

IA158 Real Time Systems

Tomáš Brázdil

Organization of This Course

Sources:

- ▶ Lectures (slides, notes)
 - ▶ based on several sources (hard to obtain)
 - ▶ slides are prepared for lectures, lots of stuff on the greenboard
(\Rightarrow attend the lectures)

Organization of This Course

Sources:

- ▶ Lectures (slides, notes)
 - ▶ based on several sources (hard to obtain)
 - ▶ slides are prepared for lectures, lots of stuff on the greenboard
(\Rightarrow attend the lectures)

Homework:

- ▶ a larger homework project

Organization of This Course

Sources:

- ▶ Lectures (slides, notes)
 - ▶ based on several sources (hard to obtain)
 - ▶ slides are prepared for lectures, lots of stuff on the greenboard
(\Rightarrow attend the lectures)

Homework:

- ▶ a larger homework project

Evaluation:

- ▶ Homework project
(have to do to be allowed to the exam)
- ▶ Oral exam

Definition 1 (Time)

Miriam-Webster: Time is the measured or measurable period during which an action, process, or condition exists or continues.

Real-Time Systems

Definition 1 (Time)

Miriam-Webster: Time is the measured or measurable period during which an action, process, or condition exists or continues.

Definition 2 (Real-time)

Real-time is a quantitative notion of time measured using a physical clock.

Example: After an event occurs (eg. temperature exceeds 500 degrees) the corresponding action (cooling) must take place within 100ms.

Compare with qualitative notion of time (before, after, eventually, etc.)

Real-Time Systems

Definition 1 (Time)

Miriam-Webster: Time is the measured or measurable period during which an action, process, or condition exists or continues.

Definition 2 (Real-time)

Real-time is a quantitative notion of time measured using a physical clock.

Example: After an event occurs (eg. temperature exceeds 500 degrees) the corresponding action (cooling) must take place within 100ms.

Compare with qualitative notion of time (before, after, eventually, etc.)

Definition 3 (Real-time system)

A *real-time system* must deliver services in a timely manner.

Not necessarily fast, must satisfy some *quantitative* timing constraints

Definition 4 (Embedded system)

An *embedded system* is a computer system designed for specific control functions within a larger system, usually consisting of electronic as well as mechanical parts.

Real-time Embedded Systems

Definition 4 (Embedded system)

An *embedded system* is a computer system designed for specific control functions within a larger system, usually consisting of electronic as well as mechanical parts.

Most (not all) real-time systems are embedded

Most (not all) embedded systems are real-time



(Few) Examples of Real-time Embedded Systems

- ▶ Industrial
 - ▶ chemical plant control
 - ▶ automated assembly line (e.g. robotic assembly, inspection)

(Few) Examples of Real-time Embedded Systems

- ▶ Industrial
 - ▶ chemical plant control
 - ▶ automated assembly line (e.g. robotic assembly, inspection)
- ▶ Medical
 - ▶ pacemaker,
 - ▶ medical monitoring devices

(Few) Examples of Real-time Embedded Systems

- ▶ Industrial
 - ▶ chemical plant control
 - ▶ automated assembly line (e.g. robotic assembly, inspection)
- ▶ Medical
 - ▶ pacemaker,
 - ▶ medical monitoring devices
- ▶ Transportation systems
 - ▶ computers in cars (ABS, MPFI, cruise control, airbag ...)
 - ▶ aircraft (FMS, fly-by-wire ...)

(Few) Examples of Real-time Embedded Systems

- ▶ Industrial
 - ▶ chemical plant control
 - ▶ automated assembly line (e.g. robotic assembly, inspection)
- ▶ Medical
 - ▶ pacemaker,
 - ▶ medical monitoring devices
- ▶ Transportation systems
 - ▶ computers in cars (ABS, MPFI, cruise control, airbag ...)
 - ▶ aircraft (FMS, fly-by-wire ...)
- ▶ Military applications
 - ▶ controllers in weapons, missiles, ...
 - ▶ radar and sonar tracking

(Few) Examples of Real-time Embedded Systems

- ▶ Industrial
 - ▶ chemical plant control
 - ▶ automated assembly line (e.g. robotic assembly, inspection)
- ▶ Medical
 - ▶ pacemaker,
 - ▶ medical monitoring devices
- ▶ Transportation systems
 - ▶ computers in cars (ABS, MPFI, cruise control, airbag ...)
 - ▶ aircraft (FMS, fly-by-wire ...)
- ▶ Military applications
 - ▶ controllers in weapons, missiles, ...
 - ▶ radar and sonar tracking
- ▶ Multimedia – multimedia center, videoconferencing
- ▶ ...

(Non-)Real-time (non-)embedded systems

There are real time systems that are not embedded:

- ▶ trading systems
- ▶ ticket reservation
- ▶ multimedia (on PC)
- ▶ ...

(Non-)Real-time (non-)embedded systems

There are real time systems that are not embedded:

- ▶ trading systems
- ▶ ticket reservation
- ▶ multimedia (on PC)
- ▶ ...

There are embedded systems that are (possibly) not real-time

e.g. a weather station sends data once a day without any deadline – not really real-time system

Caveat: Aren't all systems real-time in a sense?

Characteristics of Real-Time Embedded Systems

Real-time systems often are

- ▶ **safety critical**

- ▶ Serious consequences may result if services are not delivered on timely basis
- ▶ Bugs in embedded real-time systems are often difficult to fix

... need to validate their correctness

Characteristics of Real-Time Embedded Systems

Real-time systems often are

- ▶ **safety critical**

- ▶ Serious consequences may result if services are not delivered on timely basis
- ▶ Bugs in embedded real-time systems are often difficult to fix

... need to validate their correctness

- ▶ **concurrent**

- ▶ Real-world devices operate in parallel – better to model this parallelism by concurrent tasks in the program

... validation may be difficult, formal methods often needed

Characteristics of Real-Time Embedded Systems

Real-time systems often are

- ▶ **safety critical**

- ▶ Serious consequences may result if services are not delivered on timely basis
- ▶ Bugs in embedded real-time systems are often difficult to fix

... need to validate their correctness

- ▶ **concurrent**

- ▶ Real-world devices operate in parallel – better to model this parallelism by concurrent tasks in the program

... validation may be difficult, formal methods often needed

- ▶ **reactive**

- ▶ Interact continuously with their environment (as opposed to information processing systems)

... “traditional” validation methods do not apply

Validating Time Requirements and Predictability

- ▶ Given real-time requirements and an implementation on HW and SW, how to show that the requirements are met?

Validating Time Requirements and Predictability

- ▶ Given real-time requirements and an implementation on HW and SW, how to show that the requirements are met?

... testing might not suffice:

Maiden flight of space shuttle, 12 April 1981: 1/67 probability that a transient overload occurs during initialization; and it actually did!

Validating Time Requirements and Predictability

- ▶ Given real-time requirements and an implementation on HW and SW, how to show that the requirements are met?

... testing might not suffice:

Maiden flight of space shuttle, 12 April 1981: 1/67 probability that a transient overload occurs during initialization; and it actually did!

- ▶ We need a formal model and validation ...

Validating Time Requirements and Predictability

- ▶ Given real-time requirements and an implementation on HW and SW, how to show that the requirements are met?

... testing might not suffice:

Maiden flight of space shuttle, 12 April 1981: 1/67 probability that a transient overload occurs during initialization; and it actually did!

- ▶ We need a formal model and validation ...
- ▶ ... we need **predictable** behavior!
It is difficult to obtain
 - ▶ caches, DMA, unmaskable interrupts
 - ▶ memory management
 - ▶ scheduling anomalies
 - ▶ difficult to compute worst-case execution time
 - ▶ ...

Types of Timing Requirements

Time sharing systems: minimize average response time

The goal of scheduling in standard op. systems such as Linux and Windows

Types of Timing Requirements

Time sharing systems: minimize average response time

The goal of scheduling in standard op. systems such as Linux and Windows

Often it is **not** enough to minimize average response time!

(A man drowned crossing a stream with an average depth of 15cm.)

Types of Timing Requirements

Time sharing systems: minimize average response time

The goal of scheduling in standard op. systems such as Linux and Windows

Often it is **not** enough to minimize average response time!

(A man drowned crossing a stream with an average depth of 15cm.)

“hard” real-time tasks must be **always** finished before their deadline!

e.g. airbag in a car: whenever a collision is detected, the airbag must be deployed within 10ms

Types of Timing Requirements

Time sharing systems: minimize average response time

The goal of scheduling in standard op. systems such as Linux and Windows

Often it is **not** enough to minimize average response time!

(A man drowned crossing a stream with an average depth of 15cm.)

“hard” real-time tasks must be **always** finished before their deadline!

e.g. airbag in a car: whenever a collision is detected, the airbag must be deployed within 10ms

Not all tasks in a real-time system are critical, only the quality of service is affected by missing a deadline

Types of Timing Requirements

Time sharing systems: minimize average response time

The goal of scheduling in standard op. systems such as Linux and Windows

Often it is **not** enough to minimize average response time!

(A man drowned crossing a stream with an average depth of 15cm.)

“hard” real-time tasks must be **always** finished before their deadline!

e.g. airbag in a car: whenever a collision is detected, the airbag must be deployed within 10ms

Not all tasks in a real-time system are critical, only the quality of service is affected by missing a deadline

Most **“soft” real-time tasks** should finish before their deadlines.

e.g. frame rate in a videoconf. should be kept above 15fps most of the time

Types of Timing Requirements

Time sharing systems: minimize average response time

The goal of scheduling in standard op. systems such as Linux and Windows

Often it is **not** enough to minimize average response time!

(A man drowned crossing a stream with an average depth of 15cm.)

“**hard**” **real-time tasks** must be **always** finished before their deadline!

e.g. airbag in a car: whenever a collision is detected, the airbag must be deployed within 10ms

Not all tasks in a real-time system are critical, only the quality of service is affected by missing a deadline

Most “**soft**” **real-time tasks** should finish before their deadlines.

e.g. frame rate in a videoconf. should be kept above 15fps most of the time

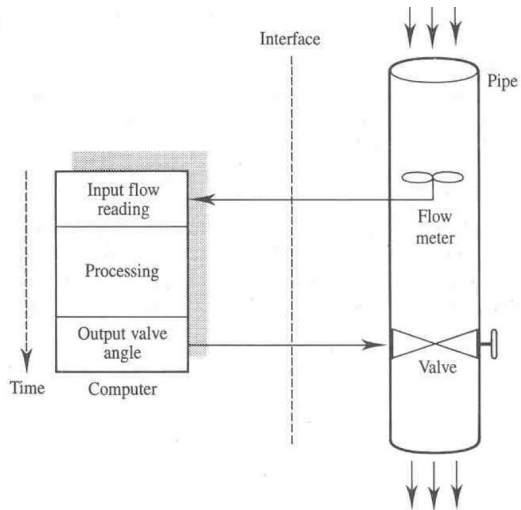
Many real-time systems combine “hard” and “soft” real-time tasks.

i.e. we optimize performance w.r.t. “soft” real-time tasks under the constraint that “hard” real-time tasks are finished before their deadlines

Examples of Real-Time Systems

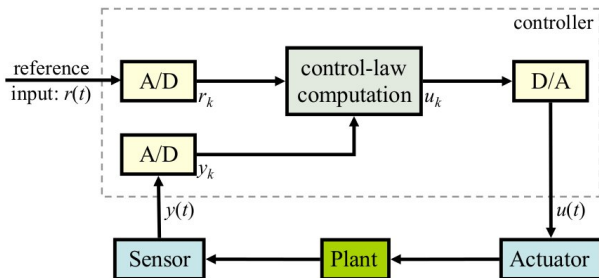
- ▶ Digital process control
 - ▶ anti-lock braking system
- ▶ Higher-level command and control
 - ▶ helicopter flight control
- ▶ Real-time databases
 - ▶ Stock trading systems

Digital Process Control



Computer controls the flow in the pipe in real-time

Digital Process Control



The controller (computer) controls the plant using the actuator (valve) based on sampled data from the sensor (flow meter)

- ▶ $y(t)$ – the measured state of the plant
- ▶ $r(t)$ – the desired state of the plant
- ▶ Calculate control output $u(t)$ as a function of $y(t), r(t)$
e.g. $u_k = u_{k-2} + \alpha(r_k - y_k) + \beta(r_{k-1} - y_{k-1}) + \gamma(r_{k-2} - y_{k-2})$
where α, β, γ are suitable constants

Digital Process Control

- ▶ Pseudo-code for the controller:

set timer to interrupt periodically with period T

foreach timer interrupt **do**

analogue-to-digital conversion of $y(t)$ to get y_k

compute control output u_k based on r_k and y_k

digital-to-analogue conversion of u_k to get $u(t)$

end

Digital Process Control

- ▶ Pseudo-code for the controller:

set timer to interrupt periodically with period T

foreach timer interrupt **do**

analogue-to-digital conversion of $y(t)$ to get y_k

compute control output u_k based on r_k and y_k

digital-to-analogue conversion of u_k to get $u(t)$

end

- ▶ Effective control of the plant depends on:
 - ▶ The correct reference input and control law computation
 - ▶ The accuracy of the sensor measurements
 - ▶ Resolution of the sampled data (i.e. bits per sample)
 - ▶ Frequency of interrupts (i.e. $1/T$)

