IN4343 Real-Time Systems, Lecture 5

Scheduling of Real-Time Systems

Contact

m.nasri@tudelft.nl
(appointment by email)
Online scheduling of periodic tasks

Buttazzo’s book, chapter 4

Disclaimer: A few slides have been taken from Giorgio Buttazzo’s website: http://retis.sssup.it/~giorgio/rts-MECS.html
Agenda

• Scheduling algorithms for periodic real-time systems (chapter 4):
  • Processor sharing
  • Cyclic scheduling
  Quiz
• Fixed-priority scheduling
  • Rate monotonic priority assignment method (RM)
  • Deadline monotonic priority assignment method (DM)
• Dynamic-priority scheduling (will be discussed in the next lecture)
  • EDF

• RM schedulability tests
  • Necessary v.s. sufficient tests
  • Liu and Layland’s test [1973]
  • Hyperbolic bound [2000]

If you want to pass the course, learn this lecture very well
Definitions

We consider a computing system that has to execute a set $\tau$ of $n$ periodic real-time tasks:

$$\tau = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$$

$T_i = \text{period}$
$C_i = \text{worst-case execution time (WCET)}$
$D_i = \text{relative deadline}$
$\phi_i = \text{offset (or phase)}$
$U_i = \frac{C_i}{T_i} = \text{utilization}$

What are $T_1$ and $T_2$?
What are $D_1$ and $D_2$?
What are $\phi_1$ and $\phi_2$?
Assumptions

• For this lecture, we assume
  
  **A1.** $C_i$ and $T_i$ are constant for every job of $\tau_i$
  
  **A2.** Tasks are fully preemptive
  
  **A3.** Context switch, preemption, and scheduling overheads are zero
  
  **A4.** Tasks are independent:
  
  • no precedence relations
  • no resource constraints
  • no blocking on I/O operations
  • no self suspension
Formulating the arrival time and absolute deadline

What is the arrival time of the third job ($J_{i,3}$) of the following task?

$$
\tau_i = (C_i = 3, T_i = 10, D_i = 7, \phi_i = 2)
$$

$$
a_{i,3} = 22
$$

Build a formula for the arrival time and absolute deadline of the $k^{th}$ job of $\tau_i = (C_i, T_i, D_i, \phi_i)$, i.e., $J_{i,k}$:

$$
a_{i,k} = \phi_i + (k - 1) \cdot T_i
$$

$$
d_{i,k} = \phi_i + (k - 1) \cdot T_i + D_i
$$

What about $U_{i,k}$?

Utilization is defined for the task not for the job: $U_i = \frac{C_i}{T_i}$

Note: in Giorgio’s book, $\tau_{ik}$ is used instead of $J_{i,k}$. They both refer to the $k^{th}$ job of task $\tau_i$. 
Feasibility of a periodic task set

A periodic task set $\tau$ is feasible if and only if there exists a schedule in which each task $\tau_i \in \tau$ can execute for $C_i$ units of time within every interval $[a_{i,k}, d_{i,k})$ for all $k \in \mathbb{N}$. 
Scheduling solutions for periodic tasks

1. Proportional share algorithm
2. Cyclic scheduling (a.k.a. timeline scheduling, table-driven scheduling)
3. Online scheduling policies
   • Fixed-priority policy
     • Rate monotonic (RM)
     • Deadline monotonic (DM)
   • Dynamic policy
     • EDF
Proportional sharing

Feed cow for 20 min / 40 min

$C_{cow}/T_{cow}$

Feed pig for 4 min / 16 min

$C_{pig}/T_{pig} = 0.25$
Proportional share algorithm

• Basic idea
  • Divide the timeline into **slots** of equal length.
  • Within each slot **serve each task for a time proportional to its utilization:**

  \[
  \text{Slot length: } \Delta = \gcd(T_i, \forall i)
  \]

  \[
  \text{Task share in each slot: } \delta_i = U_i \cdot \Delta
  \]

If a task set is feasible, then proportional sharing can also schedule it as long as there is no context switch overhead and \( D = T \).
Downside of proportional share algorithm

• if periods are not harmonic, then
  $\Delta = \text{GCD}(T_1, T_2, ..., T_n)$
  is small and a task is fragmented into many chunks
  $\left(\frac{T_i}{\Delta}\right)$ of small duration
  $\delta_i = U_i \Delta$

Can lead to a huge number of context switches

Not practical
Cyclic scheduling

• Also known as timeline scheduling, it has been used for 30 years in military systems, navigation, and monitoring systems.

