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i$ time units to finish all jobs *released before or at t*.

Proof of 1.⇒2.

Assume that $U^{\mathcal{T}} = \sum_{i=1}^{N} \frac{e_i}{p_i} > 1$.

Consider a time instant $t > \max_i \varphi_i$

(i.e. a time when all tasks are already "running")

Observe that the number of jobs of T_i that are released in the time interval $[0, t]$ is $\left\lceil \frac{t-\varphi_i}{p_i} \right\rceil$. Thus a single processor needs $\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i$ time units to finish all jobs *released before or at t*.

However, the the total time to finish all jobs released before or at t is

$$\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i \geq \sum_{i=1}^{n} (t-\varphi_i) \cdot \frac{e_i}{p_i} = \sum_{i=1}^{n} t u_i - \varphi_i u_i = \sum_{i=1}^{n} t u_i - \sum_{i=1}^{n} \varphi_i u_i = t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i$$

Here $\sum_{i=1}^{n} \varphi_i u_i$ does not depend on t.

Proof of 1.⇒2.

Assume that $U^{\mathcal{T}} = \sum_{i=1}^{N} \frac{e_i}{p_i} > 1$.

Consider a time instant $t > \max_i \varphi_i$
(i.e. a time when all tasks are already "running")

Observe that the number of jobs of T_i that are released in the time interval $[0, t]$ is $\left\lceil \frac{t-\varphi_i}{p_i} \right\rceil$. Thus a single processor needs $\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i$ time units to finish all jobs *released before or at t*.

However, the the total time to finish all jobs released before or at t is

$$\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i \geq \sum_{i=1}^{n} (t-\varphi_i) \cdot \frac{e_i}{p_i} = \sum_{i=1}^{n} t u_i - \varphi_i u_i = \sum_{i=1}^{n} t u_i - \sum_{i=1}^{n} \varphi_i u_i = t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i$$

Here $\sum_{i=1}^{n} \varphi_i u_i$ does not depend on t.

Note that $\lim_{t\to\infty} \left(t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i\right) - t = \infty$. So there exists t such that $t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i > t + \max_i D_i$.

Proof of 1.⇒2.

Assume that $U^{\mathcal{T}} = \sum_{i=1}^{N} \frac{e_i}{p_i} > 1$.

Consider a time instant $t > \max_i \varphi_i$
(i.e. a time when all tasks are already "running")

Observe that the number of jobs of T_i that are released in the time interval $[0, t]$ is $\left\lceil \frac{t-\varphi_i}{p_i} \right\rceil$. Thus a single processor needs $\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i$ time units to finish all jobs *released before or at t*.

However, the the total time to finish all jobs released before or at t is

$$\sum_{i=1}^{n} \left\lceil \frac{t-\varphi_i}{p_i} \right\rceil \cdot e_i \geq \sum_{i=1}^{n} (t-\varphi_i) \cdot \frac{e_i}{p_i} = \sum_{i=1}^{n} t u_i - \varphi_i u_i = \sum_{i=1}^{n} t u_i - \sum_{i=1}^{n} \varphi_i u_i = t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i$$

Here $\sum_{i=1}^{n} \varphi_i u_i$ does not depend on t.

Note that $\lim_{t\to\infty} \left(t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i \right) - t = \infty$. So there exists t such that $t \cdot U^{\mathcal{T}} - \sum_{i=1}^{n} \varphi_i u_i > t + \max_i D_i$.

So in order to complete all jobs released before or at t we need more time than $t + \max_i D_i$. However, the latest deadline of a job released before or at t is $t + \max_i D_i$. So at least one job misses its deadline.

Proof of 2.$\Rightarrow$3. – Simplified

Let us start with a proof of a special case (see the assumptions A1 and A2 below). Then a complete proof will be presented.

We prove $\neg$3.$\Rightarrow$ $\neg$2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$.
(Note that the general case immediately follows.)

Proof of 2.$\Rightarrow$3. – Simplified

Let us start with a proof of a special case (see the assumptions A1 and A2 below). Then a complete proof will be presented.

We prove $\neg$3.$\Rightarrow$ $\neg$2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$.
(Note that the general case immediately follows.)

Assume that $\mathcal{T}$ is not schedulable according to EDF.
(Our goal is to show that $U^{\mathcal{T}} > 1$.)

Proof of 2.$\Rightarrow$3. – Simplified

Let us start with a proof of a special case (see the assumptions A1 and A2 below). Then a complete proof will be presented.

We prove $\neg$3.$\Rightarrow$ $\neg$2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$.
(Note that the general case immediately follows.)

Assume that $\mathcal{T}$ is not schedulable according to EDF.
(Our goal is to show that $U^{\mathcal{T}} > 1$.)

This means that there must be at least one job that misses its deadline when EDF is used.

Simplifying assumptions:

A1 Suppose that all tasks are in phase, i.e. the phase $\varphi_\ell = 0$ for every task T_ℓ.

A2 Suppose that *the first job* $J_{i,1}$ of a task T_i misses its deadline.

By A1, $J_{i,1}$ is released at 0 and misses its deadline at p_i. Assume w.l.o.g. that this is the first time when a job misses its deadline.
(To simplify even further, you may (privately) assume that no other job has its deadline at p_i.)

Proof of 2.⇒3. – Simplified

Let G be the set of all jobs released in $[0, p_i]$ with deadlines in $[0, p_i]$.

Proof of 2.$\Rightarrow$3. – Simplified

Let G be the set of all jobs released in $[0, p_i]$ with deadlines in $[0, p_i]$.

Crucial observations:

- G contains $J_{i,1}$

Proof of 2.⇒3. – Simplified

Let G be the set of all jobs released in $[0, p_i]$ with deadlines in $[0, p_i]$.

Crucial observations:

- G contains $J_{i,1}$
- Only jobs of G can be executed in $[0, p_i]$

  Jobs that do not belong to G *cannot* be executed in $[0, p_i]$ as $J_{i,1}$ is not completed in $[0, p_i]$ and only jobs of G can preempt $J_{i,1}$.

Proof of 2.$\Rightarrow$3. – Simplified

Let G be the set of all jobs released in $[0, p_i]$ with deadlines in $[0, p_i]$.

Crucial observations:

- G contains $J_{i,1}$
- Only jobs of G can be executed in $[0, p_i]$

  Jobs that do not belong to G *cannot* be executed in $[0, p_i]$ as $J_{i,1}$ is not completed in $[0, p_i]$ and only jobs of G can preempt $J_{i,1}$.
- The processor is never idle in $[0, p_i]$

  The processor is not idle because $J_{i,1}$ is ready for computation throughout $[0, p_i]$.

Proof of 2.$\Rightarrow$3. – Simplified

Let G be the set of all jobs released in $[0, p_i]$ with deadlines in $[0, p_i]$.

Crucial observations:

- G contains $J_{i,1}$
- Only jobs of G can be executed in $[0, p_i]$

  Jobs that do not belong to G *cannot* be executed in $[0, p_i]$ as $J_{i,1}$ is not completed in $[0, p_i]$ and only jobs of G can preempt $J_{i,1}$.
- The processor is never idle in $[0, p_i]$

  The processor is not idle because $J_{i,1}$ is ready for computation throughout $[0, p_i]$.

Denote by E_G the total execution time of G, that is, the sum of execution times of all jobs in G.

Corollary of the crucial observation: $E_G > p_i$ because otherwise $J_{i,1}$ (and all jobs that could possibly preempt it) would be completed by p_i.

Let us compute E_G.

Proof of 2.⇒3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ, the first job of T_ℓ is released at 0, and thus $\left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

E.g., if $p_\ell = 2$ and $p_i = 5$ then three jobs of T_ℓ are released in $[0, 5]$ (at times 0, 2, 4) but only $2 = \left\lfloor \frac{5}{2} \right\rfloor = \left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ of them have their deadlines in $[0, p_i]$.