Digital Process Control

- ▶ Pseudo-code for the controller:

set timer to interrupt periodically with period T

foreach timer interrupt **do**

analogue-to-digital conversion of $y(t)$ to get y_k

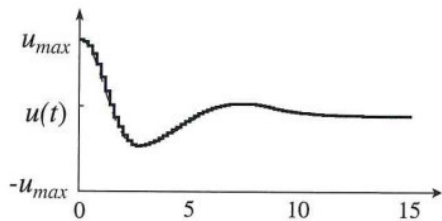
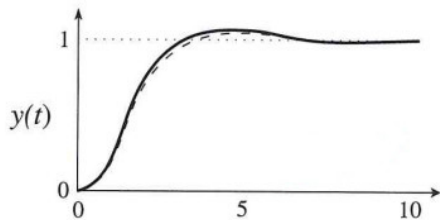
compute control output u_k based on r_k and y_k

digital-to-analogue conversion of u_k to get $u(t)$

end

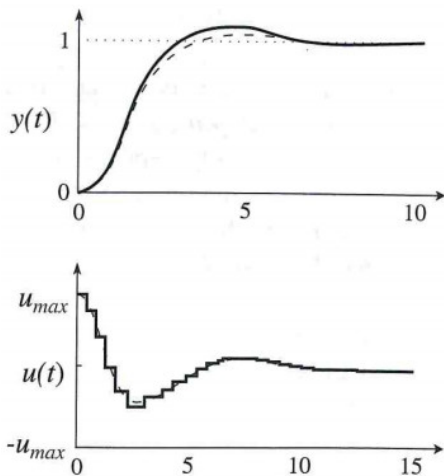
- ▶ Effective control of the plant depends on:
 - ▶ The correct reference input and control law computation
 - ▶ The accuracy of the sensor measurements
 - ▶ Resolution of the sampled data (i.e. bits per sample)
 - ▶ Frequency of interrupts (i.e. $1/T$)
- ▶ T is the *sampling period*
 - ▶ Small T better approximates the analogue behavior
 - ▶ Large T means less processor-time demand
 - ... but may result in unstable control

Example



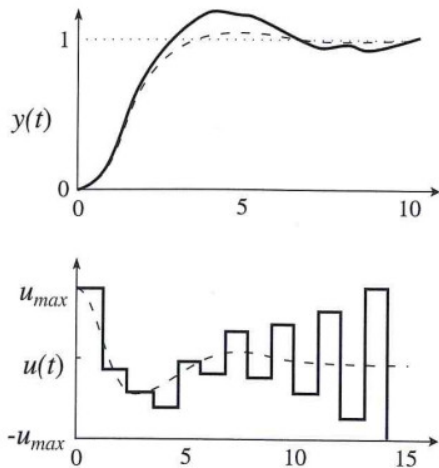
$$r(t) = 1 \text{ for } t \geq 0$$

Example



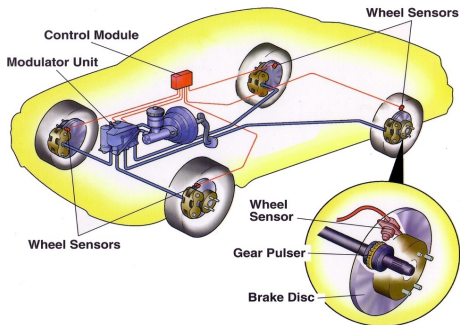
$$r(t) = 1 \text{ for } t \geq 0$$

Example



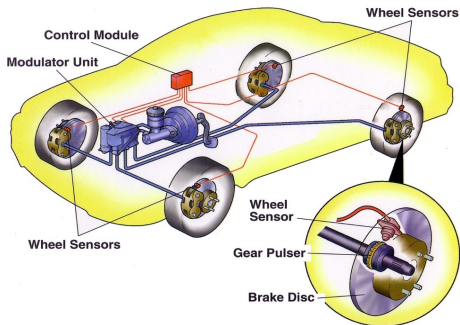
$$r(t) = 1 \text{ for } t \geq 0$$

Anti-Lock Braking System



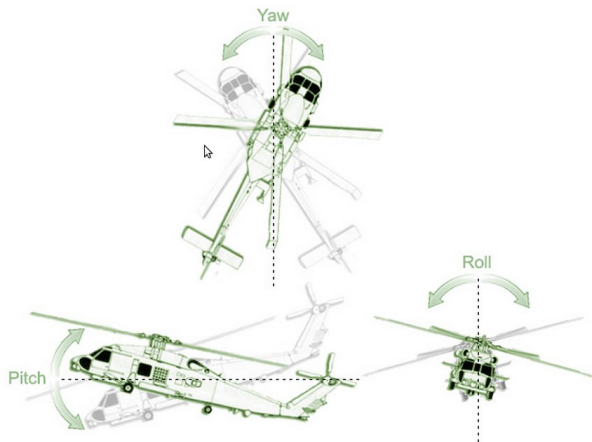
- ▶ The controller monitors the speed sensors in wheels
Right before a wheel locks up, it experiences a rapid deceleration

Anti-Lock Braking System



- ▶ The controller monitors the speed sensors in wheels
Right before a wheel locks up, it experiences a rapid deceleration
- ▶ If a rapid deceleration of a wheel is observed, the controller alternately
 - ▶ reduces pressure on the corresponding brake until acceleration is observed
 - ▶ then applies brake until deceleration is observed

Multi-Rate DPC – Helicopter Flight Control



There are also three velocity components

Two control loops: pilot's control (30Hz) and stabilization (90Hz)

Multi-Rate DPC – Helicopter Flight Control

Do the following in each 1/180-second cycle:

- ▶ Validate sensor data; in the presence of failures, reconfigure the system

Multi-Rate DPC – Helicopter Flight Control

Do the following in each 1/180-second cycle:

- ▶ Validate sensor data; in the presence of failures, reconfigure the system
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ keyboard input and mode selection
 - ▶ data normalization and coordinate transformation
 - ▶ tracking reference update

Multi-Rate DPC – Helicopter Flight Control

Do the following in each 1/180-second cycle:

- ▶ Validate sensor data; in the presence of failures, reconfigure the system
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ keyboard input and mode selection
 - ▶ data normalization and coordinate transformation
 - ▶ tracking reference update
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ control laws of the outer pitch-control loop
 - ▶ control laws of the outer roll-control loop
 - ▶ control laws of the outer yaw- and collective-control loop

Multi-Rate DPC – Helicopter Flight Control

Do the following in each 1/180-second cycle:

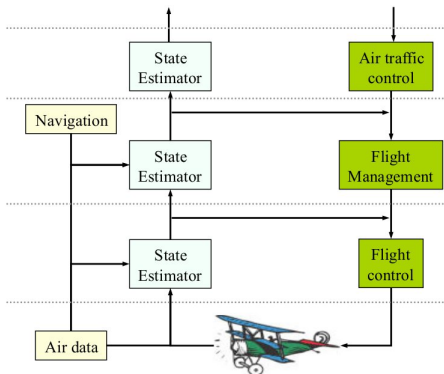
- ▶ Validate sensor data; in the presence of failures, reconfigure the system
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ keyboard input and mode selection
 - ▶ data normalization and coordinate transformation
 - ▶ tracking reference update
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ control laws of the outer pitch-control loop
 - ▶ control laws of the outer roll-control loop
 - ▶ control laws of the outer yaw- and collective-control loop
- ▶ Do each of the following 90-Hz computations once every two cycles, using outputs produced by 30-Hz computations and avionics tasks:
 - ▶ control laws of the inner pitch-control loop
 - ▶ control laws of the inner roll- and collective-control loop
- ▶ Compute the control laws of the inner yaw-control loop, using outputs produced by 90-Hz control-law computations as inputs

Multi-Rate DPC – Helicopter Flight Control

Do the following in each 1/180-second cycle:

- ▶ Validate sensor data; in the presence of failures, reconfigure the system
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ keyboard input and mode selection
 - ▶ data normalization and coordinate transformation
 - ▶ tracking reference update
- ▶ Do the following 30-Hz avionics tasks, each one every six cycles:
 - ▶ control laws of the outer pitch-control loop
 - ▶ control laws of the outer roll-control loop
 - ▶ control laws of the outer yaw- and collective-control loop
- ▶ Do each of the following 90-Hz computations once every two cycles, using outputs produced by 30-Hz computations and avionics tasks:
 - ▶ control laws of the inner pitch-control loop
 - ▶ control laws of the inner roll- and collective-control loop
- ▶ Compute the control laws of the inner yaw-control loop, using outputs produced by 90-Hz control-law computations as inputs
- ▶ Output commands
- ▶ Carry out built-in-test
- ▶ Wait until the beginning of the next cycle

Higher-Level Command and Control



Controllers organized into a hierarchy

- ▶ At the lowest level we place the digital control systems that operate on the physical environment
- ▶ Higher level controllers monitor the behavior of lower levels
- ▶ Time-scale and complexity of decision making increases as one goes up the hierarchy (from control to planning)

Real-Time Database System

- ▶ Databases that contain perishable data, i.e. relevance of data deteriorates with time
Air traffic control, stock price quotation systems, tracking systems, etc.

Real-Time Database System

- ▶ Databases that contain perishable data, i.e. relevance of data deteriorates with time
Air traffic control, stock price quotation systems, tracking systems, etc.
- ▶ The temporal quality of data is quantified by *age of an image object*, i.e. the length of time since last update

Real-Time Database System

- ▶ Databases that contain perishable data, i.e. relevance of data deteriorates with time
Air traffic control, stock price quotation systems, tracking systems, etc.
- ▶ The temporal quality of data is quantified by *age of an image object*, i.e. the length of time since last update
- ▶ temporal consistency
 - ▶ **absolute** = max. age is bounded by a fixed threshold
 - ▶ **relative** = max. difference in ages is bounded by a threshold
e.g. planning system correlating traffic density and flow of vehicles

Applications	Size	Ave. Resp. Time	Max Resp. Time	Abs. Cons.	Rel. Cons.
Air traffic control	20,000	0.50 ms	5.00 ms	3.00 sec.	6.00 sec.
Aircraft mission	3,000	0.05 ms	1.00 ms	0.05 sec.	0.20 sec.
Spacecraft control	5,000	0.05 ms	1.00 ms	0.20 sec.	1.00 sec.
Process control		0.80 ms	5.00 sec	1.00 sec.	2.00 sec

- ▶ Users of database compete for access – various models for trading consistency with time demands exist.

Stock-Trading System

- ▶ A system for selling/buying stock at public prices

Stock-Trading System

- ▶ A system for selling/buying stock at public prices
- ▶ Prices are volatile in their movement

Stock-Trading System

- ▶ A system for selling/buying stock at public prices
- ▶ Prices are volatile in their movement
- ▶ Stop orders:
 - ▶ set upper limit on prices for buying – buy for the best available price once the limit is reached
e.g. stock currently trading at \$30 should be bought when the price rises above \$35

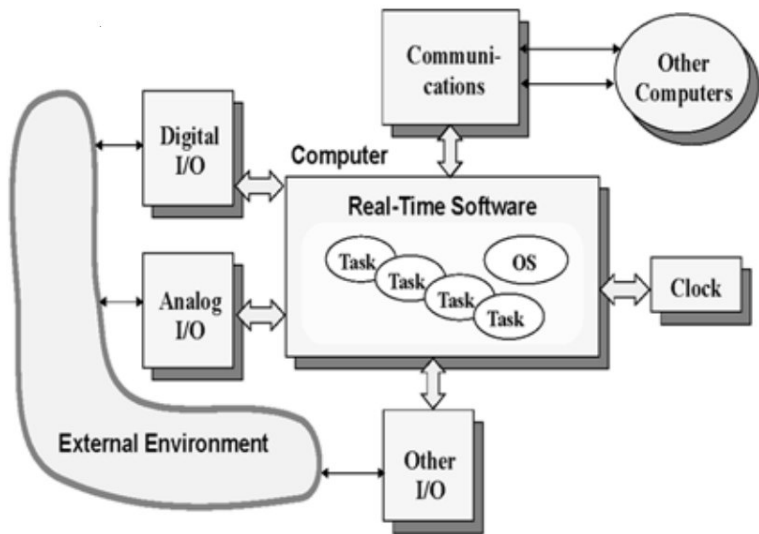
Stock-Trading System

- ▶ A system for selling/buying stock at public prices
- ▶ Prices are volatile in their movement
- ▶ Stop orders:
 - ▶ set upper limit on prices for buying – buy for the best available price once the limit is reached
e.g. stock currently trading at \$30 should be bought when the price rises above \$35
 - ▶ set lower limit on prices for selling – sell for the best available price once the limit is reached
e.g. stock currently trading at \$30 should be sold when the price sinks below \$25

Stock-Trading System

- ▶ A system for selling/buying stock at public prices
- ▶ Prices are volatile in their movement
- ▶ Stop orders:
 - ▶ set upper limit on prices for buying – buy for the best available price once the limit is reached
e.g. stock currently trading at \$30 should be bought when the price rises above \$35
 - ▶ set lower limit on prices for selling – sell for the best available price once the limit is reached
e.g. stock currently trading at \$30 should be sold when the price sinks below \$25
- ▶ Depending on the delay, the available price may be different from the limit
successful stop orders depend on the timely delivery of stock trade data and the ability to trade on the changing prices in a timely manner

