Embedded Systems Programming - PA8001

http://goo.gl/cu8OOH

Lecture 6

Mohammad Mousavi
m.r.mousavi@hh.se

HALMSTAD UNIVERSITY
Center for Research on Embedded Systems
School of Information Science, Computer and Electrical Engineering

Obtaining WCET (Recap)

Testing

is likely to find the typical execution times, but finding the worst case is much harder.

Analysis

Impossible to find precise bounds for Turing complete language (recall the halting problem)

Instead: a safe WCET approximation

Much ongoing research on obtaining WCET: mostly beyond the scope of this course, dealing with programming techniques.

In this course

Assumption: for any sequential code a safe WCET can be obtained either by meassurement or by analysis or both!

Obtaining WCET (Recap)

Testing

is likely to find the typical execution times, but finding the worst case is much harder.

Analysis

Impossible to find precise bounds for Turing complete language (recall the halting problem)

Instead: a safe WCET approximation

Much ongoing research on obtaining WCET: mostly beyond the scope of this course, dealing with programming techniques.

In this course

Assumption: for any sequential code a safe WCET can be obtained either by meassurement or by analysis or both!

Obtaining WCET (Recap)

Testing

is likely to find the typical execution times, but finding the worst case is much harder.

Analysis

Impossible to find precise bounds for Turing complete language (recall the halting problem)
Instead: a safe WCET approximation

Much ongoing research on obtaining WCET: mostly beyond the scope of this course, dealing with programming techniques.

In this course

Assumption: for any sequential code a safe WCET can be obtained either by meassurement or by analysis or both!

Intermezzo: Halting Problem

Priority assignment

Question

How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.

Priority assignment

Question

How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.

Priority assignment

Question

How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.

Priority assignment

Question

How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.

Priority assignment

Question

How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

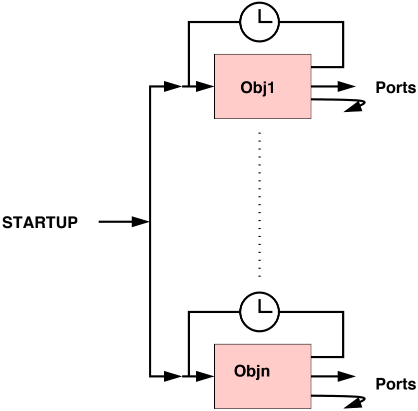
:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.

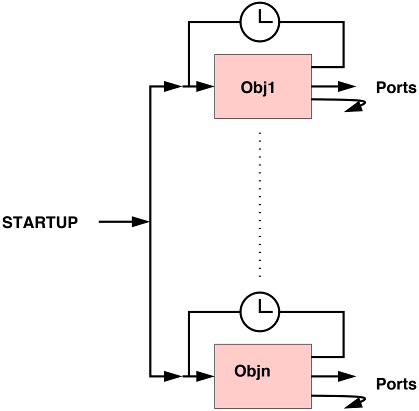
Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

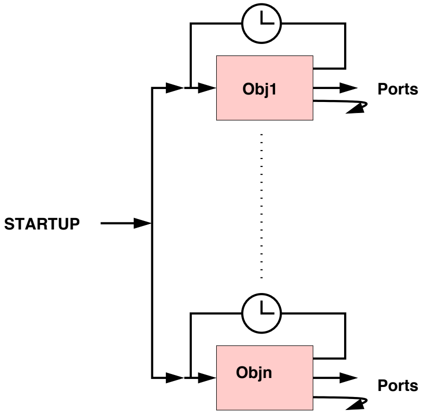
Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

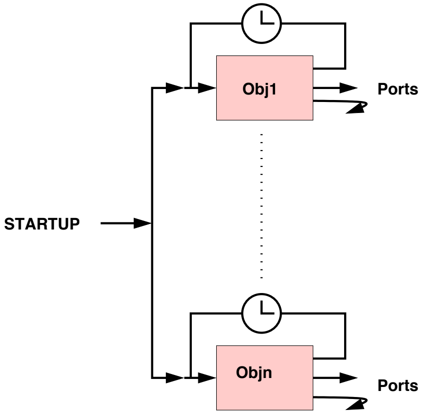
Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