Proof of 2.⇒3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ, the first job of T_ℓ is released at 0, and thus $\left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

E.g., if $p_\ell = 2$ and $p_i = 5$ then three jobs of T_ℓ are released in $[0, 5]$ (at times 0, 2, 4) but only $2 = \left\lfloor \frac{5}{2} \right\rfloor = \left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ of them have their deadlines in $[0, p_i]$.

Thus the total execution time E_G of all jobs in G is

$$E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{p_i}{p_\ell} \right\rfloor e_\ell$$

Proof of 2.⇒3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ, the first job of T_ℓ is released at 0, and thus $\left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

E.g., if $p_\ell = 2$ and $p_i = 5$ then three jobs of T_ℓ are released in $[0,5]$ (at times 0, 2, 4) but only $2 = \left\lfloor \frac{5}{2} \right\rfloor = \left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ of them have their deadlines in $[0,p_i]$.

Thus the total execution time E_G of all jobs in G is

$$E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{p_i}{p_\ell} \right\rfloor e_\ell$$

But then

$$p_i < E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{p_i}{p_\ell} \right\rfloor e_\ell \leq \sum_{\ell=1}^{n} \frac{p_i}{p_\ell} e_\ell \leq p_i \sum_{\ell=1}^{n} u_\ell \leq p_i \cdot U^{\mathcal{T}}$$

which implies that $U^{\mathcal{T}} > 1$.

Proof of 2.$\Rightarrow$3. – Complete

Now let us drop the simplifying assumptions A1 and A2 !

We prove $\neg$3.$\Rightarrow$ $\neg$2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$
(note that the general case immediately follows)

Proof of 2.⇒3. – Complete

Now let us drop the simplifying assumptions A1 and A2 !

We prove ¬3.⇒ ¬2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$
(note that the general case immediately follows)

Assume that $\mathcal{T}$ is not schedulable by EDF.
(We show that $U^{\mathcal{T}} > 1$)

Proof of 2.⇒3. – Complete

Now let us drop the simplifying assumptions A1 and A2 !

We prove ¬3.⇒ ¬2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$
(note that the general case immediately follows)

Assume that $\mathcal{T}$ is not schedulable by EDF.
(We show that $U^{\mathcal{T}} > 1$)

Suppose that a job $J_{i,k}$ of T_i misses its deadline at time $t = r_{i,k} + p_i$.
Assume that this is the earliest deadline miss.

Proof of 2.⇒3. – Complete

Now let us drop the simplifying assumptions A1 and A2 !

We prove ¬3.⇒ ¬2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$
(note that the general case immediately follows)

Assume that $\mathcal{T}$ is not schedulable by EDF.
(We show that $U^{\mathcal{T}} > 1$)

Suppose that a job $J_{i,k}$ of T_i misses its deadline at time $t = r_{i,k} + p_i$.
Assume that this is the earliest deadline miss.

Let t_- be the end of the *last interval* before t in which either jobs with deadlines after t are being executed, or the processor is idle.

Proof of 2.⇒3. – Complete

Now let us drop the simplifying assumptions A1 and A2 !

We prove ¬3.⇒ ¬2. assuming that $D_i = p_i$ for $i = 1, \dots, n$
(note that the general case immediately follows)

Assume that $\mathcal{T}$ is not schedulable by EDF.
(We show that $U^{\mathcal{T}} > 1$)

Suppose that a job $J_{i,k}$ of T_i misses its deadline at time $t = r_{i,k} + p_i$.
Assume that this is the earliest deadline miss.

Let t_{-} be the end of the *last interval* before t in which either jobs with deadlines after t are being executed, or the processor is idle.

Let G be the set of all jobs released in $[t_{-}, t]$ with deadlines in $[t_{-}, t]$.

Proof of 2.$\Rightarrow$3. – Complete (cont.)

- G contains $J_{i,k}$

  Note that $t_- \leq r_{i,k}$ because otherwise either $J_{i,k}$ or another job with a deadline at, or before t would be executed just before t_-.

Proof of 2.⇒3. – Complete (cont.)

- G contains $J_{i,k}$

  Note that $t_- \leq r_{i,k}$ because otherwise either $J_{i,k}$ or another job with a deadline at, or before t would be executed just before t_-.

- Only jobs of G can be executed in $[t_-, t]$

  Indeed, by definition of t_-:

  - All jobs (possibly) executed in $[t_-, t]$ must have their deadlines at, or before t by the definition of t_-.

Proof of 2.⇒3. – Complete (cont.)

- G contains $J_{i,k}$

  Note that $t_- \leq r_{i,k}$ because otherwise either $J_{i,k}$ or another job with a deadline at, or before t would be executed just before t_-.

- Only jobs of G can be executed in $[t_-, t]$

  Indeed, by definition of t_-:

  - All jobs (possibly) executed in $[t_-, t]$ must have their deadlines at, or before t by the definition of t_-.
  - If an idle interval precedes t_-, then all jobs with deadlines at, or before t must be released at, or after t_- because otherwise one of them would have been executed just before t_-.

Proof of 2.⇒3. – Complete (cont.)

- G contains $J_{i,k}$

  Note that $t_{-} \leq r_{i,k}$ because otherwise either $J_{i,k}$ or another job with a deadline at, or before t would be executed just before t_{-}.

- Only jobs of G can be executed in $[t_{-}, t]$

  Indeed, by definition of t_{-}:

  - All jobs (possibly) executed in $[t_{-}, t]$ must have their deadlines at, or before t by the definition of t_{-}.
  - If an idle interval precedes t_{-}, then all jobs with deadlines at, or before t must be released at, or after t_{-} because otherwise one of them would have been executed just before t_{-}.
  - If a job with its deadline after t is executed just before t_{-}, then all jobs with deadlines at, or before t must be released in $[t_{-}, t]$ because otherwise one of them would have been executed just before t_{-}.

Proof of 2.⇒3. – Complete (cont.)

- G contains $J_{i,k}$

  Note that $t_{-} \leq r_{i,k}$ because otherwise either $J_{i,k}$ or another job with a deadline at, or before t would be executed just before t_{-}.

- Only jobs of G can be executed in $[t_{-}, t]$

  Indeed, by definition of t_{-}:

  - All jobs (possibly) executed in $[t_{-}, t]$ must have their deadlines at, or before t by the definition of t_{-}.
  - If an idle interval precedes t_{-}, then all jobs with deadlines at, or before t must be released at, or after t_{-} because otherwise one of them would have been executed just before t_{-}.
  - If a job with its deadline after t is executed just before t_{-}, then all jobs with deadlines at, or before t must be released in $[t_{-}, t]$ because otherwise one of them would have been executed just before t_{-}.

- The processor is never idle in $[t_{-}, t]$ by definition of t_{-}

Denote by E_G the sum of all execution times of all jobs in G.

Proof of 2.⇒3. – Complete (cont.)

Now $E_G > t - t_-$ because otherwise $J_{i,k}$ would complete in $[t_-, t]$.

How to compute E_G?

Proof of 2.$\Rightarrow$3. – Complete (cont.)

Now $E_G > t - t_-$ because otherwise $J_{i,k}$ would complete in $[t_-, t]$.

How to compute E_G?

For a task T_ℓ, denote by R_ℓ the earliest release time of a job in T_ℓ in the interval $[t_-, t]$.

Proof of 2.⇒3. – Complete (cont.)

Now $E_G > t - t_-$ because otherwise $J_{i,k}$ would complete in $[t_-, t]$.

How to compute E_G?

For a task T_ℓ, denote by R_ℓ the earliest release time of a job in T_ℓ in the interval $[t_-, t]$.

For every T_ℓ, exactly $\left\lfloor \frac{t-R_\ell}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

Proof of 2.⇒3. – Complete (cont.)

Now $E_G > t - t_-$ because otherwise $J_{i,k}$ would complete in $[t_-, t]$.