Structure of Real-Time (Embedded) Applications



Types of Real-Time Systems

- ▶ Purely cyclic
 - ▶ every task executes periodically; I/O operations are polled; demands in resources do not vary

e.g. digital controllers

Types of Real-Time Systems

- ▶ Purely cyclic
 - ▶ every task executes periodically; I/O operations are polled; demands in resources do not vary

e.g. digital controllers

- ▶ Mostly cyclic
 - ▶ most tasks execute periodically; system also responds to external events (fault recovery and external commands) asynchronously

e.g. avionics

Types of Real-Time Systems

- ▶ Purely cyclic
 - ▶ every task executes periodically; I/O operations are polled; demands in resources do not vary

e.g. digital controllers
- ▶ Mostly cyclic
 - ▶ most tasks execute periodically; system also responds to external events (fault recovery and external commands) asynchronously

e.g. avionics
- ▶ Asynchronous and somewhat predictable
 - ▶ durations between consecutive executions of a task as well as demands in resources may vary considerably. These variations have either bounded range, or known statistics.

e.g. radar signal processing, tracking

Types of Real-Time Systems

- ▶ The type of application affects how we schedule tasks and prove correctness
- ▶ It is easier to reason about applications that are more cyclic, synchronous and predictable
 - ▶ Many real-time systems are designed in this manner
 - ▶ Safe, conservative, design approach, if it works

Real-Time Systems Failures

- ▶ AT&T *long* distance calls
- ▶ Therac-25 medical accelerator disaster
- ▶ Patriot missile mistiming

AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:



AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:

- ▶ the switch in New York City neared its load limit



AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:

- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset



AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:

- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors



AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:

- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors
- ▶ after the reset, the switch began to distribute calls (quickly)



AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:

- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors
- ▶ after the reset, the switch began to distribute calls (quickly)
- ▶ then another switch received one of these calls from New York



AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:



- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors
- ▶ after the reset, the switch began to distribute calls (quickly)
- ▶ then another switch received one of these calls from New York
- ▶ began to update its records that New York was back on line

AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:



- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors
- ▶ after the reset, the switch began to distribute calls (quickly)
- ▶ then another switch received one of these calls from New York
- ▶ began to update its records that New York was back on line
- ▶ a second call from New York arrived less than 10 milliseconds after the first, i.e. while the first hadn't yet been handled;
this together with a SW bug caused maintenance reset

AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:



- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors
- ▶ after the reset, the switch began to distribute calls (quickly)
- ▶ then another switch received one of these calls from New York
- ▶ began to update its records that New York was back on line
- ▶ a second call from New York arrived less than 10 milliseconds after the first, i.e. while the first hadn't yet been handled;
this together with a SW bug caused maintenance reset
- ▶ the error was propagated further

AT&T Long Distance Calls

114 computer-operated electronic switches scattered across USA
Handling up to 700,000 calls an hour

The problem:



- ▶ the switch in New York City neared its load limit
- ▶ entered a four-second maintenance reset
- ▶ sent “do not disturb” to neighbors
- ▶ after the reset, the switch began to distribute calls (quickly)
- ▶ then another switch received one of these calls from New York
- ▶ began to update its records that New York was back on line
- ▶ a second call from New York arrived less than 10 milliseconds after the first, i.e. while the first hadn't yet been handled;
this together with a SW bug caused maintenance reset
- ▶ the error was propagated further

The reason for failure: The system was unable to react to closely timed messages

Therac-25 medical accelerator disaster

Therac-25 = a machine for radiotherapy

- ▶ between 1985 and 1987 (at least) six accidents involving enormous radiation overdoses to patients
- ▶ Half of these patients died due to the overdoses



Therac-25 – the modes

1. electron mode

- ▶ electron beam (low current)
- ▶ various levels of energy (5 to 25-MeV)
- ▶ scanning magnets used to spread the beam to a safe concentration

Therac-25 – the modes

1. electron mode

- ▶ electron beam (low current)
- ▶ various levels of energy (5 to 25-MeV)
- ▶ scanning magnets used to spread the beam to a safe concentration

2. photon mode

- ▶ only one level of energy (25-MeV), much larger electron-beam current
- ▶ electron beam strikes a metal foil to produce X-rays (photons)
- ▶ the X-ray beam is "flattened" by a device below the foil

Therac-25 – the modes

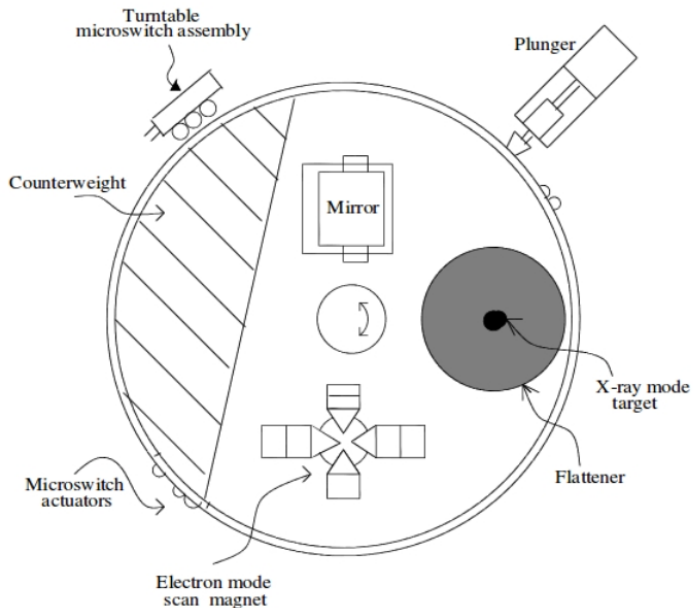
1. electron mode
 - ▶ electron beam (low current)
 - ▶ various levels of energy (5 to 25-MeV)
 - ▶ scanning magnets used to spread the beam to a safe concentration
2. photon mode
 - ▶ only one level of energy (25-MeV), much larger electron-beam current
 - ▶ electron beam strikes a metal foil to produce X-rays (photons)
 - ▶ the X-ray beam is "flattened" by a device below the foil
3. light mode – just light beam used to illuminate the field on the surface of the patient's body that will be treated

Therac-25 – the modes

1. electron mode
 - ▶ electron beam (low current)
 - ▶ various levels of energy (5 to 25-MeV)
 - ▶ scanning magnets used to spread the beam to a safe concentration
2. photon mode
 - ▶ only one level of energy (25-MeV), much larger electron-beam current
 - ▶ electron beam strikes a metal foil to produce X-rays (photons)
 - ▶ the X-ray beam is "flattened" by a device below the foil
3. light mode – just light beam used to illuminate the field on the surface of the patient's body that will be treated

All devices placed on a turntable, supposed to be rotated to the correct position before the beam is started up

Therac-25 – turntable



The Software

The software responsible for

- ▶ Operator
 - ▶ Monitoring input and editing changes from an operator
 - ▶ Updating the screen to show current status of machine
 - ▶ Printing in response to an operator commands

The Software

The software responsible for

- ▶ Operator
 - ▶ Monitoring input and editing changes from an operator
 - ▶ Updating the screen to show current status of machine
 - ▶ Printing in response to an operator commands
- ▶ Machine
 - ▶ monitoring the machine status
 - ▶ placement of turntable
 - ▶ strength and shape of beam
 - ▶ operation of bending and scanning magnets
 - ▶ setting the machine up for the specified treatment
 - ▶ turning the beam on
 - ▶ turning the beam off (after treatment, on operator command, or if a malfunction is detected)

The Software

The software responsible for

- ▶ Operator
 - ▶ Monitoring input and editing changes from an operator
 - ▶ Updating the screen to show current status of machine
 - ▶ Printing in response to an operator commands
- ▶ Machine
 - ▶ monitoring the machine status
 - ▶ placement of turntable
 - ▶ strength and shape of beam
 - ▶ operation of bending and scanning magnets
 - ▶ setting the machine up for the specified treatment
 - ▶ turning the beam on
 - ▶ turning the beam off (after treatment, on operator command, or if a malfunction is detected)

Software running several safety critical tasks in parallel!

Insufficient hardware protection (as opposed to previous models)!!

Therac-25 – software

- ▶ The Therac-25 runs on a real-time operating system

Therac-25 – software

- ▶ The Therac-25 runs on a real-time operating system
- ▶ Four major components of software: stored data, a scheduler, a set of tasks, and interrupt services (e.g. the computer clock and handling of computer-hardware-generated errors)

Therac-25 – software

- ▶ The Therac-25 runs on a real-time operating system
- ▶ Four major components of software: stored data, a scheduler, a set of tasks, and interrupt services (e.g. the computer clock and handling of computer-hardware-generated errors)
- ▶ The software segregated the tasks above into
 - ▶ critical tasks: e.g. setup and operation of the beam
 - ▶ non-critical tasks: e.g. monitoring the keyboard

Therac-25 – software

- ▶ The Therac-25 runs on a real-time operating system
- ▶ Four major components of software: stored data, a scheduler, a set of tasks, and interrupt services (e.g. the computer clock and handling of computer-hardware-generated errors)
- ▶ The software segregated the tasks above into
 - ▶ critical tasks: e.g. setup and operation of the beam
 - ▶ non-critical tasks: e.g. monitoring the keyboard
- ▶ The scheduler directs all non-interrupt events and orders simultaneous events

Therac-25 – software

- ▶ The Therac-25 runs on a real-time operating system
- ▶ Four major components of software: stored data, a scheduler, a set of tasks, and interrupt services (e.g. the computer clock and handling of computer-hardware-generated errors)
- ▶ The software segregated the tasks above into
 - ▶ critical tasks: e.g. setup and operation of the beam
 - ▶ non-critical tasks: e.g. monitoring the keyboard
- ▶ The scheduler directs all non-interrupt events and orders simultaneous events
- ▶ Every 0.1 seconds tasks are initiated and critical tasks are executed first, with non-critical tasks taking up any remaining time

Therac-25 – software

- ▶ The Therac-25 runs on a real-time operating system
- ▶ Four major components of software: stored data, a scheduler, a set of tasks, and interrupt services (e.g. the computer clock and handling of computer-hardware-generated errors)
- ▶ The software segregated the tasks above into
 - ▶ critical tasks: e.g. setup and operation of the beam
 - ▶ non-critical tasks: e.g. monitoring the keyboard
- ▶ The scheduler directs all non-interrupt events and orders simultaneous events
- ▶ Every 0.1 seconds tasks are initiated and critical tasks are executed first, with non-critical tasks taking up any remaining time

Communication between tasks based on shared variables
(without proper atomic test-and-set instructions)

What happened?

There were several accidents due to various bugs in software

What happened?

There were several accidents due to various bugs in software

One of them proceeded as follows (much simplified):

- ▶ the operator entered parameters for X-rays treatment

What happened?

There were several accidents due to various bugs in software

One of them proceeded as follows (much simplified):

- ▶ the operator entered parameters for X-rays treatment
- ▶ the machine started to set up for the treatment

What happened?

There were several accidents due to various bugs in software

One of them proceeded as follows (much simplified):

- ▶ the operator entered parameters for X-rays treatment
- ▶ the machine started to set up for the treatment
- ▶ the operator changed the mode from X-rays to electron (within the interval from 1s to 8s from the end of the original editing)

What happened?

There were several accidents due to various bugs in software

One of them proceeded as follows (much simplified):

- ▶ the operator entered parameters for X-rays treatment
- ▶ the machine started to set up for the treatment
- ▶ the operator changed the mode from X-rays to electron (within the interval from 1s to 8s from the end of the original editing)
- ▶ the patient received X-ray “treatment” with turntable in the electron position (i.e. unshielded)

What happened?

There were several accidents due to various bugs in software

One of them proceeded as follows (much simplified):

- ▶ the operator entered parameters for X-rays treatment
- ▶ the machine started to set up for the treatment
- ▶ the operator changed the mode from X-rays to electron (within the interval from 1s to 8s from the end of the original editing)
- ▶ the patient received X-ray “treatment” with turntable in the electron position (i.e. unshielded)

The cause:

- ▶ The turntable and treatment parameters were set by *different* concurrent procedures HAND and DATENT, respectively.

What happened?

There were several accidents due to various bugs in software

One of them proceeded as follows (much simplified):

- ▶ the operator entered parameters for X-rays treatment
- ▶ the machine started to set up for the treatment
- ▶ the operator changed the mode from X-rays to electron (within the interval from 1s to 8s from the end of the original editing)
- ▶ the patient received X-ray “treatment” with turntable in the electron position (i.e. unshielded)

The cause:

- ▶ The turntable and treatment parameters were set by *different* concurrent procedures `HAND` and `DATENT`, respectively.
- ▶ If the change in parameters came in the “right” time, only `HAND` reacted to the change.

Patriot missile mistiming



VS



Patriot missile mistiming

- ▶ Patriot – Air defense missile system

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Simplified principle of function:

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Simplified principle of function:

- ▶ Patriot's radar detects an airborne object

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Simplified principle of function:

- ▶ Patriot's radar detects an airborne object
- ▶ the object is identified as a scud missile (according to speed, size, etc.)

