Embedded Systems Programming

Lecture 9

www2.hh.se/staff/vero/embeddedProgramming

Verónica Gaspes
www2.hh.se/staff/vero

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

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.

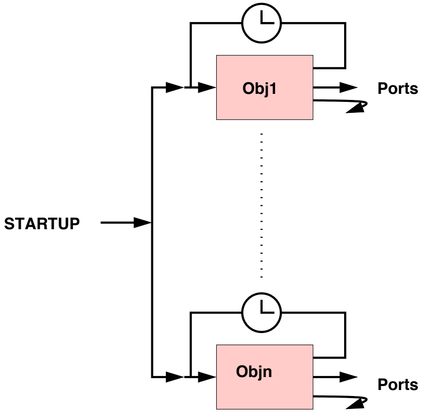
:-(

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.

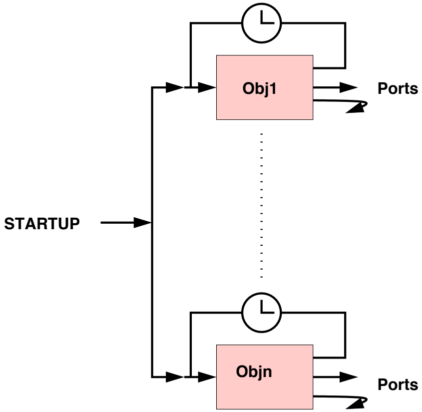
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

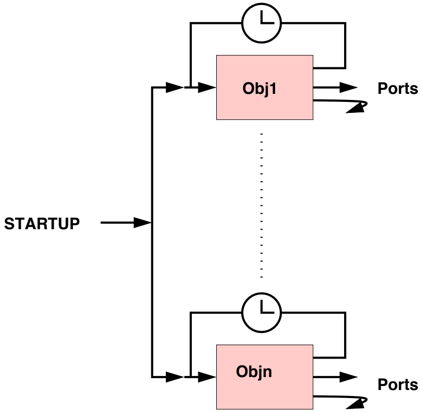
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

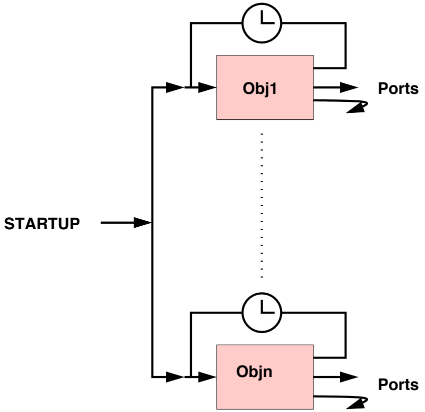
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

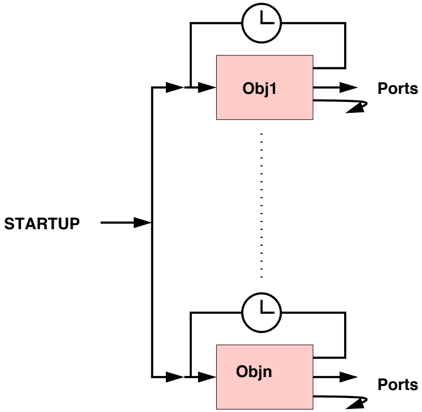
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

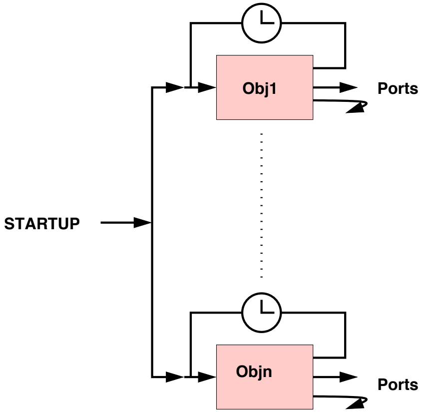
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

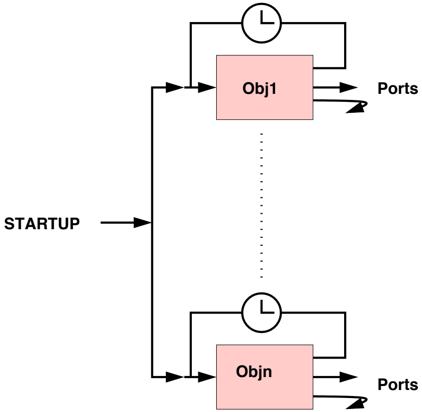
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

