CSC 714: Final Project Report

Preemption Threshold Aware Task Scheduling Simulator

http://www4.ncsu.edu/~sykang/

4/21/2009
North Carolina State University
Sangyeol Kang (sykang@ncsu.edu)
Kinjal Bhavsar (kabhavsa@ncsu.edu)

Overview

Project Description

A task scheduling simulator for timing analysis of the various task sets with different scheduling policies was implemented and tested. The simulator supports the various scheduling policies like Rate Monotonic, deadline monotonic and EDF, widely used for scheduling real time applications. It simulates the task execution for the time period, and then it outputs the time based representation of the scheduling of set of inputs tasks. It also calculates the optimal preemption threshold values for the task sets, and uses it to prevent the unnecessary preemptions.

Motivation

Scheduling real-time tasks is complicated work since it requires pre-computed information about the task's timing properties. Although we know the system's timing properties, scheduling tasks is highly concentrated mental work. Moreover the system designer is responsible to see that system meets real time constraints. For guaranteeing feasibility of generated scheduling, several static and dynamic priority scheduling algorithms and corresponding schedulability tests are introduced such as *Rate-Monotonic* (RM), *Deadline-Monotonic* (DM) and *Earliest Deadline First* (EDF).

For developing a system with real time constraints, the preemptability is considered a necessary requirement. But in reality, there is execution overhead associated with the preemptions. And as the number of tasks in task set are increased, the run time overheads increases exponentially. The *Preemption Threshold Scheduling* (PTS) is suggested to help fixed-priority scheduling to improve the overall system's timing performance by removing unwanted preemptions [2]. The preemption threshold brings one more priority, preemption threshold, to the system. While the regular priority is used when scheduling the task, preemption threshold is used when being preempted by another task. Therefore while meeting task's deadline, we reduce useless preemption, which frequently occurs with traditional fixed priority scheduling. PTS slightly lengthen the execution time of higher tasks, but it reduces that of lower tasks. This helps enhancing the response time and utilization of overall system. Also it improves the scalability of the system.

Introduction

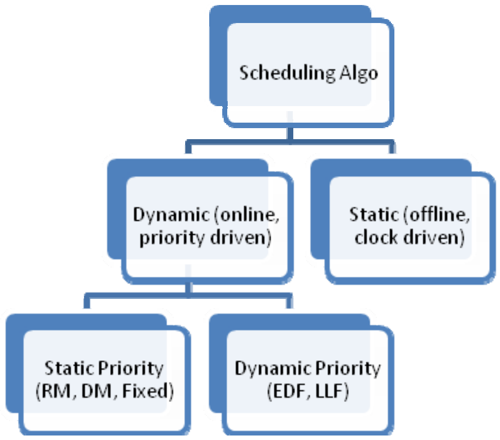
Many embedded systems require hard or soft real-time execution that must meet rigid timing constraints. Further complicating the issue is that for a variety of reasons, most of these same embedded systems have very limited processing power.

Real-time systems theory advocates the use of an appropriate scheduling algorithm and performing a schedulability analysis prior to building the system. Adherence to this theory alone does not lead to working embedded systems, and thus use of this theory is often dismissed by practitioners.

Practitioners, on the other hand, spend days - if not weeks - of testing and debugging hard-to-find and difficult-to- replicate problems because their system is not performing to specifications. Often, these problems are related to the system's timing, because functional testing was done using good tools, and the system usually produces a correct response.

There exists a balance between theory and practice, where proper design of real-time code enables the real-time analysis of it. Systematic techniques for measuring execution time can then be used alongside the guidelines provided by real-time systems theory to help an engineer design, analyze, and if necessary quickly fix timing problems in real-time embedded systems.

Several other activities of the development process can benefit from estimating and measuring execution time using the simulator described here. This includes debugging hard-to-find timing errors that result in hiccups in the system, estimating processing needs of software, and determining the hardware needs when enhancing functionality of an existing system or reusing code in subsequent generations of embedded systems.


Scheduling Algo
Dynamic (online, priority driven)
Static (offline, clock driven)
Static Priority (RM, DM, Fixed)
Dynamic Priority (EDF, LLF)

Also to make real time scheduling possible, there are various scheduling algorithms published. They can be classified as preemptive or non-preemptive, and fixed priority or dynamic priority scheduling. Usually the Fixed priority algorithms are preferred due to their qualities like the ease of use and simple ways to analyze the time response. On the other hand the static priority scheduler cannot provide the optimal utilization of the system. It guarantees the schedulability only under suboptimal utilizations, and above that; the designer needs to hand optimize and simulate the results to see whether it meets real time constraints. For example, general version of RMA cannot commit to the feasibility of a system that uses more than 69% of available CPU time. If the architect falls back on old fashioned time-line analysis, RMA can be pushed all the way to 100% utilization, but we're into tedious hand simulation, not elegant algebra. While the dynamic priority scheduling makes the effective utilization of the resources. But it is very complex and makes the timing analysis really difficult.

Also the common belief is that the preemptive schedulers give better schedulability than non preemptive schedulers. But it is not always true, i.e. it can be proved that under the context of fixed priority scheduling, preemptive schedulers do not dominate non-preemptive schedulers. The schedulability of a task set under non-preemptive scheduling does not imply the schedulability of the task set under preemptive scheduling, and vice-versa. Also for the sake of timing analysis, it is assumed that the context switching costs are zero, while in reality there are runtime overheads associated with it.

So the best way to schedule is having a hybrid policy to take advantage of both preemptive and non-preemptive schedulers. The idea of preemption threshold extends this concept. It introduces another priority level into the system called the preemption threshold. The tasks are scheduled normally according to their priorities. But while running, if another task tries to preempt, then the priority of the new task is checked with the preemption threshold of the running task. This way we can reduce the costs at runtime, but still achieving the optimal usage of the system.

Design

Problem Description

The schedulability can be enhanced by using preemption thresholds. But the questions remains that how to find the optimal assignment of task priorities and preemption thresholds.

We consider a set of N independent tasks, each having its computation time, period and deadline. The tasks are independent i.e. no resource sharing and thus no blocking effects. The tasks do not suspend themselves and the context switching overheads are negligible or zero. We also assign priorities such that two tasks do not have the same priorities. The lower number denotes the higher priority.

The run time model employed with preemption threshold is fixed priority and preemptive scheduling. Whenever task starts running, its priority level is changed to its preemption threshold, and it is preserved at that level until it completes its execution.

Solution Approach

- For a given set of tasks, the priority for each task is assigned according to the scheduling policy decided by the user. The initial preemption thresholds are set by the initial priority.
- Iteratively go through all the tasks in the task set, assign them higher preemption thresholds and compute the response time for the task and higher priority's task (since due to the preemption threshold, higher priority task's execution and scheduling will be affected). If these two tasks are schedulable, raise the preemption threshold.
- Finally simulate the tasks with assigned priorities and find the worst case response times for the each individual task.

Here is the algorithm for assigning the Preemption Thresholds to the tasks.

```
Algorithm: AssignThresholds
// assumes that task priorities are already known
(1) for ( i := n to 1)
(2)     γi := πi                        // start from initial assignment
(3)     Ri := WCRT(γi, πi) ;            // compute initial worst-case response time
(4)     while (Ri < Di) do              // while schedulable
(5)         γi ++ ;                     // increase preemption threshold
(6)         if γi >= n then
(7)             return FAIL;            // not schedulable or reach the highest priority
(8)         endif
(9)         Ri := WCRT(Ti, γi) ;        // recomputed worst-case response time
(10)    end
(11) end
(12) return SUCCESS
```

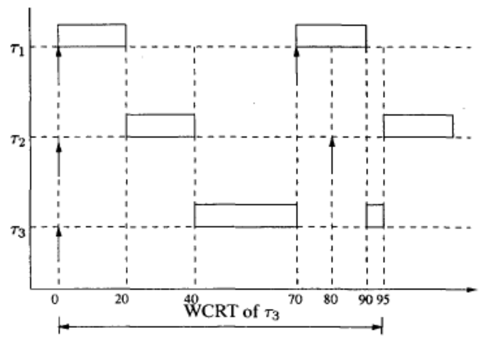
Overall Accomplishments

- Designed and implemented the scheduling simulator which reads the formatted input from user and based on the parameters passed, it schedules the set of tasks for the time period.
- Designed and implemented the Preemption threshold finding algorithm, and integrated it with the simulator.
- Implemented the graphical representation of the clock based scheduling.

Issues Solved

Ready Task Queue

While implementing the simulator, other issues are minor or software bugs which were solved without difficulty except the issue about ready task queue. Initially the simulator has maintained only the current running task's information. But while comparing the simulated results to already published test cases, it was noticed that the single information from the current running task is not enough for the simulation. The triggered example is shown below, which is from [3].



In this example, the tasks are listed in priority order; task1 is the highest and the task3 is the lowest. The preemption threshold values are 1, 1 and 2 respectively for task1, task2 and task3. At 90th scheduler tic, task3's job should be resumed since its preemption threshold value is 2. But since the simulator maintains only the current running task's priority (here 1 from task1) as the system priority, task2 is executed rather than task3 resumes its execution. So the simulator is modified to keep the queue for keeping ready task list.

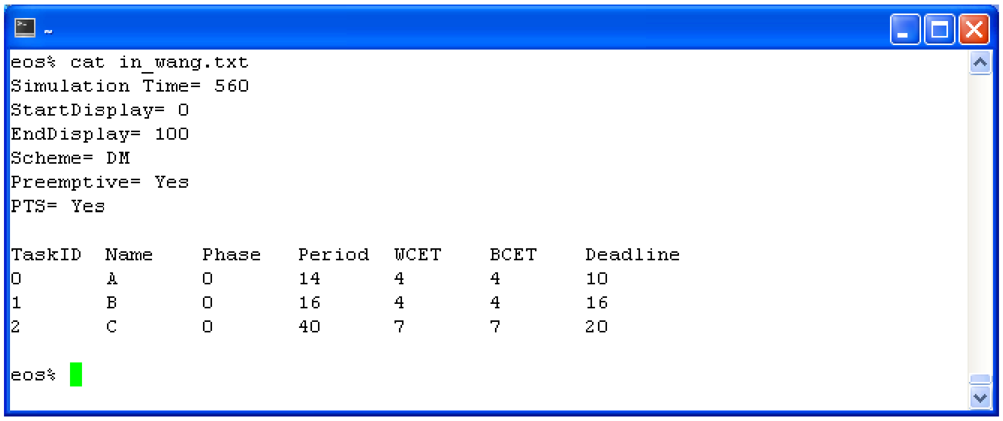
Results

Usage

The simulator can accept a one input file which describes the task set and simulator parameters. For example,

```
> sched in_wang.txt
```

The example of the input file is shown below.



```
eos% cat in_wang.txt
Simulation Time= 560
StartDisplay= 0
EndDisplay= 100
Scheme= DM
Preemptive= Yes
PTS= Yes

TaskID  Name    Phase   Period  WCET    BCET    Deadline
0       A       0       14      4       4       10
1       B       0       16      4       4       16
2       C       0       40      7       7       20

eos%
```

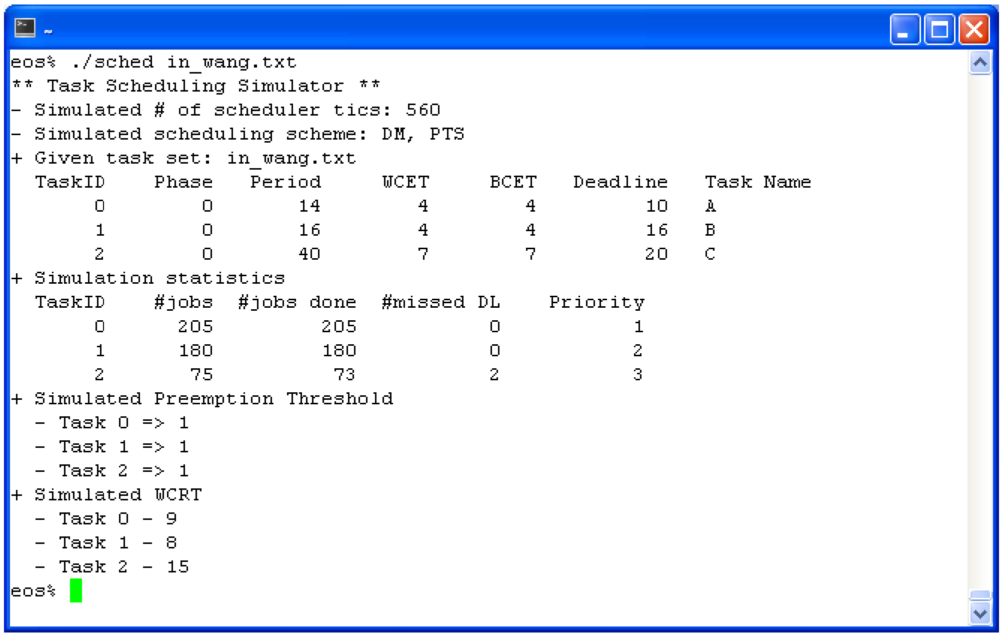
Simulation Time is the total number of scheduler tics for simulation. *StartDisplay* and *EndDisplay* specify the starting point and end point of displaying window for the graphical representation of the simulation in terms of scheduler tics. *Scheme* specifies the simulated scheduling scheme, which is one of *RM*, *DM* and *EDF*. If *Preemptive* is set to *Yes*, the scheduler will allows preemptive scheduling, and vice versa. If *PTS* is set to *Yes*, only for preemptive scheduling, preemption threshold scheduling will be enabled.

After simulator parameters, the target tasks will be given. All values are represented in scheduler tics.

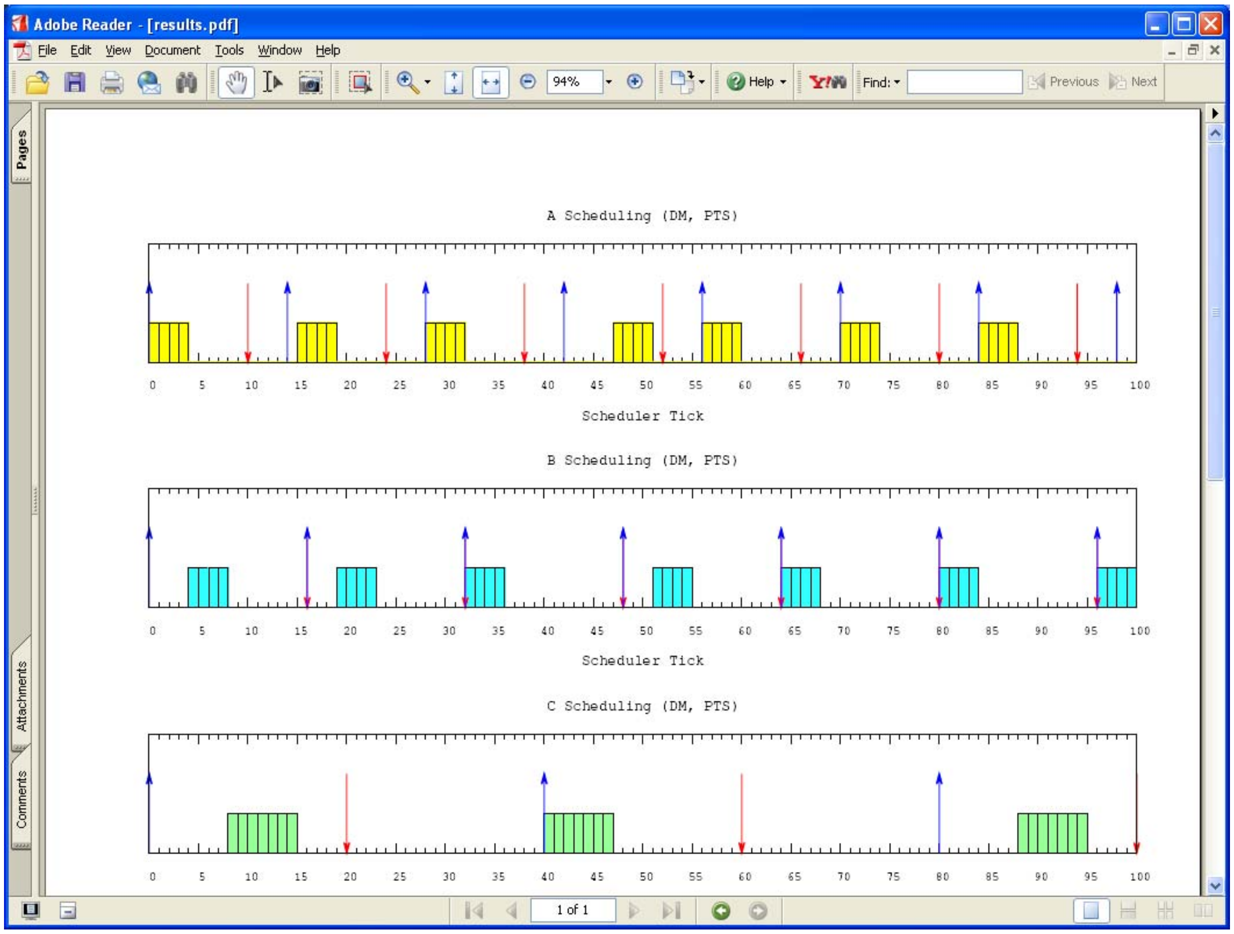
Output

With the given input file, the simulator will generate two kinds of output; simulation statistics and the GNUPLOT script named "plot.do" and the scheduling data, named "TASK-[task ID].dat", which will be fed into the GNUPLOT script. The script will generate the graphical representation of the scheduling in postscript format named "results.ps" with help of GNUPLOT.

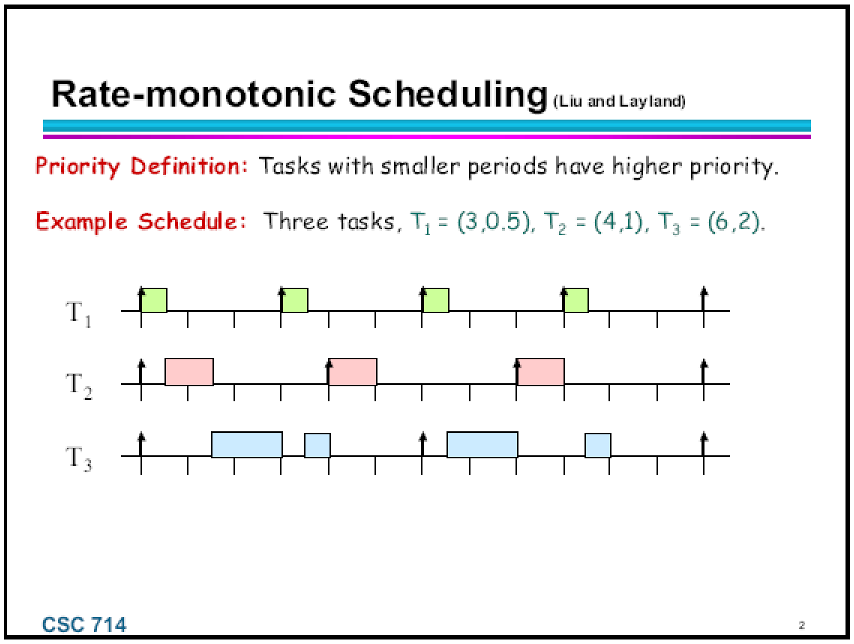
> gnuplot plot.do



```
eos% ./sched in_wang.txt
** Task Scheduling Simulator **
- Simulated # of scheduler tics: 560
- Simulated scheduling scheme: DM, PTS
+ Given task set: in_wang.txt
  TaskID    Phase   Period     WCET     BCET   Deadline   Task Name
       0        0       14        4        4         10   A
       1        0       16        4        4         16   B
       2        0       40        7        7         20   C
+ Simulation statistics
  TaskID    #jobs  #jobs done  #missed DL    Priority
       0      205         205           0           1
       1      180         180           0           2
       2       75          73           2           3
+ Simulated Preemption Threshold
  - Task 0 => 1
  - Task 1 => 1
  - Task 2 => 1
+ Simulated WCRT
  - Task 0 - 9
  - Task 1 - 8
  - Task 2 - 15
eos%
```



To verify the functionality of the implemented simulator, several test cases are test. The above example is from [3]. Another example shown below is from the class lecture note.



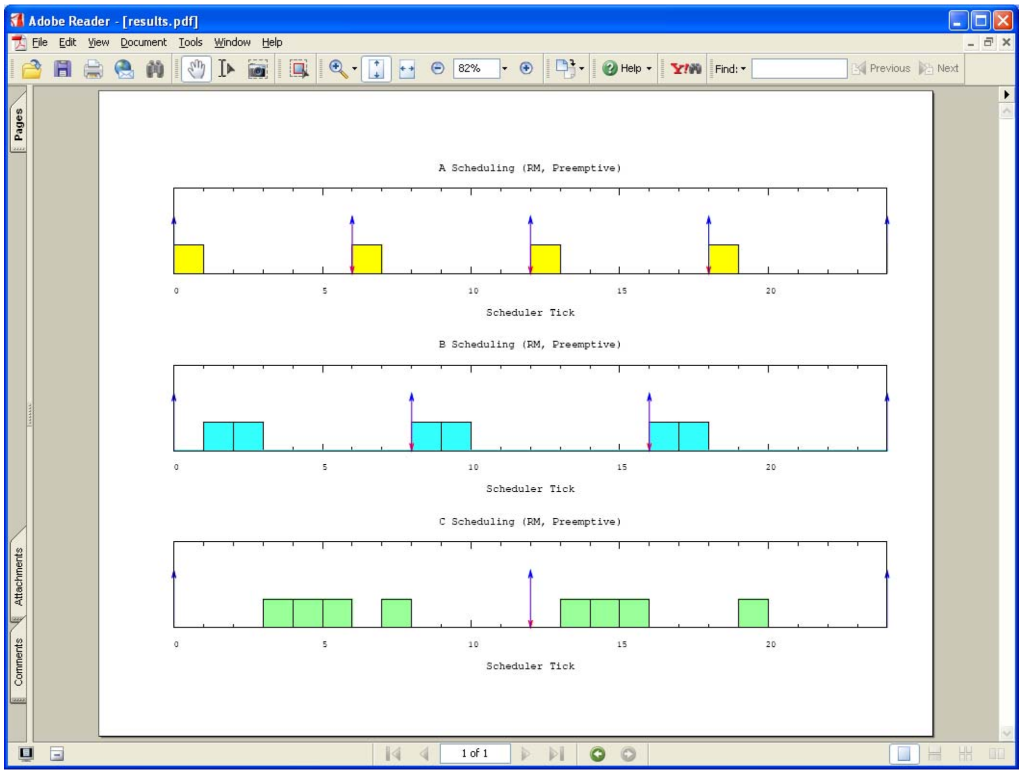
Rate-monotonic Scheduling (Liu and Layland)

Priority Definition: Tasks with smaller periods have higher priority.

Example Schedule: Three tasks, $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (6,2)$.

CSC 714 2

Since the simulator can work only with integer values, all values are multiplied by 2, and the following graph is obtained as the simulation result.



Adobe Reader - [results.pdf]
File Edit View Document Tools Window Help
82%
Help
Find:
Previous
Next
Pages
Attachments
Comments
1 of 1
A Scheduling (RM, Preemptive)
0
5
10
15
20
Scheduler Tick
B Scheduling (RM, Preemptive)
0
5
10
15
20
Scheduler Tick
C Scheduling (RM, Preemptive)
0
5
10
15
20
Scheduler Tick

Individual Contributions

By Sangyeol Kang (sykang)

- Implementation
  Frontend of simulator
  Output generator
  Preemption Threshold Assigning Algorithm
- Design
  Time-based task scheduling simulator for fixed priority scheduling
- Test and verification

By Kinjal Bhavsar (kabhavsa)

- Implementation
  Time-based task scheduling simulator for fixed priority scheduling
  Time-based task scheduling simulator for dynamic priority scheduling
  Integration of individual implementation
- Design
  Time-based task scheduling simulator for fixed priority scheduling
  Time-based task scheduling simulator for dynamic priority scheduling
- Test and Verification

Future Work

The idea of the preemption threshold is limited to the fixed priority systems. It can be extended to dynamic priority scheduling i.e. EDF. The DPT scheduling can effectively reduce context switching by threads assignment and changing task dynamic preemption threshold at runtime. Meanwhile, because the algorithm is based on dynamic scheduling, it can achieve higher processor utilization with relatively low costs in preemption switching and memory requirements. The DPT scheduling can also perfectly schedule a mixed task set with preemptive and non-preemptive tasks, and subsumes both as special cases.

References

[1] Jane W. S. Liu: *Real-Time Systems*, Prentice Hall, 2000 (ISBN-10: 0130996513)

[2] M. Saksena and Y. Wang: Scalable Real-Time System Design Using Preemption Thresholds, In Proceedings of IEEE Real-Time Systems Symposium, pages 25–36, November 2000.

[3] Y. Wang and M. Saksena: Scheduling Fixed-Priority Tasks with Preemption Threshold, In Real-Time Computing Systems and Applications, pages 328-335, December 1999.

[4] D. He, F. Wang, W. Li, and X. Zhang: Hybrid earliest deadline first/preemption threshold scheduling for real-time systems, In proceedings of 2004 International Conference on Machine Learning and Cybernetics, page(s): 433- 438, vol.1 Aug. 2004.