Verification of Real Time Systems - CS5270
3rd lecture

Hugh Anderson

National University of Singapore
School of Computing

January, 2007

A warning...

Outline

1. Administration
   - Assignment 1
   - Introduction to Uppaal
2. Scheduling
   - Scheduling concepts
   - Critical sections and Semaphores
3. Scheduling algorithms
   - RMS - Rate Monotonic Scheduling
   - Schedulability
   - EDF Earliest Deadline First

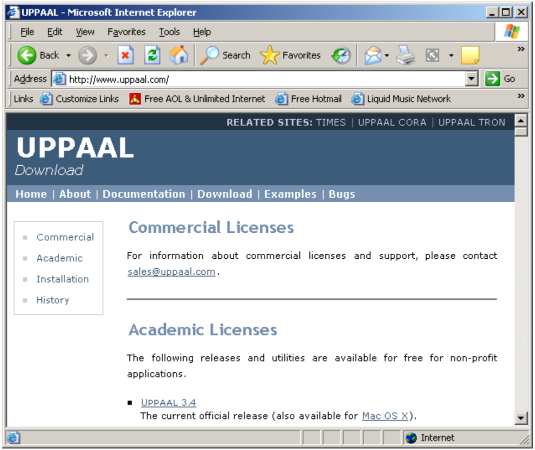
Assignment 1

Assignment number 1 is out

- Seven questions
- Some reading may be required?
- Hand in Feb 18

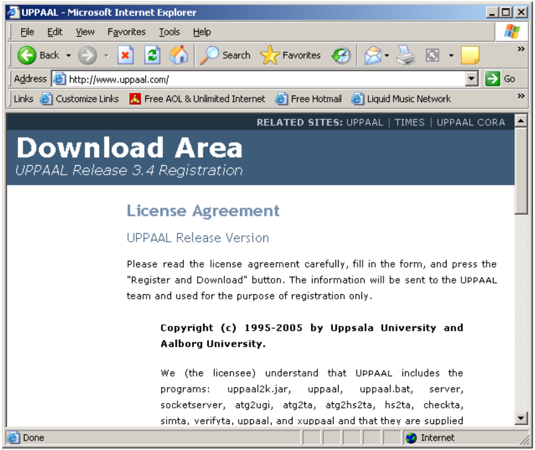
Uppaal

The website:



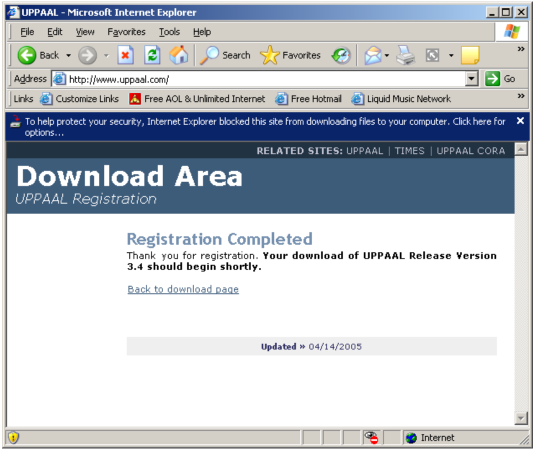
Uppaal

The license:



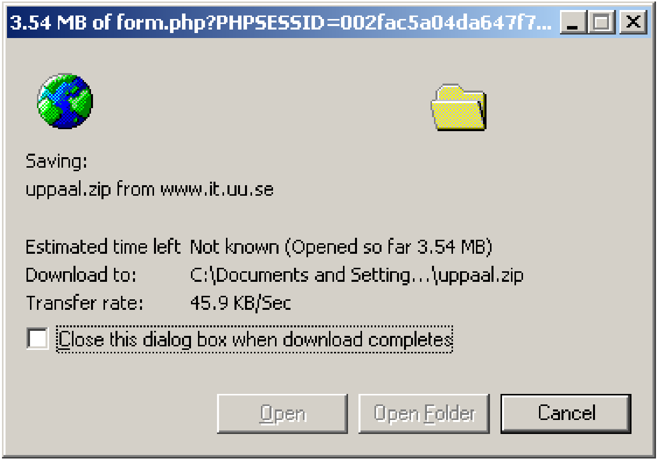
Uppaal

Complete registration:



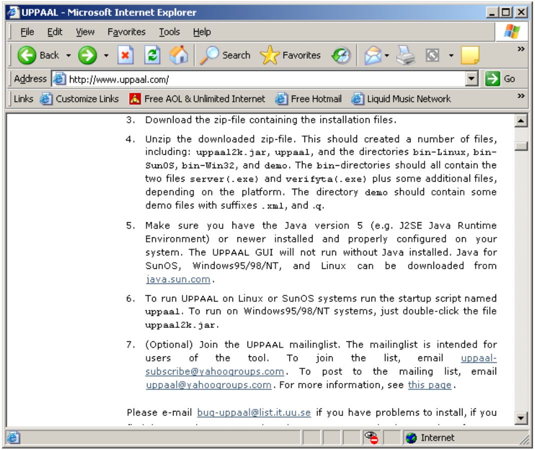
Uppaal

Download:



Uppaal

Instructions:

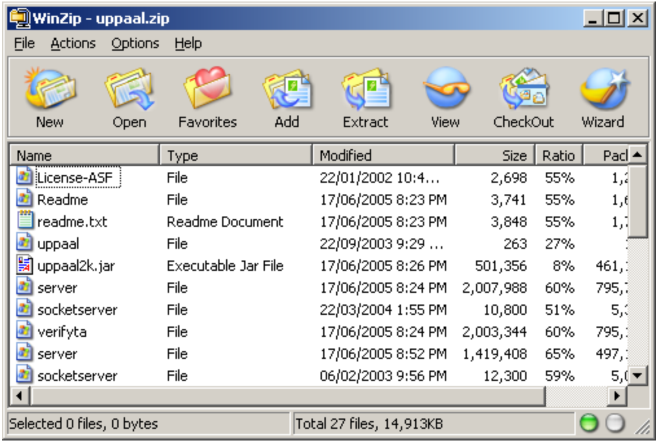


3. Download the zip-file containing the installation files.
4. Unzip the downloaded zip-file. This should created a number of files, including: `uppaal2k.jar`, `uppaal`, and the directories `bin-Linux`, `bin-SunOS`, `bin-Win32`, and `demo`. The `bin`-directories should all contain the two files `server(.exe)` and `verifyta(.exe)` plus some additional files, depending on the platform. The directory `demo` should contain some demo files with suffixes `.xml`, and `.q`.
5. Make sure you have the Java version 5 (e.g. J2SE Java Runtime Environment) or newer installed and properly configured on your system. The UPPAAL GUI will not run without Java installed. Java for SunOS, Windows95/98/NT, and Linux can be downloaded from java.sun.com.
6. To run UPPAAL on Linux or SunOS systems run the startup script named `uppaal`. To run on Windows95/98/NT systems, just double-click the file `uppaal2k.jar`.
7. (Optional) Join the UPPAAL mailinglist. The mailinglist is intended for users of the tool. To join the list, email uppaal-subscribe@yahoogroups.com. To post to the mailing list, email uppaal@yahoogroups.com. For more information, see this page.

Please e-mail bug-uppaal@list.it.uu.se if you have problems to install, if you

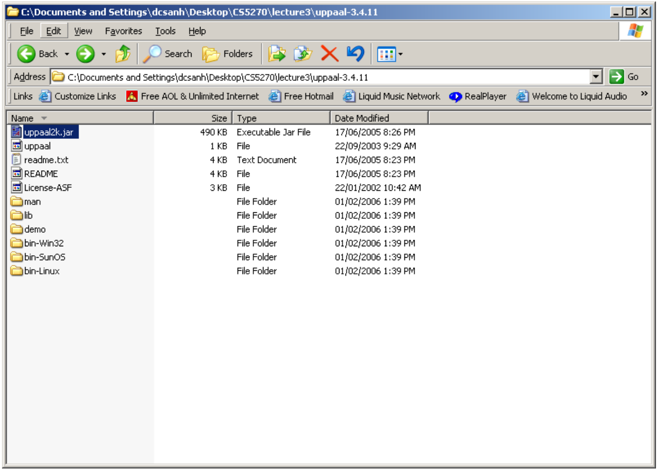
Uppaal

Extract/unzip:



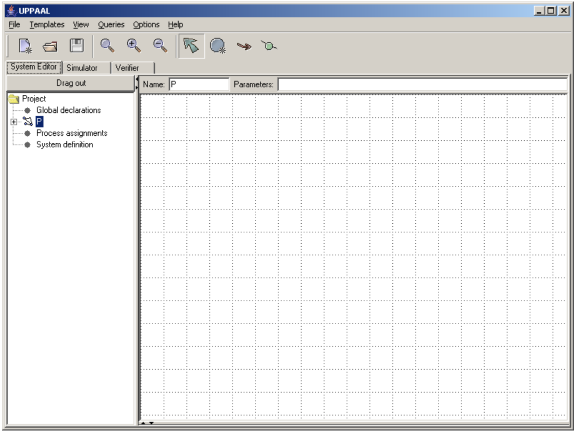
Uppaal

Click on jar file:



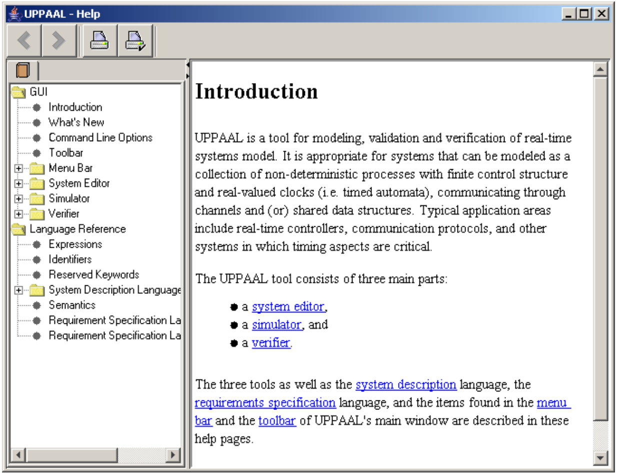
Uppaal

The application:



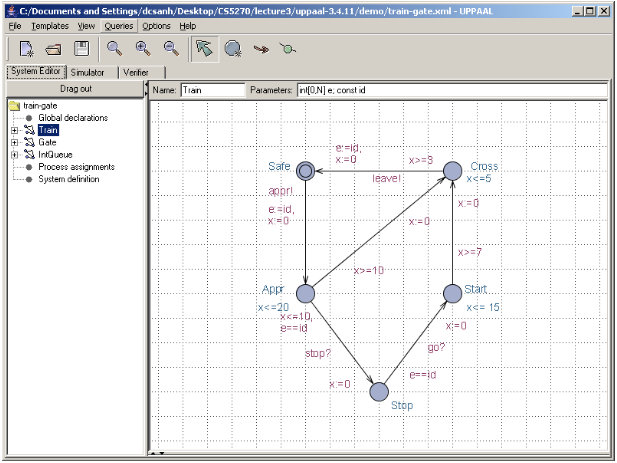
Uppaal

Help:


UPPAAL - Help
GUI
Introduction
What's New
Command Line Options
Toolbar
Menu Bar
System Editor
Simulator
Verifier
Language Reference
Expressions
Identifiers
Reserved Keywords
System Description Language
Semantics
Requirement Specification La
Requirement Specification La
Introduction
UPPAAL is a tool for modeling, validation and verification of real-time systems model. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks (i.e. timed automata), communicating through channels and (or) shared data structures. Typical application areas include real-time controllers, communication protocols, and other systems in which timing aspects are critical.
The UPPAAL tool consists of three main parts:
a system editor,
a simulator, and
a verifier.
The three tools as well as the system description language, the requirements specification language, and the items found in the menu bar and the toolbar of UPPAAL's main window are described in these help pages.