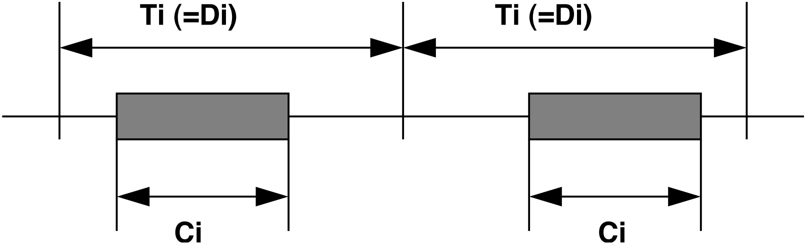
Initial restricted model



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

More advanced courses on real-time systems discuss how to remove some of these restrictions.

Notation



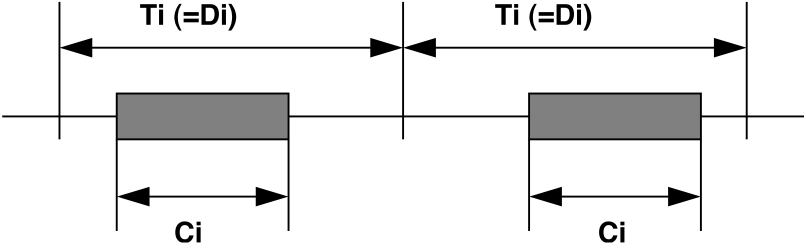
Each reactive object obj_i executes a message (thread/task/job) m_i in a periodic fashion.

For each message m_i

- We know its period T_i (given, determines the AFTER offset)
- We know its WCET C_i (meassured or analyzed)
- We know its relative deadline D_i (given, equal to T_i for now)

We want to determine its priority P_i!

Notation



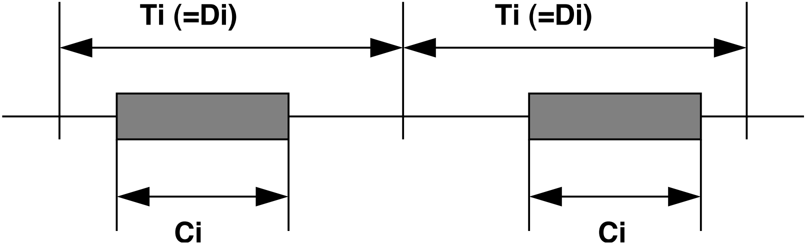
Each reactive object obj_i executes a message (thread/task/job) m_i in a periodic fashion.

For each message m_i

- We know its period T_i (given, determines the AFTER offset)
- We know its WCET C_i (meassured or analyzed)
- We know its relative deadline D_i (given, equal to T_i for now)

We want to determine its priority P_i!

Notation



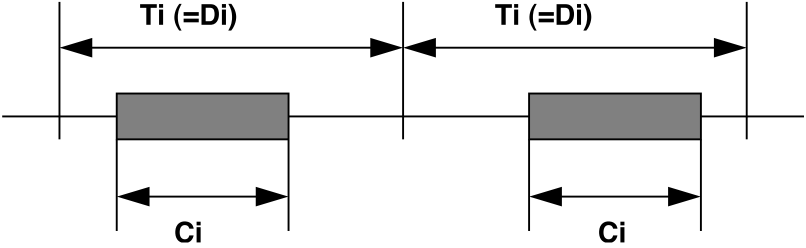
Each reactive object obj_i executes a message (thread/task/job) m_i in a periodic fashion.

For each message m_i

- We know its period T_i (given, determines the `AFTER` offset)
- We know its WCET C_i (meassured or analyzed)
- We know its relative deadline D_i (given, equal to T_i for now)

We want to determine its priority P_i!

Notation



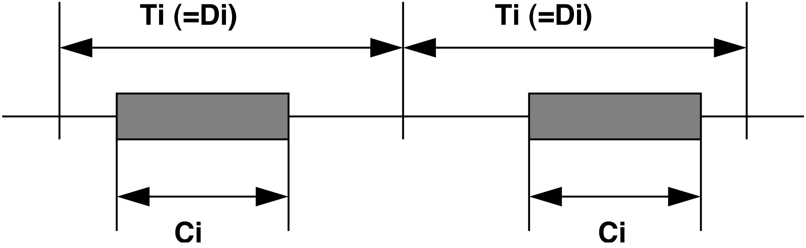
Each reactive object obj_i executes a message (thread/task/job) m_i in a periodic fashion.