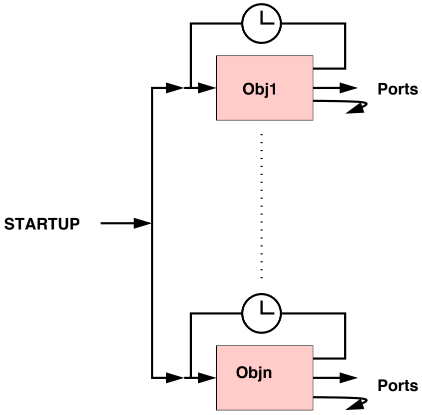
Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

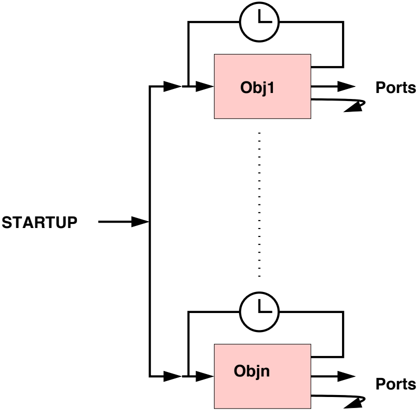
Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

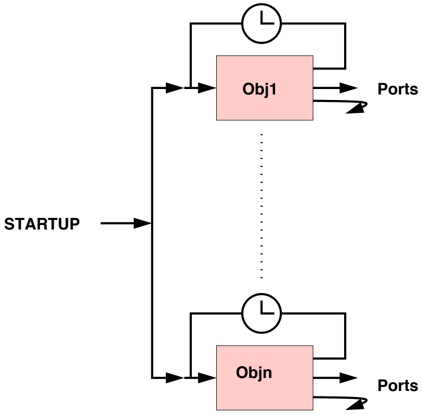
Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

Initial restricted model



- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

If time allows, we will discuss how to remove these restrictions.

Static priorities – method

Rate monotonic (RM)

Under the given assumptions, there exists a static priority assignment rule that is really simple

The shorter the period, the higher the priority

For RM, the actual priority values do not matter, only their relative order.

Because of our inverse priority scale, we can simply implement RM by letting $P_i = D_i$ $(=T_i)$

Static priorities – method

Rate monotonic (RM)

Under the given assumptions, there exists a static priority assignment rule that is really simple

The shorter the period, the higher the priority

For RM, the actual priority values do not matter, only their relative order.

Because of our inverse priority scale, we can simply implement RM by letting $P_i = D_i$ $(=T_i)$

Static priorities – method

Rate monotonic (RM)

Under the given assumptions, there exists a static priority assignment rule that is really simple

The shorter the period, the higher the priority

For RM, the actual priority values do not matter, only their relative order.

Because of our inverse priority scale, we can simply implement RM by letting $P_i = D_i$ $(=T_i)$