• Examples
  – Air traffic control systems
  – Space Shuttle
  – Boeing 777
  – Airbus navigation system

An extension of table-driven scheduling
Cyclic scheduling

- **Method**
  - The time axis is divided in intervals of equal length (*time slots*).
  - Each task is statically allocated in a slot in order to meet the desired request rate.
  - The execution in each slot is activated by a timer.

**Example:**

<table>
<thead>
<tr>
<th>task</th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10 ms</td>
<td>25 ms</td>
</tr>
<tr>
<td>B</td>
<td>10 ms</td>
<td>50 ms</td>
</tr>
<tr>
<td>C</td>
<td>10 ms</td>
<td>100 ms</td>
</tr>
</tbody>
</table>

**Step 1:** obtain the length of minor and major cycles:

- $\Delta = \text{GCD}(T_i, \forall i)$
- $T = \text{LCM}(T_i, \forall i)$

**Step 2:** assign tasks to time slots:

Ensure:

- $C_A + C_B \leq \Delta$
- $C_A + C_C \leq \Delta$
Cyclic scheduling

implementation

coding

What type of memory is used to store this schedule?

Notes:

• Assigning tasks to slots is not easy and the designer must come up with such solution.
• Cyclic scheduling does not tell you how to assign tasks to slots.
• There is no automatic support for preemption! You need to cut your code into two parts by yourself, e.g., function_A_part1(); and function_A_part2();

FLASH
Cyclic scheduling

Problems of cyclic scheduling during overload

• What to do during task overruns?
  • **Let the task continue**
    • we can have a *domino effect* on all the other tasks (timeline break)
  • **Abort the task**
    • the system can remain in inconsistent states.

Summarizing cyclic scheduling:

**Advantages**

• Simple implementation (no RTOS is required).
• Low run-time overhead.
• All tasks run with very low jitter.

**Disadvantages**

• It is not robust during overloads.
• It is difficult to expand the schedule.
• It is difficult to update the code since it may totally change the schedule
• It is not easy to handle aperiodic activities.
Step 1: open www.kahoot.it in your browser (phone or laptop)
Step 2: enter the pin code and then a nickname
What defines an optimal scheduling policy (in the sense of feasibility)?

• It can generate a feasible schedule even for infeasible task sets (seriously?!

• It always generates a feasible schedule

• It generates a feasible schedule for a feasible task set
Which policy is optimal for preemptive task sets?

- Fixed-priority scheduling
- FIFO
- EDF
- Non-preemptive EDF
Which policy is optimal for non-preemptive task sets?

- Non-preemptive EDF
- FIFO
- Non-preemptive fixed-priority
- None of these
What is the utilization of this task?

\[ \tau_1: (C_1 = 5, \quad T_1 = 20, \quad D_1 = 10, \quad \phi_1 = 2) \]
What is the **slot length** in the proportional sharing for the following task set?

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$\phi_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

10
20
40
5
Which task will have the largest execution time in each slot in the proportional sharing scheme?

<table>
<thead>
<tr>
<th>( \tau_i )</th>
<th>( C_i )</th>
<th>( T_i )</th>
<th>( D_i )</th>
<th>( \phi_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>15</td>
<td>80</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Task 1
Task 2
Task 3
They will have an equal share
Agenda

• Scheduling algorithms for periodic real-time systems (chapter 4):
  • Processor sharing
  • Cyclic scheduling
  Quiz
  • Rate monotonic (RM)
  • Deadline monotonic (DM)
  • (EDF will be discussed in the next lecture)

• RM schedulability tests
  • Necessary v.s. sufficient tests
  • Liu and Layland’s test [1973]
  • Hyperbolic bound [2000]

If you want to pass the course, learn this lecture very well
Fixed-priority scheduling

Method

1. Assign priorities to each task based on its timing constraints.
   1. Rate monotonic
   2. Deadline monotonic

2. Verify the feasibility of the schedule using analytical techniques.

3. Execute tasks on a priority-based kernel.
Priority assignment for fixed-priority scheduling

**Rate monotonic**

- Assign priorities monotonically with the activation frequency, a.k.a., rate ($\sim 1 / T$) such that a task with a smaller period gets a higher priority.

- Example: $T = 10 \Rightarrow rate = \frac{1}{10}$

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0.2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>10</td>
<td>40</td>
<td>15</td>
<td>0.25</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Priority ordering? $P_1 < P_3 < P_2$
Priority assignment for fixed-priority scheduling

Deadline monotonic

- Assign priorities monotonically with the relative deadline of the task, \( \sim 1 / D \) such that a task with a smaller relative deadline gets a higher priority.

<table>
<thead>
<tr>
<th>( \tau_i )</th>
<th>( C_i )</th>
<th>( T_i )</th>
<th>( D_i )</th>
<th>( U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0.2</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>10</td>
<td>40</td>
<td>15</td>
<td>0.25</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Priority ordering?  
\( P1 < P2 < P3 \)
Is rate-monotonic an optimal scheduling policy (in the sense of feasibility)?
Is rate-monotonic an optimal scheduling policy (in the sense of feasibility)?

Is this task set feasible? \((\forall i, D_i = T_i)\) Yes!

Is this task set schedulable by EDF? Yes! (the schedule shown here is EDF)

Is this task set schedulable by RM?

\[
U = \sum_{i=1}^{n} U_i = 0.5 + 0.44 = 0.94
\]
Is rate-monotonic an optimal scheduling policy (in the sense of feasibility)?

Is this task set feasible? \((\forall i, D_i = T_i)\)  Yes!

Is this task set schedulable by EDF?  Yes! (the schedule shown here is EDF)

Is this task set schedulable by RM?  No!

\[ U = \sum_{i=1}^{n} U_i = 0.5 + 0.44 = 0.94 \]
Fixed-priority scheduling with RM priorities is not an optimal scheduling policy (in the sense of feasibility).

However, if $\forall i, D_i = T_i$ then the rate monotonic priority assignment is an optimal priority assignment among all other priority assignment methods.
Rate Monotonic is optimal

RM is optimal among all fixed priority algorithms (if $D_i = T_i$):

If there exists a fixed priority assignment which leads to a feasible schedule for $\Gamma$, then the RM assignment is feasible for $\Gamma$.

If $\Gamma$ is not schedulable by RM, then it cannot be scheduled by any fixed priority assignment.
Deadline Monotonic is optimal

If $D_i \leq T_i$ then the optimal priority assignment is given by **Deadline Monotonic (DM)**:

**DM**

$P2 < P1$

$\tau_1$

$\tau_2$

**RM**

$P1 < P2$

$\tau_1$

$\tau_2$

Deadline miss
EDF Optimality

EDF is optimal among all algorithms:

If there exists a feasible schedule for \( \Gamma \), then EDF will generate a feasible schedule.

If \( \Gamma \) is not schedulable by EDF, then it cannot be scheduled by any algorithm.
Optimality
Agenda

- Scheduling algorithms for periodic real-time systems (chapter 4):
  - Processor sharing
  - Cyclic scheduling
  - Quiz
  - Rate monotonic (RM)
  - Deadline monotonic (DM)
  - (EDF will be discussed in the next lecture)

- **RM schedulability tests**
  - Necessary v.s. sufficient tests
  - Liu and Layland’s test [1973]
  - Hyperbolic bound [2000]

If you want to pass the course, learn this lecture very well
Necessary and sufficient tests

**Necessary test (checks the feasibility):**
If the condition in the test **is NOT satisfied**, then the task set is **not feasible** (it is impossible to find any feasible schedule for the task set)

**Sufficient schedulability test for algorithm A:**
If the condition in the test **is satisfied**, then the task set is **certainly schedulable** by the given scheduling algorithm
Utilization-based schedulability tests

A **utilization-based schedulability test** for an **algorithm A** is a test that checks whether or not a **specific relation** $f_A$ holds between the **utilization values** of the tasks in the task set.

If that relation is satisfied, then the task set is schedulable by the given scheduling algorithm.

**Example of a utilization-based test for EDF:**

$$\sum_{i=1}^{n} U_i \leq 1$$

This test is both necessary and sufficient for EDF algorithm (proof in the next lecture).
Why utilization?

• Each task uses the processor for a fraction of time:
  \[ U_i = \frac{C_i}{T_i} \]

• Hence the total processor utilization is:
  \[ U = \sum_{i=1}^{n} \frac{C_i}{T_i} \]

• \( U \) is a measure of the processor load

It is easy to use in the early stages of system design.
First bound: can we find a feasible schedule for a task set with $U > 1$?

A: yes
B: depends on the scheduling policy
C: depends on the task set
D: no

If $U > 1$, then the amount of work need to be done per unit of time ($= U$) is larger than the unit of time itself!
A necessary condition

A necessary condition for having a feasible schedule is that $U \leq 1$.

Is that enough to ensure RM schedulability? No
A necessary condition for having a feasible schedule is that $U \leq 1$.

Is that enough to ensure RM schedulability?

No

$U = \sum_{i=1}^{n} U_i = 0.5 + 0.44 = 0.94$
Let’s have a look at RM scheduling

$U = 0.94$ (smaller than 1) but not schedulable!
Let’s have a look at RM scheduling

\[ U = 0.94 \text{ (smaller than 1) but not schedulable!} \]

\[ U = 1, \text{ but it is schedulable by RM!} \]
So, how to design a utilization-based schedulability test for RM?