For each message m_i

- We know its period T_i (given, determines the `AFTER` offset)
- We know its WCET C_i (meassured or analyzed)
- We know its relative deadline D_i (given, equal to T_i for now)

We want to determine its priority P_i!

Notation



Each reactive object obj_i executes a message (thread/task/job) m_i in a periodic fashion.

For each message m_i

- We know its period T_i (given, determines the `AFTER` offset)
- We know its WCET C_i (meassured or analyzed)
- We know its relative deadline D_i (given, equal to T_i for now)

We want to determine its priority P_i!

In concrete code

The application

```
void ignite(){
  BEFORE(D_1, &obj_1, m_1, arg_1);
  BEFORE(D_2, &obj_2, m_2, arg_2);
     ⋮
  BEFORE(D_n, &obj_n, m_n, arg_n);
}


STARTUP(ignite());
```

In concrete code

The objects

```
Class_i obj_i = initClass_i();


int m_i(Class_i *self, int arg){
  // read ports
  // compute
  // update self state
  // write ports
  WITHIN(T_i, D_i, self, m_i,arg);
}
```

Each $D_i = T_i$

In concrete code

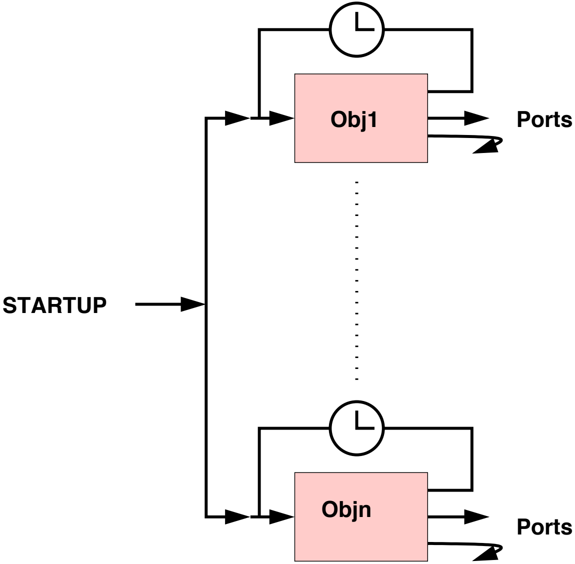
The objects

```
Classi obji = initClassi();


int mi(Classi *self, int arg){
  // read ports
  // compute
  // update self state
  // write ports
  WITHIN(Ti, Di, self, mi,arg);
}
```

Each $D_i = T_i$

Schematically (again)



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

$P1 = 10$	$P1 = 1$	$P1 = 25$
$P2 = 19$	$P2 = 3$	$P2 = 60$
$P3 = 12$	$P3 = 2$	$P2 = 45$

RM example

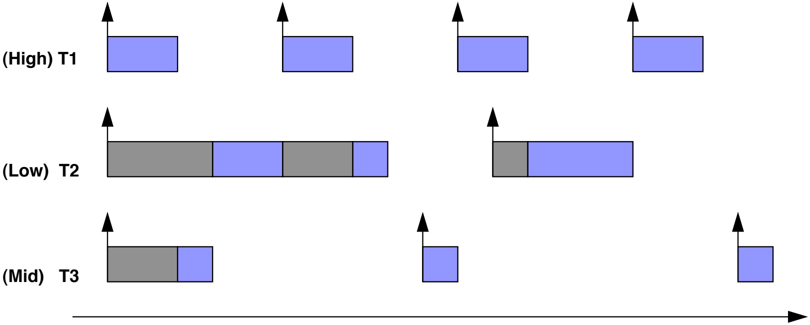
Given a set of periodic tasks with periods

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

Valid priority assignments

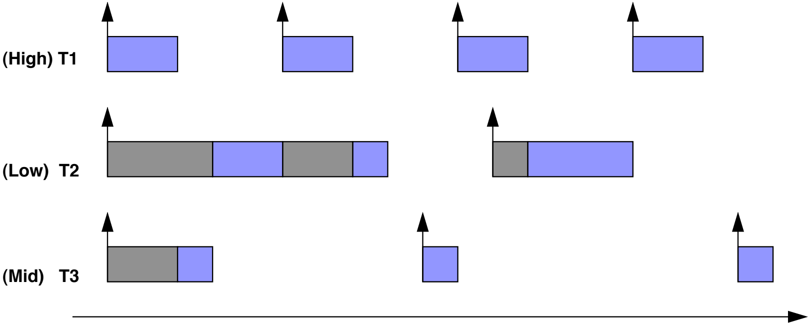
$$\begin{aligned} P1 &= 10 \\ P2 &= 19 \\ P3 &= 12 \end{aligned} \qquad \begin{aligned} P1 &= 1 \\ P2 &= 3 \\ P3 &= 2 \end{aligned} \qquad \begin{aligned} P1 &= 25 \\ P2 &= 60 \\ P2 &= 45 \end{aligned}$$

RM example



For each task, the period is equal to its deadline. Arrows mark start of period. Blue when they get to execute. Gray when they have to wait for higher prio tasks to complete.