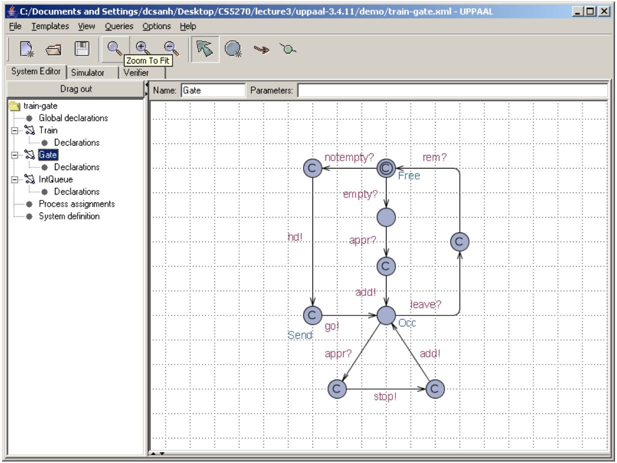
Uppaal

Load up a demo:



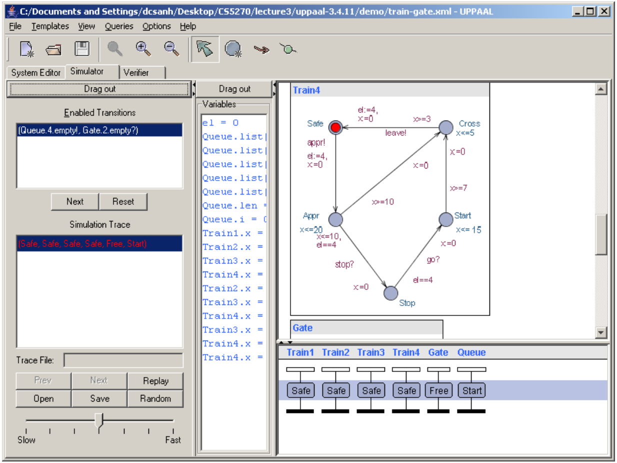
Uppaal

Look at TTS:



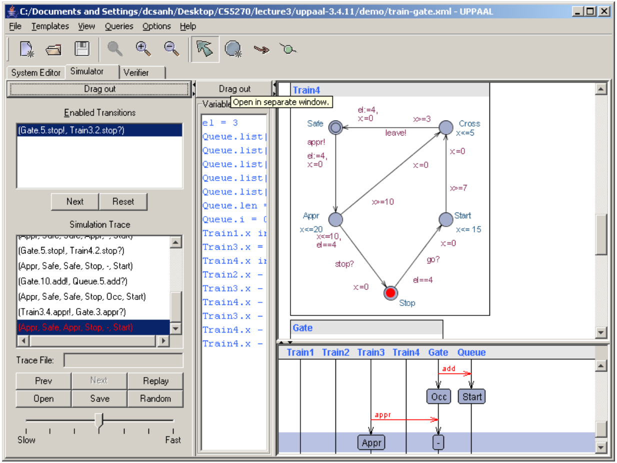
Uppaal

Simulation:



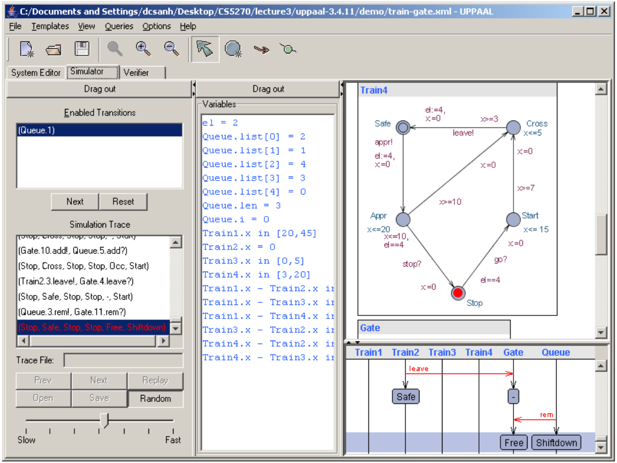
Uppaal

Simulation:



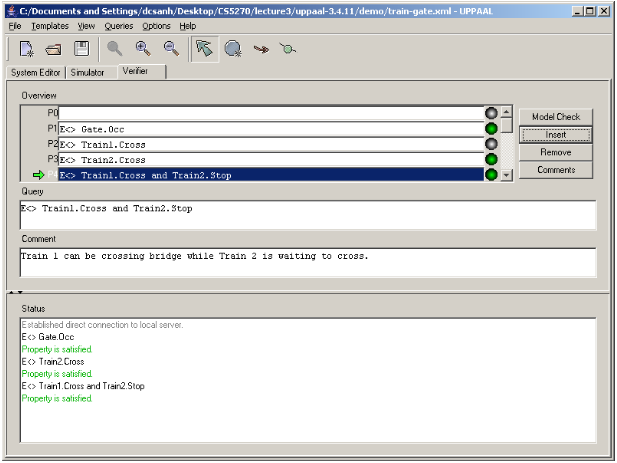
Uppaal

Simulation:



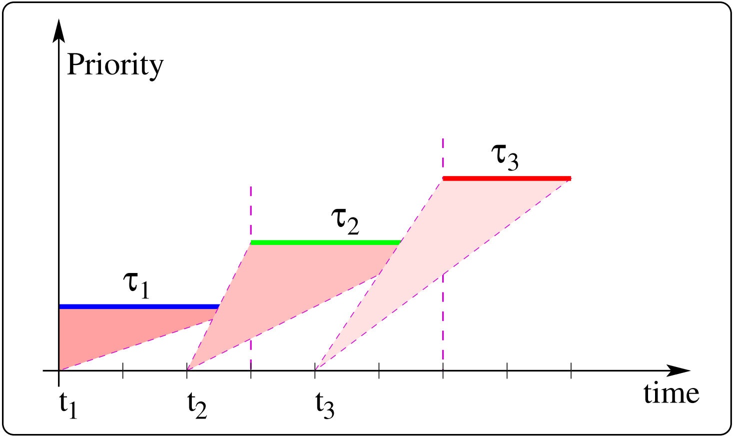
Uppaal

Verification:



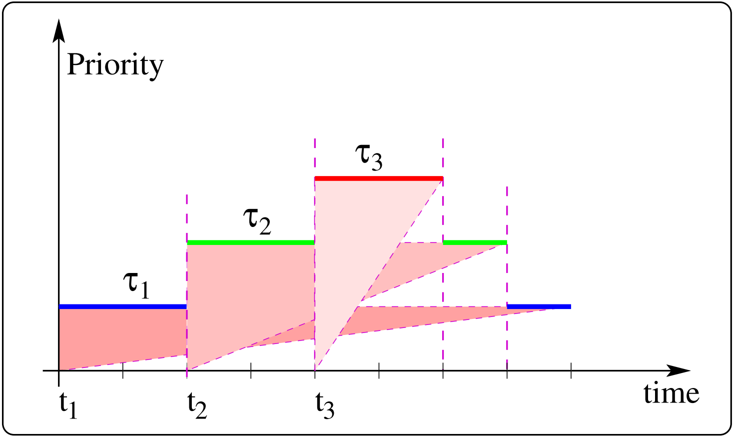
Non-preemptive scheduling

Tasks are delayed until other tasks complete:



Preemptive scheduling

Tasks preempt lower priority tasks:



Scheduling terms

Definitions:

Feasible: a schedule is termed feasible if all tasks can be completed within the constraints specified

