VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering
Digital Assignment, JUNE 2020
B.Tech, Winter-2019-2020

TITLE	Classful EDF Scheduling Algorithm
COURSE CODE	ITE2002
COURSE NAME	OPERATING SYSTEMS
SLOT	D1+TD1
FACULTY	Prof. SUDHA S

TEAM DETAILS:

REG. NO.	NAME
18BIT0226	HRITISHA
18BIT0253	ABHINABA MITRA
18BIT0272	PRIYAL BHARDWAJ
18BIT0249	ROHAN JAIN

Classful EDF Scheduling Algorithm

Abstract — **Earliest Deadline First Scheduling Algorithm (EDF) is widely considered the best scheduling algorithm for a uniprocessor system. However, the problem in the original EDF algorithm is that in transient overload condition if one process misses its deadline then other processes are also very likely to miss their deadline, such an effect is called the Domino Effect. Since the different process have different priorities so we classify those different process on basis of 3 kind of priorities namely Low-class (lowest Priority), Mid-Class, High-Class (Highest Priority). In original EDF it is very likely that if a "low-class" process misses its deadline then it causes a "high-class" process to miss its deadline too because of the domino effect during the transient overload condition. Whereas Classful EDF algorithm combines elements of Priority Scheduling Algorithm and time slicing property of Round-Robin Scheduling Algorithm into the existing EDF Scheduling Algorithm. The resultant is a scheduling algorithm which solves the above-mentioned problem. Another main advantage of Classful EDF is that it also reduces the number of processes which miss their deadline. In this algorithm the state of the processes is constantly checked for any possible process about to miss its deadline, if any such process is found then appropriate actions are taken depending upon the "class" of the process and the state of the other processes being executed.**

Keywords— EDF, Domino Effect, classes, intentional delay

I. Introduction

Earliest Deadline First (EDF) is a dynamic priority scheduling algorithm for real time operating systems. Earliest Deadline First selects a task according to its deadline such that a task with the earliest deadline has higher priority than others. It means priority of a task is inversely proportional to its absolute deadline. Since absolute deadline of a task depends on the current instant of time, so every instant is a scheduling event in EDF as deadline of task changes with time. A task which has a higher priority due to earliest deadline at one instant it may have low priority at next instant due to early deadline of another task. EDF typically executes in pre-emptive mode
i.e. currently executing task is pre-empted whenever another task with earliest deadline becomes active. It is an optimal algorithm which means if a task set is feasible then it is surely scheduled by EDF. Another thing is that EDF does not specifically take any assumption on periodicity of tasks so it is independent of Period of task and therefore can be used to schedule aperiodic tasks as well. If two tasks have same absolute deadline choose one of them on first-come-first-served basis.

Transient over load is a short time over load on the processor. Transient overload condition occurs when the computation time demand of a task set at an instant exceeds the processor timing capacity available at that instant. Due to transient over load tasks miss their deadline. This transient over load may occur due to many reasons such as changes in the environment, simultaneous arrival of asynchronous jobs, system exception. In real time operating systems under EDF, whenever a task in Transient overload condition misses its deadline then as result each of other tasks may start missing their deadlines one after the other in a sequence, such an effect is called domino effect which jeopardizes the behaviour of the whole system.

Research works done in [1], [2], and [3] tackle the issue of domino effect in EDF Scheduling. In the above-mentioned research papers attempts have been made to reduce the domino effect in EDF scheduling as a whole but in our proposed algorithm we have countered the domino effect in a case-by-case manner where the specific response to a process missing its deadline depends upon its "class", to not only minimize the number of processes missing their deadline but also to make sure that the more important processes are least affected.

II. Domain

Our Classful EDF aims to be applied on the same domain as the Original EDF.

Although EDF implementations are not common in commercial real-time kernels, here are a few open-source and real-time kernels implementing EDF:

SHARK: The SHaRK RTOS, implementing various versions of EDF scheduling and resource reservation scheduling algorithms

ERIKA Enterprise: ERIKA Enterprise, which provides an implementation of EDF optimized for small microcontrollers with an API similar to the OSEK API.

