ULB UNIVERSITÉ LIBRE DE BRUXELLES

Partitioned EDF scheduling

Real time operating systems

BOUGLAM Sara

11/2020

Table of Contents

1 Introduction

A real time operating system, also referred to as RTOS is an operating system that deals with real time applications and process data as it comes, giving the impressions that everything is executed in the same time, it is characterized by a fixed time constraints where the processing should be accomplished, otherwise, the whole System would fail.

What make the RTOS so different from the typical OS like windows and Unix is the response time, while typical OS provide a non-deterministic soft real time response, RTOS provides a highly deterministic fast hard real time response to the upcoming events.

Preemption refers to the act of interrupting a specific task with intention of resuming its execution, for this, when it comes to switching between several tasks and choosing the appropriate one at a given time, usually RTOS is based on preemptive scheduling algorithm. Several scheduling algorithm can be distinguished, namely: rate monotonic, round robin and fixed priority preemptive scheduling.

In this Report, the focus will be essentially on implementing in python the earliest deadline first (EDF) approach where priorities are assigned online based on the absolute deadline, additionally, implementing the tasks parameter generator as well as tasks partitioning for multicore processors.

2 Background

2.1 Multicore processor:

Multicore processor: is a single computing component comprised one or multiple CPU that execute and read programs instruction. (1)

2.2 Processor scheduling:

The main aim of processor scheduling is to make the system as efficient as possible, this mechanism consist of allocating the processor to a specific task at a specific time

2.3 Tasks

A task can be defined by the following parameters:

Parameter	Description
$\boldsymbol{ti} = (\boldsymbol{Pi}, \boldsymbol{Wceti}, \boldsymbol{Di})$	*The ith computation time of* $\boldsymbol{ti}$
$\boldsymbol{Pi}$	*The period of* $\boldsymbol{ti}$
$\boldsymbol{Wcet\ i}$	*The worst execution time* $\boldsymbol{ti}$
$\boldsymbol{Di}$	*The deadline of* $\boldsymbol{ti}$
$\boldsymbol{Ui} = \boldsymbol{Ci} / \boldsymbol{Ti}$	*The utilization of the task* $\boldsymbol{ti}$
$\mathbf{T} = \{\boldsymbol{t1}, \boldsymbol{t2}, \boldsymbol{t3}, \boldsymbol{tn}\}$	*Task set*
$\boldsymbol{Ui} = \sum_{1}^{n} \boldsymbol{Ui}$	*The task set utilization*

2.4 EDF scheduling (Earliest Deadline First)

EDF scheduling is a dynamic priority scheduling algorithm used in real time operating system s to place processes in a priority queue, whenever, scheduling event occur. (2)

2.5 Bin packing algorithm

In a bin packing problem, items of different volume must be packed into a single or several bin. This algorithm is de characterized by several heuristics such that: first fit, worst fit, best fit and next fit (3).

3 Project Implementation

3.1 Overview

The overall project implementation was based on the OOP principles for the sake of maintainability and readability which consisted of dividing functionalities into a several classes, eventually the whole classes consisted of the following diagram