How to compute E_G?

For a task T_ℓ, denote by R_ℓ the earliest release time of a job in T_ℓ in the interval $[t_-, t]$.

For every T_ℓ, exactly $\left\lfloor \frac{t-R_\ell}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

Thus

$$E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{t - R_\ell}{p_\ell} \right\rfloor e_\ell$$

Proof of 2.⇒3. – Complete (cont.)

Now $E_G > t - t_-$ because otherwise $J_{i,k}$ would complete in $[t_-, t]$.

How to compute E_G?

For a task T_ℓ, denote by R_ℓ the earliest release time of a job in T_ℓ in the interval $[t_-, t]$.

For every T_ℓ, exactly $\left\lfloor \frac{t-R_\ell}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

Thus

$$E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{t - R_\ell}{p_\ell} \right\rfloor e_\ell$$

As argued above:

$$t - t_- < E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{t - R_\ell}{p_\ell} \right\rfloor e_\ell \le \sum_{\ell=1}^{n} \frac{t - t_-}{p_\ell} e_\ell \le (t - t_-) \sum_{\ell=1}^{n} u_\ell \le (t - t_-) U^{\mathcal{T}}$$

which implies that $U^{\mathcal{T}} > 1$.

Density and EDF

What about tasks with $D_i < p_i$?

Density and EDF

What about tasks with $D_i < p_i$?

Density of a task T_i with period p_i, execution time e_i and relative deadline D_i is defined by

$$e_i / \min(D_i, p_i)$$

Total density $\Delta^{\mathcal{T}}$ of a set of tasks $\mathcal{T}$ is the sum of densities of tasks in $\mathcal{T}$

Note that if $D_i < p_i$ for some i, then $\Delta^{\mathcal{T}} > U^{\mathcal{T}}$

Density and EDF

What about tasks with $D_i < p_i$?

Density of a task T_i with period p_i, execution time e_i and relative deadline D_i is defined by

$$e_i / \min(D_i, p_i)$$

Total density $\Delta^{\mathcal{T}}$ of a set of tasks $\mathcal{T}$ is the sum of densities of tasks in $\mathcal{T}$

Note that if $D_i < p_i$ for some i, then $\Delta^{\mathcal{T}} > U^{\mathcal{T}}$

Theorem 3

A set $\mathcal{T}$ of independent, preemptable, periodic tasks can be feasibly scheduled on one processor if $\Delta^{\mathcal{T}} \leq 1$.

Note that this is NOT a necessary condition!

Schedulability Test For EDF

The problem: Given a set of independent, preemptable, periodic tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ where each T_i has a period p_i, execution time e_i, and relative deadline D_i, decide whether $\mathcal{T}$ is schedulable by EDF.

Schedulability Test For EDF

The problem: Given a set of independent, preemptable, periodic tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ where each T_i has a period p_i, execution time e_i, and relative deadline D_i, decide whether $\mathcal{T}$ is schedulable by EDF.

Solution using utilization and density:

If $p_i \leq D_i$ for each i, then it suffices to decide whether $U^{\mathcal{T}} \leq 1$.

Otherwise, decide whether $\Delta^{\mathcal{T}} \leq 1$:

- If yes, then $\mathcal{T}$ is schedulable with EDF
- If not, then $\mathcal{T}$ does not have to be schedulable

Schedulability Test For EDF

The problem: Given a set of independent, preemptable, periodic tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ where each T_i has a period p_i, execution time e_i, and relative deadline D_i, decide whether $\mathcal{T}$ is schedulable by EDF.

Solution using utilization and density:

If $p_i \leq D_i$ for each i, then it suffices to decide whether $U^{\mathcal{T}} \leq 1$.

Otherwise, decide whether $\Delta^{\mathcal{T}} \leq 1$:

- If yes, then $\mathcal{T}$ is schedulable with EDF
- If not, then $\mathcal{T}$ does not have to be schedulable

Note that

- Phases of tasks do not have to be specified
- Parameters may vary: increasing periods or deadlines, or decreasing execution times does not prevent schedulability

Schedulability Test for EDF – Example

Consider a digital robot controller

- A control-law computation
  - takes no more than 8 ms
  - the sampling rate: 100 Hz, i.e. computes every 10 ms

Schedulability Test for EDF – Example

Consider a digital robot controller

- A control-law computation
  - takes no more than 8 ms
  - the sampling rate: 100 Hz, i.e. computes every 10 ms

  Feasible? Trivially yes ....
- Add Built-In Self-Test (BIST)
  - maximum execution time 50 ms
  - want a minimal period that is feasible (max one second)

Schedulability Test for EDF – Example

Consider a digital robot controller

- A control-law computation
  - takes no more than 8 ms
  - the sampling rate: 100 Hz, i.e. computes every 10 ms

  Feasible? Trivially yes ....
- Add Built-In Self-Test (BIST)
  - maximum execution time 50 ms
  - want a minimal period that is feasible (max one second)

  With 250 ms still feasible ....
- Add a telemetry task
  - maximum execution time 15 ms
  - want to minimize the deadline on telemetry
    period may be large

Schedulability Test for EDF – Example

Consider a digital robot controller

- A control-law computation
  - takes no more than 8 ms
  - the sampling rate: 100 Hz, i.e. computes every 10 ms

  Feasible? Trivially yes ....
- Add Built-In Self-Test (BIST)
  - maximum execution time 50 ms
  - want a minimal period that is feasible (max one second)

  With 250 ms still feasible ....
- Add a telemetry task
  - maximum execution time 15 ms
  - want to minimize the deadline on telemetry
    period may be large

  Reducing BIST to once a second, deadline on telemetry may be set to 100 ms ....

Real-Time Scheduling

Priority-Driven Scheduling

Fixed-Priority

Fixed-Priority Algorithms

Recall that we consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$

Fixed-Priority Algorithms

Recall that we consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$

Any fixed-priority algorithm schedules tasks of $\mathcal{T}$ according to fixed (distinct) priorities *assigned to tasks.*

Fixed-Priority Algorithms

Recall that we consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$

Any fixed-priority algorithm schedules tasks of $\mathcal{T}$ according to fixed (distinct) priorities *assigned to tasks.*

We write $T_i \sqsupset T_j$ whenever T_i has a higher priority than T_j.

Fixed-Priority Algorithms

Recall that we consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$

Any fixed-priority algorithm schedules tasks of $\mathcal{T}$ according to fixed (distinct) priorities *assigned to tasks.*

We write $T_i \sqsupset T_j$ whenever T_i has a higher priority than T_j.

To simplify our reasoning, assume that

all tasks are in phase, i.e. $\varphi_k = 0$ **for all** T_k**.**

We will remove this assumption at the end.

Fixed-Priority Algorithms – Reminder

Recall that Fixed-Priority Algorithms do not have to be optimal.
Consider $\mathcal{T} = \{T_1, T_2\}$ where $T_1 = (2, 1)$ and $T_2 = (5, 2.5)$

$U^{\mathcal{T}} = 1$ and thus $\mathcal{T}$ is schedulable by EDF

If $T_1 \sqsupset T_2$, then $J_{2,1}$ misses its deadline
If $T_2 \sqsupset T_1$, then $J_{1,1}$ misses its deadline

Fixed-Priority Algorithms – Reminder

Recall that Fixed-Priority Algorithms do not have to be optimal.
Consider $\mathcal{T} = \{T_1, T_2\}$ where $T_1 = (2, 1)$ and $T_2 = (5, 2.5)$

$U^{\mathcal{T}} = 1$ and thus $\mathcal{T}$ is schedulable by EDF

If $T_1 \sqsupset T_2$, then $J_{2,1}$ misses its deadline
If $T_2 \sqsupset T_1$, then $J_{1,1}$ misses its deadline

We consider the following algorithms:

- **RM** = assigns priorities to tasks based on their periods
  the priority is inversely proportional to the period p_i
- **DM** = assigns priorities to tasks based on their relative deadlines
  the priority is inversely proportional to the relative deadline D_i

(In all cases, ties are broken arbitrarily.)