The Everyman Kernel: The Everyman Kernel implements either EDF or Deadline Monotonic scheduling depending on the user's configuration.

MaRTE OS: MaRTE OS acts as a runtime for Ada applications and implements a wide range of scheduling algorithms including EDF.

The AQuoSA project constitutes a modification to the Linux kernel enriching the process scheduler with EDF scheduling capabilities. The timing of the scheduling cannot be as precise as in the case of the above hard real-time Operating Systems, yet it is sufficiently precise so as to greatly enhance predictability, and thus fulfil the real-time requirements, of multimedia applications. AQuoSA is one of a few projects that provides real-time scheduling capabilities to unprivileged users on a system in a controlled way, by means of a properly designed access-control model.[2]

The Linux kernel has an earliest deadline first implementation named SCHED DEADLINE which is available since the release 3.14.

The real-time scheduler developed in the context of the IRMOS European Project is a multi-processor real-time scheduler for the Linux kernel, particularly suitable for temporal isolation and provisioning of QoS guarantees to complex multi-threaded software components and also entire virtual machines. For example, when using Linux as host OS and KVM as hypervisor, IRMOS can be used to provide scheduling guarantees to individual VMs and at the same time isolate their performance so as to avoid undesired temporal interferences. IRMOS features a combined EDF/FP hierarchical scheduler. At the outer level there is a partitioned EDF scheduler on the available CPUs. However, reservations are multi-CPU, and global FP over multi-processors is used at the inner level in order to schedule the threads (and/or processes) attached to each outer EDF reservation. See also this article on lwn.net for a general overview and a short tutorial about the subject.

Xen has had an EDF scheduler for some time now. The man page contains a short description.

The Plan 9 OS from Bell Labs incorporates EDFI, a "lightweight real-time scheduling protocol that combines EDF with deadline inheritance over shared resources."[3]

RTEMS: The EDF scheduler will be available in version 4.11. RTEMS SuperCore

Our algorithm is also applicable on Switched Real-Time Ethernet. The role of our algorithm is for communication in the beginning between the nodes. If original EDF is used and there is a deadline miss then results could be catastrophic if the RTOS is a hard or firm RTOS, due to the domino effect of the EDF. But our modified EDF is especially designed to minimize the Domino effect.

Classful EDF can also be used in Real-Time Traffic handling in LTE Networks.

III. Terminology And assumptions

Before we describe the scheduling algorithm, we define the following notation to refer the parameters of task:

1. Bt_i means "burst time of the process [i]"
2. Ct means "current time"
3. Dt_i means "deadline time of process [i]"
4. Dt_{max} means "maximum deadline among the deadlines of all processes at a given instance"

Moreover, we assume that tasks are preemptable.

IV. Original EDF Scheduling Algorithm

Original EDFS algorithm works on continuous time. The algorithm assumes, whenever the execution of a job is ended, the next job can start without wasting any time. Whenever CPU becomes idle, the algorithm runs and selects the job with the earliest deadline. The algorithm does not avoid deadline-misses. Thus, execution of a job can be started at current iteration, even though there is not enough time for it to finish in that iteration. This leads to deadlinemisses. The original EDFS algorithm is optimal [11]. With the continuous time assumption, original EDFS algorithm can schedule any set of jobs if the scheduling is possible. For this reason, original EDFS algorithm is basis for our modified scheduling algorithm.

V. Classful EDF Scheduling Algorithm

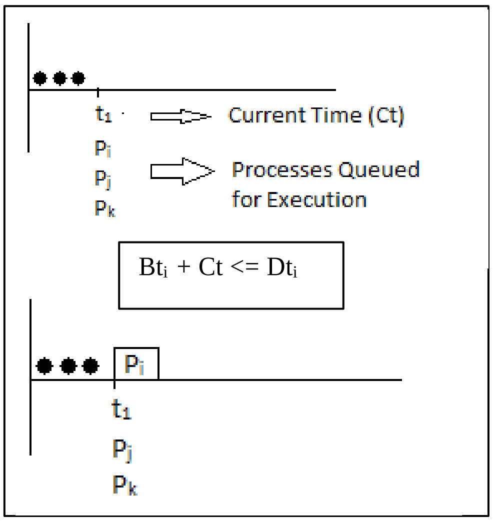
As the name suggests, processes are first segregated into classes, specifically into three classes which are:

1) *High-class – Processes which cannot afford any amount of delays possible.*
2) *Mid-class – Processes that can afford some delays.*
3) *Low-class – Processes that can afford to have long delays.*

Classful EDF Scheduling Algorithm works similar to the original Earliest Deadline First Scheduling Algorithm , the major difference is that it constantly checks whether the process being executed is going to miss its deadline or not.

It uses this formula whenever a new process starts its execution:

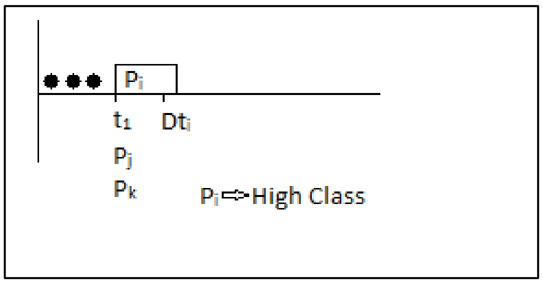
$$Bt_i + Ct \leq Dt_i$$



Original EDF Scheduling when equation is satisfied

As long as the formula returns value TRUE the algorithm will continue to work as the original Earliest Deadline First Algorithm. Once the formula returns the value FALSE for a process then it checks the process' "class".

If the process, that will miss its deadline, belon gs to the "high-class" then it is allowed to execute without any modifications, because as stated before high-class processes are the one that cannot afford any major delay, so if a high-class process is about to miss its deadline then it is only logical to let it execute and handle the domino effect in the upcoming processes.



"High-Class" process missing its deadline

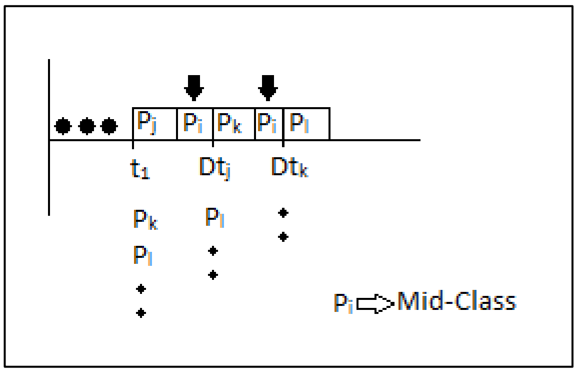
If the process, that will miss its deadline, belongs to the "mid-class" then it is removed from the queue and other processes are allowed to execute. When there comes a case in which:

$$Bt_i + Ct < Dt_i$$

(i.e. there will be some time left after the execution of the process before its deadline is reached) then a part of the "mid-class" process that can be processed within:

$$Bt_i + Ct - Dt_i$$

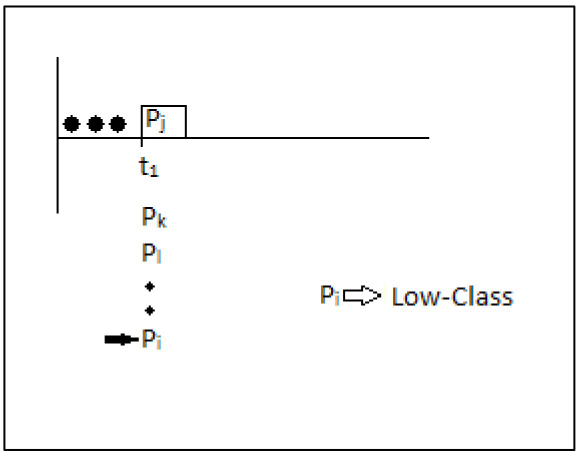
will be allowed to execute, this will happen until the "midclass" process is completely executed. If more than one midclass processes missed their deadline then they will be put in a separate queue and they will have to wait till previous mid-class process which missed its deadline is completely executed, then the process at the front of the queue will be allowed to be executed in the similar manner.