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Simplified principle of function:

- ▶ Patriot's radar detects an airborne object
- ▶ the object is identified as a scud missile (according to speed, size, etc.)
- ▶ the range gate computes an area in the air space where the system should next look for it

Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Simplified principle of function:

- ▶ Patriot's radar detects an airborne object
- ▶ the object is identified as a scud missile (according to speed, size, etc.)
- ▶ the range gate computes an area in the air space where the system should next look for it
- ▶ finding the object in the calculated area confirms that it is a scud

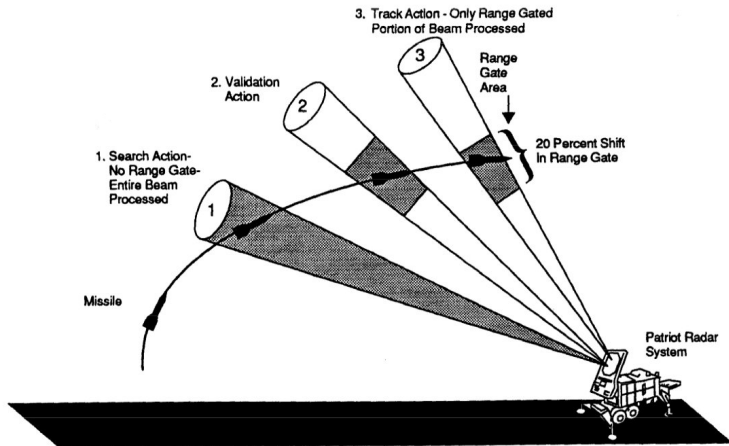
Patriot missile mistiming

- ▶ Patriot – Air defense missile system
- ▶ Failed to intercept a scud missile on February 25, 1991 at Dhahran, Saudi Arabia
(missile hit US army barracks, 28 persons killed)
- ▶ The problem was caused by incorrect measurement of time

Simplified principle of function:

- ▶ Patriot's radar detects an airborne object
- ▶ the object is identified as a scud missile (according to speed, size, etc.)
- ▶ the range gate computes an area in the air space where the system should next look for it
- ▶ finding the object in the calculated area confirms that it is a scud
- ▶ then the scud is intercepted

Patriot Missile Mistiming



Patriot Missile Mistiming

Prediction of the new area:

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**
- ▶ computation in 24bit fixed floating point numbers

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**
- ▶ computation in 24bit fixed floating point numbers

The time converted to 24bit real number and multiplied with 1/10 represented in 24bit (i.e. the real value of 1/10 was 0.099999905)

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**
- ▶ computation in 24bit fixed floating point numbers

The time converted to 24bit real number and multiplied with 1/10 represented in 24bit (i.e. the real value of 1/10 was 0.099999905)

- ▶ the system was already running for 100 hours, i.e. the counter value was 360000, i.e. $360000 \cdot 0.099999905 = 35999.6568$

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**
- ▶ computation in 24bit fixed floating point numbers

The time converted to 24bit real number and multiplied with 1/10 represented in 24bit (i.e. the real value of 1/10 was 0.099999905)

- ▶ the system was already running for 100 hours, i.e. the counter value was 360000, i.e. $360000 \cdot 0.099999905 = 35999.6568$
- ▶ the error was 0.3432 seconds, which means 687 m off MACH 5 scud missile

Patriot Missile Mistiming

Prediction of the new area:

- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**
- ▶ computation in 24bit fixed floating point numbers

The time converted to 24bit real number and multiplied with 1/10 represented in 24bit (i.e. the real value of 1/10 was 0.099999905)

- ▶ the system was already running for 100 hours, i.e. the counter value was 360000, i.e. $360000 \cdot 0.099999905 = 35999.6568$
- ▶ the error was 0.3432 seconds, which means 687 m off MACH 5 scud missile
- ▶ the problem was not only in wrong conversion but in the fact that at some points correct conversion was used (after incomplete bug fix), so the errors did not cancel out

Patriot Missile Mistiming

Prediction of the new area:

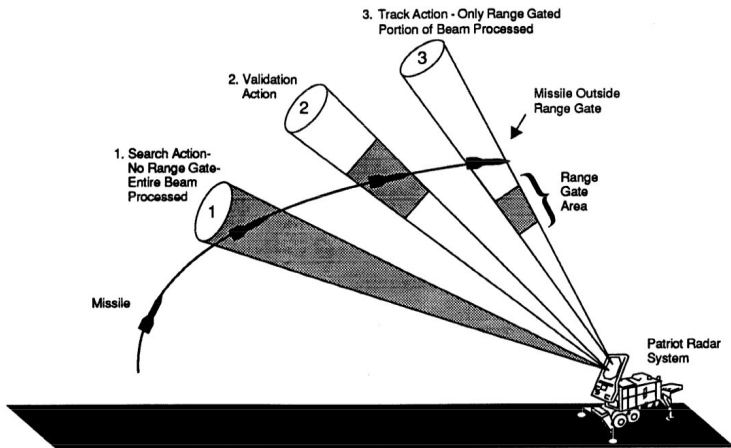
- ▶ a function of *velocity* and *time* of the last radar detection
- ▶ velocity represented as a real number
- ▶ **the current time kept by incrementing whole number counter counting tenths of seconds**
- ▶ computation in 24bit fixed floating point numbers

The time converted to 24bit real number and multiplied with 1/10 represented in 24bit (i.e. the real value of 1/10 was 0.099999905)

- ▶ the system was already running for 100 hours, i.e. the counter value was 360000, i.e. $360000 \cdot 0.099999905 = 35999.6568$
- ▶ the error was 0.3432 seconds, which means 687 m off MACH 5 scud missile
- ▶ the problem was not only in wrong conversion but in the fact that at some points correct conversion was used (after incomplete bug fix), so the errors did not cancel out

As a result, the tracking gate looked into wrong area

Patriot Missile Mistiming



Starliner

- ▶ Developed by Boeing & NASA
- ▶ Seven passengers, or a mix of crew and cargo, for missions to low-Earth orbit



Starliner

- ▶ Developed by Boeing & NASA
- ▶ Seven passengers, or a mix of crew and cargo, for missions to low-Earth orbit
- ▶ A timing issue occurred on the last Orbital Flight Test on December 20, 2019



- ▶ Developed by Boeing & NASA
- ▶ Seven passengers, or a mix of crew and cargo, for missions to low-Earth orbit
- ▶ A timing issue occurred on the last Orbital Flight Test on December 20, 2019
- ▶ What is supposed to happen:
 - ▶ Atlas V leaves Starliner on a suborbital trajectory.
 - ▶ Starliner's own propulsion system takes the spacecraft into orbit and to ISS.



- ▶ Developed by Boeing & NASA
- ▶ Seven passengers, or a mix of crew and cargo, for missions to low-Earth orbit
- ▶ A timing issue occurred on the last Orbital Flight Test on December 20, 2019
- ▶ What is supposed to happen:
 - ▶ Atlas V leaves Starliner on a suborbital trajectory.
 - ▶ Starliner's own propulsion system takes the spacecraft into orbit and to ISS.
- ▶ What happened:
 - ▶ Mission Elapsed Timer (MET), or clock, on Starliner was set to the wrong time and did not trigger the engines to fire correctly.
 - ▶ Other onboard systems compensated and it reached orbit, but had depleted so much fuel there was not enough to continue the journey.



(Rough) Course Outline

- ▶ Real-time scheduling
 - ▶ Time and priority driven
 - ▶ Resource control
 - ▶ Multi-processor (a bit)

(Rough) Course Outline

- ▶ Real-time scheduling
 - ▶ Time and priority driven
 - ▶ Resource control
 - ▶ Multi-processor (a bit)

- ▶ A little bit on programming real-time systems
 - ▶ Real-time operating systems

Outline – Scheduling

The Scheduling problem:

Input:

- ▶ available processors, resources
- ▶ set of tasks/jobs
with their requirements, deadlines, etc.

Outline – Scheduling

The Scheduling problem:

Input:

- ▶ available processors, resources
- ▶ set of tasks/jobs
with their requirements, deadlines, etc.

Question: How to assign processors and resources to tasks/jobs so that all requirements are met?

Outline – Scheduling

The Scheduling problem:

Input:

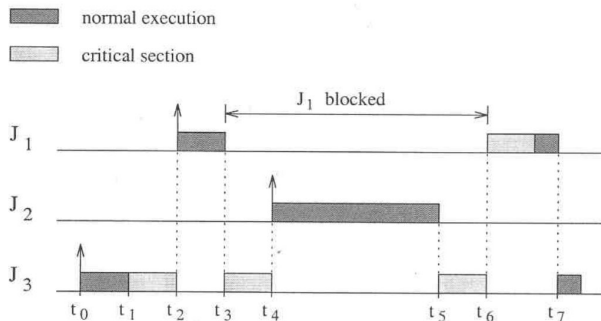
- ▶ available processors, resources
- ▶ set of tasks/jobs
with their requirements, deadlines, etc.

Question: How to assign processors and resources to tasks/jobs so that all requirements are met?

Example:

- ▶ 1 processor, one critical section shared by job 1 and job 3
- ▶ job 1: release time 1, computation time 4, deadline 8
- ▶ job 2: release time 1, computation time 2, deadline 5
- ▶ job 3: release time 0, computation time 3, deadline 4
- ▶ ...

Outline – Scheduling



- ▶ We consider a formal model of systems with parallel jobs that possibly contend for shared resources
consider periodic as well as aperiodic jobs
- ▶ Consider various algorithms that schedule jobs to meet their timing constraints
offline and online algorithms, RM, EDF, etc.

Outline – Programming

The screenshot shows the Microsoft Support website interface. At the top left is the Microsoft logo and the word 'Support'. To the right is a search input field. Below this is a blue navigation bar with three options: 'Find it myself' (with a dropdown arrow), 'Ask the community', and 'Get live help'. To the right of these options is the text 'Select the product you need help with' followed by a row of product icons: Windows, Internet Explorer, Office, Surface, Xbox, and Skype. Below the navigation bar, the article title 'Windows Does Not Support Real-Time Programming' is displayed in a large font. Underneath the title is the article ID 'Article ID: 22523' and a link 'View products that this article applies to'. At the bottom of the screenshot, there is a small icon and the text 'Retired KB Content Disclaimer'.

Basic information about RTOS and RT programming languages

- ▶ RTOS – overview
 - ▶ real-time in non-real-time operating systems
 - ▶ **implementation of theoretical concepts in freeRTOS**
- ▶ RT in programming languages – short overview

Real-Time Scheduling

Formal Model

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

<http://csparks.org/teaching/rtes/index.html>]

Real-Time Scheduling – Formal Model

- ▶ Introduce an abstract model of real-time systems
 - ▶ abstracts away unessential details
 - ▶ sets up consistent terminology

Real-Time Scheduling – Formal Model

- ▶ Introduce an abstract model of real-time systems
 - ▶ abstracts away unessential details
 - ▶ sets up consistent terminology

- ▶ Three components of the model
 - ▶ A workload model that describes applications supported by the system
i.e. jobs, tasks, ...
 - ▶ A resource model that describes the system resources available to applications
i.e. processors, passive resources, ...
 - ▶ Algorithms that define how the application uses the resources at all times
i.e. scheduling and resource access protocols

- ▶ A *job* is a unit of work that is scheduled and executed by a system
compute a control law, transform sensor data, etc.

Basic Notions

- ▶ A *job* is a unit of work that is scheduled and executed by a system
compute a control law, transform sensor data, etc.
- ▶ A *task* is a set of related jobs which jointly provide some system function
check temperature periodically, keep a steady flow of water

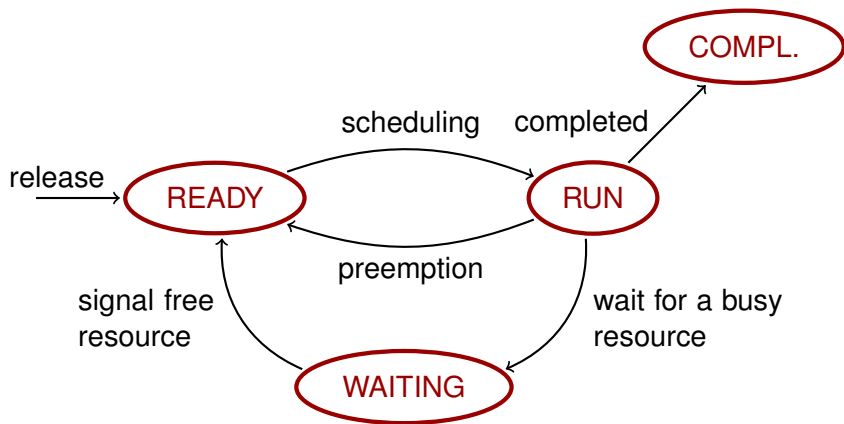
Basic Notions

- ▶ A *job* is a unit of work that is scheduled and executed by a system
compute a control law, transform sensor data, etc.
- ▶ A *task* is a set of related jobs which jointly provide some system function
check temperature periodically, keep a steady flow of water
- ▶ A job executes on a *processor*
CPU, transmission link in a network, database server, etc.

Basic Notions

- ▶ A *job* is a unit of work that is scheduled and executed by a system
compute a control law, transform sensor data, etc.
- ▶ A *task* is a set of related jobs which jointly provide some system function
check temperature periodically, keep a steady flow of water
- ▶ A job executes on a *processor*
CPU, transmission link in a network, database server, etc.
- ▶ A job may use some (shared) passive *resources*
file, database lock, shared variable etc.

Life Cycle of a Job



Jobs – Parameters

We consider finite, or countably infinite number of jobs J_1, J_2, \dots

Each job has several parameters.