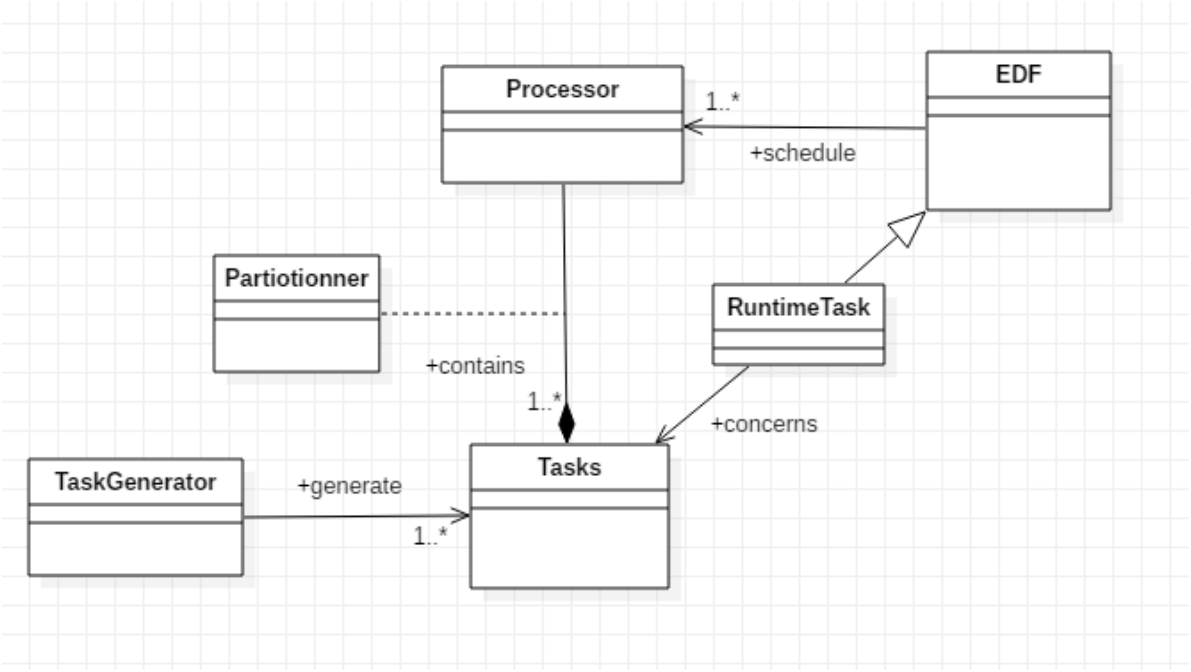

Figure 1: A simplified diagram to represent the whole implementation

The TaskGenerator Class: this class receives as a parameter the global utilization as well as the number of the tasks need and returns a CSV file containing the whole tasks

The Partitioner class: receives as a parameter the tasks file with CSV extension, and has as attributes the tasks array list which are the tasks that would be extracted from the file, and the processors attribute which represents the result of partitioning the whole task, Moreover, this class also include the bin packing algorithm methods such as *FirstFit*, *WorstFit*, *bestFit* and *nextFit*.

The Task class: represents a single task composed of several private attributes accessed by methods.

The Processor class: illustrate single process which may contain the tasks characterized by capacity and tasks attributes.

The EDF class: is the class responsible for scheduling each processor.

3.2 Generating tasks

3.2.1 Utilization generation

Firstly, in order to apply the Scheduling algorithm, a reliable task set is needed for the effectiveness of the whole implementation (excluding manual method which is a tedious job). Generating a task set based on pure randomness which is an unbiased method, may not lead to the intended result

Initially, In order to generate efficiently tasks utilization the *UUnifast* algorithm was used, which comprises of generating uniformly distributed random variables on a given interval with $\boldsymbol{O(n)}$ complexity. (4)

However, the downside of this algorithm is that it can only generate task where the global utilization<= 1, inserting as an input utilization>1 may lead to generate $\boldsymbol{U(Ti) > 0}$ which is not convenient, eventually this algorithm was slightly modified to suit the intended system.

UUnifast-discarded consist of generating tasks using the *UUnifast* algorithm, but discarding the $\boldsymbol{U(Ti) > 0}$ (5)**,** and handling endless loops by some way.

```
def uniFastDiscarded(self):  # implementation of the uuniFast algorithm for
generating the appropriate task set
    sum_u = self.__utilization__
    n = self.__number__ + 1
    i = 1  # the index
    utilization_set = []
    while i < n:
        next_sum_u = sum_u * pow(random(), 1 / (n - 1))
        new_sum_u = sum_u - next_sum_u
        if new_sum_u < 1:  # Discarding the tasks with utilization > 1
            utilization_set.append(new_sum_u)
            i = i + 1  # The loop is only incremented when u<1 in order for
the |vector| = n
            sum_u = next_sum_u
    self.__utilization_set__ = utilization_set
```

3.2.1.1 Observation

As one can see that $\boldsymbol{U(Ti) > 1}$ were not appended to the utilization array, in another word, discarded.

For large data set, this algorithm may not be efficient which is considered as a weak spot for this last, however, *Stafford random fixed sum*[1] might be used instead (6)

3.2.2 Other parameters generation

For generating the other parameter $\boldsymbol{Pi = Di}$ where periods were generated according to a log-uniform distribution on a specific interval band, such that, for a random variable$\boldsymbol{x}$, $\boldsymbol{ln(x)}$ has uniform distribution.

[1]**Stafford random fixed sum**: generates a set of vectors which are uniformly distributed in (N-1) dimensional space with a constant given sum. The vectors represent the utilization factors of the tasks and are limited between 0 and 1[2] (6)

$\boldsymbol{Wceti}$ was easily obtained through$\boldsymbol{Ti} * \boldsymbol{Ui}$.