RM example

Given a set of periodic tasks with periods

T1 = 25ms
T2 = 60ms
T3 = 45ms

Valid priority assignments

P1 = 10
P2 = 19
P3 = 12

P1 = 1
P2 = 3
P3 = 2

P1 = 25
P2 = 60
P2 = 45

RM example

Given a set of periodic tasks with periods

$$\begin{aligned} T1 &= 25ms \\ T2 &= 60ms \\ T3 &= 45ms \end{aligned}$$

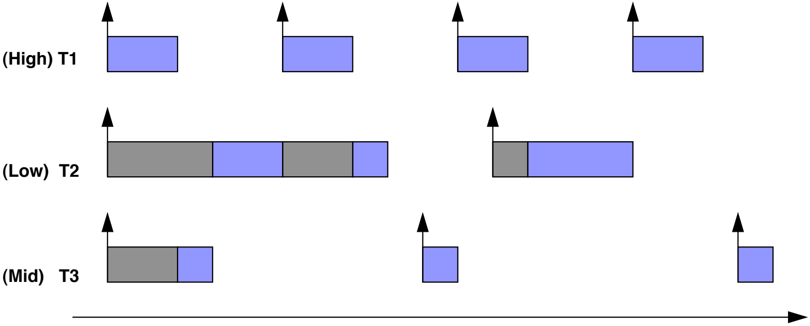
Valid priority assignments

$$\begin{aligned} P1 &= 10 \\ P2 &= 19 \\ P3 &= 12 \end{aligned}$$

$$\begin{aligned} P1 &= 1 \\ P2 &= 3 \\ P3 &= 2 \end{aligned}$$

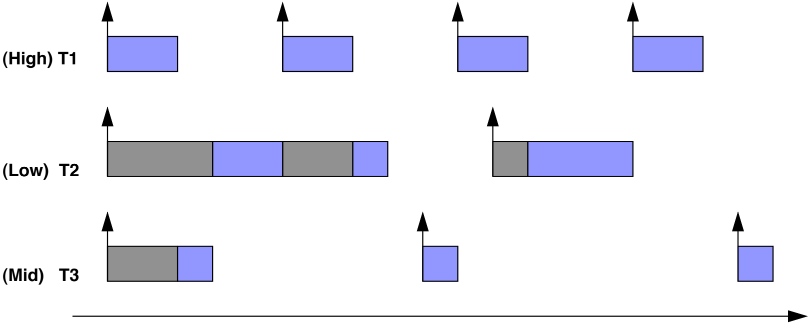
$$\begin{aligned} P1 &= 25 \\ P2 &= 60 \\ P2 &= 45 \end{aligned}$$

RM example



Period = Deadline. Arrows mark start of period.
Blue: running. Gray: waiting.

RM example



Period = Deadline. Arrows mark start of period.
Blue: running. Gray: waiting.

Dynamic priorities – method

Earliest Deadline First – EDF

Dynamic priority assignment rule:

> The shorter the time remaining until deadline, the higher the priority

To use absolute deadlines: priorities = remaining clock cycles (before missing the deadline)

Under EDF, each activation n of periodic task i will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$

Dynamic priorities – method

Earliest Deadline First – EDF

Dynamic priority assignment rule:

> The shorter the time remaining until deadline, the higher the priority

To use absolute deadlines: priorities = remaining clock cycles (before missing the deadline)

Under EDF, each activation n of periodic task i will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$

Dynamic priorities – method

Earliest Deadline First – EDF

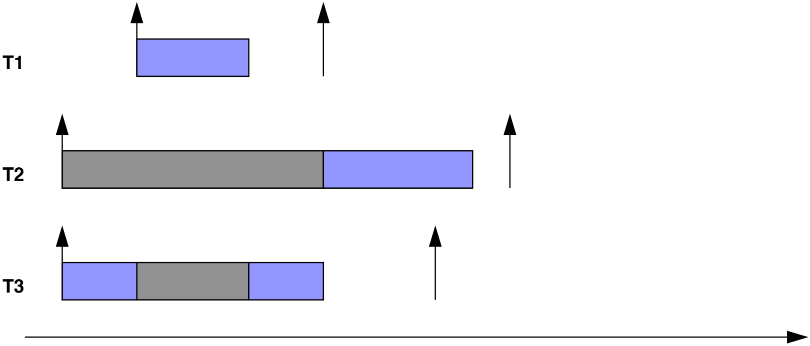
Dynamic priority assignment rule:

> The shorter the time remaining until deadline, the higher the priority

To use absolute deadlines: priorities = remaining clock cycles (before missing the deadline)

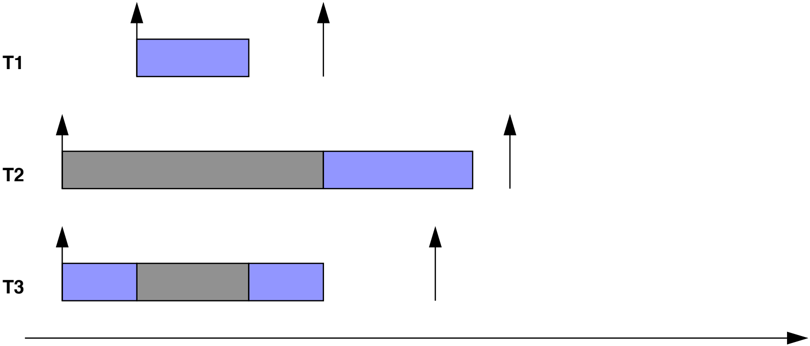
Under EDF, each activation n of periodic task i will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$

EDF example



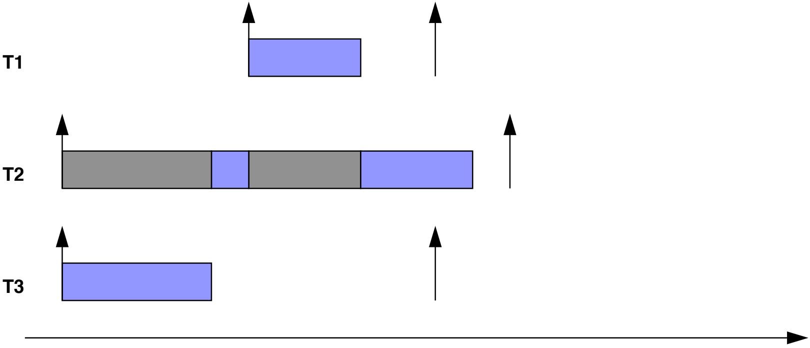
T1 arrives later, but its deadline is earlier than both T2's and T3's absolute deadlines!