RM example



For each task, the period is equal to its deadline. Arrows mark start of period. Blue when they get to execute. Gray when they have to wait for higher prio tasks to complete.

Dynamic priorities – method

Earliest Deadline First – EDF

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

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

Because EDF will want to distinguish between messsages on basis of their absolute deadlines, priority values must use the same units as the system clock.

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

Dynamic priorities – method

Earliest Deadline First – EDF

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

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

Because EDF will want to distinguish between messsages on basis of their absolute deadlines, priority values must use the same units as the system clock.

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

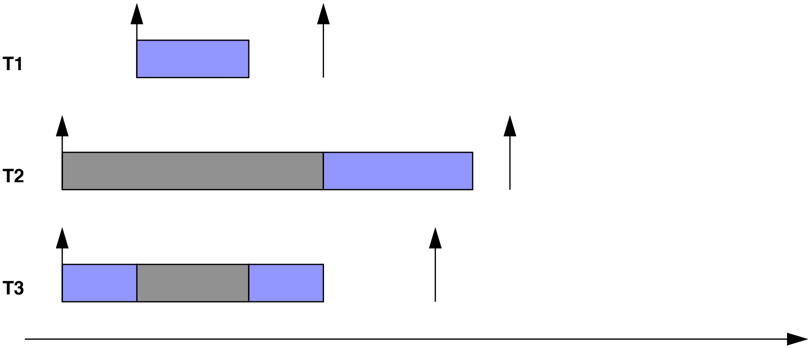
Under the given assumptions, there exists a dynamic priority assignment rule that is really simple:

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

Because EDF will want to distinguish between messsages on basis of their absolute deadlines, priority values must use the same units as the system clock.

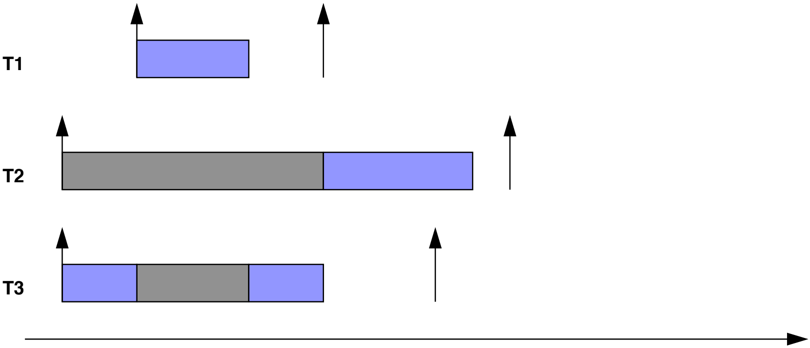
Under EDF, each activation n of periodic task i will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$

EDF example



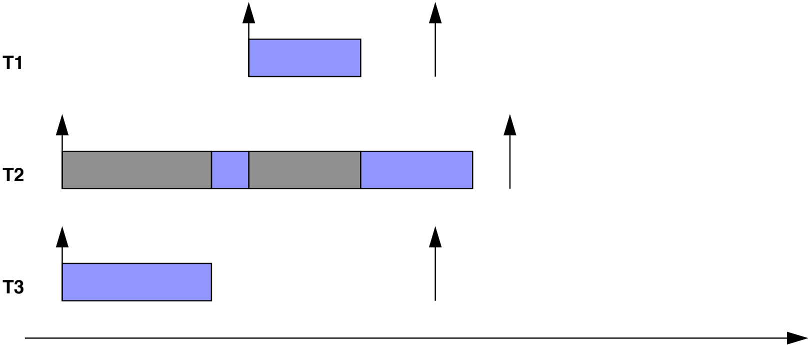
T1 arrives later, but its deadline is earlier than both T2's and T3's absolute deadlines!

EDF example



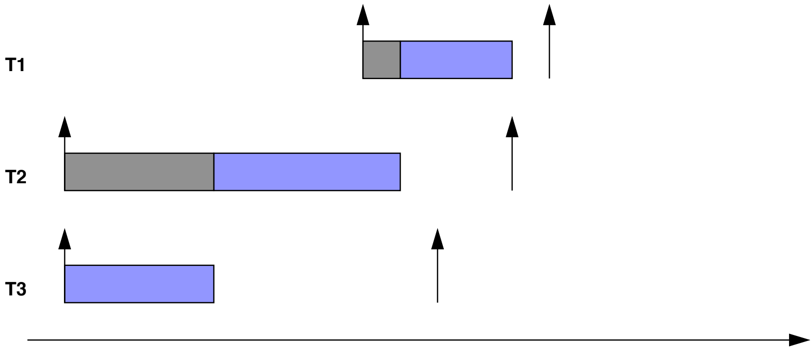
T1 arrives later, but its deadline is earlier than both T2's and T3's absolute deadlines!

EDF example



Deadline of T1 < Deadline of T2

EDF example



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

Optimality

Under some given assumptions, there might be several ways of assigning priorities so that deadlines are met.

Clearly, a method that only fails if every other method also fails is preferred — such a method is called optimal

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

Optimality

Under some given assumptions, there might be several ways of assigning priorities so that deadlines are met.

Clearly, a method that only fails if every other method also fails is preferred — such a method is called optimal

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

Optimality

Under some given assumptions, there might be several ways of assigning priorities so that deadlines are met.

Clearly, a method that only fails if every other method also fails is preferred — such a method is called optimal

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

Schedulability

However, knowing that a priority assignment is the best one possible is not the same thing as knowing that it is good enough, i.e. knowing that deadlines actually will be met!

Assume all we know is that our priority assignment method is optimal. This is like knowing where the shortest path from A to B lies, but still not knowing if the path is short enough so that B can be reached in time

Schedulability

However, knowing that a priority assignment is the best one possible is not the same thing as knowing that it is good enough, i.e. knowing that deadlines actually will be met!

Assume all we know is that our priority assignment method is optimal. This is like knowing where the shortest path from A to B lies, but still not knowing if the path is short enough so that B can be reached in time

Schedulability

To answer whether our tasks will actually meet their deadlines at run-time, we need to determine if our task set is at all schedulable (recall that an optimal priority assignment method will produce a successful schedule if such a schedule exists)

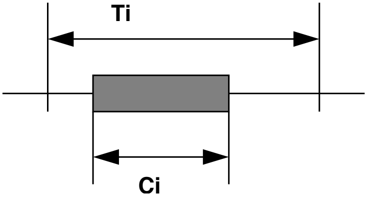
Clearly, the question of schedulability must take the WCETs of tasks into account!

Schedulability

To answer whether our tasks will actually meet their deadlines at run-time, we need to determine if our task set is at all schedulable (recall that an optimal priority assignment method will produce a successful schedule if such a schedule exists)

Clearly, the question of schedulability must take the WCETs of tasks into account!

Utilization-based analysis



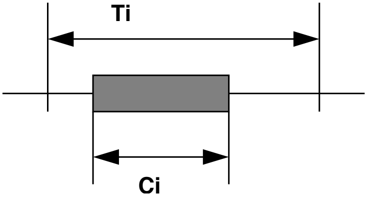
For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

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



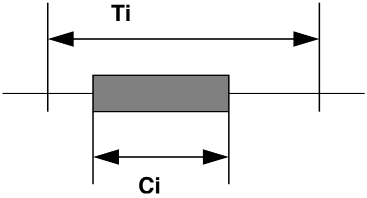
For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

