Real time scheduling for embedded systems
Introduction

- In many critical systems, tasks must compute a value within a given time bound.
- The correctness of the system relies on these constraints.
Introduction

• Example: An ABS system is made up of sensors on the wheels (to detect blocking) and a microcontroller, that controls the brakes.
Introduction

• The microcontroller must execute several tasks:
  • **Read the values** from the sensors
  • **Compute** the appropriate action
  • **Send** the commands to the brakes
  • **Test the systems** on a regular basis
  • **Send** relevant information to the diagnostic system
Introduction

• All those tasks share the same microcontroller

• We need to rely on a time sharing mechanism, to provide the microcontroller to the tasks in a cyclic fashion...

• ... while satisfying the constraints on response times
Introduction

• The timing constraints can be of different types and may be more or less urgent:
  • The task controlling the brakes must enforce strict constraints
  • While the tasks that communicates with the diagnostic system can be delayed
Introduction

• Most of the time, these tasks are cyclic:
  • A task is modelled as a given computation time
  • an instance of that task becomes active in the system periodically (e.g. every 10 t.u.)
  • each instance has to be finished within a fixed time delay
Introduction

• The basic problems:
  • Given a set of tasks (modelled as before), is there a schedule that enforces all the constraints?
  • A schedule is a function that associates, to each point in time the active task
  • If yes, can we compute it?
Definitions
Task

• We consider cyclic tasks

• Each task generates a new job on a regular basis

• e.g.: the task “read the sensor” generates a job that communicates with the sensor every 5 ms

• All the job have the same characteristics
Jobs

- A task $T_i$ is characterised by three values that also characterise the jobs of $T_i$:
  - A (worst case) execution time $C_i$.
  - A deadline $D_i$. It is the time, relative to the activation of the job, at which the job should be completed.
  - A periode $P_i$. 
Jobs

• We also assume:
  • All the tasks start in $t=0$
  • Tasks are completely independent
  • A task can’t block another one
Jobs

• **Example:** Let $T=(5,7,10)$ be a task
  
  • The **first job** appears in $t=0$. Next jobs are activated in $t=10, 20, 30...$
  
  • Each job needs at most 5 t.u. of computation time
  
  • The first job **should complete before** $t=7$. The next ones before $t=17, 27, 27...$

\[0 \quad 7 \quad 10 \quad 17 \quad 20 \quad 27 \quad 30\]
Jobs

• **Remark:** in the sequel, we’ll assume that each job must always be completed when the next job of the same task gets activated.

• That is: $D=P$

• We thus consider tasks of the form $T=(C,P)$
Scheduler

• Once it is active, a job can be:
  • running: it uses the CPU and consumes computing time
  • inactive

• The scheduler is the part of the OS that decides which job should be granted the CPU
Schedule

• The scheduler thus computes a schedule

• **Definition:** a schedule is a function, that associates, to each point in time, the running job

• **Definition:** A set of tasks \( \{T_1, \ldots, T_n\} \) is schedulable iff there is a schedule that satisfies the deadlines of all the jobs generated by all the tasks.
Scheduler

• A classical technique consists, for the scheduler, to assign a priority to each job

• At any time, the CPU runs the job with the highest priority

• Each priority assignment thus defines a unique schedule

• Finding a schedule thus amounts to finding a proper priority assignment
Questions

• How to assign priorities to job to satisfy all the deadlines

• And maybe other constraints?

• For a fixed scheduling policy, can we determine whether a system is schedulable?
Example

$T_1 = (2, 6)$  $T_2 = (3, 8)$  $T_3 = (1, 10)$

Priorities: $T_1 > T_2 > T_3$
Static vs Dynamic

• In the previous example, we have assigned the same priority to all the jobs of the same task

• It is a static policy

• We would also assign different priorities to different jobs of the same task

• This is a dynamic policy
Utilisation

• **Definition:** The **utilisation** of a task $T=(C,P)$ is $U(T)=C/P$

• **Definition:** the **utilisation** of a system with $n$ tasks $T_i=(C_i, P_i)$ ($i=1,...,n$) is $U=\sum_{1 \leq i \leq n} C_i/P_i$

• **Theorem:** If $U > 1$, the system is not schedulable
Two basic algorithms
RM(S)

- RMS = *rate monotonic scheduling*
- It is a *static algorithm*
- **Principle:** priority is *proportional* to the **activation rate** of the jobs
- “Frequent” jobs thus have higher priority
Example

$T_1=(2,6)$
$T_2=(3,8)$
$T_3=(1,10)$

Priorities: $T_1 > T_2 > T_3$
RM(S) - Feasibility

• How can we check whether a system is schedulable under RM?