As a final step, the tasks were transferred to a CSV file [2] for a later use in scheduling.

3.3 Tasks partitioning

Considering that, the Bin packing algorithm was used for partitioning tasks into processors, there 5 distinguishable methods used for this, and are as follows:

3.3.1 Next Fit

For the next fit algorithm, only the last added processor is taken into consideration, if it doesn't fit then a new processor is added which corresponds to the next one, this is illustrated by the pseudocode provided:

Algorithm 1 next fit

procedure NE TF.

```
p →processor                                  ▷ initialize the processor
processors → []
for task ∈ tasks do
    if task.utilization ≥ 1 then
        continue
    end if
    if p.addTask(tasl ) == 0 then             ▷ when tasl doesn't fir in p
        processors.add(p)
        p →processor                          assigning a new processor to p
        p.addTask(tasl )
    end if
end for
return processors
```

Figure 2 : Next Fit Algorithm

3.3.2 First Fit Algorithm

For a given list of processors, the first processor, the task is added to the first processor fits in, else a new one is created and the task is added to it, this idea, can be illustrated such that:

[2] **CSV files (Comma Separated Values):** are comma delimited text files where each line represent the a record, and each record is characterized by fields that are separated by commas (9)

Algorithm 1 First fit

```
procedure FIRSTF
    p →processor                                  ▷ initialize the processor
    processors → []
    processors.add(p)                             ▷ initializing the processors array
    added → false                                 ▷ keeping trac! whether the tasl was added
    for task ∈ tasks do
        if task.utilization == 1 then
            continue
        end if
        for processor ∈ processors do
            if processor.addTask(tasl ) == 1) then
                added → true
                breal
            end if
        end for
        if added == false then
            p → processor
            p.addTask(tasl )
            processors.add(p)
        end if
    end for
return processors
```

Figure 3 First fit algorithm

3.3.3 Worst and best fit

The algorithm for both Worst and best fit is similar, considering that its principle relies on the idea of assigning tasks to processors with that are best or worst or fit, eventually the processors are constantly sorted in each iteration in a decreasing manner for best fit and in an increasing manner for worst fit. For further clarification to this idea, the pseudocode below is given:

Algorithm 1 Best Fit and Worst Fit

```
procedure BES WORSTF (order)
    p →processor                                  ▷ initialize the processor
    processors → []
    processors.add(p)                             ▷ initializing the processors array
    added = false                                 ▷ keeping trac! whether the tasl was added
    for task ∈ tasks do
        if task.utilization > 1 then
            continue
        end if
        sort(processors,order)                    ▷ sorting processors according to the capacity
        for processor ∈ processors do
            if processor.addTask(tasl ) == 1) then
                added → true
                breal
            end if
        end for
        if added == false then
            p → processor
            p.addTask(tasl )
            processors.add(p)
        end if
    end for
return procecessors
end procedure
```

Figure 4: Pseudocode for both worst fit and best fit partitioning

3.4 Task scheduling

EDF scheduling was used to schedule the tasks on each processor. The implementation of this algorithm consisted of creating an extra internal class entitled $RuntimeTask$ which holds as attributes: a *task*, the *number of* executions i *and the absolute deadline*, which would be updated in each iteration:

The absolute deadline (upcoming deadline of a task) is calculated such that: $\boldsymbol{absDeadline = (i + 1) \times periodi}$

Another point to consider is that the execution time (the launch) of each task is constantly updated, for this, a method is added to the $RuntimeTask$ class, entitled $computeExecution$ time; this method consists of estimating the suitable execution time such that no task can be executed more than once in its period Pi, to illustrate this

mechanism, the above pseudocode is given which corresponds to scheduling tasks on a single processor:

Algorithm 1 Schedule one processor

```
procedure SCHEDULEONE(processor,interval)
    queue → []
    for task ∈processor.tasks do
        t− >runtimeTask(task, 0, task.deadline)
        queue.add(t)
    end for
    time → 0
    while time ≤interval do
        tasks = min(queue , time)
                                ▷ Selecting the minimum absolute deadline
        for task ∈tasks do
            time = task.computeExecution(time)
            task.i →task.i + 1
            task.absDeadline →tasks.period ×(task.i + 1 )
            time →time + task.task.Wcet
        end for
    end while
end procedure
```

Figure 5 a simplified algorithm that describes scheduling a processor

Observation: As seen in the above, several tasks were returned from the min methods which refers to selecting also the tasks with equal absolute deadline in order for them to be executed simultaneously

3.5 External libraries

3.5.1 Scipy

Scipy is a library that offers several numerical routines such as numerical integration, interpolation, optimization, linear algebra and statistics. During task generation process, Scipy library was used to generate period parameters according to a Log uniform distribution (7)

3.5.2 Matplotlib

Matplotlib is a python plotting library, that provides object oriented API to embed plots into application (8).

During the scheduler implementation finalization, this library was helpful to visualize the tasks on the given interval.

3.6 Program execution

For the program execution the argument below should be provided:

$$-\boldsymbol{u} < \boldsymbol{number} > -\boldsymbol{n} < \boldsymbol{number} > -\boldsymbol{h}\ \boldsymbol{ff}|\boldsymbol{wf}|\boldsymbol{bf}|\boldsymbol{nf} -\boldsymbol{s}\ \boldsymbol{iu}|\boldsymbol{du} -\boldsymbol{l} < \boldsymbol{limit} >$$

$-\boldsymbol{u}$ Represents the Global utilization

$-\boldsymbol{n}$ Represent the number of tasks to display

$-\boldsymbol{h}$ Defines heuristics, $\boldsymbol{ff}$ for First fit, $\boldsymbol{wf}$ for worst fit, $\boldsymbol{bf}$ for best fit and $\boldsymbol{nf}$ for next fit,

$-\boldsymbol{s}$ Defines the order of tasks, either increasingly $\boldsymbol{iu}$ or decreasingly $\boldsymbol{du}$

$-\boldsymbol{l}$ Refers to the time interval where the task would be executed

Note: preferably, for the time interval, insert an input between 1 and 4 to get a better view of plotting execution

Make sure to respect the syntax of these arguments, else the program won't work as intended

In case where you are executing from the command line make sure Scipy and Matplotlib are installed[3]

When executing the program, make sure that you are in same the folder as Main.py class

3.6.1 Example Execution

Here is an example of executing the program from the command line, typed using the pyCharm editor's terminal

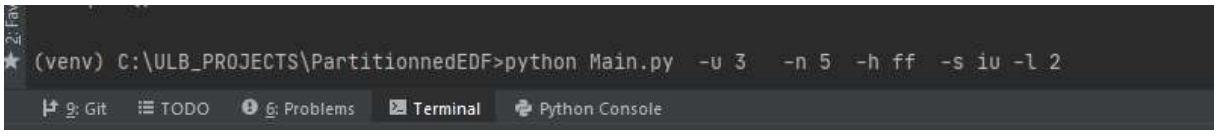

Figure 6: Example execution

This would output:

[3] You can install Scipy and Matplotlib as mentioned in this page
https://solarianprogrammer.com/2017/02/25/install-numpy-scipy-matplotlib-python-3-windows/

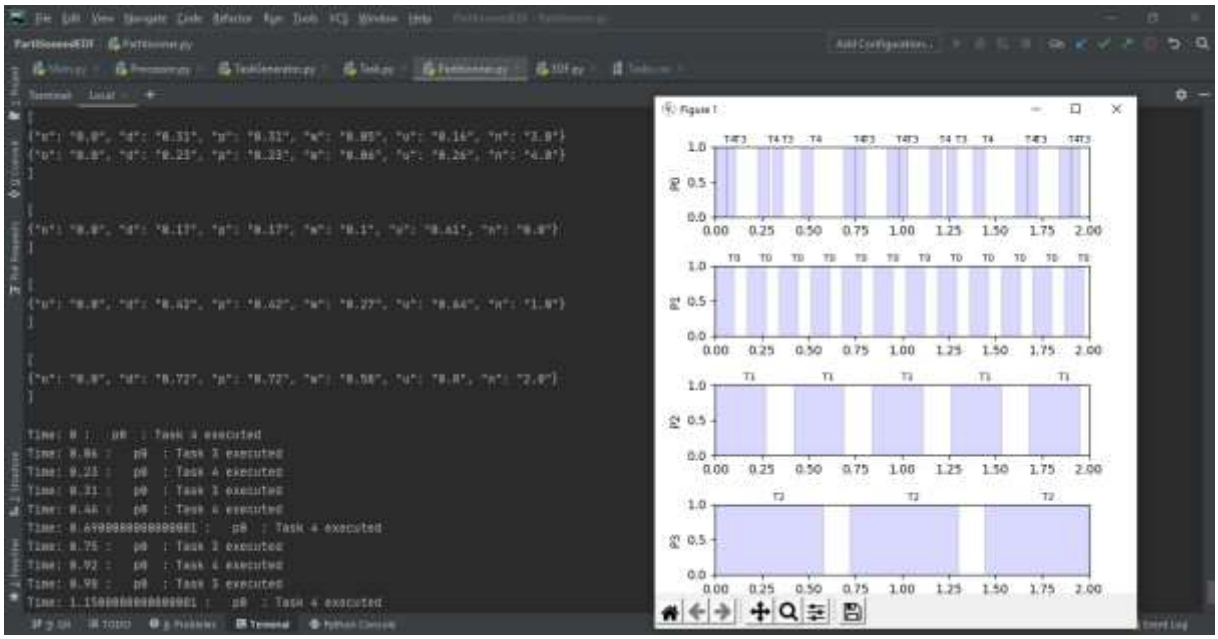

Figure 7: output Example

Observation: In the right is the result plotting of Matplotlib and in the left is simply standard output stream; in the top portion there is task partitioning and on its bottom, there is task execution, you can scroll down to access the full view

You can display the tasks file through the Windows and Mac command type or the UNIX command cat, as shown below (make sure access the $mainPackage$ folder):

```
(venv) C:\ULB_PROJECTS\PartitionnedEDF\mainPackage>cat Tasks.csv
offset,deadline,period,wcet,utilization,number
0,0.17,0.17,0.07,0.41,0
0,0.32,0.32,0.03,0.08,1
0,0.49,0.49,0.39,0.79,2
0,0.65,0.65,0.04,0.07,3
0,0.15,0.15,0.09,0.61,4

(venv) C:\ULB_PROJECTS\PartitionnedEDF\mainPackage>type Tasks.csv
offset,deadline,period,wcet,utilization,number
0,0.17,0.17,0.07,0.41,0
0,0.32,0.32,0.03,0.08,1
0,0.49,0.49,0.39,0.79,2
0,0.65,0.65,0.04,0.07,3
0,0.15,0.15,0.09,0.61,4
```

Figure 8: Task display from the command line

Note that in every execution the Tasks.csv file is modified which leads to different results each time, however, in the case where keeping the previously defined file tasks set is needed, then setting $-\boldsymbol{n}\ \mathbf{0}$ and $-\boldsymbol{u}\ \mathbf{0}$ would allow it , here is an example for that, below :

```
(venv) C:\ULB_PROJECTS\PartitionnedEDF>python Main.py  -u 0   -n 0  -h ff  -s iu -l 2
```

Figure 9: Input example

4 Conclusion

To sum up, the task generation problem has been treated although not fully optimal, nevertheless, totally convenient for a limited small dataset, then, the task partition into processors, where the bin packing problem has been introduced accompanied with some heuristics; online ones that refer to the algorithms namely first fit, worst fit, best fit and next fit, and offline one which correspond to tasks sorting. As final step, the EDF algorithm scheduling has been brought up as well as implemented.

EDF algorithm is considered optimal whenever the feasibility criteria is satisfied, without the need to define the priorities offline, however, this may also result in downsides such as unpredictability of tasks execution and processor overhead .

5 Bibliography

1. Computer Hope . *Multicore processor.* [Online] https://www.computerhope.com/jargon/m/multcore.htm#:~:text=A%20multicore%20processor%20is%20a,advantage%20of%20the%20unique%20architecture..

2. Earliest deadline first scheduling. *Wikipedia.* [Online] https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling.

3. Wikipedia. *Bin packing problem.* [Online] https://en.wikipedia.org/wiki/Bin_packing_problem.

4. *Measuring the Performance of Schedulability Tests.* **ENRICO BINI, GIORGIO C. BUTTAZZO.** 2005.

5. *Improved priority assignment for global fixed priority.* **Burns, Robert I. Davis · Alan.** 2010.

6. *Efficient data generation for the testing of real-time multiprocessor scheduling algorithms.* **M. Naeem Shehzad1, A.M. Déplanche2, Yvon Trinquet2, Umer Farooq1.**

7. Scipy. [Online] https://scipy.org/scipylib/.

8. Matplotlib. [Online] https://matplotlib.org/3.3.3/tutorials/index.html.

9. Wikipedia. *Comma separated values.* [Online] https://en.wikipedia.org/wiki/Comma-separated_values.