

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



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



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



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



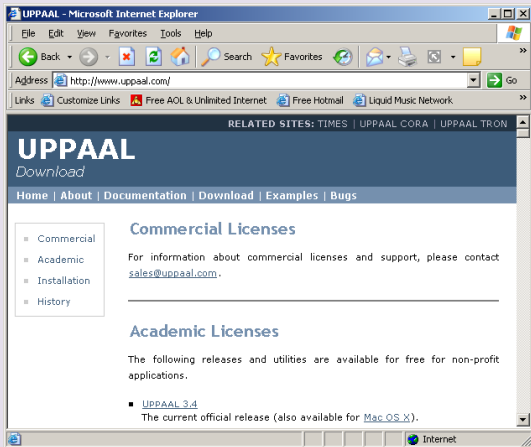
Outline

- 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



Uppaal

The website:



Uppaal

The license:

UPPAAL - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites

Address <http://www.uppaal.com/> Go

Links Customize Links Free AOL & Unlimited Internet Free Hotmail Liquid Music Network

RELATED SITES: UPPAAL | TIMES | UPPAAL CORA

Download Area

UPPAAL Release 3.4 Registration

License Agreement

UPPAAL Release Version

Please read the license agreement carefully, fill in the form, and press the "Register and Download" button. The information will be sent to the UPPAAL team and used for the purpose of registration only.

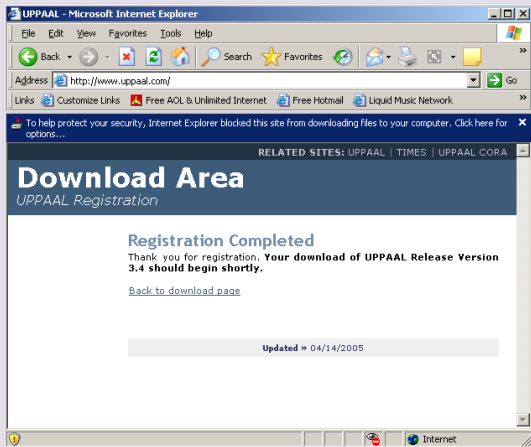
Copyright (c) 1995-2005 by Uppsala University and Aalborg University.

We (the licensee) understand that UPPAAL includes the programs: uppaal2k.jar, uppaal, uppaal.bat, server, socketserver, atg2ugi, atg2ta, atg2hs2ta, hs2ta, checkta, simta, verifyta, uppaal, and xuppaal and that they are supplied

Done Internet

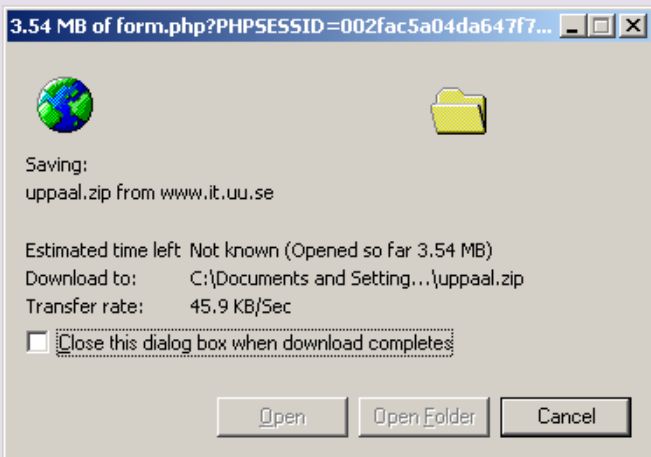
Uppaal

Complete registration:



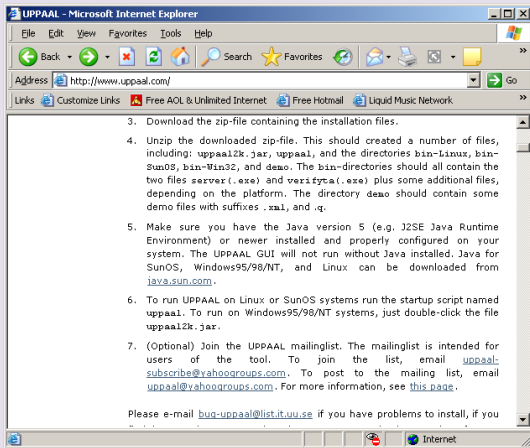
Uppaal

Download:



Uppaal

Instructions:



UPPAAL - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites

Address <http://www.uppaal.com/> Go

Links Customize Links Free AOL & Unlimited Internet Free Hotmail Liquid Music Network

3. Download the zip-file containing the installation files.
4. Unzip the downloaded zip-file. This should create 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 `.xal`, and `.g`.
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 buq-uppaal@list.it.uu.se if you have problems to install, if you

Internet

Uppaal

Extract/unzip:

WinZip - uppaal.zip

File Actions Options Help

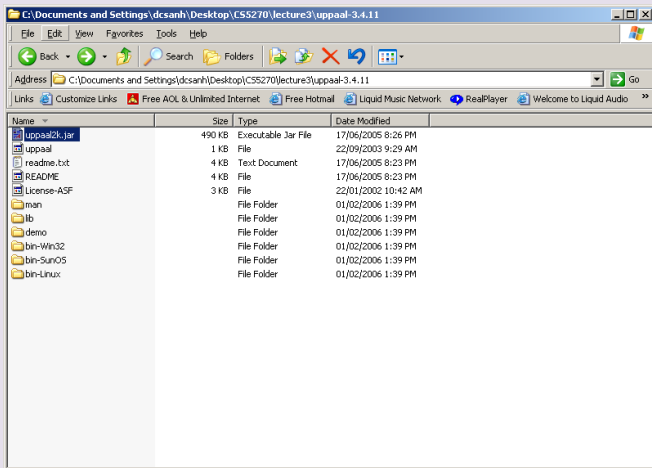
New Open Favorites Add Extract View CheckOut Wizard

Name	Type	Modified	Size	Ratio	Pack
License-ASF	File	22/01/2002 10:4...	2,698	55%	1,2
Readme	File	17/06/2005 8:23 PM	3,741	55%	1,6
readme.txt	Readme Document	17/06/2005 8:23 PM	3,848	55%	1,7
uppaal	File	22/09/2003 9:29 ...	263	27%	
uppaal2k.jar	Executable Jar File	17/06/2005 8:26 PM	501,356	8%	461,
server	File	17/06/2005 8:24 PM	2,007,988	60%	795,
socketserver	File	22/03/2004 1:55 PM	10,800	51%	5,
verifyta	File	17/06/2005 8:24 PM	2,003,344	60%	795,
server	File	17/06/2005 8:52 PM	1,419,408	65%	497,
socketserver	File	06/02/2003 9:56 PM	12,300	59%	5,0

Selected 0 files, 0 bytes Total 27 files, 14,913KB

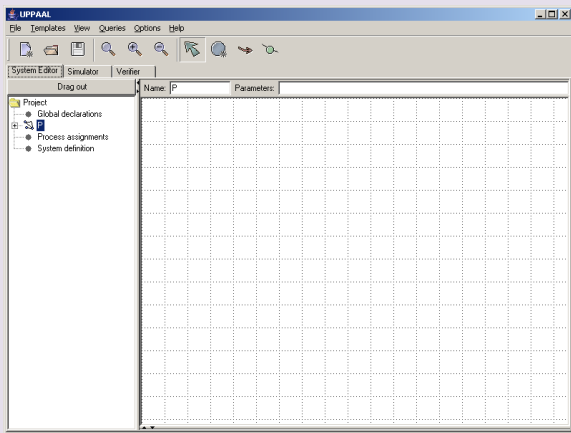
Uppaal

Click on jar file:



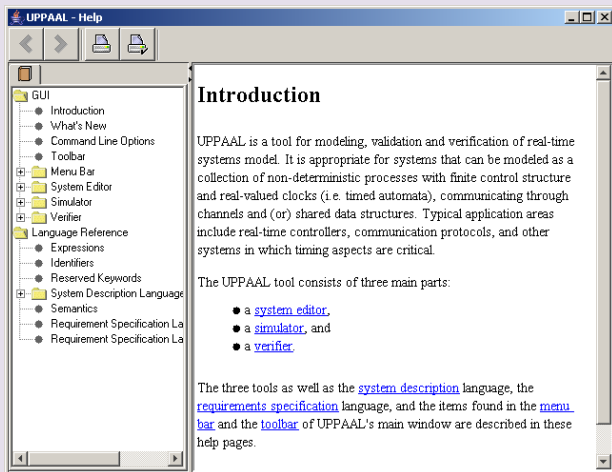
Uppaal

The application:



Uppaal

Help:



The screenshot shows a window titled "UPPAAL - Help" with a standard Windows-style title bar. On the left is a tree view of the help contents, and on the right is the main text area.

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.

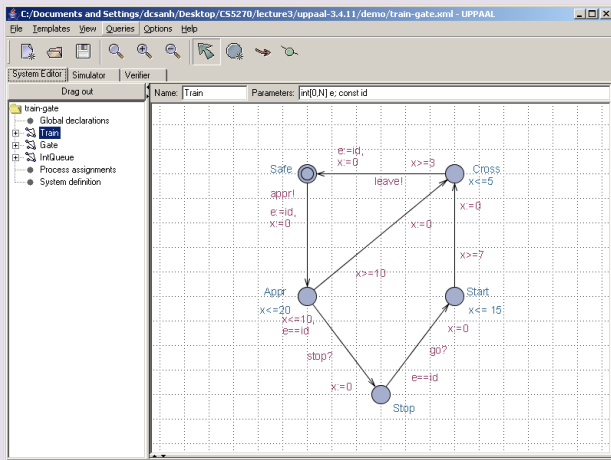
The UPPAAL tool consists of three main parts:

- 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.

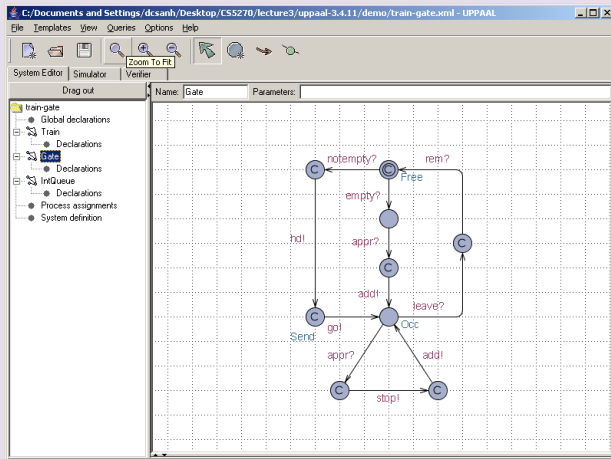
Uppaal

Load up a demo:



Uppaal

Look at TTS:



Uppaal

Simulation:

The screenshot shows the Uppaal simulation environment. The main window displays a Petri net diagram for a train gate system. The diagram includes places: Safe (with 1 token), Appr, Stop, Cross, and Start. Transitions are labeled: appr!, leave!, go?, and stop?. Guards and actions are associated with these transitions. For example, the 'leave!' transition has guard $x=3$ and action $x+=5$. The 'go?' transition has guard $x=0$ and action $x+=15$. The 'stop?' transition has guard $x=0$ and action $x=0$. The 'appr!' transition has guard $el=4, x=0$ and action $x=0$. The 'leave!' transition also has guard $x=0$. The 'go?' transition has guard $x=0$. The 'stop?' transition has guard $x=0$. The 'leave!' transition has guard $x=3$. The 'go?' transition has guard $x=0$. The 'stop?' transition has guard $x=0$. The 'leave!' transition has guard $x=3$. The 'go?' transition has guard $x=0$. The 'stop?' transition has guard $x=0$.

Below the diagram is a table showing the state of the system:

Train1	Train2	Train3	Train4	Gate	Queue
Safe	Safe	Safe	Safe	Free	Start

The interface also includes a menu bar (File, Templates, View, Queries, Options, Help), a toolbar, and a control panel with buttons for Next, Reset, Prev, Next, Replay, Open, Save, and Random. A simulation trace window shows the sequence of states: (Safe, Safe, Safe, Safe, Free, Start).

Uppaal

Simulation:

The screenshot displays the Uppaal simulation environment. The main window shows a state transition graph with nodes labeled 'Safe', 'Cross', 'Appr', 'Start', and 'Stop'. Transitions are labeled with guard conditions and actions, such as 'appr!' and 'leave!'. The 'Stop' node is highlighted with a red circle, indicating the current state of the simulation.

On the left side, the 'Enabled Transitions' panel shows a list of transitions, with '(Gate.5.stopl, Train3.2.stopl?)' selected. Below this, the 'Simulation Trace' panel shows a sequence of events, including '(Appr, Safe, Safe, Stop, -, Start)'. The 'Trace File' panel at the bottom left contains buttons for 'Prev', 'Next', 'Replay', 'Open', 'Save', and 'Random', along with a speed control slider ranging from 'Slow' to 'Fast'.

At the bottom of the interface, a 'Train' panel shows a sequence of components: Train1, Train2, Train3, Train4, Gate, and Queue. Below this, a 'Gate' panel shows a sequence of components: Appr, -, Occ, and Start. Red arrows indicate the flow of data or control between these components.

Uppaal

Simulation:

The screenshot displays the Uppaal simulator interface for a file named 'train-gate.xml'. The interface is divided into several panels:

- System Editor:** Contains 'Enabled Transitions' (currently showing 'Queue.1') and 'Simulation Trace' (listing events like '(Gate.10.add, Queue.5.add?)', '(Stop, Cross, Stop, Stop, Occ, Start)', etc.).
- Variables:** Lists variables such as `#1 = 2`, `Queue.list[0] = 2`, `Queue.len = 3`, `Train1.x in [20, 45]`, and `Train4.x in [3, 20]`.
- Train4 State Transition Diagram:** A state machine with states:
 - Safe:** Initial state with `appri!` and `x=0`.
 - Appri:** Reached via `leave` from Safe. Contains `x=20` and `el=10`.
 - Cross:** Reached via `go?` from Appri. Contains `x=5` and `x=0`.
 - Start:** Reached via `stop?` from Appri. Contains `x=15` and `x=0`.
 - Stop:** Reached via `stop?` from Appri. Contains `x=0`.
- Gate:** A sequence diagram showing the interaction between the train and the gate. It includes components for 'Safe', 'Free', and 'Shutdown', with transitions labeled 'leave' and 'rem'.

Uppaal

Verification:

The screenshot shows the Uppaal System Editor window with the following configuration:

- Overview:** A list of properties with status indicators (green for satisfied, grey for not checked).
 - P0: [grey circle]
 - P1: E<> Gate.0cc [green circle]
 - P2: E<> Train1.Cross [grey circle]
 - P3: E<> Train2.Cross [green circle]
 - E<> Train1.Cross and Train2.Stop [green circle]
- Query:** E<> Train1.Cross and Train2.Stop
- Comment:** Train 1 can be crossing bridge while Train 2 is waiting to cross.
- Status:**
 - Established direct connection to local server.
 - E<> Gate.0cc
 - Property is satisfied.
 - E<> Train2.Cross
 - Property is satisfied.
 - E<> Train1.Cross and Train2.Stop
 - Property is satisfied.

Buttons on the right include Model Check, Insert, Remove, and Comments.

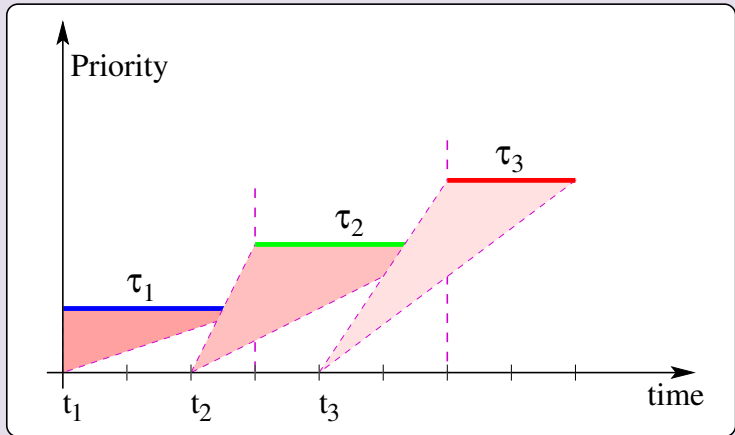
Outline

- 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



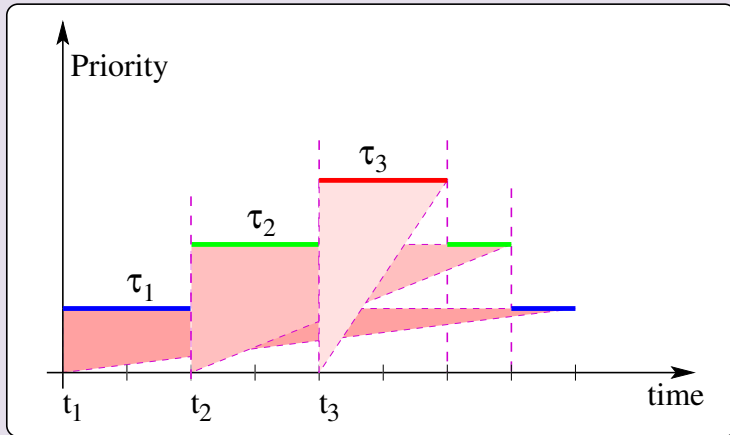
Non-preemptive scheduling

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)

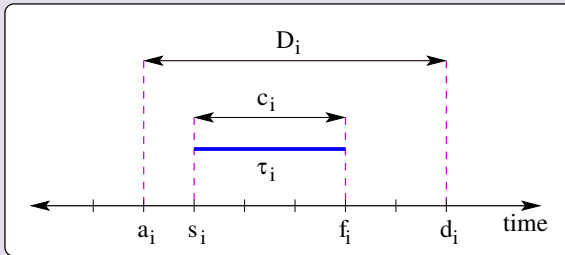
Resource (shared access)

Hard/Soft constraints



Scheduling terms

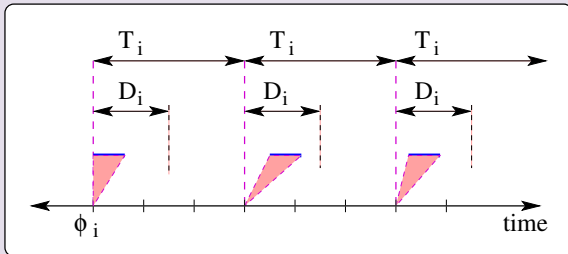
Deadlines:



- 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

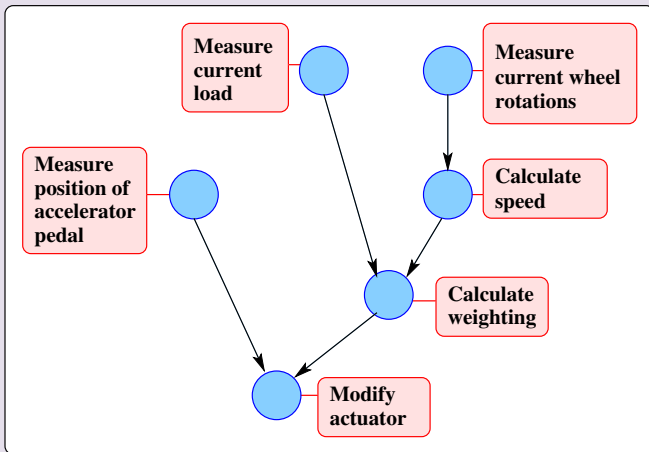
Periodic tasks:



- 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_j is used to control access to an associated CS_j .

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



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:

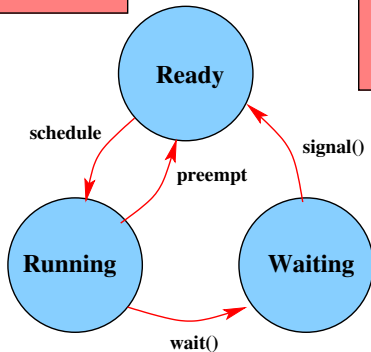
- A task waiting for an exclusive resource is **blocked** on that resource.
- Tasks blocked on the same resource are kept in a **wait queue** associated with the **semaphore** protecting the resource.
- A task in the running state executing **wait(s)** on a locked semaphore ($s = 0$) enters the **waiting** 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:

For each task:

Only 1 running at a time
Queue of Ready tasks
Tasks block on semaphore



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.

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



Assumptions for RMS

In RMS:

- assume a set of tasks $\{\tau_1, \dots, \tau_m\}$ with periods T_1, \dots, 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.

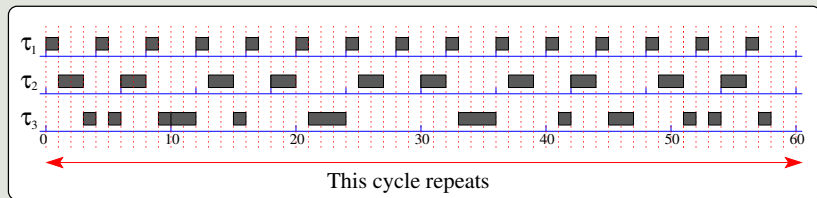
(A5) Any nonperiodic tasks in the system are special; they are initialization or failure-recovery routines; they displace periodic tasks while they themselves are being run, and do not themselves have hard, critical deadlines.

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 is:



RMS

Properties of RMS:

- RMS is **optimal** (Given the previous constraints)
 - If a set 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.



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



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:

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

If this factor is *greater* than 1 then of course, the task set can not be scheduled. However if $U \leq 1$ 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 $U = \sum_{i=1}^m \frac{C_i}{T_i} = 0.833$.

Schedulability

The least upper bound of processor utilization:

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

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

m	1	2	3	4	5	6	∞
U_{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 $U = \sum_{i=1}^m \frac{C_i}{T_i} = 0.833$.

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

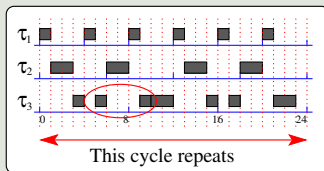
Schedulability

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.



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



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.



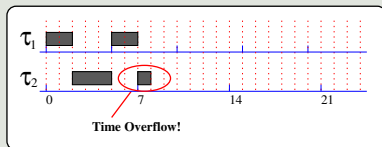
Earliest Deadline First

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

- EDF scheduling succeeds:

