School of Computer Science and Engineering, VIT University

OPERATING SYSTEMS (CSE2005)
PROJECT COMPONENT

Review Report

Submitted By

Mohit Chotani (19BCE0893)

Akashdeep (19BCI0036)

Deepak Pandey (19BCI0003)

Rohit Mantri (19BCE2673)

Review Report Submitted To

Prof. Vijayarajan V

Table of Contents

TITLE

EARLY DEADLINE FIRST (EDF) CPU SCHEDULING ALGORITHM
OR
PRIORITY AND DEADLINE CPU SCHEDULING ALGORITHM

AIM

To design and implement Priority and Deadline Algorithm for greater CPU utilization. To guarantee that all deadlines are met provided that the total CPU utilization is not more than 100%.

ABSTRACT

With the help of this project we are trying to make a new CPU scheduling algorithm that might find some uses in some sectors. The problem with the best algorithms is that the program having high burst time will take a long time to be executed.

Therefore, we are trying to make a CPU scheduling algorithm focusing mainly on the priority and deadline of the process to be executed. All processes need to be executed before the deadline arrives. With this algorithm the important processes will be dealt first and the waiting time of the process with high burst time will be less. Our proposed technique provide a way which leads toward a good solution, as we know that each job takes some duration to execute (i.e. Burst time) and each job is associated with some priority, there are many other factor associated with job but these two are important. So considering both factor we provide a sequence of jobs which is towards desired optimal. This is the whole reason for working on EDF algorithm.

KEYWORDS:

Deadline: The target completion time for a task is called its **deadline**.

Scheduler: **Schedulers** are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Preemptive: In computing, **preemption** is the act of temporarily interrupting a task being carried out by a computer system, without requiring its cooperation, and with the intention of resuming the task at a later time.

Non – Preemptive: Non – Preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.

Real Time-Systems: A real time operating system (RTOS) is an operating system that guarantees a certain capability within a specified time constraint.

Rate-Monotonic Scheduling: In computer science, **rate-monotonic scheduling** (RMS) is a priority assignment algorithm used in real-time operating systems (RTOS) with a static-priority **scheduling** class

Sporadic task: Sporadic tasks can be scheduled within a dedicated periodic task called a sporadic server. The priorities of all the periodic tasks, including the sporadic server, can be statically determined.

Latency: The **OS latency** is defined as the time between an event and the execution of the application that will respond to that event

Worst-case execution time: Worst-case execution time is the maximum length of time a task takes to execute on a specific hardware platform.

INTRODUCTION

Scheduling is one of the fundamental and most significant OS Function which is important to an operating system's design. Scheduling refers to set of rules, policies and mechanism that govern the order within which resource is allocated to the various processes and the work is to be done. The scheduling is a method of managing multiple queues of processes in order to minimize delay and to enhance and optimize the performance of the system. Scheduling of jobs plays a vital role as utilization of resources to a large extent depends on the efficiency of scheduling. A scheduler is an OS module that implements the scheduling policies. Its primary objective is to optimize the system performance according to the standards set by the system designers. It selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

EDF algorithm is Earliest Deadline First Algorithm. It is a dynamic priority scheduling algorithm used in real-time operating systems to place processes in a priority queue. Earliest deadline first selects a task according to its deadline such that a task with 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 present instance of time so every instant could be a scheduling event in EDF as deadline of task changes with time. A task which incorporates 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 is an ideal scheduling algorithm and typically executes in preemptive mode i.e. currently executing task is preempted whenever another task with earliest deadline becomes active. EDF can provide assurance that all deadlines are met with taking into consideration that the total CPU utilization is not more than 100%. In contrast to fixed priority arranging techniques like rate-monotonic of scheduling, EDF can certain all the deadlines in the system at higher loading.

MOTIVATION

The problem of getting a flexible or robust solution for scheduling problems is of uttermost importance for real-life applications. The major functionality of this task is to maximize the efficiency and finally enhance the performance. In real life problems we have a deadline to complete certain set of jobs, we have to decide the sequence of job in which we complete our jobs before or nearby deadline. Thus our interest to learn operating systems and its applications in real world has prompted us to take this project, which would set the stage for the applications we would be developing in the near future.