• Observation: since tasks are cyclic, so is the behaviour of the system

• From $t=\text{LCM}(P_1,\ldots,P_n)$, the sequence of activation of the job is the same as from $t=0$
RM(S) - hyperperiod

$T_1 = (2,6)$

$T_2 = (3,8)$

$\text{LCM}(6,8) = 24$
RM(S) - Feasibility

• The time $H = \text{LCM}(P_1, ..., P_n)$ is called the hyperperiod of the system

• Thus, if the system is schedulable on $[0...H]$, it is schedulable

• We can thus simulate the system on that interval
RM(S) - Hyperperiod

\[ T_1 = (2,6) \quad T_2 = (3,8) \]

The system is schedulable
RM(S) - Feasibility

• Unfortunately, $H$ can be huge

• Liu et Layland have shown in 1973 (J. ACM 20(1)) that there is a sufficient condition for schedulability which is easy to check:

• **Theorem**: In a system with $n$ tasks: If $U \leq n \times (n\sqrt{2} - 1)$ then, the system can be scheduled under RM
RM(S) - Feasibility

• We can observe that \( n \times (n \sqrt{2} - 1) \) tends (by above) to \( \ln(2) \) when \( n \) goes to infinity.

• Thus, for any \( n \): \( n \times (n \sqrt{2} - 1) \geq \ln(2) > 0.69 \)

• In practice we can use:

• **Theorem**: A system is schedulable under RM if \( U \leq 0.69 \)
RM(S) - Optimality

• Can we do better?

• **Definition**: A static scheduling policy is **optimal** iff, any system which is **schedulable** (under any policy R’) is **schedulable under R** too

• **Theorem** (Liu et Layland): RMS is **optimal**

• Not worth trying another algorithm!
RM(S) - in practice

• RMS is implemented is real time OSes like RTEMS

• http://www.rtems.com/

• That OS has been applied to aerospatial cases (NASA, ESA)

• It was also the scheduling algorithm of Mars PathFinder...
EDF

- EDF = *earliest deadline first*
- It is a *dynamic* scheduling algorithm
- **Idea:** The job which is *closest to its deadline* has *highest priority*
- Intuitively, *urgent jobs* get higher priority
- Following the state of the system, *different jobs* of the *same task* can be assigned different priorities
EDF

• In practice, the scheduler has to maintain a queue of waiting jobs.

• When a job is activated, it is inserted in the queue, at a position that depends on its priority.

• If the new job has to be inserted in the first position, the running job is preempted.
Example

$T_1 = (2,6)$  
$T_2 = (3,8)$  
$T_3 = (1,10)$
Example

$T_1=(2,6)$  \hspace{1.5cm}  $T_2=(3,8)$  \hspace{1.5cm}  $T_3=(1,10)$
Example

$T_1=(2,6)$  \hspace{1cm}  $T_2=(3,8)$  \hspace{1cm}  $T_3=(1,10)$
Example

\( T_1 = (2,6) \)

\( T_2 = (3,8) \)

\( T_3 = (1,10) \)
Example

$T_1=(2,6)$  $T_2=(3,8)$  $T_3=(1,10)$
Example

$T_1 = (2,6)$

$T_2 = (3,8)$

$T_3 = (1,10)$
EDF - Feasibility

• The necessary condition $U \leq 1$ for schedulability is now **sufficient**:  

• **Theorem:** A system is **schedulable** under EDF if and only if $U \leq 1$

• EDF is also **optimal** for dynamic policies
EDF vs RMS

• The sufficient condition $U \leq 1$ suggests that there are systems with $U=1$ that are still schedulable under EDF