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

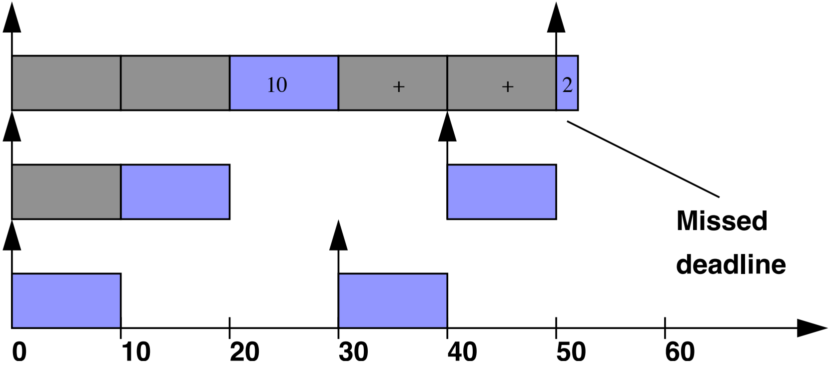
Example A

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



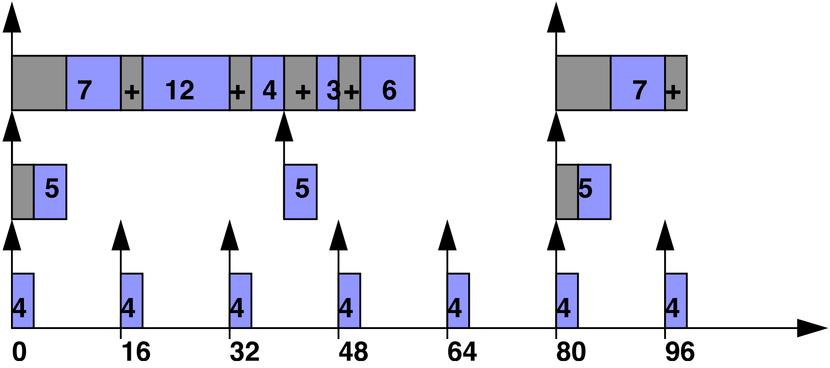
Example B

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

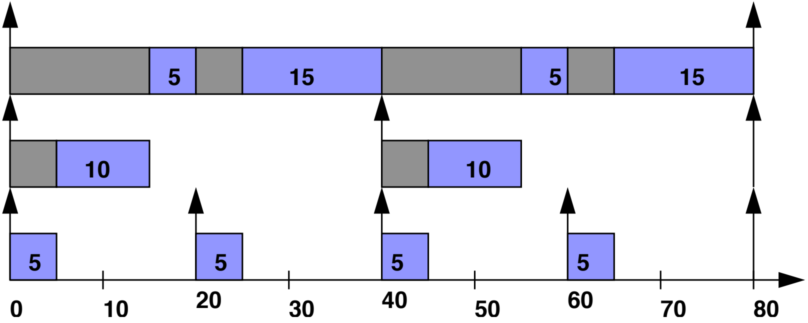
Task	Period	WCET	Utilization
i	T_i	C_i	U_i
1	80	40	50%
2	40	10	25%
3	20	5	25%

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)

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

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

That is, the sum of all CPU utilizations must be less than or equal 100%, independent of the number of tasks.

Unlike the case for RM, the utilization-based test for EDF is both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)

Utilization-based analysis (EDF)

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

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

That is, the sum of all CPU utilizations must be less than or equal 100%, independent of the number of tasks.

Unlike the case for RM, the utilization-based test for EDF is both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)

Utilization-based analysis (EDF)

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

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

That is, the sum of all CPU utilizations must be less than or equal 100%, independent of the number of tasks.

Unlike the case for RM, the utilization-based test for EDF is both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)

EDF vs RM

Similarities

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended 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

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task priorities instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don't support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

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

Implementation (RM)

In `TinyTimber.c`

```
struct msg_block{
  ...
  Time baseline;
  Time priority;
  ...
};

void async(Time offset, Time prio,
           OBJECT *to, METHOD meth, int arg){
   ...
   m->baseline=MAX(TIMERGET(),
                   current->baseline+offset);
   m->priority = prio;
   ...
}
```

Implementation (EDF)

In `TinyTimber.c`

```
struct msg_block{
  ...
  Time baseline;
  Time deadline;
  ...
};

void async(Time BL, Time DL,
           OBJECT *to, METHOD meth, int arg){
   ...
   m->baseline=MAX(TIMERGET(),
                   current->baseline+BL);
   m->deadline = m->baseline+DL;
   ...
}
```

More on real-time

Loosening the restrictions

What can be said if we allow tasks/threads/messages that are not periodic or for which deadline is not equal to period or that can block?

Other analysis

Response-time analysis, a more powerful technique than utilization based, is needed.

We leave this for more specialized courses on real-time (such as distributed real time systems)

More on real-time

Loosening the restrictions

What can be said if we allow tasks/threads/messages that are not periodic or for which deadline is not equal to period or that can block?

Other analysis

Response-time analysis, a more powerful technique than utilization based, is needed.

We leave this for more specialized courses on real-time (such as distributed real time systems)

More on real-time

Loosening the restrictions

What can be said if we allow tasks/threads/messages that are not periodic or for which deadline is not equal to period or that can block?

Other analysis

Response-time analysis, a more powerful technique than utilization based, is needed.

We leave this for more specialized courses on real-time (such as distributed real time systems)