Schedulable: a task set is schedulable if a particular scheduling algorithm produces a feasible schedule

Scheduling terms

Constraints found in various areas:

Timing (deadlines for tasks)

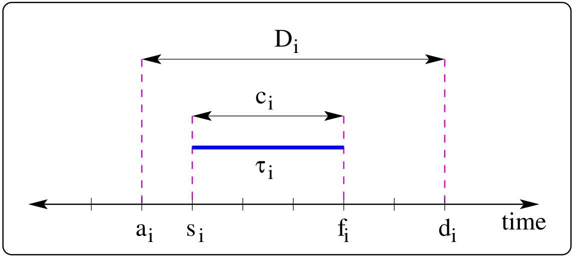
Precedence (which task comes first)

Resource (shared access)

Hard/Soft constraints

Scheduling terms

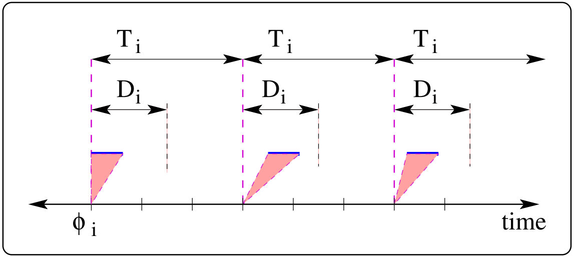
Deadlines:



- If a task t_i needs to finish before some time d_i, then this is called a **deadline**. A **relative deadline** D_i for the task is $D_i = d_i - a_i$.
- Tasks run for time c_j, and must complete before a deadline

Scheduling terms

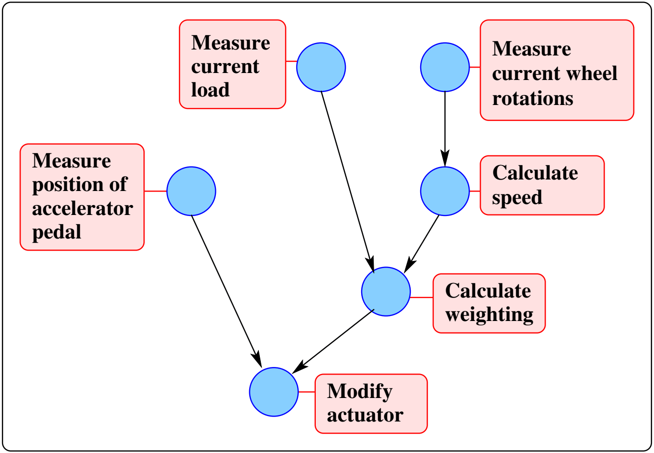
Periodic tasks:



- A **periodic** task is one that is regularly activated at a constant rate.
- Its period is T_i, and the time of first activation (its **phase**) is ϕ_i.

Scheduling terms

Precedence between tasks - visualize as a graph:



Scheduling terms

Resource access:

A *resource constraint*, may be some variable or device or some other structure in the system. Resources only become *critical resource constraints* when they are shared with other tasks.

- An *exclusive resource* is one which may require exclusion of all other tasks when the resource is accessed. This is called *mutual exclusion* (OS normally provide mechanisms to assist tasks to provide mutually exclusive access to a resource). The code which requires this mutually exclusive access is termed a *critical section* (CS).
- The most common mechanism for this purpose is called the semaphore, where a semaphore variable s_i is used to control access to an associated CS_i.

Critical section

A critical section is:

- A piece of code belonging to task executed under mutual exclusion constraints.
- Mutual exclusion is enforced by semaphores.
  - wait(s)
    - Blocked if s = 0.
  - signal(s)
    - s is set to 1 when signal(s) executes.