OBJECTIVES

- To provide an optimal and feasible schedule of all processes within the deadline.
- To ensure that all the jobs are completed prior to the deadline.
- To increase the efficiency and throughput of the processor.
- To completely utilize the CPU.

SOCIAL IMPACT

A user can schedule a real-time job in an IOT space like smart home and made them run using any of the real-time tasks scheduling algorithms using EDF algorithm. The system will provide the interface for the user to schedule these jobs and track them even if the user is not on his premises. This way the IOT smart space will behave autonomously without the interaction of users and more importantly without the need to be on the location. We have tried to add a real time application where it will schedule the complete real time tasks in a priority queue according to their deadline and will form a complete schedule of that time for the individual user.

ECONOMIC IMPACT

EDF scheduling is very efficient as compared to other scheduling algorithms in real-time systems. It can make the CPU utilization to about 100% while still guaranteeing the deadlines of all the tasks. Compared to fixed priority scheduling techniques like rate-monotonic scheduling **EDF** can guarantee all the deadlines in the system at higher loading.
Thus, the schedulability test for **EDF** is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 - p,$$

Where the *{Ci}* are the worst-case computation-times of the n processes and the *{Ti}* are their respective inter-arrival periods (assumed to be equal to the relative deadlines).

EDF is also an *optimal* scheduling algorithm on non-preemptive uniprocessors, but only among the class of scheduling algorithms that do not allow inserted idle time. When scheduling periodic processes that have deadlines equal to their periods, a sufficient (but not necessary) schedulability test for **EDF** becomes:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 - p,$$

Where *p* represents the penalty for non-preemption, given by max *{Ci} / min {Ti}*. If this factor can be kept small, non-preemptive **EDF** can be beneficial as it has low implementation overhead.