Fixed-Priority Algorithms – Reminder

Recall that Fixed-Priority Algorithms do not have to be optimal.

Consider $\mathcal{T} = \{T_1, T_2\}$ where $T_1 = (2, 1)$ and $T_2 = (5, 2.5)$

$U^{\mathcal{T}} = 1$ and thus $\mathcal{T}$ is schedulable by EDF

If $T_1 \sqsupset T_2$, then $J_{2,1}$ misses its deadline

If $T_2 \sqsupset T_1$, then $J_{1,1}$ misses its deadline

We consider the following algorithms:

- **RM** = assigns priorities to tasks based on their periods

  the priority is inversely proportional to the period p_i
- **DM** = assigns priorities to tasks based on their relative deadlines

  the priority is inversely proportional to the relative deadline D_i

(In all cases, ties are broken arbitrarily.)

We consider the following questions:

- Are the algorithms optimal?
- How to efficiently (or even online) test for schedulability?

Maximum Response Time

Which job of a task T_i has the maximum response time?

Maximum Response Time

Which job of a task T_i has the maximum response time?

As all tasks are in phase, the first job of T_i is released together with (first) jobs of all tasks that have higher priority than T_i.

Maximum Response Time

Which job of a task T_i has the maximum response time?

As all tasks are in phase, the first job of T_i is released together with (first) jobs of all tasks that have higher priority than T_i.

This means, that $J_{i,1}$ is the most preempted of jobs in T_i.

Maximum Response Time

Which job of a task T_i has the maximum response time?

As all tasks are in phase, the first job of T_i is released together with (first) jobs of all tasks that have higher priority than T_i.

This means, that $J_{i,1}$ is the most preempted of jobs in T_i.

It follows, that $J_{i,1}$ has the maximum response time.
Note that this relies heavily on the assumption that tasks are in phase!

Maximum Response Time

Which job of a task T_i has the maximum response time?

As all tasks are in phase, the first job of T_i is released together with (first) jobs of all tasks that have higher priority than T_i.

This means, that $J_{i,1}$ is the most preempted of jobs in T_i.

It follows, that $J_{i,1}$ has the maximum response time.
Note that this relies heavily on the assumption that tasks are in phase!

Thus in order to decide whether $\mathcal{T}$ is schedulable, it suffices to test for schedulability of the first jobs of all tasks.

Optimality of RM for Simply Periodic Tasks

Definition 4

A set $\{T_1, \ldots, T_n\}$ is **simply periodic** if for every pair T_i, T_ℓ satisfying $p_i > p_\ell$ we have that p_i is an integer multiple of p_ℓ

Example 5

The helicopter control system from the first lecture.

Optimality of RM for Simply Periodic Tasks

Definition 4

A set $\{T_1, \ldots, T_n\}$ is **simply periodic** if for every pair T_i, T_ℓ satisfying $p_i > p_\ell$ we have that p_i is an integer multiple of p_ℓ

Example 5

The helicopter control system from the first lecture.

Theorem 6

A set $\mathcal{T}$ of n simply periodic, independent, preemptable tasks with $D_i = p_i$ is schedulable on one processor according to RM **iff** $U^{\mathcal{T}} \leq 1$.

i.e. on simply periodic tasks RM is as good as EDF

Note: Theorem 6 is true in general, no "in phase" assumption is needed.

Proof of Theorem 6

By Theorem 2, every schedulable set $\mathcal{T}$ satisfies $U^{\mathcal{T}} \leq 1$.

Proof of Theorem 6

By Theorem 2, every schedulable set $\mathcal{T}$ satisfies $U^{\mathcal{T}} \leq 1$.

We prove that if $\mathcal{T}$ is not schedulable *according to RM*, then $U^{\mathcal{T}} > 1$.

Proof of Theorem 6

By Theorem 2, every schedulable set $\mathcal{T}$ satisfies $U^{\mathcal{T}} \leq 1$.

We prove that if $\mathcal{T}$ is not schedulable *according to RM*, then $U^{\mathcal{T}} > 1$.

Assume that a job $J_{i,1}$ of T_i misses its deadline at $D_i = p_i$. W.l.o.g., we assume that $T_1 \sqsupset \cdots \sqsupset T_n$ according to RM.

Proof of Theorem 6

By Theorem 2, every schedulable set $\mathcal{T}$ satisfies $U^{\mathcal{T}} \leq 1$.

We prove that if $\mathcal{T}$ is not schedulable *according to RM*, then $U^{\mathcal{T}} > 1$.

Assume that a job $J_{i,1}$ of T_i misses its deadline at $D_i = p_i$. W.l.o.g., we assume that $T_1 \sqsupset \cdots \sqsupset T_n$ according to RM.

Let us compute the total execution time of $J_{i,1}$ and all jobs that preempt it:

$$E = e_i + \sum_{\ell=1}^{i-1} \left\lceil \frac{p_i}{p_\ell} \right\rceil e_\ell = \sum_{\ell=1}^{i} \frac{p_i}{p_\ell} e_\ell = p_i \sum_{\ell=1}^{i} u_\ell \leq p_i \sum_{\ell=1}^{n} u_\ell = p_i U^{\mathcal{T}}$$

Proof of Theorem 6

By Theorem 2, every schedulable set $\mathcal{T}$ satisfies $U^{\mathcal{T}} \leq 1$.

We prove that if $\mathcal{T}$ is not schedulable *according to RM*, then $U^{\mathcal{T}} > 1$.

Assume that a job $J_{i,1}$ of T_i misses its deadline at $D_i = p_i$. W.l.o.g., we assume that $T_1 \sqsupset \cdots \sqsupset T_n$ according to RM.

Let us compute the total execution time of $J_{i,1}$ and all jobs that preempt it:

$$E = e_i + \sum_{\ell=1}^{i-1} \left\lceil \frac{p_i}{p_\ell} \right\rceil e_\ell = \sum_{\ell=1}^{i} \frac{p_i}{p_\ell} e_\ell = p_i \sum_{\ell=1}^{i} u_\ell \leq p_i \sum_{\ell=1}^{n} u_\ell = p_i U^{\mathcal{T}}$$

Now $E > p_i$ because otherwise $J_{i,1}$ meets its deadline.

Proof of Theorem 6

By Theorem 2, every schedulable set $\mathcal{T}$ satisfies $U^{\mathcal{T}} \leq 1$.

We prove that if $\mathcal{T}$ is not schedulable *according to RM*, then $U^{\mathcal{T}} > 1$.

Assume that a job $J_{i,1}$ of T_i misses its deadline at $D_i = p_i$. W.l.o.g., we assume that $T_1 \sqsupset \cdots \sqsupset T_n$ according to RM.

Let us compute the total execution time of $J_{i,1}$ and all jobs that preempt it:

$$E = e_i + \sum_{\ell=1}^{i-1} \left\lceil \frac{p_i}{p_\ell} \right\rceil e_\ell = \sum_{\ell=1}^{i} \frac{p_i}{p_\ell} e_\ell = p_i \sum_{\ell=1}^{i} u_\ell \leq p_i \sum_{\ell=1}^{n} u_\ell = p_i U^{\mathcal{T}}$$

Now $E > p_i$ because otherwise $J_{i,1}$ meets its deadline. Thus

$$p_i < E \leq p_i U^{\mathcal{T}}$$

and we obtain $U^{\mathcal{T}} > 1$.

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 7
A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm.

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 7

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm*.*

Proof.

Assume a fixed-priority feasible schedule with $T_1 \sqsupset \cdots \sqsupset T_n$.

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 7
A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm.

Proof.
Assume a fixed-priority feasible schedule with $T_1 \sqsupset \cdots \sqsupset T_n$.