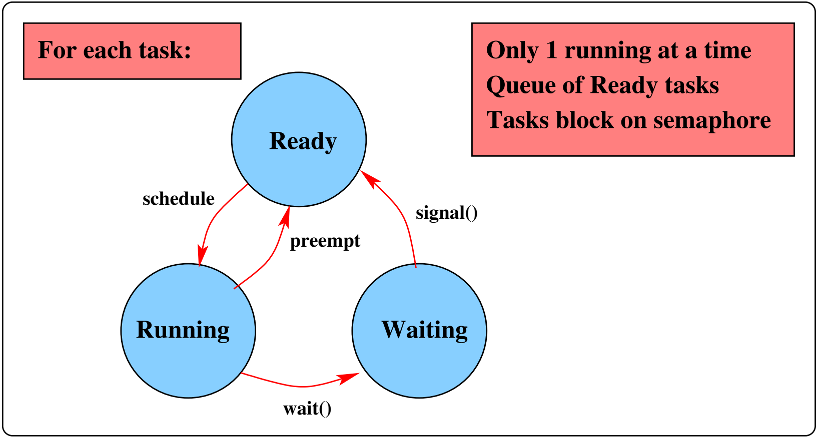
Critical sections

CS and blocking:

- A task waiting for an exclusive resource is blocked on that resource.
- Tasks blocked on the same resource are kept in a wait queue associated with the semaphore protecting the resource.
- A task in the running state executing wait(s) on a locked semaphore (s = 0) enters the waiting state.
- When a task currently using the resource executes signal(s), the semaphore is released.
- When a task leaves its waiting state (because the semaphore has been released) it goes into the ready state:

Semaphores and blocking

In an OS, tasks block when waiting for a resource:



The scheduling problem

The general scheduling problem is NP-complete:

- There is a non-deterministic Turing Machine TM and a polynomial in one variable $p(n)$ such that for each problem instance of size n, TM determines if there exists a schedule and if so outputs one in at most $p(n)$ steps. Any non-deterministic polynomial time problem can be transformed in deterministic polynomial time to the general scheduling problem, and only exponential time deterministic algorithms are known.

Hence we must find imperfect but efficient solutions to scheduling problems. A great variety of algorithms exist, with various assumptions, and with different complexities.

Assumptions for RMS

In RMS:

- assume a set of tasks $\{\tau_1, \ldots, \tau_m\}$ with periods $T_1, \ldots, T_m$, $\phi_i = 0$ and $D_i = T_i$ for each task.
  - We allow preemption,
  - there is only a single processor, and
  - we have no precedence constraints.

RMS

The RMS algorithm:

- Assign a static priority to the tasks according to their periods.
- Priority of a task does not change during execution.
- Tasks with shorter periods have higher priorities.
- Preemption policy:
  - If T_i is executing and T_j arrives which has higher priority (shorter period), then preempt T_i and start executing T_j.

Assumptions RMS

From the Liu article:

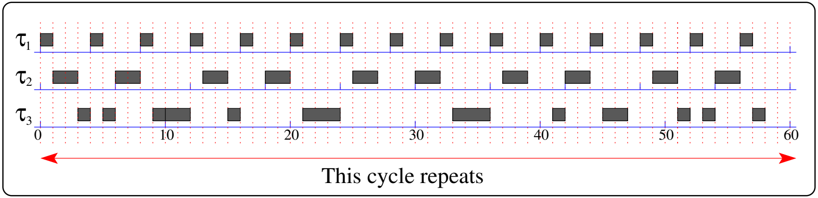
(A1) The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.
(A2) Deadlines consist of run-ability constraints only - i.e. each task must be completed before the next request for it occurs.
(A3) The tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks.
(A4) Run-time for each task is constant for that task and does not vary with time.
(A5) Any nonperiodic tasks in the system are special; they are initialization or failure-recovery routines; they displace periodic tasks while they themselves are being run, and do not themselves have hard, critical deadlines.

RMS

Given this task set:

	τ_1	τ_2	τ_3
C_i	1	2	3
T_i	4	6	10

An RMS schedule is:



RMS

Properties of RMS:

- RMS is optimal (Given the previous constraints)
  - If a set of of periodic tasks (satisfying the assumptions set out previously) is not schedulable under RMS then no static priority algorithm can schedule this set of tasks.
- RMS requires very little run time processing.