EDF example



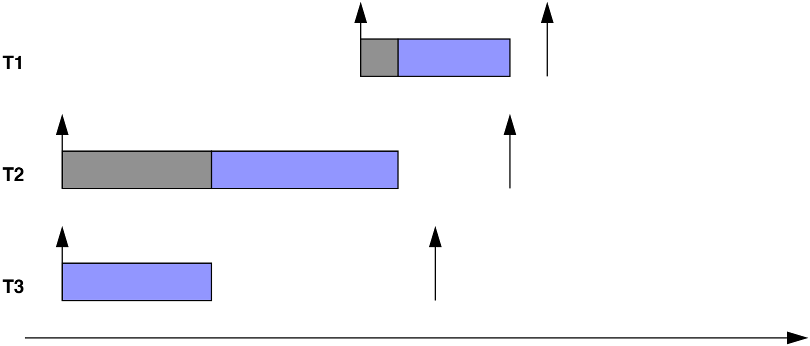
T1 arrives later, but its deadline is earlier than both T2's and T3's absolute deadlines!

EDF example



Deadline of T1 < Deadline of T2

EDF example



(absolute) Deadline of T1 $>$ (absolute) Deadline of T2

Optimality

Multiple ways assigning priorities to meet deadlines

Optimal: a method which fails only if every other method fails

- RM is optimal among static assignment methods
- EDF is optimal among dynamic methods

Optimality

Multiple ways assigning priorities to meet deadlines

Optimal: a method which fails only if every other method fails

- RM is optimal among static assignment methods
- EDF is optimal among dynamic methods

Optimality

Multiple ways assigning priorities to meet deadlines

Optimal: a method which fails only if every other method fails

- RM is optimal among static assignment methods
- EDF is optimal among dynamic methods

Schedulability

An optimal method may also fail
A set of task may not be schedulable at all

Example

The shortest path from A to B is 200km (the optimal scheduling). We have only one hour to reach the destination and the maximum speed is 120 km/h (deadline and platform constraints).
Can we be there on time (schedulability analysis)

Schedulability

An optimal method may also fail
A set of task may not be schedulable at all

Example

The shortest path from A to B is 200km (the optimal scheduling). We have only one hour to reach the destination and the maximum speed is 120 km/h (deadline and platform constraints).
Can we be there on time (schedulability analysis)

Schedulability

To determine whether task set is at all schedulable (with optimal methods)

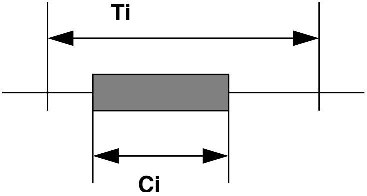
Schedulability must take the WCETs of tasks into account.

Schedulability

To determine whether task set is at all schedulable (with optimal methods)

Schedulability must take the WCETs of tasks into account.

Utilization-based analysis



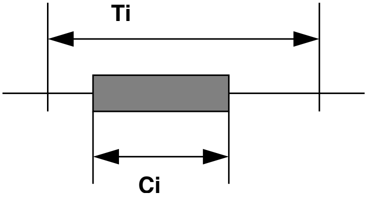
For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

That is, the CPU utilization of a periodic task i is the ratio $\frac{C_i}{T_i}$, where C_i is the WCET and T_i is the period.

Note

Any task for which $C_i = T_i$ will effectively need exclusive access to the CPU!

Utilization-based analysis



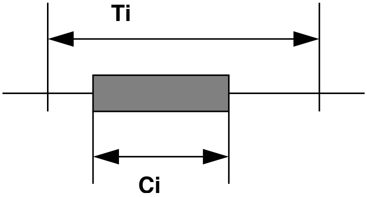
For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

That is, the CPU utilization of a periodic task i is the ratio $\frac{C_i}{T_i}$, where C_i is the WCET and T_i is the period.

Note

Any task for which $C_i = T_i$ will effectively need exclusive access to the CPU!