Consider the least i such that the relative deadline D_i of T_i is larger than the relative deadline D_{i+1} of T_{i+1}.

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 7
A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm*.*

Proof.
Assume a fixed-priority feasible schedule with $T_1 \sqsupset \cdots \sqsupset T_n$.

Consider the least i such that the relative deadline D_i of T_i is larger than the relative deadline D_{i+1} of T_{i+1}.

Swap the priorities of T_i and T_{i+1}.

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 7

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm.

Proof.

Assume a fixed-priority feasible schedule with $T_1 \sqsupset \cdots \sqsupset T_n$.

Consider the least i such that the relative deadline D_i of T_i is larger than the relative deadline D_{i+1} of T_{i+1}.

Swap the priorities of T_i and T_{i+1}.

The resulting schedule is still feasible.

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 7

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm.

Proof.

Assume a fixed-priority feasible schedule with $T_1 \sqsupset \cdots \sqsupset T_n$.

Consider the least i such that the relative deadline D_i of T_i is larger than the relative deadline D_{i+1} of T_{i+1}.

Swap the priorities of T_i and T_{i+1}.

The resulting schedule is still feasible.

DM is obtained by using finitely many swaps. $\square$

Note: If the assumptions of the above theorem hold and all relative deadlines are equal to periods, then RM is optimal among all fixed-priority algorithms.

Fixed-Priority Algorithms: Schedulability

We consider two schedulability tests:

- Schedulable utilization U_{RM} of the RM algorithm.
- Time-demand analysis based on response times.

Schedulable Utilization for RM

Theorem 8

Let us fix $n \in \mathbb{N}$ and consider only independent, preemptable periodic tasks with $D_i = p_i$.

Schedulable Utilization for RM

Theorem 8

Let us fix $n \in \mathbb{N}$ and consider only independent, preemptable periodic tasks with $D_i = p_i$.

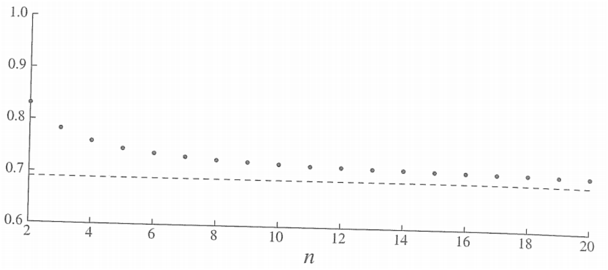
- *If $\mathcal{T}$ is a set of n tasks satisfying $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$, then $U^{\mathcal{T}}$ is schedulable according to the RM algorithm.*

Schedulable Utilization for RM

Theorem 8

Let us fix $n \in \mathbb{N}$ and consider only independent, preemptable periodic tasks with $D_i = p_i$.

- *If $\mathcal{T}$ is a set of n tasks satisfying $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$, then $U^{\mathcal{T}}$ is schedulable according to the RM algorithm.*
- *For every $U > n(2^{1/n} - 1)$ there is a set $\mathcal{T}$ of n tasks satisfying $U^{\mathcal{T}} \leq U$ that is not schedulable by RM.*



Note: Theorem 8 holds in general, no "in phase" assumption is needed.

Schedulable Utilization for RM

It follows that the maximum schedulable utilization U_{RM} over independent, preemptable periodic tasks satisfies

$$U_{RM} = \inf_{n} n(2^{1/n} - 1) = \lim_{n \to \infty} n(2^{1/n} - 1) = \ln 2 \approx 0.693$$

Note that $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$ is a sufficient but not necessary condition for schedulability of $\mathcal{T}$ using the RM algorithm (an example will be given later)

Schedulable Utilization for RM

It follows that the maximum schedulable utilization U_{RM} over independent, preemptable periodic tasks satisfies

$$U_{RM} = \inf_n n(2^{1/n} - 1) = \lim_{n \to \infty} n(2^{1/n} - 1) = \ln 2 \approx 0.693$$

Note that $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$ is a sufficient but not necessary condition for schedulability of $\mathcal{T}$ using the RM algorithm (an example will be given later)

We say that a set of tasks $\mathcal{T}$ is *RM-schedulable* if it is schedulable according to RM.

We say that $\mathcal{T}$ is *RM-infeasible* if it is not RM-schedulable.

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Outline: Given p_1, p_2, e_1, denote by max_e_2 the maximum execution time so that $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$ is RM-schedulable.

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Outline: Given p_1, p_2, e_1, denote by max_e_2 the maximum execution time so that $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$ is RM-schedulable.

We define $U_{e_1}^{p_1,p_2}$ to be $U^{\mathcal{T}}$ where $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.
We say that $\mathcal{T}$ fully utilizes the processor, any increase in an execution time causes RM-infeasibility.

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Outline: Given p_1, p_2, e_1, denote by max_e_2 the maximum execution time so that $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$ is RM-schedulable.

We define $U_{e_1}^{p_1,p_2}$ to be $U^{\mathcal{T}}$ where $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.
We say that $\mathcal{T}$ fully utilizes the processor, any increase in an execution time causes RM-infeasibility.

Now **we find the (global) minimum** $minU$ **of** $U_{e_1}^{p_1,p_2}$**.**

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Outline: Given p_1, p_2, e_1, denote by max_e_2 the maximum execution time so that $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$ is RM-schedulable.

We define $U_{e_1}^{p_1,p_2}$ to be $U^{\mathcal{T}}$ where $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.
We say that $\mathcal{T}$ fully utilizes the processor, any increase in an execution time causes RM-infeasibility.

Now **we find the (global) minimum** $minU$ **of** $U_{e_1}^{p_1,p_2}$**.**
Note that this suffices to obtain the desired result:

- Given a set of tasks $\mathcal{T} = \{(p_1, e_1), (p_2, e_2)\}$ satisfying $U^{\mathcal{T}} \leq minU$ we get $U^{\mathcal{T}} \leq minU \leq U_{e_1}^{p_1,p_2}$, and thus the execution time e_2 cannot be larger than max_e_2. Thus, $\mathcal{T}$ is RM-schedulable.

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Outline: Given p_1, p_2, e_1, denote by max_e_2 the maximum execution time so that $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$ is RM-schedulable.

We define $U_{e_1}^{p_1,p_2}$ to be $U^{\mathcal{T}}$ where $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.
We say that $\mathcal{T}$ fully utilizes the processor, any increase in an execution time causes RM-infeasibility.

Now **we find the (global) minimum** $minU$ **of** $U_{e_1}^{p_1,p_2}$**.**
Note that this suffices to obtain the desired result:

- Given a set of tasks $\mathcal{T} = \{(p_1, e_1), (p_2, e_2)\}$ satisfying $U^{\mathcal{T}} \leq minU$ we get $U^{\mathcal{T}} \leq minU \leq U_{e_1}^{p_1,p_2}$, and thus the execution time e_2 cannot be larger than max_e_2. Thus, $\mathcal{T}$ is RM-schedulable.
- Given $U > minU$, there must be p_1, p_2, e_1 satisfying $minU \leq U_{e_1}^{p_1,p_2} < U$ where $U_{e_1}^{p_1,p_2} = U^{\mathcal{T}}$ for a set of tasks $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Outline: Given p_1, p_2, e_1, denote by max_e_2 the maximum execution time so that $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$ is RM-schedulable.

We define $U_{e_1}^{p_1,p_2}$ to be $U^{\mathcal{T}}$ where $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.
We say that $\mathcal{T}$ fully utilizes the processor, any increase in an execution time causes RM-infeasibility.

Now **we find the (global) minimum** $minU$ **of** $U_{e_1}^{p_1,p_2}$**.**
Note that this suffices to obtain the desired result:

- Given a set of tasks $\mathcal{T} = \{(p_1, e_1), (p_2, e_2)\}$ satisfying $U^{\mathcal{T}} \leq minU$ we get $U^{\mathcal{T}} \leq minU \leq U_{e_1}^{p_1,p_2}$, and thus the execution time e_2 cannot be larger than max_e_2. Thus, $\mathcal{T}$ is RM-schedulable.
- Given $U > minU$, there must be p_1, p_2, e_1 satisfying $minU \leq U_{e_1}^{p_1,p_2} < U$ where $U_{e_1}^{p_1,p_2} = U^{\mathcal{T}}$ for a set of tasks $\mathcal{T} = \{(p_1, e_1), (p_2, max\_e_2)\}$.
  However, now increasing e_1 by a sufficiently small $\varepsilon > 0$ makes the set RM-infeasible without making utilization larger than U.

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$U_{e_1}^{p_1,p_2}$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$.

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_1 - e_1}{p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_1 - e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_1}{p_2} - \frac{e_1}{p_2}$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$.
Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_1 - e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_1}{p_2} - \frac{e_1}{p_2} = \frac{p_1}{p_2} + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)$$

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_1 - e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_1}{p_2} - \frac{e_1}{p_2} = \frac{p_1}{p_2} + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)$$

As $\frac{p_2}{p_1} - 1 \geq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by minimizing e_1.

Proof – Special Case (Cont.)

Consider two cases depending on e_1:

1. $e_1 < p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_2 - 2e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_2 - 2e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_2}{p_2} - \frac{2e_1}{p_2} = 1 + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 2\right)$$

As $\frac{p_2}{p_1} - 2 \leq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by maximizing e_1.

2. $e_1 \geq p_2 - p_1$:

Maximum RM-feasible max_e_2 (with p_1, p_2, e_1 fixed) is $p_1 - e_1$. Which gives the utilization

$$U_{e_1}^{p_1,p_2} = \frac{e_1}{p_1} + \frac{max\_e_2}{p_2} = \frac{e_1}{p_1} + \frac{p_1 - e_1}{p_2} = \frac{e_1}{p_1} + \frac{p_1}{p_2} - \frac{e_1}{p_2} = \frac{p_1}{p_2} + \frac{e_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)$$

As $\frac{p_2}{p_1} - 1 \geq 0$, the utilization $U_{e_1}^{p_1,p_2}$ is minimized by minimizing e_1.

The minimum of $U_{e_1}^{p_1,p_2}$ is attained at $e_1 = p_2 - p_1$.
(Both expressions defining $U_{e_1}^{p_1,p_2}$ give the same value for $e_1 = p_2 - p_1$.)

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U^{p_1,p_2}_{e_1}$:

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U_{e_1}^{p_1,p_2}$:

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2} + \frac{p_2 - p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2} + \left(1 - \frac{p_1}{p_2}\right)\left(\frac{p_2}{p_1} - 1\right)$$

$$= \frac{p_1}{p_2} + \frac{p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2}\left(1 + \left(\frac{p_2}{p_1} - 1\right)^2\right)$$

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U_{e_1}^{p_1,p_2}$:

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2} + \frac{p_2 - p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2} + \left(1 - \frac{p_1}{p_2}\right)\left(\frac{p_2}{p_1} - 1\right)$$

$$= \frac{p_1}{p_2} + \frac{p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2}\left(1 + \left(\frac{p_2}{p_1} - 1\right)^2\right)$$

Denoting $G = \frac{p_2}{p_1} - 1$ we obtain

$$U_{p_2-p_1}^{p_1,p_2}$$

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U_{e_1}^{p_1,p_2}$:

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2} + \frac{p_2 - p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2} + \left(1 - \frac{p_1}{p_2}\right)\left(\frac{p_2}{p_1} - 1\right)$$

$$= \frac{p_1}{p_2} + \frac{p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2}\left(1 + \left(\frac{p_2}{p_1} - 1\right)^2\right)$$

Denoting $G = \frac{p_2}{p_1} - 1$ we obtain

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2}(1 + G^2)$$

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U_{e_1}^{p_1,p_2}$:

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2} + \frac{p_2 - p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2} + \left(1 - \frac{p_1}{p_2}\right)\left(\frac{p_2}{p_1} - 1\right)$$
$$= \frac{p_1}{p_2} + \frac{p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2}\left(1 + \left(\frac{p_2}{p_1} - 1\right)^2\right)$$

Denoting $G = \frac{p_2}{p_1} - 1$ we obtain

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2}(1 + G^2) = \frac{1 + G^2}{p_2/p_1}$$

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U_{e_1}^{p_1,p_2}$:

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2} + \frac{p_2 - p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2} + \left(1 - \frac{p_1}{p_2}\right)\left(\frac{p_2}{p_1} - 1\right)$$

$$= \frac{p_1}{p_2} + \frac{p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2}\left(1 + \left(\frac{p_2}{p_1} - 1\right)^2\right)$$

Denoting $G = \frac{p_2}{p_1} - 1$ we obtain

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2}(1 + G^2) = \frac{1 + G^2}{p_2/p_1} = \frac{1 + G^2}{1 + G}$$

Proof – Special Case (Cont.)

Substitute $e_1 = p_2 - p_1$ into the expression for $U_{e_1}^{p_1,p_2}$:

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2} + \frac{p_2 - p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2} + \left(1 - \frac{p_1}{p_2}\right)\left(\frac{p_2}{p_1} - 1\right)$$

$$= \frac{p_1}{p_2} + \frac{p_1}{p_2}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{p_2}{p_1} - 1\right) = \frac{p_1}{p_2}\left(1 + \left(\frac{p_2}{p_1} - 1\right)^2\right)$$

Denoting $G = \frac{p_2}{p_1} - 1$ we obtain

$$U_{p_2-p_1}^{p_1,p_2} = \frac{p_1}{p_2}(1 + G^2) = \frac{1 + G^2}{p_2/p_1} = \frac{1 + G^2}{1 + G}$$

Differentiating w.r.t. G we get

$$\frac{G^2 + 2G - 1}{(1 + G)^2}$$

which attains minimum at $G = -1 \pm \sqrt{2}$. Here only $G = -1 + \sqrt{2} > 0$ is acceptable since the other root is negative.

Proof – Special Case (Cont.)

Thus the minimum value of $U_{e_1}^{p_1,p_2}$ is

$$\frac{1 + (\sqrt{2} - 1)^2}{1 + (\sqrt{2} - 1)} = \frac{4 - 2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2} - 1)$$

Proof – Special Case (Cont.)

Thus the minimum value of $U_{e_1}^{p_1,p_2}$ is

$$\frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} = \frac{4-2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2}-1)$$

It is attained at periods satisfying

$$G = \frac{p_2}{p_1} - 1 = \sqrt{2} - 1 \qquad \text{i.e. satisfying } p_2 = \sqrt{2}p_1.$$

Proof – Special Case (Cont.)

Thus the minimum value of $U_{e_1}^{p_1,p_2}$ is

$$\frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} = \frac{4-2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2}-1)$$

It is attained at periods satisfying

$$G = \frac{p_2}{p_1} - 1 = \sqrt{2} - 1 \qquad \text{i.e. satisfying } p_2 = \sqrt{2}p_1.$$

The execution time e_1 which at full utilization of the processor (due to max_e_2) gives the minimum utilization is

$$e_1 = p_2 - p_1 = (\sqrt{2}-1)p_1$$

Proof – Special Case (Cont.)

Thus the minimum value of $U_{e_1}^{p_1,p_2}$ is

$$\frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} = \frac{4-2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2}-1)$$

It is attained at periods satisfying

$$G = \frac{p_2}{p_1} - 1 = \sqrt{2} - 1 \qquad \text{i.e. satisfying } p_2 = \sqrt{2}p_1.$$

The execution time e_1 which at full utilization of the processor (due to max_e_2) gives the minimum utilization is

$$e_1 = p_2 - p_1 = (\sqrt{2}-1)p_1$$

and the corresponding $max\_e_2 = p_1 - e_1 = p_1 - (p_2 - p_1) = 2p_1 - p_2$.