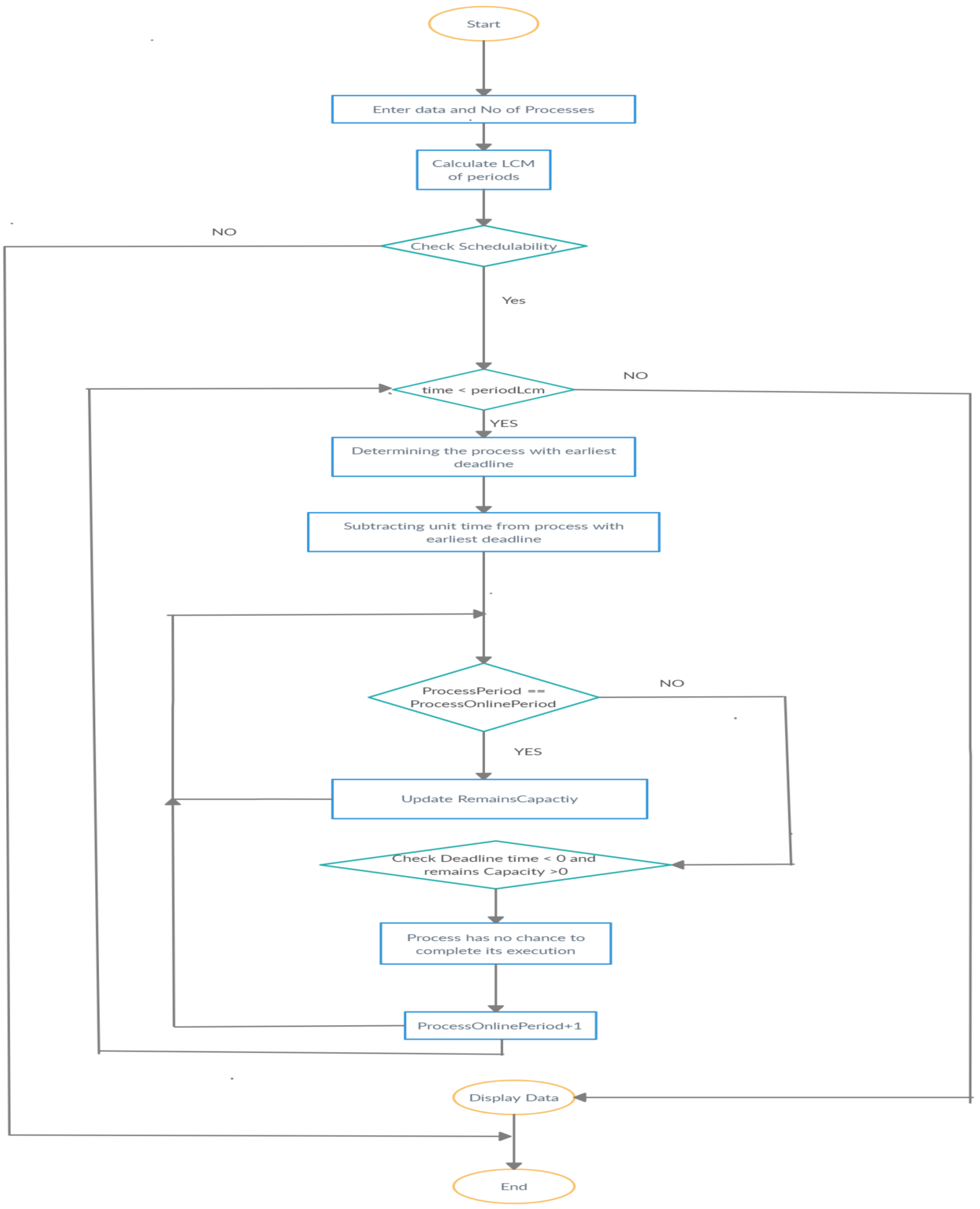
METHODOLOGY

ASSUMPTIONS:

a) The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.
b) Each task must be completed before the next requests for it occurs.
c) The tasks are independent in that requests for a certain task do not depend on the initialization or the completion of requests for other tasks.

ProcessOnlinePeriod - It keeps the check on the time in the running period of a process.
Schedulability - It keeps check if the process is schedulable or not by checking if average utilization of CPU is less than 1.
PeriodsLcm - It stores the value of LCM for the given period of processes.
remainsCapacity - To keep track of the remaining execution times of tasks.
ProcessDeadline - It stores the deadlines of processes (Deadlines mark the due-date of a process or task).

ARCHITECTURE DIAGRAM:


Start
Enter data and No of Processes
Calculate LCM of periods
Check Schedulability
NO
Yes
time < periodLcm
NO
YES
Determining the process with earliest deadline
Subtracting unit time from process with earliest deadline
ProcessPeriod == ProcessOnlinePeriod
NO
YES
Update RemainsCapactiy
Check Deadline time < 0 and remains Capacity >0
Process has no chance to complete its execution
ProcessOnlinePeriod+1
Display Data
End

LITERATURE SURVEY

Computation and communication are increasingly demanding for higher performance and reliability. This can be especially evident within the era of growing real-time and parallel data processing systems. In such systems or frameworks, violating the time limitations beyond certain thresholds is unacceptable and unsuitable. A central concept in the design and analysis of high-performance real-time systems is that the scheduling of a job that is released and must be completed within some timing constraints. Depending on job characteristics, a scheduling policy may assign priorities to jobs, which constitutes the scheduling decisions. During the past few years, there has been a growing interest in developing models for evaluating the performance of such real-time scheduling policies. Classical queueing theory usually assumes no limits or restrictions on the waiting time of jobs of the system. Such systems are studied under various different conditions where the timing requirements of jobs are not considered [8].

Key components in a real-time OS are negligible cut off latency and negligible gist swapping latency, a real-time OS is treasured more for how quickly or how typically it can reply than for the amount of work it can present in a granted time span of time. Real-time systems use arranging algorithms to conclude an alignment of execution of the jobs and an amount of time allotted for each task in the system so that no task (for hard real-time systems) or a smallest number of jobs (for soft real-time systems) misses their deadlines [14]. In alignment to verify the fulfillment of the temporal constraints, real-time systems use different exact or inexact schedulability checks. In the context of queueing theory, jobs with limited waiting time are usually referred to as impatient jobs (customers), which in practical real-time applications are called real-time jobs [10]. A real-time job encompasses a deadline before which it’s available for service and after which it must leave the system. Two models of job behaviour are usually considered: deadlines until the beginning of service (DBS) and deadlines until the end of service (DES). Within the former model, a job keeps its deadline only until the beginning of service. Accordingly, jobs remain in the system while being served until they complete their service requirements. In the latter model, a job retains its deadline until the end of service and accordingly, jobs may discontinue their service because they may have missed their deadlines [9].