Jobs – Parameters

We consider finite, or countably infinite number of jobs J_1, J_2, \dots

Each job has several parameters.

There are four types of job parameters:

- ▶ temporal
 - ▶ release time, execution time, deadlines
- ▶ functional
 - ▶ Laxity type: hard and soft real-time
 - ▶ preemptability, (criticality)
- ▶ interconnection
 - ▶ precedence constraints
- ▶ resource
 - ▶ usage of processors and passive resources

Job Parameters – Execution Time

Execution time e_i of a job J_i – the amount of time required to complete the execution of J_i when it executes alone and has all necessary resources

Execution time e_i of a job J_i – the amount of time required to complete the execution of J_i when it executes alone and has all necessary resources

- ▶ Value of e_i depends upon complexity of the job and speed of the processor on which it executes; may change for various reasons:
 - ▶ Conditional branches
 - ▶ Caches, pipelines, etc.
 - ▶ ...
- ▶ **Execution times fall into an interval $[e_i^-, e_i^+]$** ; we assume that we know this interval (WCET analysis) but not necessarily e_i

Job Parameters – Execution Time

Execution time e_i of a job J_i – the amount of time required to complete the execution of J_i when it executes alone and has all necessary resources

- ▶ Value of e_i depends upon complexity of the job and speed of the processor on which it executes; may change for various reasons:
 - ▶ Conditional branches
 - ▶ Caches, pipelines, etc.
 - ▶ ...
- ▶ **Execution times fall into an interval $[e_i^-, e_i^+]$** ; we assume that we know this interval (WCET analysis) but not necessarily e_i

We usually validate the system using only e_i^+ for each job
i.e. assume $e_i = e_i^+$

Job Parameters – Release and Response Time

Release time r_i – the instant in time when a job J_i becomes available for execution

- ▶ Release time may *jitter*, only an interval $[r_i^-, r_i^+]$ is known
- ▶ A job can be executed at any time at, or after, its release time, provided its processor and resource demands are met

Job Parameters – Release and Response Time

Release time r_i – the instant in time when a job J_i becomes available for execution

- ▶ Release time may *jitter*, only an interval $[r_i^-, r_i^+]$ is known
- ▶ A job can be executed at any time at, or after, its release time, provided its processor and resource demands are met

Completion time C_i – the instant in time when a job completes its execution

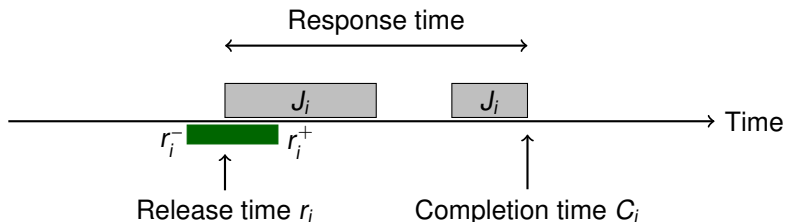
Job Parameters – Release and Response Time

Release time r_i – the instant in time when a job J_i becomes available for execution

- ▶ Release time may *jitter*, only an interval $[r_i^-, r_i^+]$ is known
- ▶ A job can be executed at any time at, or after, its release time, provided its processor and resource demands are met

Completion time C_i – the instant in time when a job completes its execution

Response time – the difference $C_i - r_i$ between the completion time and the release time



Job Parameters – Deadlines

Absolute deadline d_i – the instant in time by which a job must be completed

Job Parameters – Deadlines

Absolute deadline d_i – the instant in time by which a job must be completed

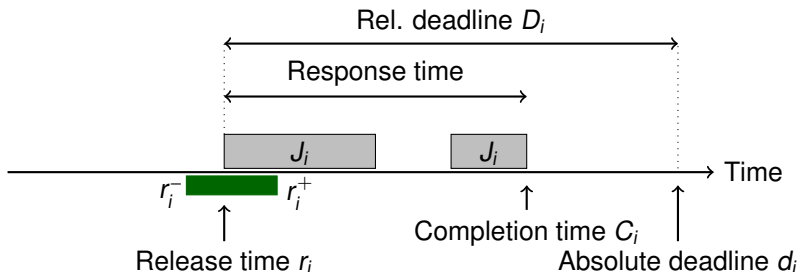
Relative deadline D_i – the maximum allowable response time
i.e. $D_i = d_i - r_i$

Job Parameters – Deadlines

Absolute deadline d_i – the instant in time by which a job must be completed

Relative deadline D_i – the maximum allowable response time
i.e. $D_i = d_i - r_i$

Feasible interval is the interval $(r_i, d_i]$

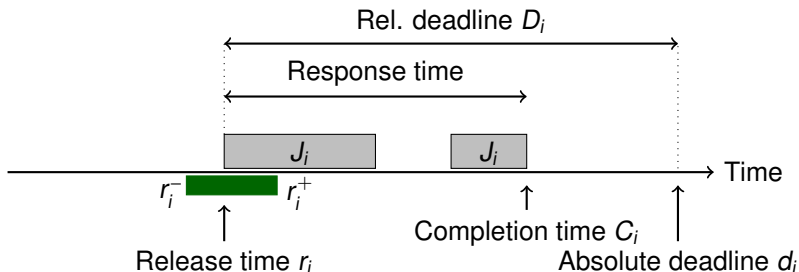


Job Parameters – Deadlines

Absolute deadline d_i – the instant in time by which a job must be completed

Relative deadline D_i – the maximum allowable response time
i.e. $D_i = d_i - r_i$

Feasible interval is the interval $(r_i, d_i]$



A *timing constraint* of a job is specified using release time together with relative and absolute deadlines.

Laxity Type – Hard Real-Time

A **hard real-time constraint** specifies that a job should never miss its deadline.

Examples: Flight control, railway signaling, anti-lock brakes, etc.

Laxity Type – Hard Real-Time

A **hard real-time constraint** specifies that a job should never miss its deadline.

Examples: Flight control, railway signaling, anti-lock brakes, etc.

Several more precise definitions occur in literature:

- ▶ A timing constraint is hard if the failure to meet it is considered a fatal error
e.g. a bomb is dropped too late and hits civilians

Laxity Type – Hard Real-Time

A **hard real-time constraint** specifies that a job should never miss its deadline.

Examples: Flight control, railway signaling, anti-lock brakes, etc.

Several more precise definitions occur in literature:

- ▶ A timing constraint is hard if the failure to meet it is considered a fatal error
e.g. a bomb is dropped too late and hits civilians
- ▶ A timing constraint is hard if the usefulness of the results falls off abruptly (may even become negative) at the deadline
Here the nature of abruptness allows to soften the constraint

Laxity Type – Hard Real-Time

A **hard real-time constraint** specifies that a job should never miss its deadline.

Examples: Flight control, railway signaling, anti-lock brakes, etc.

Several more precise definitions occur in literature:

- ▶ A timing constraint is hard if the failure to meet it is considered a fatal error
e.g. a bomb is dropped too late and hits civilians
- ▶ A timing constraint is hard if the usefulness of the results falls off abruptly (may even become negative) at the deadline
Here the nature of abruptness allows to soften the constraint

Definition 5

A *timing constraint is hard* if the user requires *formal validation* that the job meets its timing constraint.

Laxity Type – Soft Real-Time

A **soft real-time constraint** specifies that a job could occasionally miss its deadline

Examples: stock trading, multimedia, etc.

Laxity Type – Soft Real-Time

A **soft real-time constraint** specifies that a job could occasionally miss its deadline

Examples: stock trading, multimedia, etc.

Several more precise definitions occur in literature:

- ▶ A timing constraint is soft if the failure to meet it is undesirable but acceptable if the probability is low

Laxity Type – Soft Real-Time

A **soft real-time constraint** specifies that a job could occasionally miss its deadline

Examples: stock trading, multimedia, etc.

Several more precise definitions occur in literature:

- ▶ A timing constraint is soft if the failure to meet it is undesirable but acceptable if the probability is low
- ▶ A timing constraint is soft if the usefulness of the results decreases at a slower rate with *tardiness* of the job
e.g. the probability that a response time exceeds 50 ms is less than 0.2

Laxity Type – Soft Real-Time

A **soft real-time constraint** specifies that a job could occasionally miss its deadline

Examples: stock trading, multimedia, etc.

Several more precise definitions occur in literature:

- ▶ A timing constraint is soft if the failure to meet it is undesirable but acceptable if the probability is low
- ▶ A timing constraint is soft if the usefulness of the results decreases at a slower rate with *tardiness* of the job
e.g. the probability that a response time exceeds 50 ms is less than 0.2

Definition 6

A *timing constraint is soft* if either validation is not required, or only a demonstration that a *statistical constraint* is met suffices.

Jobs – Preemptability

Jobs may be interrupted by higher priority jobs

Jobs – Preemptability

Jobs may be interrupted by higher priority jobs

- ▶ A job is *preemptable* if its execution can be interrupted
- ▶ A job is *non-preemptable* if it must run to completion once started
(Some preemptable jobs have periods during which they cannot be preempted)
- ▶ The *context switch time* is the time to switch between jobs
(Most of the time we assume that this time is negligible)

Jobs – Preemptability

Jobs may be interrupted by higher priority jobs

- ▶ A job is *preemptable* if its execution can be interrupted
- ▶ A job is *non-preemptable* if it must run to completion once started
(Some preemptable jobs have periods during which they cannot be preempted)
- ▶ The *context switch time* is the time to switch between jobs
(Most of the time we assume that this time is negligible)

Reasons for preemptability:

- ▶ Jobs may have different levels of criticality
e.g. brakes vs radio tuning
- ▶ Priorities may make part of scheduling algorithm
e.g. resource access control algorithms

Jobs – Precedence Constraints

Jobs may be constrained to execute in a particular order

Jobs – Precedence Constraints

Jobs may be constrained to execute in a particular order

- ▶ This is known as a *precedence constraint*
- ▶ A job J_i is a *predecessor* of another job J_k and J_k a *successor* of J_i (denoted by $J_i < J_k$) if J_k cannot begin execution until the execution of J_i completes
- ▶ J_i is an *immediate predecessor* of J_k if $J_i < J_k$ and there is no other job J_j such that $J_i < J_j < J_k$
- ▶ J_i and J_k are *independent* when neither $J_i < J_k$ nor $J_k < J_i$

Jobs – Precedence Constraints

Jobs may be constrained to execute in a particular order

- ▶ This is known as a *precedence constraint*
- ▶ A job J_i is a *predecessor* of another job J_k and J_k a *successor* of J_i (denoted by $J_i < J_k$) if J_k cannot begin execution until the execution of J_i completes
- ▶ J_i is an *immediate predecessor* of J_k if $J_i < J_k$ and there is no other job J_j such that $J_i < J_j < J_k$
- ▶ J_i and J_k are *independent* when neither $J_i < J_k$ nor $J_k < J_i$

A job with a precedence constraint becomes ready for execution when its release time has passed and when all predecessors have completed.

Example: authentication before retrieving an information, a signal processing job in radar surveillance system precedes a tracker job

Tasks – Modeling Reactive Systems

Reactive systems – run for unlimited amount of time

Tasks – Modeling Reactive Systems

Reactive systems – run for unlimited amount of time

A system parameter: number of tasks

- ▶ may be known in advance (flight control)
- ▶ may change during computation (air traffic control)

Tasks – Modeling Reactive Systems

Reactive systems – run for unlimited amount of time

A system parameter: number of tasks

- ▶ may be known in advance (flight control)
- ▶ may change during computation (air traffic control)

We consider three types of tasks

- ▶ Periodic – jobs executed at regular intervals, hard deadlines
- ▶ Aperiodic – jobs executed in random intervals, soft deadlines
- ▶ Sporadic – jobs executed in random intervals, hard deadlines

... precise definitions later.

Processors

A **processor**, P , is an *active* component on which jobs are scheduled

Processors

A **processor**, P , is an **active** component on which jobs are scheduled

The general case considered in literature:

m processors P_1, \dots, P_m , each P_i has its *type* and *speed*.

Processors

A **processor**, P , is an **active** component on which jobs are scheduled

The general case considered in literature:

m processors P_1, \dots, P_m , each P_i has its *type* and *speed*.

We mostly concentrate on **single processor** scheduling

- ▶ Efficient scheduling algorithms
- ▶ In a sense subsumes multiprocessor scheduling where tasks are assigned *statically* to individual processors
i.e. all jobs of every task are assigned to a single processor

A **processor**, P , is an **active** component on which jobs are scheduled

The general case considered in literature:

m processors P_1, \dots, P_m , each P_i has its *type* and *speed*.

We mostly concentrate on **single processor** scheduling

- ▶ Efficient scheduling algorithms
- ▶ In a sense subsumes multiprocessor scheduling where tasks are assigned *statically* to individual processors
i.e. all jobs of every task are assigned to a single processor

Multi-processor scheduling is a rich area of current research, we touch it only lightly (later).