• EDF thus allow full utilisation of the CPU

• It is not always the case with RMS...
EDF vs RMS

\[ T_1 = (3, 6) \]
\[ T_2 = (3, 8) \]
\[ T_3 = (1, 10) \]

\[ U = \frac{1}{2} + \frac{3}{8} + \frac{1}{10} = 0.975 \]

Missed deadline for \( T_3 \)
EDF - in practice

- EDF is implemented in several real time OSes:
  - SHARK: http://shark.sssup.it/
  - RTAI: https://www.rtai.org/
Energy constraints
Motivation

• With embedded systems, we try to minimise the global power consumption

• Main power sources are:
  • batteries
  • solar panels (aerospatial)
  • ...

lundi 14 mars 2011
Motivation

• The power usage of a CPU depends on its clock frequency

• The voltage is proportional to the clock frequency

• The energy consumed during one cycle is proportional to the square of the voltage
Motivation

• However CPUs sometimes idle

• And their full power (frequency) is thus not always necessary

• Some jobs can be executed «more slowly»
Motivation

• **Main idea:** We will reduce the input voltage of the CPU

• This will **lower** the power consumption

• but it will also **increase** the response time of the tasks...

• ...which can be a **problem** in a **real time** setting.
Technology

• Not all CPUs can *vary* their *frequency* arbitrarily, but this is more and more frequent

• **Example:** At Intel’s, that technology (introduced in 2004) is called **SpeedStep**
Intel SpeedStep

• On Pentium M CPUs, the frequency can vary by steps of 200 MHz

### Supported Performance States for the Intel® Pentium® M Processor at 1.6GHz

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6 GHz (HFM)</td>
<td>1.484 V</td>
</tr>
<tr>
<td>1.4 GHz</td>
<td>1.420 V</td>
</tr>
<tr>
<td>1.2 GHz</td>
<td>1.276 V</td>
</tr>
<tr>
<td>1.0 GHz</td>
<td>1.164 V</td>
</tr>
<tr>
<td>800 MHz</td>
<td>1.036 V</td>
</tr>
<tr>
<td>600 MHz (LFM)</td>
<td>0.956 V</td>
</tr>
</tbody>
</table>

Source: White paper: *Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor*
Intel SpeedStep

Power vs. Core Voltage for the Intel® Pentium® M Processor at 1.6GHz for its six frequency/voltage operating points (not to scale). HFM and LFM power values are TDP specifications.

Source: White paper: Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor
Real-time + Energy
Introduction

• Real-time constraints and energy savings are contradictory

• By diminishing the voltage, we save energy, but we lower the clock frequency and we might miss deadlines

• At maximal frequency, energy consumption is maximal
Hypothesis

• We will first assume that the jobs always consume their worst-case execution time

• We assume that the CPU speed can vary between $S_{\text{min}}$ and $S_{\text{max}}$

• Those speeds are normalised wrt $S_{\text{max}}$: $S_{\text{max}}=1$ et $0 \leq S_{\text{min}} \leq 1$
Hypothesis

• We assume that we will change the speed only during context switches

• Changing the speed has negligible cost
Jobs

- A task $T_i$ is characterised by three values that also characterise the jobs of $T_i$:
  - A (worst case) execution time $C_i$.
  - A deadline $D_i$. It is the time, relative to the activation of the job, at which the job should be completed.
  - A periode $P_i$. 

lundi 14 mars 2011
Jobs

• As speeds are normalised, the computation time of any $T_i$ job running at speed $S$ will be $C_i/S$.

• Utilisation of a task at speed $S$ will be $U_S = U_{\text{max}}/S$, where $U_{\text{max}}$ is the utilisation at max speed, i.e. $C/P$. 
A first idea

- A first simple idea consists in:
  - Schedule the system with a well-known algorithm (e.g. EDF)
  - Choose minimal speed during idle periods
  - Choose maximal speed when the CPU runs tasks
Example

$T_1 = (1, 6)$

$T_2 = (2, 7)$

min speed
Example

$T_1=(1,6)$

$T_2=(2,7)$

$U = 0.45$

If $S_{\text{min}}=0$, and voltage is $k \times S$ we have an average voltage of $k \times 0.495$
Example

• On that example, we could have a lower energy consumption by keeping the speed constant

• We must choose an appropriate speed to respect the deadlines

• Since $U=0.45$, since we want $U \leq 1$, and since at speed $S$: $U_s=U/S$, we choose $S=0.45$
Example

$T_1=(1,6)$ \hspace{5cm} $T_2=(2,7)$

As the speed is **constant** and equal to $S=0.45$, the average volatage is $k \times 0.45$ volts
Solution 1

- A first solution consists in scheduling the system with EDF and choosing \( S = \max\{S_{\text{min}}, U\} \) for the speed.