Proof – Special Case (Cont.)

Thus the minimum value of $U_{e_1}^{p_1,p_2}$ is

$$\frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} = \frac{4-2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2}-1)$$

It is attained at periods satisfying

$$G = \frac{p_2}{p_1} - 1 = \sqrt{2} - 1 \qquad \text{i.e. satisfying } p_2 = \sqrt{2}p_1.$$

The execution time e_1 which at full utilization of the processor (due to max_e_2) gives the minimum utilization is

$$e_1 = p_2 - p_1 = (\sqrt{2}-1)p_1$$

and the corresponding $max\_e_2 = p_1 - e_1 = p_1 - (p_2 - p_1) = 2p_1 - p_2$.

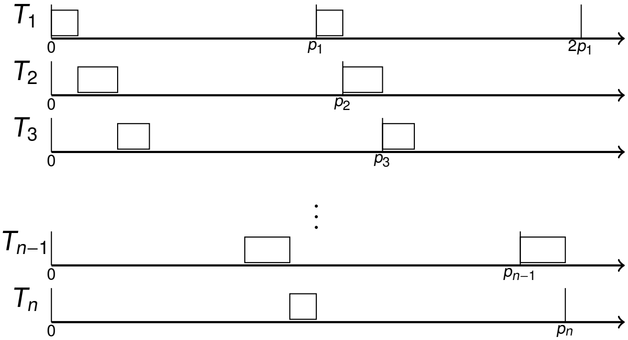
Scaling to $p_1 = 1$, we obtain a completely determined example

$$p_1 = 1 \quad p_2 = \sqrt{2} \approx 1.41 \quad e_1 = \sqrt{2} - 1 \approx 0.41 \quad max\_e_2 = 2 - \sqrt{2} \approx 0.59$$

that fully utilizes the processor (no execution time can be increased) but has the minimum utilization $2(\sqrt{2}-1)$.

Proof Idea of Theorem 8

Fix periods $p_1 < \cdots < p_n$ so that (w.l.o.g.) $p_n \leq 2p_1$. Then the following set of tasks has the smallest utilization among all task sets that fully utilize the processor (i.e., any increase in any execution time makes the set unschedulable).



$$e_k = p_{k+1} - p_k \qquad \text{for } k = 1, \ldots, n-1$$

$$e_n = p_n - 2\sum_{k=1}^{n-1} e_k = 2p_1 - p_n$$

Time-Demand Analysis

Consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$.
Recall that we consider only independent, preemptable, in phase (i.e. $\varphi_i = 0$ for all i) tasks without resource contentions.

Time-Demand Analysis

Consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$.
Recall that we consider only independent, preemptable, in phase (i.e. $\varphi_i = 0$ for all i) tasks without resource contentions.

Assume that $D_i \leq p_i$ for every i, and consider an arbitrary fixed-priority algorithm. W.l.o.g. assume $T_1 \sqsupset \cdots \sqsupset T_n$.

Time-Demand Analysis

Consider a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$.
Recall that we consider only independent, preemptable, in phase (i.e. $\varphi_i = 0$ for all i) tasks without resource contentions.

Assume that $D_i \leq p_i$ for every i, and consider an arbitrary fixed-priority algorithm. W.l.o.g. assume $T_1 \sqsupset \cdots \sqsupset T_n$.

Idea: For every task T_i and every time instant $t \geq 0$, compute the total execution time $w_i(t)$ (the time demand) of the first job $J_{i,1}$ and of all higher-priority jobs released up to time t.

If $w_i(t) \leq t$ for some time $t \leq D_i$, then $J_{i,1}$ is schedulable, and hence all jobs of T_i are schedulable.

Time-Demand Analysis

- Consider one task T_i at a time, starting with highest priority and working to lowest priority.

Time-Demand Analysis

- Consider one task T_i at a time, starting with highest priority and working to lowest priority.
- Focus on the first job $J_{i,1}$ of T_i.

  If $J_{i,1}$ makes it, all jobs of T_i will make it due to $\varphi_i = 0$.

Time-Demand Analysis

- Consider one task T_i at a time, starting with highest priority and working to lowest priority.
- Focus on the first job $J_{i,1}$ of T_i.

  If $J_{i,1}$ makes it, all jobs of T_i will make it due to $\varphi_i = 0$.
- At time t for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[0, t]$ is bounded by

$$w_i(t) = e_i + \sum_{\ell=1}^{i-1} \left\lceil \frac{t}{p_\ell} \right\rceil e_\ell \qquad \text{for } 0 < t \leq p_i$$

(Note that the smallest t for which $w_i(t) \leq t$ is the response time of $J_{i,1}$, and hence the maximum response time of jobs in T_i).

Time-Demand Analysis

- Consider one task T_i at a time, starting with highest priority and working to lowest priority.
- Focus on the first job $J_{i,1}$ of T_i.

  If $J_{i,1}$ makes it, all jobs of T_i will make it due to $\varphi_i = 0$.
- At time t for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[0, t]$ is bounded by

$$w_i(t) = e_i + \sum_{\ell=1}^{i-1} \left\lceil \frac{t}{p_\ell} \right\rceil e_\ell \qquad \text{for } 0 < t \leq p_i$$

  (Note that the smallest t for which $w_i(t) \leq t$ is the response time of $J_{i,1}$, and hence the maximum response time of jobs in T_i).
- If $w_i(t) \leq t$ for some $t \leq D_i$, the job $J_{i,1}$ meets its deadline D_i.

Time-Demand Analysis

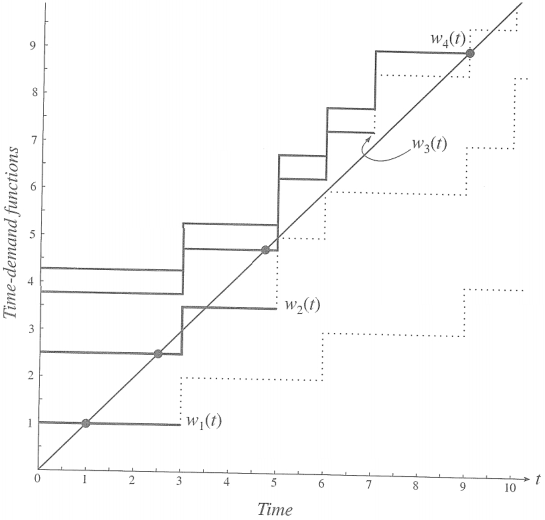
- Consider one task T_i at a time, starting with highest priority and working to lowest priority.
- Focus on the first job $J_{i,1}$ of T_i.

  If $J_{i,1}$ makes it, all jobs of T_i will make it due to $\varphi_i = 0$.
- At time t for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[0, t]$ is bounded by

$$w_i(t) = e_i + \sum_{\ell=1}^{i-1} \left\lceil \frac{t}{p_\ell} \right\rceil e_\ell \qquad \text{for } 0 < t \leq p_i$$

  (Note that the smallest t for which $w_i(t) \leq t$ is the response time of $J_{i,1}$, and hence the maximum response time of jobs in T_i).
- If $w_i(t) \leq t$ for some $t \leq D_i$, the job $J_{i,1}$ meets its deadline D_i.
- If $w_i(t) > t$ for all $0 < t \leq D_i$, then the first job of the task cannot complete by its deadline.