Based on the EDF policy the job with the earliest deadline is the one with the highest priority [12]. It has been shown that EDF stochastically minimizes the fraction of lost jobs in both pre-emptive and non-pre-emptive models [5, 6, 7]. Because of the optimality of EDF, its analysis is particularly valuable. In spite of its importance, there are few papers on the probabilistic analysis of EDF. This may be due to the complexity of such analysis. Some of the works done in this area (e.g., [11, 13]) have concentrated on the probabilistic analysis of EDF for periodic task models. For a schedulable periodic task set that is feasibly scheduled by a certain algorithm, it is interesting to determine the particular time intervals called idle times during which the processor is not occupied and hence idle. Idle times can then be recovered to process additional tasks. A periodic task set is said to be feasibly scheduled by a certain algorithm if the computation of each request can be completed prior to the arrival of the next request of the same task. In this project, we investigated the problem of estimating localization and duration of idle times when tasks are scheduled according to the EDF scheduling algorithm [4] [3]. Under EDF, denoted priority is the earliest deadline, at each instant of time t, higher priority is assigned to the task whose deadline is closer to t. An illustrative example concerns a real-time system in which sporadic tasks may occur at unpredictable times and be required to be processed before a specified deadline. Such a system was the subject of studies reported in [2] and [1]. Our outcomes enable us to propose that a proportion of the maximum amount of processor's idle time, available for processing a sporadic task can be obtained in assuming that periodic tasks are processed according to earliest deadline from the arrival time of that sporadic task. Since EDF has been demonstrated as to be an optimal scheduling policy, its analysis is very important. To the best of our knowledge, in spite of its importance, there has been no exact analytical solution for the analysis of EDF. We have proposed an approximation method for this problem which we believe is quite accurate and at the same time very simple and straightforward. Some of the future works to continue this study report includes, extending the presented approach to certain kinds of multi-queue framework systems with dynamic routing of real-time jobs among the queues, systems with multiple priority classes of jobs, and other types of arrival designs.

REAL TIME APPLICATION OF THE PROJECT

There is only one computer in the office. Employees will be given a task to complete on that computer. In a period each employee will be alloted one task only. Every employee have to complete the task in the given deadline. So the mechanishm will allow the employees to work on that common computer in such a way that employee with closest deadline will be allowed to work first and by this every employee will complete his work in his given period of time and before his deadline.

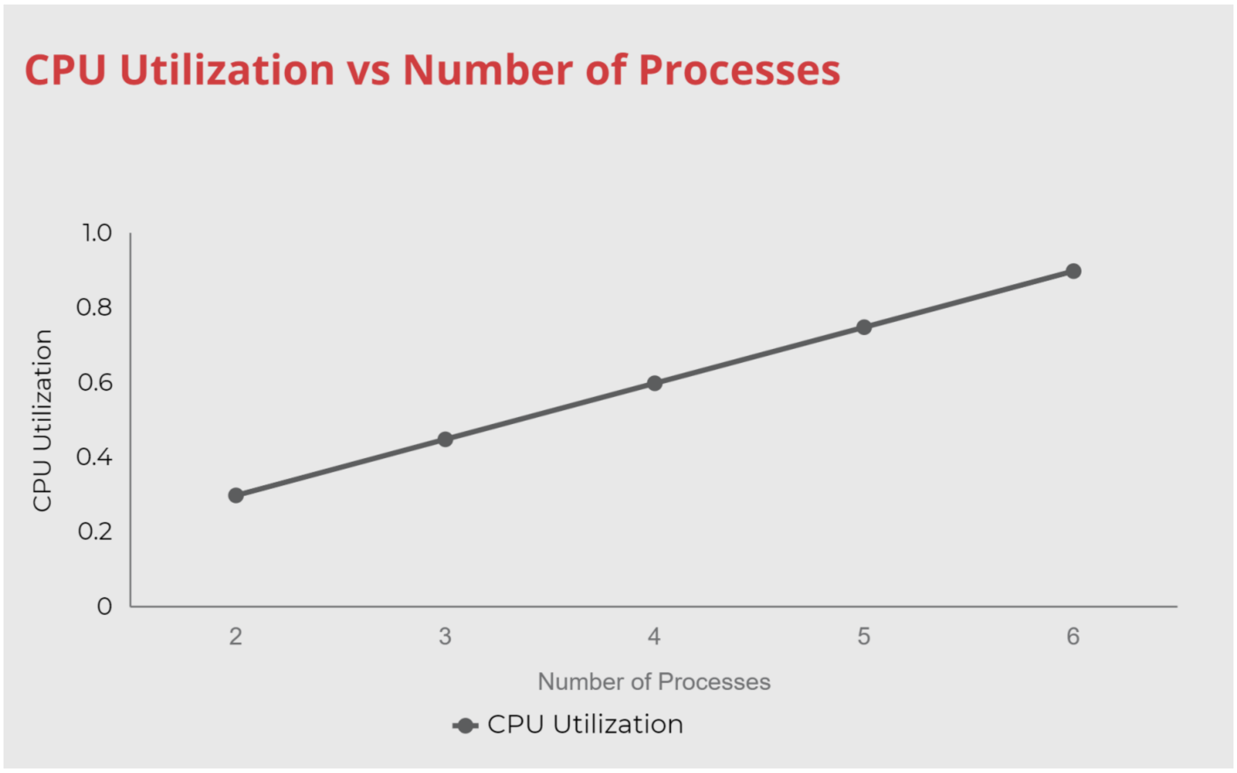
We have also added a schedulability condition in which it will not allow employees to work in overtime.

