Lecture 13

Real-Time Scheduling

Daniel Kästner
AbsInt GmbH
2013
Model-based Software Development

SCADE Suite

Application Model in SCADE (data flow + SSM)

Generator

Astrée

System Model (tasks, interrupts, buses, …)

SymTA/S

System-level Schedulability Analysis

C-Code

Compiler

Runtime Error Analysis

SCADE Suite

Generator

Astrée

System Model (tasks, interrupts, buses, …)

SymTA/S

System-level Schedulability Analysis

C-Code

Compiler

Runtime Error Analysis

Worst-Case Execution Time Analysis

Stack Usage Analysis
Setting the scene

- Hard real-time systems can be designed as a set of cooperating sequential processes (tasks).

Questions:
- In which order to execute tasks?
- How to deal with shared resources?
- How to guarantee timely execution?
The Endless Loop

Do forever
  request input device;
  fetch input value;
  do computation;
  request output device;
  write output;
End
The Basic Cyclic Executive

- Let three procedures A, B, and C be given.

```
Do forever
  call A;
  call B;
  call C;
End
```
The Time-Driven Cyclic Executive

- Let three procedures A, B, and C be given.

```
Do forever
    wait for timer interrupt;
    call A;
    call B;
    call C;
End
```

- The rate of hardware timer interrupts is the rate at which the procedures (tasks) must execute.
Multi-Rate Cyclic Executive

- Let the following task system be given:

  \[
  \text{Do forever} \quad \text{// The major cycle} \\
  \quad \text{wait for timer interrupt;} \quad \text{//1st minor cycle} \\
  \quad \text{A; B; C;} \\
  \quad \text{wait for timer interrupt;} \quad \text{//2nd minor cycle} \\
  \quad \text{A; B; D; E;} \\
  \quad \text{wait for timer interrupt;} \quad \text{//3rd minor cycle} \\
  \quad \text{A; B; C;} \\
  \quad \text{wait for timer interrupt;} \quad \text{//4th minor cycle} \\
  \quad \text{A; B; D;} \\
  \text{End}
  \]

- Procedures are mapped onto a set of minor cycles that together constitute the complete schedule (or major cycle).

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>60</td>
<td>1</td>
</tr>
</tbody>
</table>
The Cyclic Executive

- **Naive, but common** way to implement concurrent hard real-time systems.
- **No actual processes** exist at run-time; each minor cycle is just a sequence of procedure calls.
- Procedures share a **common address space** and can thus pass data between themselves. Concurrent access is not possible, thus no protection (e.g. semaphores) required.
- All process periods must be a **multiple of the minor cycle time**.
The Cyclic Executive

- Simple process modell:
  - Application consists of fixed set of processes
  - All processes are periodic
  - All processes are independent from each other
  - Context-switching times and other overhead is ignored
  - All processes have a deadline equal to their period
  - All processes have known worst-case execution time
The Cyclic Executive - Problems

- System is deterministic, but only fully so for the first task (at begin of major/minor cycle). All later tasks start to run whenever the preceding ones have ended.
- Hardware devices are polled. If they are not polled frequently enough, important events might be missed. If they are polled too frequently, processing power is wasted.
- Difficult to incorporate processes with long periods.
- If procedures are split up to form tasks with lower execution times, finding the right granularity of “processes” is difficult.
- Code for logically independent tasks is interleaved.
- Sporadic activities cannot be incorporated.
- Difficult to construct (NP complete) and difficult to maintain.
The Scheduling Problem: Classification

- Scheduling problems usually are classified according to a set of criteria:
  - the **cost function**
  - **hard** deadlines vs. **soft** deadlines
  - **periodic** vs. **aperiodic** vs. **sporadic** events
  - **preemptive** vs. **non-preemptive**
  - **static** vs. **dynamic**
  - **online** vs. **offline**
The Scheduling Problem: Classification

- Tasks which must be executed once every $p$ units of time are called **periodic**, and $p$ is called their **period**. Each execution of a periodic task is called a **job**.
- Tasks which are not periodic are called **aperiodic**.
- Aperiodic tasks requesting the processor at unpredictable times are called **sporadic**, if there is a minimum separation between the times at which they request the processor.

- A **preemptive** scheduler can arbitrarily suspend a process’s execution and restart it later without affecting the functional behavior of the process. Preemption typically occurs when a higher priority process becomes runnable. **Non-preemptive** schedulers do not suspend processes in this way.
The Scheduling Problem: Classification

- An **offline** scheduling algorithm makes all scheduling decisions prior to the running of the system. **Online** scheduling algorithms schedule tasks at run-time; they can be either **static** or **dynamic**.

- In a **static** scheduling algorithm calculating the schedules is based on a process’s characteristics available before the system is run. It requires little runtime overhead.

- A **dynamic** method schedules at run-time, taking into account both process characteristics and the current state of the system. It has higher run-time cost but can deal with non-predicted events and can give greater processor utilization.
The Task Model

- Let $\Gamma = \{ T_i \}$ be a set of tasks. Then let
  - $r_i$ be the release time (or arrival time) which is the time at which $T_i$ is ready for processing
  - $c_i$ be the worst-case execution time of $T_i$
  - $d_i$ be the deadline interval, i.e., the time between $T_i$ becoming available and the time until which $T_i$ has to finish execution
  - $l_i = d_i - c_i$ be the laxity or slack of $T_i$.
  - In $\{ T_i \}$ precedence constraints among tasks may be defined. $T_i \rightarrow T_j$ means that the processing of $T_i$ must be completed before $T_j$ can be started.

![Diagram of task release and execution](image)
Task Model

- The following parameters can be calculated from a given schedule:
  - Completion Time $C_i$
  - Response Time $R_i = C_i - r_i$
  - Lateness $L_i = C_i - d_i$
  - Tardiness $D_i = \max\{C_i - d_i, 0\}$

- Some performance measures / goal functions:
  - Schedule Length (makespan) $C_{max} = \max\{C_i\}$
  - Maximum Lateness $L_{max} = \max\{L_i\}$

- Critical instant: That time at which the release of a task will produce the largest response time.

- Scheduling to minimize the makespan with release times and deadlines is NP hard.
Overview

- Static-Priority Scheduling (Fixed-priority Scheduling)
- Dynamic-Priority Scheduling
- Schedulability and Response Time Analysis

Further reading:
Fixed-Priority Scheduling

- Under fixed-priority scheduling, different jobs of a task are assigned the same priority.

- A fixed-priority scheduling scheme S is optimal if the following criterion is satisfied:
  If any process can be scheduled with some fixed-priority assignment scheme,
  then the given process can also be scheduled with scheme S.
Rate Monotonic Scheduling

- Let each process have a unique priority $P_i$ based on its period $\pi_i$.
- We assume that the shorter the period, the higher the priority, i.e., $\pi_i < \pi_j \Leftrightarrow P_i > P_j$.
- Further assume $d_i = \pi_i$ for all tasks $T_i$.
- Example schedule: $T_1$ with $\pi_1=3$ and $c_1=0.5$, $T_2$ with $\pi_2=4$ and $c_2=1$ and $T_3$ with $\pi_3=6$ and $c_3=2$. 

![Diagram of task schedules](image-url)
Rate Monotonic Scheduling

- The priority of a process is derived from its temporal requirements, not its importance to the system, nor its integrity.
- Note: priority 1 is lowest (least) priority.

<table>
<thead>
<tr>
<th>Task</th>
<th>Period $\pi$</th>
<th>Priority P</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>45</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>35</td>
<td>3</td>
</tr>
</tbody>
</table>

- The schedulability depends on the period and the maximal computational requirements of each process.
Processor Utilization

- Let $\Gamma = \{T_i\}$ be a set of tasks. The utilization $U$ of a task set is defined as $U = \sum_{i=1}^{N} \frac{C_i}{\pi_i}$

- Corollary: If the utilization factor of a task set $\Gamma = \{ T_i \}_{i=1..N}$ is greater than one, the task set cannot be scheduled by any algorithm.

- PROOF: Let $\Pi = \pi_1 \pi_2 ... \pi_N$ be the product of all periods. If $U > 1$, then also $U \Pi > \Pi$, which can be written as:

$$\sum_{i=1}^{N} \frac{\Pi}{\pi_i} c_i > \Pi$$

- $\Pi/\pi_i$ is the number of times task $T_i$ is executed in the interval $\Pi$.
- $(\Pi/\pi_i)c_i$ is the total computation time requested by $T_i$ in the interval $\Pi$.

- Thus: if the total demand in computation time is higher than the available processor time, there can be no feasible schedule for the task set. ■
Processor Utilization

- There exists a maximum value of $U$ below which $\Gamma$ is schedulable and above which $\Gamma$ is not schedulable. This limit depends on:
  - the task set, i.e. the relations among task's periods
  - and on the algorithm used to schedule the tasks.

- Let $U_{ub}(\Gamma, A)$ be this upper bound of the processor utilization factor for a task set $\Gamma$ under an algorithm $A$.

- When $U = U_{ub}(\Gamma, A)$, $\Gamma$ fully utilizes the processor. Then $\Gamma$ is schedulable but an increase in computation time in any of the tasks will make the set infeasible.

- For a given algorithm $A$, the least upper bound $U_{lub}(A)$ is the minimum of the utilization factors over all task sets that fully utilize the processor:
  $$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$$

- Any task set whose processor utilization factor is below $U_{lub}(A)$ is schedulable by $A \implies$ With $U_{lub}$ schedulability can be easily verified!
Rate Monotonic Scheduling

- Theorem [Liu and Layland]: A system of $N$ independent, preemptable periodic tasks $T_i$ with $d_i = \pi_i$ can be feasibly scheduled on a processor according to the rate monotonic algorithm if its total utilization $U$ is at most

$$U_{RM} = N(2^N - 1)$$

- Note: $U_{RM}$ asymptotically approaches $ln2$ (69.3%).

<table>
<thead>
<tr>
<th>$N$</th>
<th>$U_{RM}(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.828</td>
</tr>
<tr>
<td>3</td>
<td>0.779</td>
</tr>
<tr>
<td>4</td>
<td>0.756</td>
</tr>
<tr>
<td>5</td>
<td>0.743</td>
</tr>
<tr>
<td>6</td>
<td>0.734</td>
</tr>
</tbody>
</table>
Example: Process Set A

The combined utilization is $U = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} = 0.823$.

Since this is above the threshold for three processes ($U_{RM}(3) = 0.78$), this process set fails the utilization test.

<table>
<thead>
<tr>
<th>Process</th>
<th>Period $\pi$</th>
<th>WCET $c$</th>
<th>Priority $P$</th>
<th>Utilization $U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>50</td>
<td>12</td>
<td>1</td>
<td>0.240</td>
</tr>
<tr>
<td>T2</td>
<td>40</td>
<td>10</td>
<td>2</td>
<td>0.250</td>
</tr>
<tr>
<td>T3</td>
<td>30</td>
<td>10</td>
<td>3</td>
<td>0.333</td>
</tr>
</tbody>
</table>
Example: Process Set B

<table>
<thead>
<tr>
<th>Process</th>
<th>Period $\pi$</th>
<th>WCET $c$</th>
<th>Priority $P$</th>
<th>Utilization $U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>80</td>
<td>32</td>
<td>1</td>
<td>0.400</td>
</tr>
<tr>
<td>T2</td>
<td>40</td>
<td>5</td>
<td>2</td>
<td>0.125</td>
</tr>
<tr>
<td>T3</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>0.250</td>
</tr>
</tbody>
</table>

- The combined utilization is $U = \frac{32}{80} + \frac{5}{40} + \frac{5}{20} = 0.775$.
- Since this is below the threshold for three processes ($U_{RM}(3)=0.78$), this process set will meet all its deadlines.
Example: Process Set C

The combined utilization is 1.0.

Since this is **above** the threshold for three processes \(U_{RM}(3)=0.78\), this process set **fails** the utilization test. Nevertheless the process set will **meet all its deadlines**.

<table>
<thead>
<tr>
<th>Process</th>
<th>Period (\pi)</th>
<th>WCET (c)</th>
<th>Priority (P)</th>
<th>Utilization (U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>80</td>
<td>40</td>
<td>1</td>
<td>0.500</td>
</tr>
<tr>
<td>B</td>
<td>40</td>
<td>10</td>
<td>2</td>
<td>0.250</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>0.250</td>
</tr>
</tbody>
</table>
**Critical Instants**

- **Corollary:** A critical instant for a task occurs whenever the task is released simultaneously with all higher-priority tasks.

- Let $\Gamma = \{ T_i \}_{i=1..N}$ be a set of periodic tasks, ordered by increasing periods, i.e. $\pi_1 < \pi_2 < ... < \pi_n$ and, thus $P_1 > P_2 > ... > P_N$ and let $n > i$.

  ![Diagram](image)

  **Intuition:**
  - The response time of task $T_n$ is delayed by the interference of a task $T_i$ with higher priority.
  - Advancing the release time of $T_i$ may increase the completion time of $T_n$. 
Optimality of Rate Monotonic Scheduling

- Observation: If all tasks are feasible at their critical instants, then the task set is schedulable in any other condition.
- Theorem: If a task set is schedulable by an arbitrary fixed priority assignment, then it is also schedulable by RM.

PROOF:
- Let $T_1$ and $T_2$ be two periodic tasks with $\pi_1 < \pi_2$. Assume that their priorities are not assigned according to RM, i.e., $P_2 > P_1$.
- At a critical instant, the schedule is feasible if the following inequality is satisfied: $c_1 + c_2 \leq \pi_1$ (Eq. 1)
Optimality of Rate Monotonic Scheduling

Now we want to show that $T_1$ and $T_2$ are also schedulable with the RM priority scheme, ie when $P_1 > P_2$.

Let $m = \left\lfloor \frac{\pi_2}{\pi_1} \right\rfloor$ be the number of periods of $T_1$ entirely contained in $\pi_2$.

Then two cases have to be distinguished:

- **Case 1**: The computation time $c_1$ is short enough that all requests of $T_1$ within the critical time zone of $T_2$ are completed before the second request of $T_2$. That is: $c_1 \leq \pi_2 - m\pi_1$
Then the task set is schedulable if 

\[(m + 1)c_1 + c_2 \leq \pi_2 \quad (Eq. 2)\]

We have to show that Eq. 1 \(\Rightarrow\) Eq. 2.

\[c_1 + c_2 \leq \pi_1 \quad (Eq. 1)\]

\[\Leftrightarrow mc_1 + mc_2 \leq m\pi_1\]

\[\Leftrightarrow mc_1 + c_2 \leq mc_1 + mc_2 \leq m\pi_1, \text{ since } m \geq 1\]

\[\Leftrightarrow (m + 1)c_1 + c_2 \leq m\pi_1 + c_1\]

\[\Leftrightarrow (m + 1)c_1 + c_2 \leq m\pi_1 + c_1 \leq \pi_2, \text{ since } c_1 \leq \pi_2 - m\pi_1\]
Case 2: The execution of the last request of $T_1$ in the critical time zone of $T_2$ overlaps the second request of $T_2$. That is:

$$c_1 > \pi_2 - m\pi_1$$

Then the task set obviously is schedulable, if

$$mc_1 + c_2 \leq m\pi_1 \quad (Eq. 3)$$

We have to show that Eq.1 $\Rightarrow$ Eq.3.
Consider again Eq. 1.

\[ c_1 + c_2 \leq \pi_1 \quad \text{(Eq. 1)} \]

\[ \iff mc_1 + mc_2 \leq m\pi_1 \]

\[ \iff mc_1 + c_2 \leq mc_1 + mc_2 \leq m\pi_1, \text{ since } m \geq 1 \]

This directly shows

\[ mc_1 + c_2 \leq m\pi_1 \quad \text{(Eq. 3)} \]
Deadline Monotonic Scheduling

- Let each process have a unique priority $P_i$ based on its relative deadline $d_i$.
- Same as rate monotonic, if each task’s relative deadline equals its period.
- We assume that the shorter the deadline, the higher the priority, i.e. $d_i < d_j \iff P_i > P_j$.
- Example schedule: $T_1$ with $\pi_1 = d_1 = 3$ and $c_1 = 0.5$, $T_2$ with $\pi_2 = 4$, $d_2 = 2$ and $c_2 = 1$ and $T_3$ with $\pi_3 = d_3 = 6$ and $c_3 = 2$. 

![Diagram of T1, T2, T3 with scheduling intervals]
Schedulability Analysis

- The rate monotonic schedulability test can be applied also to deadline monotonic scheduling, by reducing periods to relative deadlines:

\[ \sum_{i=1}^{N} \frac{c_i}{d_i} \leq N \left( \frac{1}{2^N} - 1 \right) \]

- However, this test significantly overestimates the workload on the processor.

Observations:
- The worst-case processor demand occurs when all tasks are released at their critical instants.
- For each task \( T_i \), the sum of its processing time and the interference (preemption) imposed by higher priority tasks must be less than or equal to its deadline \( d_i \).
Schedulability Analysis

- Assume that tasks are ordered by increasing relative deadlines: $i < j \iff d_i < d_j \iff P_i > P_j$.

- Then a task set $\Gamma = \{ T_i \}_{i=1..N}$ is schedulable if the following condition is satisfied:
  $$\forall 1 \leq i \leq N : c_i + I_i \leq d_i$$

- where $I_i$ is a measure of the interference of $T_i$, which can be computed as the sum of the processing times of all higher-priority tasks released before $d_i$:
  $$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{d_i}{\pi_j} \right\rceil c_j$$
Schedulability Analysis

- Note that this test is **sufficient but not necessary**.
- \( I_i \) is calculated by assuming that each higher-priority task exactly interferes \( \left\lfloor \frac{d_i}{\pi_j} \right\rfloor \) times during the execution time of \( T_i \). However, since \( T_i \) may terminate earlier, the **actual interference** may be smaller.
- A sufficient and necessary schedulability test for DM must take the **exact** interleaving of higher-priority tasks into account for each process.