Resources

A resource, R , is a *passive* entity upon which jobs may depend

Resources

A resource, R , is a *passive* entity upon which jobs may depend

In general, **we consider n resources R_1, \dots, R_n of distinct types**

Resources

A resource, R , is a *passive* entity upon which jobs may depend

In general, **we consider n resources R_1, \dots, R_n of distinct types**

Each R_i is used in a mutually exclusive manner

- ▶ A job that acquires a free resource locks the resource
- ▶ Jobs that need a busy resource have to wait until the resource is released
- ▶ Once released, the resource may be used by another job (i.e. it is not consumed)

(More generally, each resource may be used by k jobs concurrently, i.e., there are k units of the resource)

Resources

A **resource**, R , is a *passive* entity upon which jobs may depend

In general, **we consider n resources R_1, \dots, R_n of distinct types**

Each R_i is used in a mutually exclusive manner

- ▶ A job that acquires a free resource locks the resource
- ▶ Jobs that need a busy resource have to wait until the resource is released
- ▶ Once released, the resource may be used by another job (i.e. it is not consumed)

(More generally, each resource may be used by k jobs concurrently, i.e., there are k units of the resource)

Resource requirements of a job specify

- ▶ which resources are used by the job
- ▶ the time interval(s) during which each resource is required (precise definitions later)

Scheduling

Schedule assigns, in every time instant, processors and resources to jobs.

More formally, a schedule is a function

$$\sigma : \{J_1, \dots\} \times \mathbb{R}_0^+ \rightarrow \mathcal{P}(\{P_1, \dots, P_m, R_1, \dots, R_n\})$$

so that for every $t \in \mathbb{R}_0^+$ there are rational $0 \leq t_1 \leq t < t_2$ such that $\sigma(J_i, \cdot)$ is constant on $[t_1, t_2)$.

(We also assume that there is the least time quantum in which scheduler does not change its decisions, i.e. each of the intervals $[t_1, t_2)$ is larger than a fixed $\varepsilon > 0$.)

Valid and Feasible Schedule

A schedule is *valid* if it satisfies the following conditions:

- ▶ Every processor is assigned to at most one job at any time
- ▶ Every job is assigned to at most one processor at any time
- ▶ No job is scheduled before its release time
- ▶ The total amount of processor time assigned to a given job is equal to its actual execution time
- ▶ *All the precedence and resource usage constraints are satisfied*

Valid and Feasible Schedule

A schedule is *valid* if it satisfies the following conditions:

- ▶ Every processor is assigned to at most one job at any time
- ▶ Every job is assigned to at most one processor at any time
- ▶ No job is scheduled before its release time
- ▶ The total amount of processor time assigned to a given job is equal to its actual execution time
- ▶ *All the precedence and resource usage constraints are satisfied*

A schedule is *feasible* if *all jobs with hard real-time constraints* complete before their deadlines

Valid and Feasible Schedule

A schedule is *valid* if it satisfies the following conditions:

- ▶ Every processor is assigned to at most one job at any time
- ▶ Every job is assigned to at most one processor at any time
- ▶ No job is scheduled before its release time
- ▶ The total amount of processor time assigned to a given job is equal to its actual execution time
- ▶ *All the precedence and resource usage constraints are satisfied*

A schedule is *feasible* if *all jobs with hard real-time constraints* complete before their deadlines

A set of jobs is *schedulable* if there is a feasible schedule for the set.

Scheduling – Algorithms

Scheduling algorithm computes a schedule for a set of jobs

A set of jobs is *schedulable according to a scheduling algorithm* if the algorithm produces a feasible schedule

Scheduling – Algorithms

Scheduling algorithm computes a schedule for a set of jobs

A set of jobs is *schedulable according to a scheduling algorithm* if the algorithm produces a feasible schedule

Scheduling – Algorithms

Scheduling algorithm computes a schedule for a set of jobs

A set of jobs is *schedulable according to a scheduling algorithm* if the algorithm produces a feasible schedule

Definition 7

A scheduling algorithm is *optimal* if it always produces a feasible schedule whenever such a schedule exists.

Real-Time Scheduling

Individual Jobs

Scheduling of Individual Jobs

We start with scheduling of finite sets of jobs $\{J_1, \dots, J_m\}$ for execution on **single processor** systems.

Scheduling of Individual Jobs

We start with scheduling of finite sets of jobs $\{J_1, \dots, J_m\}$ for execution on **single processor** systems.

Each J_i has a release time r_i , an execution time e_i and an absolute deadline d_i .

We assume hard real-time constraints.

The question: Is there an optimal scheduling algorithm?

Scheduling of Individual Jobs

We start with scheduling of finite sets of jobs $\{J_1, \dots, J_m\}$ for execution on **single processor** systems.

Each J_i has a release time r_i , an execution time e_i and an absolute deadline d_i .

We assume hard real-time constraints.

The question: Is there an optimal scheduling algorithm?

We proceed in the direction of growing generality:

1. No resources, independent, synchronized (i.e. $r_i = 0$ for all i)
2. No resources, independent but not synchronized
3. No resources but possibly dependent
4. The general case

No resources, Independent, Synchronized

	J_1	J_2	J_3	J_4	J_5
e_j	1	1	1	3	2
d_j	3	10	7	8	5

Is there a feasible schedule?

No resources, Independent, Synchronized

	J_1	J_2	J_3	J_4	J_5
e_j	1	1	1	3	2
d_j	3	10	7	8	5

Is there a feasible schedule?

Note: Preemption does not help in synchronized case.

No resources, Independent, Synchronized

	J_1	J_2	J_3	J_4	J_5
e_j	1	1	1	3	2
d_j	3	10	7	8	5

Is there a feasible schedule?

Note: Preemption does not help in synchronized case.

Theorem 8

If there are no resource contentions, then executing independent jobs in the order of non-decreasing deadline (EDD) produces a feasible schedule (if it exists).

No resources, Independent, Synchronized

	J_1	J_2	J_3	J_4	J_5
e_j	1	1	1	3	2
d_j	3	10	7	8	5

Is there a feasible schedule?

Note: Preemption does not help in synchronized case.

Theorem 8

If there are no resource contentions, then executing independent jobs in the order of non-decreasing deadline (EDD) produces a feasible schedule (if it exists).

Proof.

Let σ be a schedule. **Inversion** is a pair (J_a, J_b) such that J_a precedes J_b in σ but $d_b < d_a$.

Note that σ is EDD iff it does not contain any inversion.

Proof cont.

Assume $k > 0$ inversions in σ .

Let (J_a, J_b) be an inversion such that J_a is scheduled right before J_b .

There is always at least one such inversion (homework).

Let $t_a < t_b$ be the time instants when J_a, J_b start to be executed in σ .

Recall: C_a, C_b are completion times of J_a, J_b , and e_a, e_b are execution times.

Note that $C_a \leq d_a$ and that $C_b \leq d_b < d_a$.

Proof cont.

Assume $k > 0$ inversions in σ .

Let (J_a, J_b) be an inversion such that J_a is scheduled right before J_b .

There is always at least one such inversion (homework).

Let $t_a < t_b$ be the time instants when J_a, J_b start to be executed in σ .

Recall: C_a, C_b are completion times of J_a, J_b , and e_a, e_b are execution times.

Note that $C_a \leq d_a$ and that $C_b \leq d_b < d_a$.

Define a new schedule σ' in which:

- ▶ All jobs except J_a, J_b are scheduled as in σ ,
- ▶ J_b starts at t_a ,
- ▶ J_a starts at $t_a + e_b$.

Observe that σ' is still feasible:

- ▶ J_b is completed at $t_a + e_b < t_a + e_b + e_a = t_b + e_b = C_b \leq d_b$
- ▶ J_a is completed at $t_a + e_b + e_a = C_b \leq d_b < d_a$

Note that σ' has $k - 1$ inversions. By repeating the above procedure k times, we obtain an EDD schedule. □

No resources, Independent, Synchronized

Is there any simple schedulability test?

$\{J_1, \dots, J_n\}$ where $d_1 \leq \dots \leq d_n$ is schedulable iff
 $\forall i \in \{1, \dots, n\} : \sum_{k=1}^i e_k \leq d_i$

No resources, Independent (No Synchro)

	J_1	J_2	J_3
r_i	0	0	2
e_i	1	2	2
d_i	2	5	4

- ▶ find a (feasible) schedule (with and without preemption)
- ▶ determine response time of each job in your schedule

No resources, Independent (No Synchro)

	J_1	J_2	J_3
r_i	0	0	2
e_i	1	2	2
d_i	2	5	4

- ▶ find a (feasible) schedule (with and without preemption)
- ▶ determine response time of each job in your schedule

Preemption makes a difference.

No resources, Independent (No Synchro)

Earliest Deadline First (EDF) scheduling:

At any time instant, a job with the earliest absolute deadline is executed

Here EDF works in the preemptive case but not in the non-preemptive one.

	J_1	J_2
r_i	0	1
e_i	4	2
d_i	7	5

No Resources, Independent (No Synchro)

Theorem 9

If there are no resource contentions, jobs are independent and preemption is allowed, the EDF algorithm finds a feasible schedule (if it exists).

Proof.

We show that any feasible schedule σ can be transformed in finitely many steps to EDF schedule which is feasible.

No Resources, Independent (No Synchro)

Theorem 9

If there are no resource contentions, jobs are independent and preemption is allowed, the EDF algorithm finds a feasible schedule (if it exists).

Proof.

We show that any feasible schedule σ can be transformed in finitely many steps to EDF schedule which is feasible.

Let σ be a feasible schedule but not EDF. Assume, w.l.o.g., that for every $k \in \mathbb{N}$ at most one job is executed in the interval $[k, k + 1)$ and that all release times and deadlines are in \mathbb{N} .

(Otherwise rescale by the least common multiple.)

No Resources, Independent (No Synchro)

Proof cont.

We say that σ **violates** EDF at k if one of the following conditions holds:

1. No job is executed in $[k, k + 1)$ and there is a job J_b ready for execution in $[k, k + 1)$
2. There are two jobs J_a and J_b that satisfy:
 - ▶ J_a and J_b are ready for execution at k
 - ▶ J_a is executed in $[k, k + 1)$
 - ▶ $d_b < d_a$

No Resources, Independent (No Synchro)

Proof cont.

We say that σ **violates** EDF at k if one of the following conditions holds:

1. No job is executed in $[k, k + 1)$ and there is a job J_b ready for execution in $[k, k + 1)$
2. There are two jobs J_a and J_b that satisfy:
 - ▶ J_a and J_b are ready for execution at k
 - ▶ J_a is executed in $[k, k + 1)$
 - ▶ $d_b < d_a$

Let $k \in \mathbb{N}$ be the *least* time instant such that σ violates EDF at k .

Assume, w.l.o.g. that J_b has the minimum deadline among all jobs ready for execution at k .

No Resources, Independent (No Synchro)

Proof cont.

Consider the above two cases separately:

- ad 1.** Let us define a new schedule σ' which is the same as σ except that J_b is executed in the empty interval $[k, k + 1)$.
- ad 2.** There is $k < \ell < d_b$ such that J_b is executed in $[\ell, \ell + 1)$.

Let us define a new schedule σ' which is the same as σ except:

- ▶ executes J_b in $[k, k + 1)$
- ▶ executes J_a in $[\ell, \ell + 1)$

In both cases the σ' is feasible and does not violate EDF at any $k' \leq k$.

Finitely many steps transform any feasible schedule to EDF. □

No resources, Independent (No Synchro)

The **non-preemptive** case is NP-hard.

No resources, Independent (No Synchro)

The **non-preemptive** case is **NP-hard**.

Heuristics are needed, such as the **Spring algorithm**, that usually work in much more general setting (with resources etc.)

No resources, Independent (No Synchro)

The **non-preemptive** case is **NP-hard**.

Heuristics are needed, such as the **Spring algorithm**, that usually work in much more general setting (with resources etc.)

Use the notion of *partial schedule* where only a subset of jobs has been scheduled.

Exhaustive search through partial schedules

- ▶ start with an empty schedule

No resources, Independent (No Synchro)

The **non-preemptive** case is **NP-hard**.

Heuristics are needed, such as the **Spring algorithm**, that usually work in much more general setting (with resources etc.)

Use the notion of *partial schedule* where only a subset of jobs has been scheduled.

Exhaustive search through partial schedules

- ▶ start with an empty schedule
- ▶ in every step either
 - ▶ add a job which maximizes a *heuristic function* H among jobs that have not yet been tried in this partial schedule
 - ▶ or backtrack if there is no such a job

No resources, Independent (No Synchro)

The **non-preemptive** case is **NP-hard**.

Heuristics are needed, such as the **Spring algorithm**, that usually work in much more general setting (with resources etc.)

Use the notion of *partial schedule* where only a subset of jobs has been scheduled.

Exhaustive search through partial schedules

- ▶ start with an empty schedule
- ▶ in every step either
 - ▶ add a job which maximizes a *heuristic function* H among jobs that have not yet been tried in this partial schedule
 - ▶ or backtrack if there is no such a job
- ▶ After failure, backtrack to previous partial schedule

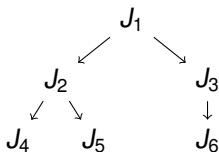
Heuristic function identifies plausible jobs to be scheduled (earliest release, earliest deadline, etc.)

No Resources, Dependent (No Synchro)

Example:

	J_1	J_2	J_3	J_4	J_5	J_6
e_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

Dependencies:



Does EDF work?

No resources, Dependent (No Synchro)

Theorem 10

Assume that there are no resource contentions and jobs are preemptable. There is a polynomial time algorithm which decides whether a feasible schedule exists and if yes, then computes one.

Idea: Reduce to independent jobs by changing release times and deadlines. Then use EDF.

No resources, Dependent (No Synchro)

Theorem 10

Assume that there are no resource contentions and jobs are preemptable. There is a polynomial time algorithm which decides whether a feasible schedule exists and if yes, then computes one.

Idea: Reduce to independent jobs by changing release times and deadlines. Then use EDF.

Observe that if $J_i < J_k$ then replacing

- ▶ r_k with $\max\{r_k, r_i + e_i\}$
(J_k cannot be scheduled for execution before $r_i + e_i$ because J_i cannot be finished before $r_i + e_i$)
- ▶ d_i with $\min\{d_i, d_k - e_k\}$
(J_i must be finished before $d_k - e_k$ so that J_k can be finished before d_k)

does not change feasibility.

Replace systematically according to the precedence relation.

No Resources, Dependent (No Synchro)

Define r_k^* , d_k^* systematically as follows:

- ▶ Pick J_k whose all predecessors have been processed and compute $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$. Repeat for all jobs.
- ▶ Pick J_k whose all successors have been processed and compute $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$. Repeat for all jobs.

No Resources, Dependent (No Synchro)

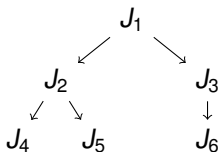
Define r_k^* , d_k^* systematically as follows:

- ▶ Pick J_k whose all predecessors have been processed and compute $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$. Repeat for all jobs.
- ▶ Pick J_k whose all successors have been processed and compute $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$. Repeat for all jobs.

Example:

	J_1	J_2	J_3	J_4	J_5	J_6
e_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

Dependencies:



No Resources, Dependent (No Synchro)

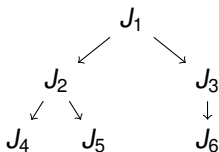
Define r_k^* , d_k^* systematically as follows:

- ▶ Pick J_k whose all predecessors have been processed and compute $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$. Repeat for all jobs.
- ▶ Pick J_k whose all successors have been processed and compute $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$. Repeat for all jobs.

Example:

	J_1	J_2	J_3	J_4	J_5	J_6
e_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

Dependencies:



Do you need the precedence constraints?

No Resources, Dependent (No Synchro)

Define r_k^* , d_k^* systematically as follows:

- ▶ Pick J_k whose all predecessors have been processed and compute $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$. Repeat for all jobs.
- ▶ Pick J_k whose all successors have been processed and compute $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$. Repeat for all jobs.

No Resources, Dependent (No Synchro)

Define r_k^* , d_k^* systematically as follows:

- ▶ Pick J_k whose all predecessors have been processed and compute $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$. Repeat for all jobs.
- ▶ Pick J_k whose all successors have been processed and compute $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$. Repeat for all jobs.

This gives a new set of jobs J_1^*, \dots, J_m^* where each J_k^* has the release time r_k^* and the absolute deadline d_k^* .

We impose **no precedence constraints** on J_1^*, \dots, J_m^* .

No Resources, Dependent (No Synchro)

Define r_k^* , d_k^* systematically as follows:

- ▶ Pick J_k whose all predecessors have been processed and compute $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$. Repeat for all jobs.
- ▶ Pick J_k whose all successors have been processed and compute $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$. Repeat for all jobs.

This gives a new set of jobs J_1^*, \dots, J_m^* where each J_k^* has the release time r_k^* and the absolute deadline d_k^* .

We impose **no precedence constraints** on J_1^*, \dots, J_m^* .

Lemma 11

$\{J_1, \dots, J_m\}$ is feasible iff $\{J_1^, \dots, J_m^*\}$ is feasible. If EDF schedule is feasible on $\{J_1^*, \dots, J_m^*\}$, then the same schedule is feasible on $\{J_1, \dots, J_m\}$.*

The same schedule means that whenever J_i^ is scheduled at time t , then J_i is scheduled at time t .*

No Resources, Dependent (No Synchro)

Recall: $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$ and
 $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$

Proof of Lemma 11.

\Rightarrow : It is easy to show that in *no feasible schedule* on $\{J_1, \dots, J_m\}$ any job J_k can be executed before r_k^* and completed after d_k^* (otherwise, precedence constraints would be violated).

No Resources, Dependent (No Synchro)

Recall: $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$ and
 $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$

Proof of Lemma 11.

\Rightarrow : It is easy to show that in *no feasible schedule* on $\{J_1, \dots, J_m\}$ any job J_k can be executed before r_k^* and completed after d_k^* (otherwise, precedence constraints would be violated).

\Leftarrow : Assume that EDF σ is feasible on $\{J_1^*, \dots, J_m^*\}$. Let us use σ on $\{J_1, \dots, J_m\}$.

I.e. J_i is executed iff J_i^* is executed.

No Resources, Dependent (No Synchro)

Recall: $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$ and
 $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$

Proof of Lemma 11.

\Rightarrow : It is easy to show that in *no feasible schedule* on $\{J_1, \dots, J_m\}$ any job J_k can be executed before r_k^* and completed after d_k^* (otherwise, precedence constraints would be violated).

\Leftarrow : Assume that EDF σ is feasible on $\{J_1^*, \dots, J_m^*\}$. Let us use σ on $\{J_1, \dots, J_m\}$.

i.e. J_i is executed iff J_i^* is executed.

Timing constraints of $\{J_1, \dots, J_m\}$ are satisfied since $r_k \leq r_k^*$ and $d_k \geq d_k^*$ for all k .

No Resources, Dependent (No Synchro)

Recall: $r_k^* := \max\{r_k, \max_{J_i < J_k} r_i^* + e_i\}$ and
 $d_k^* := \min\{d_k, \min_{J_k < J_i} d_i^* - e_i\}$

Proof of Lemma 11.

\Rightarrow : It is easy to show that in *no feasible schedule* on $\{J_1, \dots, J_m\}$ any job J_k can be executed before r_k^* and completed after d_k^* (otherwise, precedence constraints would be violated).

\Leftarrow : Assume that EDF σ is feasible on $\{J_1^*, \dots, J_m^*\}$. Let us use σ on $\{J_1, \dots, J_m\}$.

i.e. J_i is executed iff J_i^ is executed.*

Timing constraints of $\{J_1, \dots, J_m\}$ are satisfied since $r_k \leq r_k^*$ and $d_k \geq d_k^*$ for all k .

Precedence constraints: Assume that $J_s < J_t$. Then J_s^* executes completely before J_t^* since $r_s^* < r_s^* + e_s \leq r_t^*$ and $d_s^* \leq d_t^* - e_t < d_t^*$ and σ is EDF on $\{J_1^* \dots, J_m^*\}$.

Resources, Dependent, Not Synchronized

Even the preemptive case is NP-hard

- ▶ reduce the non-preemptive case without resources to the preemptive with resources
- ▶ Use a common resource R .
 - ▶ Whenever a job starts its execution it locks the resource R .
 - ▶ Whenever a job finishes its execution it releases the resource R .

Could be solved using heuristics, e.g. the Spring algorithm.

Real-Time Scheduling

Scheduling of Reactive Systems

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

<http://csparks.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

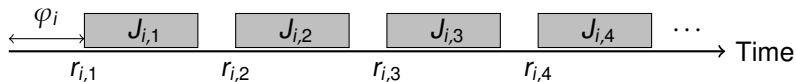
i.e. systems that run for unlimited amount of time

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

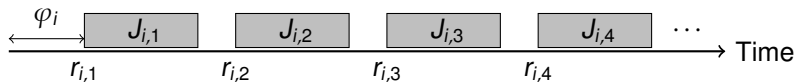
Periodic Tasks

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



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, ...,
- ▶ 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, ...,
- ▶ 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 $d_i = p_i$

$T_2 = (0, 10, 3, 6)$ 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, ...)

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 $d_i = p_i$

$T_2 = (0, 10, 3, 6)$ 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, ...)

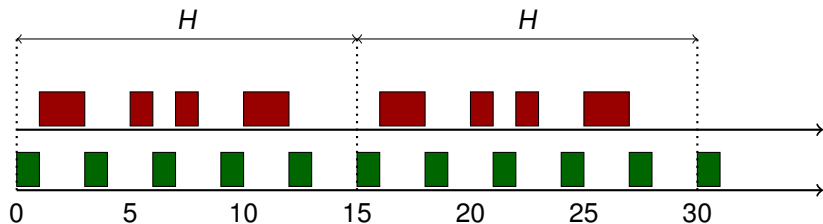
$T_3 = (0, 10, 3, 10)$ satisfies $\varphi = 0$, $p_i = 10$, $e_i = 3$, $D_i = 10$, i.e. jobs of T_3 are

- ▶ 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 surveillance 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

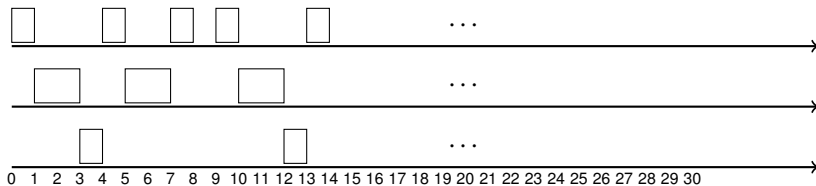
- ▶ Assign priorities to jobs, based on some algorithm
- ▶ Make scheduling decisions based on the priorities, when events such as releases and job completions occur
 - ▶ Priority scheduling algorithms are *event-driven*
 - ▶ Jobs are placed in one or more queues; at each event, the ready job with the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

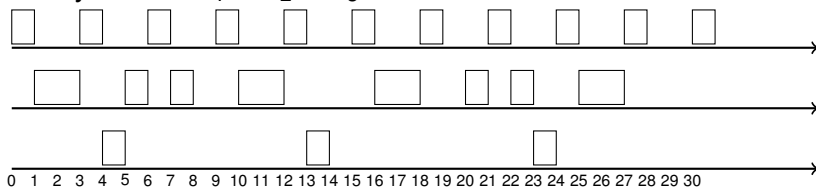
Clock-Driven & Priority-Driven Example

	T_1	T_2	T_3
p_i	3	5	10
e_i	1	2	1

Clock-Driven:



Priority-driven: $T_1 > T_2 > T_3$



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

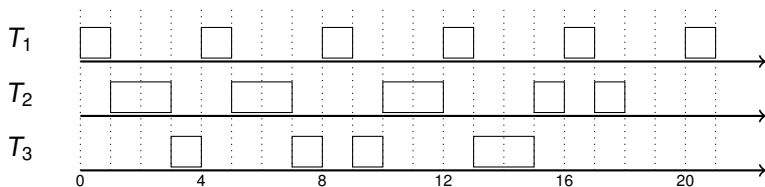
- ▶ The shorter the period, the higher the priority
- ▶ The *rate* is the inverse of the period, so jobs with higher rate have higher priority

RM is very widely studied and used

Example 12

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

The priorities: $T_1 > T_2 > T_3$



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 $T_2 = (2, 1)$. □

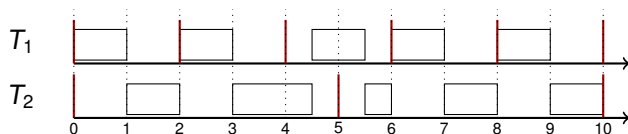
Earliest Deadline First (EDF) assigns priorities to jobs based on their *current absolute deadlines*

- ▶ 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 $\mathcal{T} = \{T_1, \dots, T_n\}$* is defined as the sum of utilizations of all tasks of \mathcal{T} , i.e. by

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

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 $U^{\mathcal{T}} < U_{\text{ALG}}$, then \mathcal{T} is schedulable by ALG.
 - ▶ If $U > U_{\text{ALG}}$, 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 13

Let $\mathcal{T} = \{T_1, \dots, T_n\}$ be a set of independent, preemptable periodic tasks with $D_i \geq p_i$ for $i = 1, \dots, n$. 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. \Rightarrow 2. We prove that $U^{\mathcal{T}} > 1$ implies that \mathcal{T} is not schedulable
2. \Rightarrow 3. We prove that if EDF fails to feasibly schedule, then $U^{\mathcal{T}} > 1$
3. \Rightarrow 1. Trivial



Proof of 1. \Rightarrow 2.

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

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. \Rightarrow 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. \Rightarrow 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. \Rightarrow 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 $\lim_{t \rightarrow \infty} (t \cdot U^{\mathcal{T}} - \sum_{i=1}^n \varphi_i u_i) - t = \infty$. 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. \Rightarrow 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 $\lim_{t \rightarrow \infty} (t \cdot U^{\mathcal{T}} - \sum_{i=1}^n \varphi_i u_i) - t = \infty$. So there exists t such that $t \cdot U^{\mathcal{T}} - \sum_{i=1}^n \varphi_i u_i > t + \max_j D_j$.

So in order to complete all jobs released before or at t we need more time than $t + \max_j D_j$. However, the latest deadline of a job released before or at t is $t + \max_j D_j$. 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 $i = 1, \dots, n.$
(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 $i = 1, \dots, n.$

(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 $i = 1, \dots, n.$

(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 $T_\ell.$

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. \Rightarrow 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. \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 $J_{i,1}$.

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. \Rightarrow 3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ , the first job of T_ℓ is released at 0, and thus $\lfloor \frac{p_i}{p_\ell} \rfloor$ 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 $2 = \lfloor \frac{5}{2} \rfloor = \lfloor \frac{p_i}{p_\ell} \rfloor$ of them have their deadlines in $[0, p_i]$.