- Remark that any speed strictly smaller than \( S \) will yield \( U > 1 \), and the system is thus not schedulable.
In general

- Given a **schedulability test** for a scheduling policy R:
  - **Schedule** (if possible) the system at maximal speed
  - Fix an **increment** i
  - Initialise $S = S_{\text{max}} - i$
  - **While** $S \geq S_{\text{min}}$ **do**:
    - If the system is **schedulable** at speed $S$, decrement $S$ of i and go on
    - Otherwise, use speed $S+i$
In general

- Given a schedulability test for a scheduling policy R:
  - Schedule (if possible) the system at maximal speed

That can be too costly (depending on the schedulability test)

- If the system is schedulable at speed $S$, decrement $S$ of $i$ and go on
- Otherwise, use speed $S+i$
Hypothesis

• It might be the case that some jobs do not run for their full worst-case execution time

• That idle time of the CPU should be reclaimed

• We will rely on the hypothesis that the speeds have fixed steps.
Remark

• The algorithms we are about to present have been introduced in *Real-Time Dynamic Scaling for Low-Power Embedded Operating Systems* de Padmanabhan Pillai et Kang G. Shin (2001, ACM)
Reclaiming

$T_1 = (3, 8)$  
$T_2 = (3, 10)$  
$T_3 = (1, 14)$

$U = 0.746...$  
Statical solution: $S = 0.75$

Possible speeds: 0.1  0.25  0.5  0.75  1
Reclaiming

\[ T_1 = (3, 8) \quad \quad T_2 = (3, 10) \quad \quad T_3 = (1, 14) \]
Reclaiming

$T_1 = (3, 8)$  $T_2 = (3, 10)$  $T_3 = (1, 14)$

If the first job of $T_1$ consumes less time (for instance $0.75 \times C_1$), the system is in a situation «as if» $C_1$ was smaller.
Reclaiming

$T_1 = (3, 8)$  $T_2 = (3, 10)$  $T_3 = (1, 14)$

If the first job of $T_1$ consumes less time (for instance $0.75 \times C_1$), the system is in a situation “as if” $C_1$ was smaller. We can thus recompute $U$ with that new $C_1$ and deduce a new speed. That speed will be used up to the introduction of a new job of $T_1$, that could consume $C_1$.
Reclaiming

T_1=(2,8)  T_2=(3,10)  T_3=(1,14)

2.25

✗

U=0.746...

✗

0.65...

Possible speeds: 0.1  0.25  0.75  1

We keep S=0.75

lundi 14 mars 2011
Reclaiming

$T_1 = (1, 8)$  $T_2 = (3, 10)$  $T_3 = (1, 14)$

Possible speeds: 0.1  0.25  0.5  0.75  1

We keep $S = 0.75$

$U = 0.46\ldots$  $0.65\ldots$
Reclaiming

$T_1 = (2, 8)$

$T_2 = (3, 10)$

$T_3 = (1, 14)$

$U = 0.746...$

Possible speeds: 0.1 0.25 0.5 0.75 1

We keep $S = 0.75$

$T_2$'s first job uses 2 t.u. @ $S = 0.75$

This is «as if» $C_2 = 1.5$
Reclaiming

$T_1=(2,8)$  $T_2=(3,10)$  $T_3=(1,14)$

Possible speeds: 0.1  0.25  0.5  0.75  1

We use $S=0.5$
Reclaiming

\[ T_1 = (2, 8) \]
\[ T_2 = (3, 10) \]
\[ T_3 = (1, 14) \]

Possible speeds: 0.1 0.25 0.5 0.75 1

We use \( S = 0.5 \)

\[ U = 0 \times 46 \ldots 0.5 \ldots \]

lundi 14 mars 2011
Reclaiming

\[ T_1 = (2, 8) \]
\[ T_2 = (3, 10) \]
\[ T_3 = (1, 14) \]

\[ U = 0.746... \]
\[ 0.5... \]

Possible speeds: 0.1  0.25  0.5  0.75  1

We have managed to lower the CPU frequency and to spread \( T_3 \)'s first job on 2 t.u. instead of 1.