Time-Demand Analysis – Example



Example: $T_1 = (3, 1)$, $T_2 = (5, 1.5)$, $T_3 = (7, 1.25)$, $T_4 = (9, 0.5)$

This set of tasks is schedulable by RM even though $U^{\{T_1,\dots,T_4\}} = 0.85 > 0.757 = U_{RM}(4)$

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function
  - Steps in the time-demand for a task occur at multiples of the period for higher-priority tasks

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function
  - Steps in the time-demand for a task occur at multiples of the period for higher-priority tasks
  - The value of $w_i(t) - t$ linearly decreases from a step until the next step

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function
  - Steps in the time-demand for a task occur at multiples of the period for higher-priority tasks
  - The value of $w_i(t) - t$ linearly decreases from a step until the next step

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function
  - Steps in the time-demand for a task occur at multiples of the period for higher-priority tasks
  - The value of $w_i(t) - t$ linearly decreases from a step until the next step
- If our interest is the schedulability of a task, it suffices to check if $w_i(t) \leq t$ at the time instants when a higher-priority job is released and at D_i

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function
  - Steps in the time-demand for a task occur at multiples of the period for higher-priority tasks
  - The value of $w_i(t) - t$ linearly decreases from a step until the next step
- If our interest is the schedulability of a task, it suffices to check if $w_i(t) \leq t$ at the time instants when a higher-priority job is released and at D_i
- Our schedulability test becomes:
  - Compute $w_i(t)$
  - Check whether $w_i(t) \leq t$ for some t equal either to D_i, or to $j \cdot p_k$ where $k = 1, 2, \ldots, i$ and $j = 1, 2, \ldots, \lfloor D_i / p_k \rfloor$

Time-Demand Analysis – Comments

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:

Time-Demand Analysis – Comments

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:
  - Works for *any* fixed-priority scheduling algorithm, provided the tasks have short response time ($D_i \leq p_i$)

    Can be extended to tasks with arbitrary deadlines

Time-Demand Analysis – Comments

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:
  - Works for *any* fixed-priority scheduling algorithm, provided the tasks have short response time ($D_i \leq p_i$)
    Can be extended to tasks with arbitrary deadlines
- Still more efficient than exhaustive simulation.

Time-Demand Analysis – Comments

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:
  - Works for *any* fixed-priority scheduling algorithm, provided the tasks have short response time ($D_i \leq p_i$)
    Can be extended to tasks with arbitrary deadlines
- Still more efficient than exhaustive simulation.
- Assuming that the tasks are in phase the time demand analysis is complete.

Time-Demand Analysis – Comments

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:
  - Works for *any* fixed-priority scheduling algorithm, provided the tasks have short response time ($D_i \leq p_i$)
    Can be extended to tasks with arbitrary deadlines
- Still more efficient than exhaustive simulation.
- Assuming that the tasks are in phase the time demand analysis is complete.

We have considered the time demand analysis for tasks in phase. In particular, we used the fact that the first job has the maximum response time.

Time-Demand Analysis – Comments

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:
  - Works for *any* fixed-priority scheduling algorithm, provided the tasks have short response time ($D_i \leq p_i$)
    Can be extended to tasks with arbitrary deadlines
- Still more efficient than exhaustive simulation.
- Assuming that the tasks are in phase the time demand analysis is complete.

We have considered the time demand analysis for tasks in phase. In particular, we used the fact that the first job has the maximum response time.

This is not true if the jobs are not in phase, we need to identify the so called *critical instant*, the time instant in which the system is most loaded, and has its worst response time.

Critical Instant – Formally

Definition 9
A **critical instant** t_{crit} of a task T_i is a time instant in which a job $J_{i,k}$ in T_i is released so that $J_{i,k}$ either does not meet its deadline, or has the maximum response time of all jobs in T_i.

Critical Instant – Formally

Definition 9
A **critical instant** t_{crit} of a task T_i is a time instant in which a job $J_{i,k}$ in T_i is released so that $J_{i,k}$ either does not meet its deadline, or has the maximum response time of all jobs in T_i.

Theorem 10
In a fixed-priority system where every job completes before the next job in the same task is released, a critical instant of a task T_i occurs when one of its jobs $J_{i,k}$ is released at the same time with a job from every higher-priority task.

Critical Instant – Formally

Definition 9
A **critical instant** t_{crit} of a task T_i is a time instant in which a job $J_{i,k}$ in T_i is released so that $J_{i,k}$ either does not meet its deadline, or has the maximum response time of all jobs in T_i.

Theorem 10
In a fixed-priority system where every job completes before the next job in the same task is released, a critical instant of a task T_i occurs when one of its jobs $J_{i,k}$ is released at the same time with a job from every higher-priority task.

Note that the situation described in the theorem does not have to occur if tasks are not in phase!

Critical Instant and Schedulability Tests

We use critical instants to get upper bounds on schedulability as follows:

Critical Instant and Schedulability Tests

We use critical instants to get upper bounds on schedulability as follows:

- Set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T'_1, \ldots, T'_n\}$

  By Theorem 10, the response time of the first job $J'_{i,1}$ of T'_1 in $\mathcal{T}'$ is at least as large as the response time of every job of T_i in $\mathcal{T}$.

Critical Instant and Schedulability Tests

We use critical instants to get upper bounds on schedulability as follows:

- Set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T_1', \dots, T_n'\}$

  By Theorem 10, the response time of the first job $J_{i,1}'$ of T_1' in $\mathcal{T}'$ is at least as large as the response time of every job of T_i in $\mathcal{T}$.

- Decide schedulability of $\mathcal{T}'$, e.g. using the timed-demand analysis.

Critical Instant and Schedulability Tests

We use critical instants to get upper bounds on schedulability as follows:

- Set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T'_1, \ldots, T'_n\}$

  By Theorem 10, the response time of the first job $J'_{i,1}$ of T'_1 in $\mathcal{T}'$ is at least as large as the response time of every job of T_i in $\mathcal{T}$.

- Decide schedulability of $\mathcal{T}'$, e.g. using the timed-demand analysis.
  - If $\mathcal{T}'$ if schedulable, then also $\mathcal{T}$ is schedulable.

Critical Instant and Schedulability Tests

We use critical instants to get upper bounds on schedulability as follows:

- Set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T'_1, \ldots, T'_n\}$

  By Theorem 10, the response time of the first job $J'_{i,1}$ of T'_1 in $\mathcal{T}'$ is at least as large as the response time of every job of T_i in $\mathcal{T}$.

- Decide schedulability of $\mathcal{T}'$, e.g. using the timed-demand analysis.
  - If $\mathcal{T}'$ if schedulable, then also $\mathcal{T}$ is schedulable.
  - If $\mathcal{T}'$ is not schedulable, then $\mathcal{T}$ does not have to be schedulable.

    But may be schedulable, which make the time-demand analysis incomplete in general for tasks not in phase.

Dynamic vs Fixed Priority

- EDF
  - pros:
    - optimal
    - very simple and complete test for schedulability

Dynamic vs Fixed Priority

- EDF
  - pros:
    - optimal
    - very simple and complete test for schedulability
  - cons:
    - difficult to predict which job misses its deadline
    - strictly following EDF in case of overloads assigns higher priority to jobs that missed their deadlines
    - larger scheduling overhead

Dynamic vs Fixed Priority

- EDF
  - pros:
    - optimal
    - very simple and complete test for schedulability
  - cons:
    - difficult to predict which job misses its deadline
    - strictly following EDF in case of overloads assigns higher priority to jobs that missed their deadlines
    - larger scheduling overhead
- DM (RM)
  - pros:
    - easier to predict which job misses its deadline (in particular, tasks are not blocked by lower priority tasks)
    - easy implementation with little scheduling overhead
    - (optimal in some cases often occurring in practice)

Dynamic vs Fixed Priority

- EDF
  - pros:
    - optimal
    - very simple and complete test for schedulability
  - cons:
    - difficult to predict which job misses its deadline
    - strictly following EDF in case of overloads assigns higher priority to jobs that missed their deadlines
    - larger scheduling overhead
- DM (RM)
  - pros:
    - easier to predict which job misses its deadline (in particular, tasks are not blocked by lower priority tasks)
    - easy implementation with little scheduling overhead
    - (optimal in some cases often occurring in practice)
  - cons:
    - not optimal
    - incomplete and more involved tests for schedulability