Utilization-based analysis



For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

That is, the CPU utilization of a periodic task i is the ratio $\frac{C_i}{T_i}$, where C_i is the WCET and T_i is the period.

Note

Any task for which $C_i = T_i$ will effectively need exclusive access to the CPU!

Utilization-based analysis (RM)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

where N is the number of threads.

That is, the sum of all CPU utilizations must be less than a certain bound that depends on N.

Utilization-based analysis (RM)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

where N is the number of threads.

That is, the sum of all CPU utilizations must be less than a certain bound that depends on N.

Utilization bounds

N	Utilization bound
1	100.0 %
2	82.8 %
3	78.0 %
4	75.7 %
5	74.3 %
10	71.8 %

Approaches 69.3% asymptotically

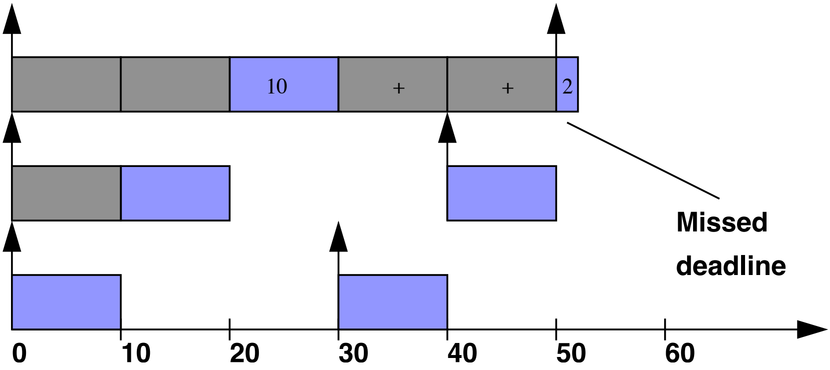
Example A

Task	Period	WCET	Utilization
i	T_i	C_i	U_i
1	50	12	24%
2	40	10	25%
3	30	10	33%

The combined utilization U is 82%, which is above the bound for 3 threads (78%).

The task set fails the utilization test.

Time-line for example A



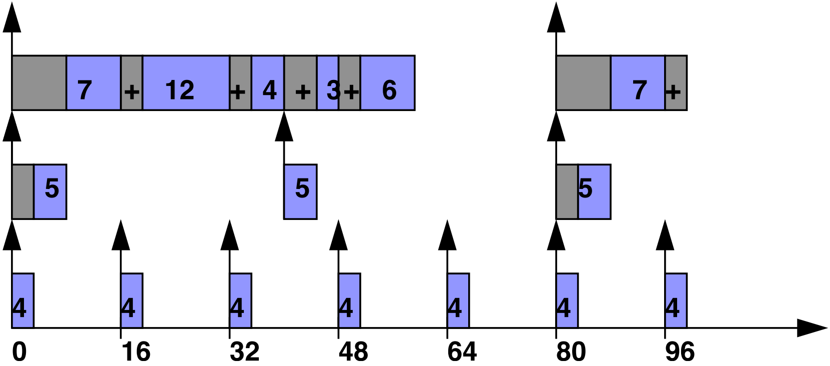
Example B

Task	Period	WCET	Utilization
i	T_i	C_i	U_i
1	80	32	40%
2	40	5	12.5%
3	16	4	25%

The combined utilization U is 77.5%, which is below the bound for 3 threads (78%).

The task set will meet all its deadlines!

Time-line for example B



Example C

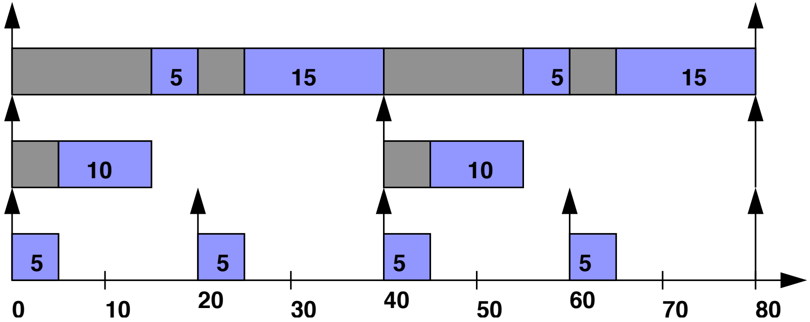
Task	Period	WCET	Utilization
i	T_i	C_i	U_i
1	80	40	50%
2	40	10	25%
3	20	5	25%

The combined utilization U is 100%, which is well above the bound for 3 threads (78%).