EXPERIMENTAL EVALUATIONS

To evaluate the effectiveness of our solution, we performed a series of experiments with randomly generated task sets. We tried creating a great number of task sets, varying the total utilization Umax with total number of tasks, execution time of tasks and periods of tasks. We used values of the total utilization Umax ranging from 0.3 to 0.9, the total number of tasks ranging from 2 to 6, the execution time ranging from 2 to 6 and period of tasks ranging from 10 to 30. From the experiments, with the generated tasksets, we observed that as the number of tasks increase, CPU utilization increases and as the execution time of processes increases CPU utilization increases. And when the periods of tasks are increased then CPU utilization will decrease. The total CPU utilization should remain less than 1 because CPU utilization greater than 1 is not possible and if CPU utilization becomes greater than 1, our program will not allow processes to schedule.

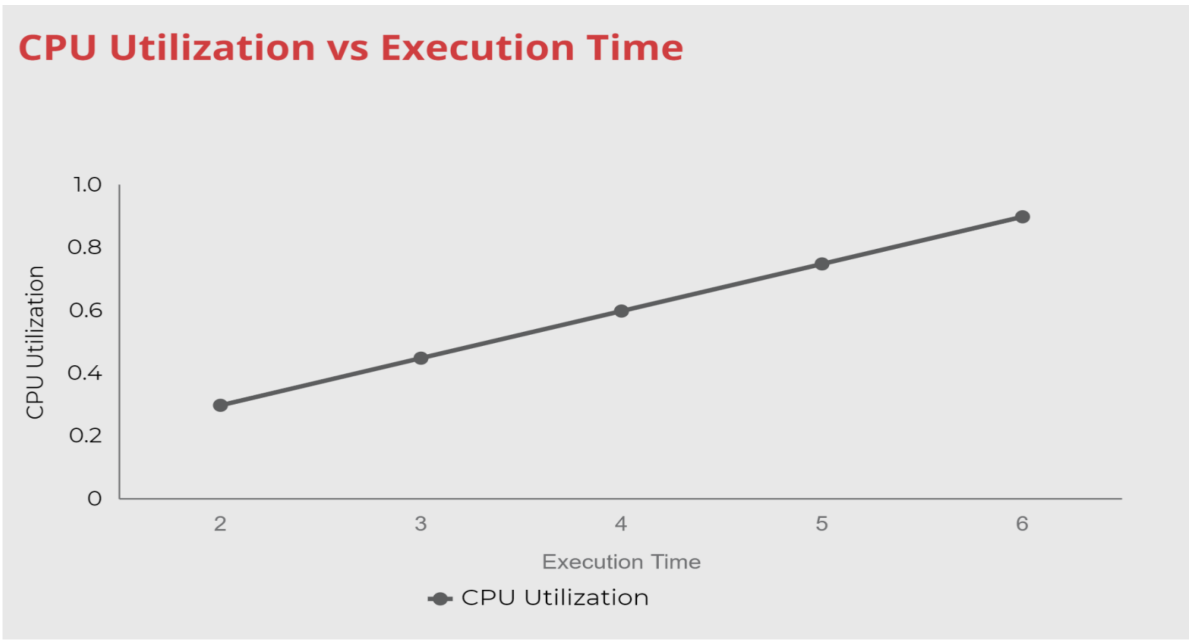
CPU Utilization vs Number of Processes-