"Mid-Class" process missing its deadline

If the process, that will miss its deadline, belongs to the "low-class" then it is simply moved to the end of the queue by giving it a new deadline which would be:

$$(new)Dt_i = Dt_{max} + Bt_i$$



"Low-Class" process missing its deadline

VI. Original EDF v/s Classful EDF

Parameters of Evaluation: Classful EDF is being Evaluated on two parameters

1. Failure Ratio: Traditional FR is calculated by number of processes that miss their deadline divided by total number of processes.
2. Failure Ratio (Classes Considered) : FR_C is a method developed just to calculate effectiveness of Classful EDF. It is used because as mentioned before "all process are not equal" thus if high-class process misses its deadline then it should be considered a higher failure than if a low-class process misses its deadline.

It is calculated by:

$$\frac{\sum(\text{Process that missed its deadline}) * (\text{Class of the process})}{\sum\ (\text{All Processes}) * (\text{Class of the respective processes})}$$

Respective values of each Class:

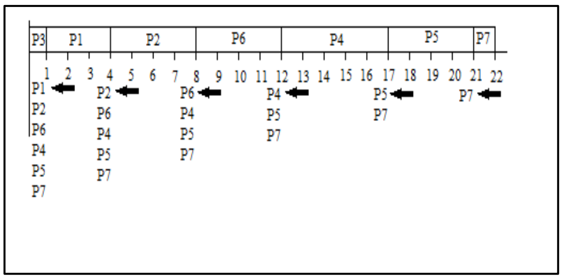
High-Class	3
Mid-Class	2
Low-Class	1

Assumption: All the Processes arrive simultaneously (i.e. Arrival time for all is "0")

Processes	Burst Time (Bt_i)	Deadline Time (Dt_i)	Class (Case 1)	Class (Case 2)
P1	3	5	Mid	Low
P2	4	7	High	Mid
P3	1	3	Low	Low
P4	5	16	Low	Mid
P5	4	20	Low	High
P6	4	12	High	Low
P7	1	21	Mid	Low

- Original EDF Scheduling Algorithm→

Priority Queue : P3, P1, P2, P6, P4, P5, P7



Original EDF Scheduling

As we can clearly see that Process P2 misses it's deadline which causes a domino effect forcing processes P4, P5, and P7 to miss their deadline too.

FR = 4/7 = 0.571428571

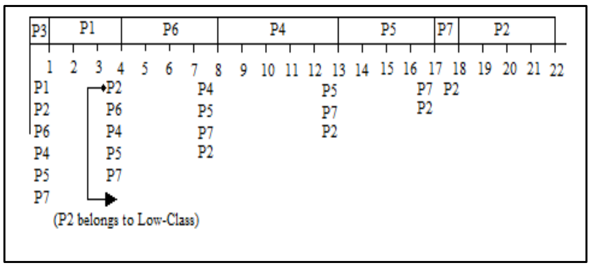
FR_C (Case 1) = (1+1+3+2)/(2+3+1+1+1+3+2) = 7/13 = 0.538461538

FR_C(Case 2) = (2+2+3+1)/(1+2+1+2+3+1+1) = 8/11 = 0.727272727

- Classful EDF Scheduling Algorithm→

Case 1

Priority Queue : P3, P1, P2, P6, P4, P5, P7



Classful EDF Scheduling (Case 1)

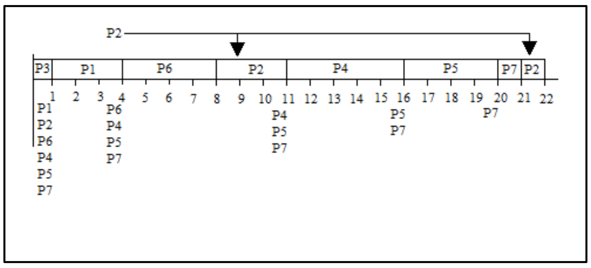
As we can see that process P2 was about to miss its deadline but according to Classful EDF when a low-class process is about to miss its deadline it moves to the end of the priority queue and prevent a domino effect from taking place.

FR = 1/7 = 0.142857143

FR_C = 1/13 = 0.0769230769

Case 2

Priority Queue : P3, P1, P2, P6, P4, P5, P7



Classful EDF Scheduling (Case 2)

FR = 1/7 = 0.142857143

FR_C = 2/11 = 0.181818182

Results→

- FR and FR_C of Classful EDF is less than Original EDF Scheduling Algorithm
- Number of Context Switches is more in Classful EDF than the Original EDF Scheduling Algorithm
- Complexity of implementation of Classful EDF algorithm is greater than Original EDF

PSEUDOCODE

```
while(TRUE)
{
    if(CPU.free())
    {
        if(!temp)
            temp=Mid_misses_queue.dequeue();
        if($Bt_i$ + Ct <= $Dt_i$)
        {
            //  Oiginal EDF Algorithm
            Ct = Ct + $Bt_i$;
            if(temp && Ct < $Dt_i$)
            {
                //execute temp from Ct till $Dt_i$
                // or until temp is completed
            }
        }
        else
        {
            switch(process[i].class())
            {
            case "high-class":
                {
                Execute(process[i]);
                break;
                }
            case "mid-class":
                {
                if(!temp) temp=process[i];
                else
                {
                Mid_misses_queue.enqueue(process[i]);
                break;
                }
            case "low-class":
                {
                EDFpriority_queue.enqueue(process[i]);
                //process sent at the end of the EDF queue
                process[i].setDeadline($Dt_{max}$+$Bt_i$);
                break;
                }
            }
        }
    }
}
```

REFERENCES

[1] Chenyang Lu , John A. Stankovic, Gang Tao and Sang H. Son, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm"

[2] Rashmi Sharma and Nitin, "Performance Evaluation of New Joint EDF-RM Scheduling Algorithm for Real Time Distributed System", Hindawi Publishing Corporation, Journal of Engineering, Volume 2014, Article ID 485361, 13 pages http://dx.doi.org/10.1155/2014/485361

[3] Giorgio Buttazzo, Marco Spuri and Fabrizio Sensini, "Value vs. Deadline Scheduling in Overload Conditions"

[4] Erhan Okuyan and Barış Kayayurt, "Earliest Deadline First Scheduling Algorithm and its use in ANKA UAV"

[5] S. Baruah, S. Funk and J. Goossens, "Robustness Results Concerning EDF Scheduling upon Uniform Multiprocessors", IEEE Transaction on computers, Vol. 52, No.9 pp. 1185-1195 September 2003.

[6] E.P.Markatos and T.J. LeBlanc, " Load Balancing versus Locality Management in Shared-Memory Multiprocessors", The 1992 International Conference on Parallel Processing, August 1992.

[7] S. Lauzac, R. Melhem and D. Mosses, "Compression of Global and Partitioning Scheme for Scheduling Rate Monotonic Task on a Multiprocessor", The 10th EUROMICRO Workshop on Real-Time Systems, Berlin, pp.188-195, June 17-18, 1998.

[8] Vahid Salmani and Mohsen Kahani, "Deadline Scheduling with Processor Affinity and Feasibility Check on Uniform Parallel Machines", Seventh International Conference on Computer and Information Technology, CIT.121.IEEE,2007.

[9] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem. Operations Research", 26(1):127–140, 1978.

[10] Y. Oh and S. Son. "Allocating fixed priority periodic tasks on multiprocessor systems", Real-Time Systems Journal, 9:207–239, 1995.

[11] John A. Stankovic, "Deadline Scheduling for Real-Time Systems EDF and Related Algorithms"

[12] Mahmoud M. Hamed, Suzan Shukry, Mohamed S. El-Mahallawy and Salwa H. El-Ramly, "Modified Earliest Deadline First Scheduling with Channel Quality Indicator for Downlink Real-Time Traffic in LTE Networks"

[13] Deming Liu and Yann-Hang Lee," An Efficient Scheduling Discipline for Packet Switching Networks Using Earliest Deadline First Round Robin"

[14] Vidya Sagar and Debabrata Das," Modified EDF Algorithm and WiMAX Architecture to Ensure End-to-End Delay in Multi-hop Networks"