Schedulability terms

Definition of PUF, the Processor Utilization Factor:

The **processor utilization factor** U is the fraction of processor time spent in the task set:

$$\mathrm{U} = \sum_{i=1}^{m} \frac{C_i}{T_i}$$

If this factor is *greater* than 1 then of course, the task set can not be scheduled. However if $\mathrm{U} \leq 1$ then it is possible that it may be RMS-schedulable. If a particular set of tasks has a feasible RMS schedule, and any increase of the runtime of any task would render the particular set infeasible, then the processor is said to be fully utilized.

Schedulability terms

Example of PUF calculation:

	τ_1	τ_2	τ_3
C_i	1	2	3
T_i	4	6	10

the processor utilization factor is $\mathrm{U} = \sum_{i=1}^{m} \frac{C_i}{T_i} = 0.833$.

Schedulability

The least upper bound of processor utilization:

The **least upper bound** $\mathrm{U}_{\mathrm{lub}}$ is the minimum of the U over all sets of tasks that fully utilize the processor.

- If $\mathrm{U} \leq \mathrm{U}_{\mathrm{lub}}$, then the set of tasks is guaranteed to be schedulable.
- Table gives a sufficient value for $\mathrm{U}_{\mathrm{lub}}$ for different numbers of tasks for RMS ($\mathrm{U}_{\mathrm{lub}} = m(2^{\frac{1}{m}} - 1)$), but note that it may be possible to schedule a task set even if the criterion fails.

m	1	2	3	4	5	6	∞
$\mathrm{U}_{\mathrm{lub}}$	1.000	0.828	0.780	0.757	0.743	0.735	0.690

Schedulability

Using our example:

	τ_1	τ_2	τ_3
C_i	1	2	3
T_i	4	6	10

the processor utilization factor is $\mathrm{U} = \sum_{i=1}^{m} \frac{C_i}{T_i} = 0.833$.

- The least upper bound for rate monotonic scheduling for 3 tasks is given in the table as $\mathrm{U}_{\mathrm{lub}} = 0.780$, and since $\mathrm{U}_{\mathrm{lub}} < \mathrm{U}$, we cannot guarantee that this task set is schedulable..

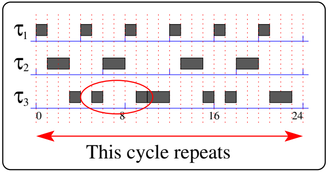
Schedulability

Another example:

	τ_1	τ_2	τ_3
C_i	1	2	3
T_i	4	6	8

the processor utilization factor is $U = \sum_{i=1}^{m} \frac{C_i}{T_i} = 0.95833$.

- With this set of tasks, we have that task τ_3 fails to complete within its period (8), task set is not schedulable using RMS.



Earliest Deadline First

The policy:

- Tasks with earlier deadlines will have higher priorities.
- Applies to both periodic and aperiodic tasks.
- EDF is optimal for dynamic priority algorithms.
- A set of periodic tasks is schedulable with EDF iff the utilization factor is not greater than 1.

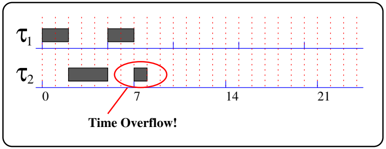
Earliest Deadline First

RMS fails:

	τ_1	τ_2
C_i	2	4
T_i	5	7

From $U = 0.4 + 0.57 = 0.97 \leq 1$ we know that it is guaranteed to be schedulable under EDF, and might be schedulable under RMS.

- It is not RMS schedulable:



Earliest Deadline First

EDF can guarantee deadlines in the system at higher loading:

	τ_1	τ_2
C_i	2	4
T_i	5	7

From $\mathrm{U} = 0.4 + 0.57 = 0.97 \leq 1$ we know that it is guaranteed to be schedulable under EDF.

- EDF scheduling succeeds: