2012-2013 Project 1: EDF-k vs. Global EDF

Cédric Lood
Jean-Luc Zirani

27 novembre 2012

Table des matières

1 Introduction **2**

2 How to use the software ? **2**

3 Implementation details **2**

3.1 Task Generator 2

3.1.1 Data Structure 2

3.1.2 Algorithms 2

3.1.3 Tweak 3

3.2 Simulator 3

3.2.1 Data Structures 3

3.2.2 Algorithms 3

3.2.3 EDF-k 4

3.2.4 GlobalEDF 4

4 Performance simulation **5**

5 Conclusion **5**

A Coding Conventions **6**

B Simulation results **7**

1 Introduction

The present report provides some documentation related to the first project in Real Time Operating Systems (Fall 2012). The purpose of the project was to compare the performances of two multiprocessor schedulers : EDF-k and Global EDF in case of asynchronous and periodic systems with constrained deadlines. The systems considered consist of n periodic, asynchronous and independent tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$

2 How to use the software ?

To compile the software please use the makefile located at the root of the project folder. The output of the compiler is four executables that can be run independently. Please note that the software is command-line oriented and GUI development is not in the pipeline for the moment.

For the *taskGenerator* and the *GlobalEDFvsEDF-k* executables, you can use the *–help* or *-h* to view the argument options that are available. For the *simGlobalEDF* and *simEDF-k* you only need to specify the path to the tasks files (e.g. : ./simGlobalEDF path_to_my_file/tasks).

Please note that a segmentation fault can arrive but we do not have any idea why. GDB says that it occured in glib.

3 Implementation details

In this section, we give some details about the data structures and algorithms used in some selected parts of the code. We mainly focus on the simulator because that is where the core of the project assignment lies.

3.1 Task Generator

3.1.1 Data Structure

A *task* is a *struct* consisting of the following attributes :

- an offset (*m_ iOffset*) that represents the offset in time between the moment the simulation started and the moment when the task entered the system
- a period (*m_ iPeriod*) that represents the periodicity of the jobs that will be submitted by the tasks
- a deadline (*m_ iDeadline*) that represents the deadline at which the job will have to be finished
- a worst case execution time (*m_ iWCET*)

3.1.2 Algorithms

In order to set the period of the tasks, we divide the utilization by the number of jobs chosen. The intuition behind the division is akin to taking a straight line of length equal to the utilization and then randomly placing n-1 separations on the line with the condition that you cannot have 2 separations at the same place. All the intervals now defined (n of them) are then set to the periods of each tasks. We also make sure that there is no single job that has an utilization over 100 percent (otherwise it would not fit on any processor).

Another interesting fact to note is that the deadline is random and belongs to the time interval between WCET and the period.

3.1.3 Tweak

We have chosen the periods to be multiples of 10 in order to avoid that the least common multiple (LCM) would be a number of a very large magnitude. This is a difficulty that we discovered as we were developing the software. For example, in some case we mainly had periods consisting of prime numbers and the LCM was very large.

3.2 Simulator

3.2.1 Data Structures

SJob

- *m_psTask* : a pointer to the task to which the job belongs
- *m_uiStart* : moment at which it arrives in the system
- *m_uiDeadline* : deadline for the calculation
- *m_uiRemCost* : remaining cost (in processor time)
- *m_uiAssignedProc* : id of the processor to which it is assigned
- *m_bPriority* : fixed priority or not for EDF-k (boolean value)

SProc

This data structure is used to represent a processor with the following characteristics

- *m_uiProcId* : an Id specific to the processor.
- *m_uiIdle* : the amount of time that the processor was idle.
- *m_sCurrJob* : assigned job (if any)

CSimulator

This class is used to run the simulator. The following variables are used for the simulation (note that some variables used for statistics are not listed here).

- *m_vecsTask* : the initial vector of tasks (sorted by utilization)
- *m_vecPassive* : vector of jobs that have not yet appeared in the system
- *m_vecWaitJob* : vector containing active jobs that do not have a processor
- *m_vecActiveProc* : vector of processor(s) that execute jobs
- *m_vecIdleProc* : vector of idle processor(s)
- *m_uiInternalTime* : represent the considered moment during the simulation.
- *m_uiMaxTime* : represent the study interval
- *m_bFeasible* : boolean to know if the system is schedulable or not.

3.2.2 Algorithms

We provide here a quick description of the different functions of importance to the simulation of the scheduler. We have chosen to implement a simulator with variable steps.

simulate

This is the core function of the simulator. It consists in a loop that runs until the simulation proves unschedulable or until it has reached the end of the study interval. The end of the study interval is reached when we have gone through the periodicity of the whole batch of tasks. In our case, we stop after 2*(least common multiple) + greatest offset of all the tasks. That gives us some safety that we have at least gone through a full cycle.

init

This function is responsible for sorting the different vectors of passive and active tasks. The passive tasks are the ones that have not yet entered the system. The active tasks are being scheduled. The first thing that we do is to sort the passive tasks by priority (EDF-k and Global EDF).

Note that this function also has the responsability to reset all the vectors and variables of the simulator and then set them properly.

reorganizeSystem

This function has two responsabilities. The first one is to assign idle processors to waiting active jobs that have higher priorities. The second one is to verify that there are no waiting active jobs with higher priorities than at least one of the jobs currently executed. If it is the case, it swaps the waiting active job with the executed job that has the least priority.

updateJobs

Ths function starts by updating the remaining cost of all the executed jobs by substracting the minimal amount that the simulator was able to carry. It then verifies if one or more jobs are finished. In that case, it sets the given processors in idle mode. It finishes by verifying that we have not missed a deadline, which would be considered a failure for the system's schedulability.

3.2.3 EDF-k

How do we set priority ?

If the position of the task is inferior to k then its job has a higher priority then the following tasks' jobs. In order to simplify the code, jobs that have the same priority are sorted by deadline (EDF). Please note that the vector of tasks are already sorted by utilization.

Calculation of k and of the minimum number of processors

Using the formula described in the reference material, we can easily calculate the number of processors and the k parameter that will be needed for a specific system to possibly be schedulable.

3.2.4 GlobalEDF

How do we set priority ?

Global EDF is assimilated to EDF-1. The EDF-k algorithm will set all jobs with the same type of priority (deadline).

Calculation of the number of Processors needed

Using the formula described in the reference material, we can easily calculate the number of processors that will be needed for a specific system to possibly be schedulable.

4 Performance simulation

For each tests we generated systems with different combinations of parameters and then ran several simulations. On these simulations, we got the number of schedulable systems by edf-k and global edf and the average number of processors, preemptions, migrations and finally the total amount of idle time.

The values that we used for utilization are : 120, 150, 180, 220 and 250 and those for the number of tasks are : 8, 10 and 15. This gives us 15 different categories of systems. For each combination, we proceeded with 100 simulations.

See Appendix B for the results or see the "Results" folder (format is "utilization.numberOfTasks.txt)

5 Conclusion

The simulation revealed that in most cases Global EDF has better chances to reach succesful schedulability of the system then its counterpart EDF-k. Nevertheless, when EDF-k can schedule a given system, it usually does it with slightly better performances. The proportion of systems succesfully schedulable by EDF-k is highly dependant on the number of tasks submitted and their utilization. When one or the other increases, the rate of succesful schedulability decreases.

A Coding Conventions

In order to be consistant in the naming of variables throughout the code, we have decided to use the following naming conventions :

Variable Names

The format used is : $[scoping][static][const]$_$[type][nameOfVariable]$

Scoping

- g : global
- m : class variables
- f : functions
- nothing : local to a scope
- static : s
- const : c

Primitives

- vec : vector
- str : std string and C-style string
- ui : unsigned int
- i : int
- OGS : Oppa Gangnam Style

Classes, Structures

- class : CName
- struct : SName
- enum : EName

Examples

- CSimEdf_k : Class named "SimEdf_k"
- m_vecPassive : Class attribute of type vector, named "Passive"
- m_uiMMin : Class attribute of type unsigned int named "MMin"

B Simulation results

```
Utilization target : 120
Nbr of tasks : 8
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 67 86
Avg nbr Core 2 2
Avg Tot Idle 10156 10046
Avg nbr of preemptions 317 345
Avg nbr of migrations 117 135

Utilization target : 120
Nbr of tasks : 10
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 63 88
Avg nbr Core 2 2
Avg Tot Idle 33519 28703
Avg nbr of preemptions 1445 1557
Avg nbr of migrations 396 392

Utilization target : 120
Nbr of tasks : 15
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 57 90
Avg nbr Core 2 2
Avg Tot Idle 93618 85919
Avg nbr of preemptions 4758 5167
Avg nbr of migrations 1320 1494

Utilization target : 150
Nbr of tasks : 8
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 28 78
Avg nbr Core 2 2
Avg Tot Idle 32088 42470
Avg nbr of preemptions 1833 553
Avg nbr of migrations 515 198
```

```
Utilization target : 150
Nbr of tasks : 10
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 15 69
Avg nbr Core 2 2
Avg Tot Idle 13620 30437
Avg nbr of preemptions 1021 1681
Avg nbr of migrations 202 558

Utilization target : 150
Nbr of tasks : 15
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 2 77
Avg nbr Core 2 2
Avg Tot Idle 28440 48054
Avg nbr of preemptions 2317 7900
Avg nbr of migrations 968 2345

Utilization target : 180
Nbr of tasks : 8
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 36 82
Avg nbr Core 2 3
Avg Tot Idle 9630 20571
Avg nbr of preemptions 174 158
Avg nbr of migrations 92 68

Utilization target : 180
Nbr of tasks : 10
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 25 80
Avg nbr Core 2 3
Avg Tot Idle 20657 39793
Avg nbr of preemptions 507 678
Avg nbr of migrations 269 226

Utilization target : 180
Nbr of tasks : 15
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 15 83
Avg nbr Core 2 2
Avg Tot Idle 8260 67594
Avg nbr of preemptions 2638 1949
Avg nbr of migrations 816 894
```

```
Utilization target : 220
Nbr of tasks : 8
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 28 90
Avg nbr Core 3 5
Avg Tot Idle 11586 99666
Avg nbr of preemptions 114 29
Avg nbr of migrations 75 12

Utilization target : 220
Nbr of tasks : 10
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 17 82
Avg nbr Core 3 5
Avg Tot Idle 17542 61588
Avg nbr of preemptions 79 99
Avg nbr of migrations 51 48

Utilization target : 220
Nbr of tasks : 15
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 5 70
Avg nbr Core 3 3
Avg Tot Idle 23322 51027
Avg nbr of preemptions 994 970
Avg nbr of migrations 644 398

Utilization target : 250
Nbr of tasks : 8
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 29 94
Avg nbr Core 3 6
Avg Tot Idle 8894 106625
Avg nbr of preemptions 61 5
Avg nbr of migrations 41 4

Utilization target : 250
Nbr of tasks : 10
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 22 92
Avg nbr Core 3 6
Avg Tot Idle 26074 155236
Avg nbr of preemptions 106 43
Avg nbr of migrations 73 21
```

```
Utilization target : 250
Nbr of tasks : 15
Nbr of systems to test: 100
EDFK GlobalEDF
Schedulable 8 73
Avg nbr Core 3 5
Avg Tot Idle 102526 298061
Avg nbr of preemptions 1298 832
Avg nbr of migrations 582 443
```