We use \( S = 0.5 \)
Reclaiming technique for EDF

- Let us assume a fixed set \{S_1, S_2, \ldots, S_m\} of normalised speeds for the CPU
- We compute, for each \(T_i\), its utilisation \(U_i\)
- When a job of \(T_i\) enters the system, we set \(U_i\) at \(C_i/P_i\) and adapt the speed (cfr. infra)
Reclaiming technique for EDF

• When a job of $T_i$ finishes, we compute $cc_i$, i.e. the time that has really been used by the job, set $U_i$ at $(cc_i \times S)/P_i$ and adapt the speed

• Here, $S$ is the speed at which the job has run

• $cc_i \times S$ is thus the equivalent running time at maximal speed
Reclaiming technique for EDF

- To adapt the speed: we select the smallest speed \( S \) s.t.
  \[ S \geq U_1 + U_2 + \cdots + U_n \]
Reclaiming technique for EDF

- Although we use lower speeds, we are still sure to respect deadlines

Job of $T = (C,P)$

Consumes $C' < C$

At that point the system is equivalent to a system where $T$ is replaced by $T' = (C',P)$

The lower speed corresponds to the scheduling in this new system.
Reclaiming technique for EDF

- Although we use lower speeds, we are still sure to respect deadlines.

Job of $T=(C,P)$

However, when a new job of $T$ enters the system, the hypothesis that it consumes at most $C'$ t.u. does not hold anymore!
Reclaiming technique for EDF

• **Remark**: That technique could be applied to RM too

• But the efficiency of the solution relies on the fact that the schedulability test is very efficient for EDF (one just has to compute $U$)

• But that test is quite costly for RM

• **Better suited** solutions exist for RM
Reclaiming technique for RM

$T_1=(3,8)$  $T_2=(3,10)$  $T_3=(1,14)$

At maximal speed

If all the tasks use the worst case, we can’t do better
Reclaiming technique for RM

$T_1=(3,8)$  $T_2=(3,10)$  $T_3=(1,14)$
Reclaiming technique for RM

\[ T_1 = (3,8) \quad T_2 = (3,10) \quad T_3 = (1,14) \]

If \( T_1 \)'s first job finishes sooner, we have «more room» for the jobs of \( T_2 \) and \( T_3 \) before the next deadline of \( T_1 \).

We choose \( S = 0.75 \)
Reclaiming technique for RM

$T_1 = (3, 8)$

$T_2 = (3, 10)$

$T_3 = (1, 14)$

This is the worst case

lundi 14 mars 2011
Reclaiming technique for RM

$T_1 = (3, 8)$  
$T_2 = (3, 10)$  
$T_3 = (1, 14)$
Reclaiming technique for RM

\[ T_1 = (3, 8) \quad T_2 = (3, 10) \quad T_3 = (1, 14) \]

If \( T_2 \)'s first job finishes sooner too, we can further lower the speed, and still place \( T_3 \) before the next deadline.

lundi 14 mars 2011
Reclaiming technique for RM

$T_1 = (3, 8)$  $T_2 = (3, 10)$  $T_3 = (1, 14)$
Reclaiming technique for RM

• **Contrary to the EDF technique**, that algorithm is **local**:

• **At any time we consider only the interval from present time to the next deadline**

• **At each deadline, we “start from scratch” for all the tasks…**

• **…whereas in the EDF technique, only the task that has reached its deadline gets adapted**
Reclaiming technique for RM

• Although the algorithm is **conceptually simple**, we need to **maintain** several information about the running tasks.

• That techniques **avoids** relying on the **schedulability test** for RM, but is still **heavier** than the EDF technique.