However, this task set still meets all its deadlines!

How can this be??

Time-line for example C



Characteristics

The utilization-based test

- Is sufficient (pass the test and you are OK)
- Is not necessary (fail, and you might still have a chance)

Why bother with such a test?

- Because it is so simple!
- Because only very specific sets of tasks fail the test and still meet their deadlines!

Characteristics

The utilization-based test

- Is sufficient (pass the test and you are OK)
- Is not necessary (fail, and you might still have a chance)

Why bother with such a test?

- Because it is so simple!
- Because only very specific sets of tasks fail the test and still meet their deadlines!

Utilization-based analysis (EDF)

For a set of periodic tasks, EDF scheduling succeeds if

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

Sum of CPU utilizations $\leq$ 100%, independent of the number of tasks.

Utilization-based test for EDF: both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)

Utilization-based analysis (EDF)

For a set of periodic tasks, EDF scheduling succeeds if

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

Sum of CPU utilizations $\leq$ 100%, independent of the number of tasks.

Utilization-based test for EDF: both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)

Utilization-based analysis (EDF)

For a set of periodic tasks, EDF scheduling succeeds if

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

Sum of CPU utilizations $\leq$ 100%, independent of the number of tasks.

Utilization-based test for EDF: both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Similarities

- Optimal within their class
- Easy to implement in terms of priority queues
- Utilization-based schedulability tests
- Extensible in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- superior CPU utilization

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

EDF vs RM

Drawbacks of EDF

- Exhibits random behaviour under transient overload (the same for RM, in a different way)
- Dynamic change of task priorities
- More elaborate utilization-based test when $D_i \leq T_i$ (but is still feasible)
- Meager OS and language support (priority scales lack granularity, no automatic time-stamping)

However, for reactive objects, EDF fits nice as an alternative to RM

Loosening the assumptions

$T_i \neq D_i$

Deadlines less than periods: infrequent, urgent tasks

Sporadic Tasks

Sporadic tasks: no fixed period (interrupt handlers), urgent deadlines

Deadline Monotonic

Basic Principle

$C_i < D_i < T_i$
Lower deadline values get higher priority: a priority assignment is valid when $P_i < P_j$ iff $D_i < D_j$.

Naive Schedulability Analysis

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{D_i} \leq N(2^{1/N} - 1)$$

More Precise Schedulability Analysis

Pre-Processing

Sort the tasks by increasing order of deadlines:

$$i < j \text{ iff } D_i < D_j$$

Schedulability Analysis

For each and every $i \leq n$:

$$C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_j}{T_j} \right\rceil C_j \leq D_i$$

Loosening the assumptions

Sporadic Tasks

Sporadic tasks: no fixed period (interrupt handlers), urgent deadlines

Characteristics needed for schedulability analysis

Characteristics

Minimum inter-arrival time: minimum time between two events causing sporadic tasks (e.g., key strokes, signal updates)

Period T interpreted as inter-arrival time

For sporadic tasks: $D < T$

Loosening the assumptions

Sporadic Tasks

Sporadic tasks: no fixed period (interrupt handlers), urgent deadlines
Characteristics needed for schedulability analysis

Characteristics

Minimum inter-arrival time: minimum time between two events causing sporadic tasks (e.g., key strokes, signal updates)
Period T interpreted as inter-arrival time
For sporadic tasks: $D < T$

Scheduling Sporadic Tasks

Polling Servers

A task with period T_s
Fixed capacity C_s

Scheduling

Sporadic events scheduled in the server when there is capacity left
Capacity is replenished every T units

Scheduling Sporadic Tasks

Polling Servers

A task with period T_s
Fixed capacity C_s

Scheduling

Sporadic events scheduled in the server when there is capacity left
Capacity is replenished every T units

Polling Servers

Schedulability Analysis

$$U \equiv \frac{C_s}{T_s} + \sum_{i=1}^{N} \frac{C_i}{T_i} \leq (N+1)(2^{1/(N+1)} - 1)$$

More on real-time

Other analysis

Response-time analysis: more powerful technique than utilization based

More on this in specialized courses on real-time (such as distributed real time systems)

More on real-time

Other analysis

Response-time analysis: more powerful technique than utilization based

More on this in specialized courses on real-time (such as distributed real time systems)