Try to find a utilization threshold (lower bound) such that ANY task set with utilization lower than that bound is CERTAINLY schedulable.
The simplest utilization-based test

For a given task set, check whether or not

$$U \leq U_{lb}$$

where $U_{lb}$ is the largest utilization such that any task set with $U \leq U_{lb}$ is always schedulable by RM.
The simplest utilization-based test

For a given task set, check whether or not
\[ U \leq U_{lb} \]

where \( U_{lb} \) is the largest utilization such that any task set with \( U \leq U_{lb} \) is always schedulable by RM.
Liu and Layland test for RM

Liu and Layland [1973] derived the largest value for $U_{lb}$ for the rate monotonic scheduling under certain assumptions:

**A1.** $C_i$ and $T_i$ are constant for every job of $\tau_i$

**A2.** For each task, $T_i = D_i$

**A3.** Tasks are fully preemptive

**A4.** Context switch, preemption, and scheduling overheads are zero

**A5.** Tasks are independent:
  - no precedence relations
  - no resource constraints
  - no blocking on I/O operations
  - no self suspension

Hint for the exam: remember the assumptions
Liu and Layland’s test for RM

\[ U \leq n \cdot (2^{1/n} - 1) \]

- \( n \to \infty \)
  - \( U \to ? \)
- \( n \to 2 \)
  - \( U \to ? \)
- \( n \to \infty \)
  - \( U \to \ln 2 \sim 0.691 \)
- \( n \to 2 \)
  - \( U \to 0.83 \)
Liu and Layland’s test accepts more task sets when $n$ is small
How happy are we with the L&L RM test?

$U \leq 1$ is necessary and sufficient for EDF schedulability

$U = \sum_{i=1}^{2} U_i = U_1 + U_2$
How happy are we with the L&L RM test?

The test rejects any task set with \( U > 0.83 \)

L&L Test for RM: \[ \sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \]

EDF bound

L&L bound (RM)

\[ U = \sum_{i=1}^{2} U_i = U_1 + U_2 \]
Hyperbolic bound

\[ \sum_{i=1}^{n} U_i \leq 1 \]

\[ \sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \]

\[ \prod_{i=1}^{n} (U_i + 1) \leq 2 \]
The Hyperbolic Bound

- In 2000, Bini et al. proved that a set of $n$ periodic tasks is schedulable with RM if:

$$\prod_{i=1}^{n} (U_i + 1) \leq 2$$
Example

Is the task set feasible?
Yes \( U \leq 1 \)

Does the task set pass the Liu & Layland’s test?
No because \( U = 0.85 > 2(2^{1/2} - 1) \approx 0.83 \)

Does the task set pass the hyperbolic-bound test?
Yes because \((0.8 + 1) \times (0.05 + 1) = 1.89 < 2\)

Is the task set schedulable by RM?
Yes!

\[
\sum_{i=1}^{n} U_i \leq 1 \quad \text{necessary} \\
\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \quad \text{Liu and Layland test} \\
\prod_{i=1}^{n} (U_i + 1) \leq 2 \quad \text{Hyperbolic bound}
\]
Example

Is the task set feasible?
Yes $U \leq 1$

Does the task set pass the Liu & Layland’s test?
No because $U > 2(2^{1/2} - 1) \approx 0.83$

Does the task set pass the hyperbolic-bound test?
No because $(0.4 + 1) \times (0.6 + 1) = 2.24 > 2$

Is the task set schedulable by RM (according to your brain)?
Yes! (hint: the task set is harmonic)

\[
\sum_{i=1}^{n} U_i \leq 1 \\
\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \\
\prod_{i=1}^{n} (U_i + 1) \leq 2
\]

necessary
Liu and Layland test
Hyperbolic bound
Hyperbolic bound is tight

It means that it is impossible to invent a new utilization-based test $A$ such that $A$ accepts a task set that the hyperbolic bound rejects!

In other words, as long as you only consider task utilizations into account, it is impossible to invent a test that is better than the hyperbolic bound!
Summary

Scheduling algorithms for periodic real-time systems (chapter 4):

- Processor sharing
- Cyclic scheduling
- Rate monotonic (RM)
- Deadline monotonic (DM)
- EDF

RM schedulability tests

- Necessary v.s. sufficient tests

- Liu and Layland’s test [1973]
  - It is a sufficient utilization-based test for RM

- Hyperbolic bound [2000]
  - It is a sufficient utilization-based test for RM
  - It is the tightest test among all utilization-based tests for RM
Disclaimer

• Thanks to
  • Giorgio Buttazzo
  • Koen Langendoen