Proof of 2. \Rightarrow 3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ , the first job of T_ℓ is released at 0, and thus $\lfloor \frac{p_i}{p_\ell} \rfloor$ 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 $2 = \lfloor \frac{5}{2} \rfloor = \lfloor \frac{p_i}{p_\ell} \rfloor$ 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. \Rightarrow 3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ , the first job of T_ℓ is released at 0, and thus $\lfloor \frac{p_i}{p_\ell} \rfloor$ 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 $2 = \lfloor \frac{5}{2} \rfloor = \lfloor \frac{p_i}{p_\ell} \rfloor$ 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 $i = 1, \dots, n$
(note that the general case immediately follows)

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 $i = 1, \dots, n$
(note that the general case immediately follows)

Assume that \mathcal{T} is not schedulable by EDF.

(We show that $U^{\mathcal{T}} > 1$)

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 $i = 1, \dots, n$
(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.

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 $i = 1, \dots, n$
(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.

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 $i = 1, \dots, n$

(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. \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_- .

- ▶ 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. \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_- .

- ▶ 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. \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_- .

- ▶ 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. \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_- .

- ▶ 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. \Rightarrow 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. \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 $[t_-, t]$.

For every T_ℓ , exactly $\lfloor \frac{t-R_\ell}{p_\ell} \rfloor$ jobs of T_ℓ belong to 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 $[t_-, t]$.

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. \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 $[t_-, t]$.

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:

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

which implies that $U^T > 1$.

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 14

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 $\mathcal{T} = \{T_1, \dots, T_n\}$ 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 $\mathcal{T} = \{T_1, \dots, T_n\}$ 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 $\mathcal{T} = \{T_1, \dots, T_n\}$ 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 $\mathcal{T} = \{T_1, \dots, T_n\}$

Fixed-Priority Algorithms

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

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 $\mathcal{T} = \{T_1, \dots, T_n\}$

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 $\mathcal{T} = \{T_1, \dots, T_n\}$

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 $T_1 = (4, 2)$ and $T_2 = (6, 3)$

$U^{\mathcal{T}} = 1$ 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 $T_1 = (4, 2)$ and $T_2 = (6, 3)$

$U^{\mathcal{T}} = 1$ 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 $T_1 = (4, 2)$ and $T_2 = (6, 3)$

$U^{\mathcal{T}} = 1$ 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 15

A set $\{T_1, \dots, T_n\}$ 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 16

The helicopter control system from the first lecture.

Optimality of RM for Simply Periodic Tasks

Definition 15

A set $\{T_1, \dots, T_n\}$ 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 16

The helicopter control system from the first lecture.

Theorem 17

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 17 is true in general, no "in phase" assumption is needed.

Proof of Theorem 17

By Theorem 13, every schedulable set \mathcal{T} satisfies $U^{\mathcal{T}} \leq 1$.

Proof of Theorem 17

By Theorem 13, 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 17

By Theorem 13, 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 $T_1 \supset \dots \supset T_n$ according to RM.

Proof of Theorem 17

By Theorem 13, 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 $T_1 \supset \dots \supset T_n$ according to RM.

Let us compute the total execution time of $J_{i,1}$ and all jobs that possibly 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 17

By Theorem 13, 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 $T_1 \supset \dots \supset T_n$ according to RM.

Let us compute the total execution time of $J_{i,1}$ and all jobs that possibly 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 17

By Theorem 13, 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 $T_1 \supset \dots \supset T_n$ according to RM.

Let us compute the total execution time of $J_{i,1}$ and all jobs that possibly 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 18

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \dots, n$) 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 18

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \dots, n$) 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 $T_1 \supset \dots \supset T_n$.

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

Theorem 18

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \dots, n$) 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 $T_1 \supset \dots \supset T_n$.

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 18

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \dots, n$) 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 $T_1 \supset \dots \supset T_n$.

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 18

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \dots, n$) 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 $T_1 \supset \dots \supset T_n$.

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 18

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \dots, n$) 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 $T_1 \supset \dots \supset T_n$.

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. □

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 19

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

Schedulable Utilization for RM

Theorem 19

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

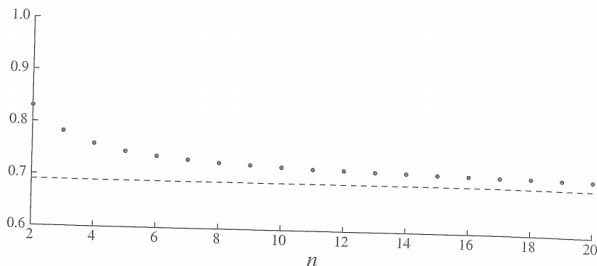
- ▶ If \mathcal{T} is a set of n tasks satisfying $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$, then $U^{\mathcal{T}}$ is schedulable according to the RM algorithm.

Schedulable Utilization for RM

Theorem 19

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 19 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

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

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

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

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 $p_1 \leq p_2 \leq 2p_1$.
(the latter condition is w.l.o.g., proof omitted)

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_1 \leq p_2 \leq 2p_1$.
(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 $p_1 \leq p_2 \leq 2p_1$.
(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 $p_1 \leq p_2 \leq 2p_1$.
(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 $U_{e_1}^{p_1, p_2}$ w.r.t. all parameters p_1, p_2, e_1 .

Proof – Special Case

To simplify, we restrict to two tasks and always assume $p_1 \leq p_2 \leq 2p_1$.
(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 $\min U$ of $U_{e_1}^{p_1, p_2}$ w.r.t. all parameters p_1, p_2, e_1 .

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 $U^{\mathcal{T}} \leq \min U$ we get $U^{\mathcal{T}} \leq \min U \leq U_{e_1}^{p_1, p_2}$, 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 $p_1 \leq p_2 \leq 2p_1$.
(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 $\min U$ of $U_{e_1}^{p_1, p_2}$ w.r.t. all parameters p_1, p_2, e_1 .

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 $U^{\mathcal{T}} \leq \min U$ we get $U^{\mathcal{T}} \leq \min U \leq U_{e_1}^{p_1, p_2}$, and thus the execution time e_2 cannot be larger than \max_{e_2} . Thus, \mathcal{T} is RM-schedulable.
- ▶ Given $U > \min U$, there must be p_1, p_2, e_1 satisfying $\min U \leq U_{e_1}^{p_1, p_2} < U$ 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 $p_1 \leq p_2 \leq 2p_1$.
(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 $\min U$ of $U_{e_1}^{p_1, p_2}$ w.r.t. all parameters p_1, p_2, e_1 .

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 $U^{\mathcal{T}} \leq \min U$ we get $U^{\mathcal{T}} \leq \min U \leq U_{e_1}^{p_1, p_2}$, and thus the execution time e_2 cannot be larger than \max_{e_2} . Thus, \mathcal{T} is RM-schedulable.
- ▶ Given $U > \min U$, there must be p_1, p_2, e_1 satisfying $\min U \leq U_{e_1}^{p_1, p_2} < U$ 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). Two cases depending on e_1 :

Proof – Special Case (Cont.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). Two cases depending on e_1 :

1. $e_1 < p_2 - p_1$:

Proof – Special Case (Cont.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) is $p_2 - 2e_1$.

Proof – Special Case (Cont.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) is $p_2 - 2e_1$. Which gives the utilization

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

Proof – Special Case (Cont.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) is $p_1 - e_1$.

Proof – Special Case (Cont.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) is $p_1 - e_1$. Which gives the utilization

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

Proof – Special Case (Cont.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) 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.)

First, minimize w.r.t. e_1 (p_1, p_2 fixed). 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) 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) 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 .

In both cases, 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 $U_{e_1}^{p_1, p_2}$:

Proof – Special Case (Cont.)

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

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

Proof – Special Case (Cont.)

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

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

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

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

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

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

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

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

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

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

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 is equal to zero 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

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

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

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

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

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

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

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

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 $\max_e_2 = p_1 - e_1 = p_1 - (p_2 - p_1) = 2p_1 - p_2$.

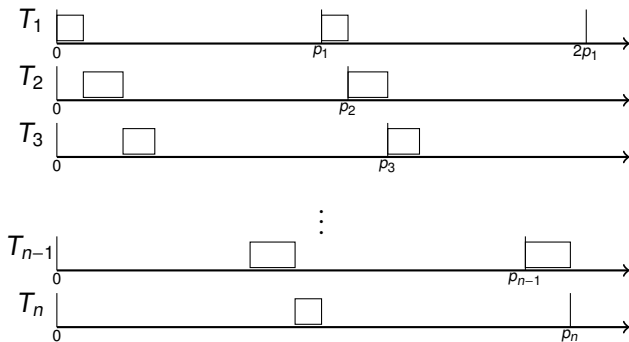
Scaling to $p_1 = 1$, we obtain a completely determined example

$$p_1 = 1 \quad p_2 = \sqrt{2} \approx 1.41 \quad e_1 = \sqrt{2} - 1 \approx 0.41 \quad \max_e_2 = 2 - \sqrt{2} \approx 0.59$$

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

Proof Idea of Theorem 19

Fix periods $p_1 < \dots < p_n$ 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).



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

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

Time-Demand Analysis

Consider a set of n tasks $\mathcal{T} = \{T_1, \dots, T_n\}$.

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 $\mathcal{T} = \{T_1, \dots, T_n\}$.

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 $T_1 \supset \dots \supset T_n$.

Time-Demand Analysis

Consider a set of n tasks $\mathcal{T} = \{T_1, \dots, T_n\}$.

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 $T_1 \supset \dots \supset T_n$.

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

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

(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

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

(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

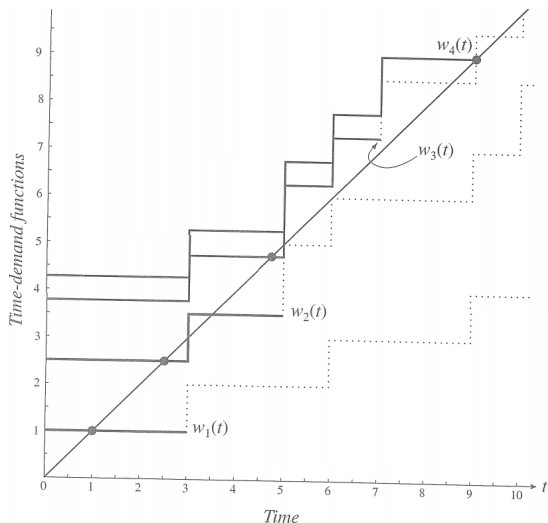
- ▶ Consider one task T_i at a time, starting with highest priority and working to lowest priority.
- ▶ Focus on the first job $J_{i,1}$ of T_i .
If $J_{i,1}$ makes it, all jobs of T_i will make it due to $\varphi_i = 0$.
- ▶ At time t for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[0, t]$ is bounded by

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

(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

$$U^{(T_1, \dots, T_4)} = 0.85 > 0.757 = U_{RM}(4)$$

- ▶ 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 $k = 1, 2, \dots, i$ and $j = 1, 2, \dots, \lfloor D_i/p_k \rfloor$

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

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 20

Assume $D_i \leq p_i$ for every i and use a fixed-priority algorithm. 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

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 20

Assume $D_i \leq p_i$ for every i and use a fixed-priority algorithm. 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 – Formally

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 20

Assume $D_i \leq p_i$ for every i and use a fixed-priority algorithm. 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!

To get such a critical instant, we set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T'_1, \dots, T'_n\}$. Denote jobs of T'_i by $J'_{i,k}$.

Critical Instant – Formally

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 20

Assume $D_i \leq p_i$ for every i and use a fixed-priority algorithm. 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!

To get such a critical instant, we set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T'_1, \dots, T'_n\}$. Denote jobs of T'_i by $J'_{i,k}$.

Corollary 21

Assume $D_i \leq p_i$ for every i and use a fixed-priority algorithm. Consider a critical instant t_{crit} of a task T_i .

- ▶ If the job $J_{i,k}$ released at t_{crit} misses its deadline, then $J'_{i,1}$ misses its deadline.
- ▶ Otherwise, the response time of $J_{i,k}$ is at most as large as the response time of $J'_{i,1}$.

Critical Instant and Schedulability Tests

Now we can (partially) decide schedulability of a given set of periodic tasks $\mathcal{T} = \{T_1, \dots, T_n\}$ (assuming $D_i \leq p_i$) as follows:

Critical Instant and Schedulability Tests

Now we can (partially) decide schedulability of a given set of periodic tasks $\mathcal{T} = \{T_1, \dots, T_n\}$ (assuming $D_i \leq p_i$) as follows:

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

Critical Instant and Schedulability Tests

Now we can (partially) decide schedulability of a given set of periodic tasks $\mathcal{T} = \{T_1, \dots, T_n\}$ (assuming $D_i \leq p_i$) as follows:

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

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

Critical Instant and Schedulability Tests

Now we can (partially) decide schedulability of a given set of periodic tasks $\mathcal{T} = \{T_1, \dots, T_n\}$ (assuming $D_i \leq p_i$) as follows:

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

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

- ▶ If \mathcal{T}' is schedulable, then also \mathcal{T} is schedulable.

Critical Instant and Schedulability Tests

Now we can (partially) decide schedulability of a given set of periodic tasks $\mathcal{T} = \{T_1, \dots, T_n\}$ (assuming $D_i \leq p_i$) as follows:

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

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

- ▶ If \mathcal{T}' is 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