We have chosen execution time and period common for all processes for better understanding and to get a clear idea of dependency of number of processes on the CPU utilization.



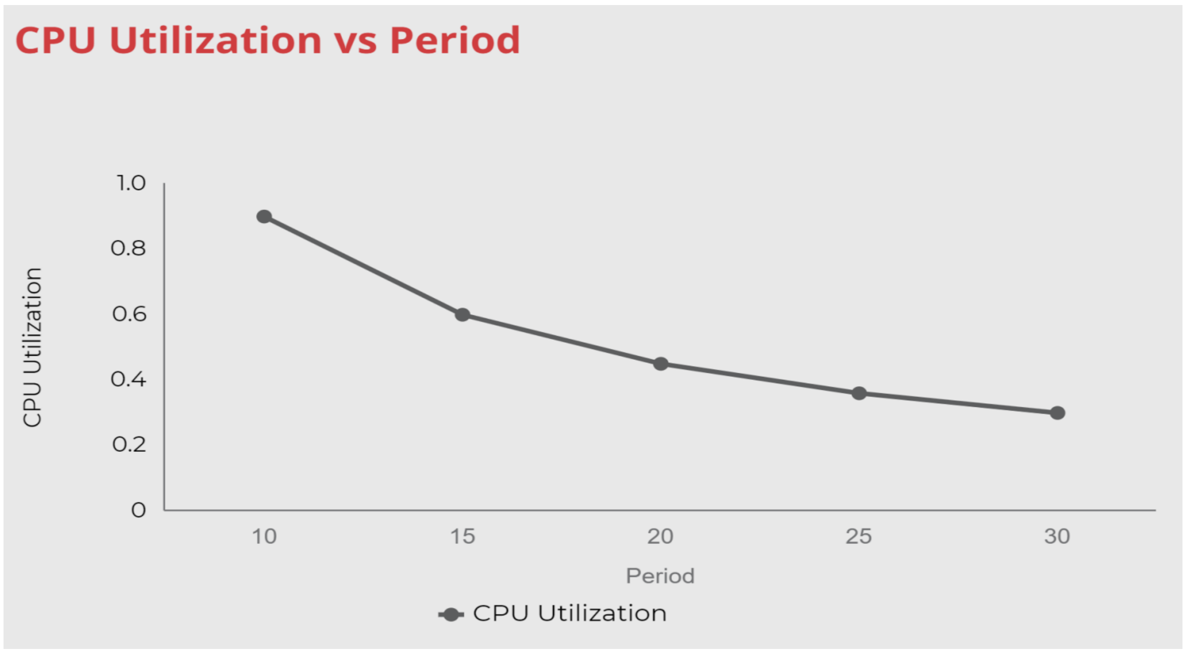
CPU Utilization vs Execution Time-

We have taken a constant number of processes and we have chosen a common period for all processes. We are also taking common execution time for all processes but we are increasing its value for each test.



CPU Utilization vs Period-

We have taken a constant number of processes and we have chosen a common execution time for all processes. We are also taking common period for all processes but we are increasing its value for each test.



CODE, INPUT & OUTPUT

SOURCE CODE

```
#include<stdio.h>
#include<string.h>
#include <stdlib.h>

int employeeNumber;
int pptGenerationTime[1000];
int pptDeadline[1000];
int pptPeriod[1000];
int PeriodsLCM;

char help[255];
int i,j,h;
float f;

void getJob()
{
	FILE *InputFile;
	InputFile = fopen("input.txt", "r");
	fgets(help, 255, InputFile);
	employeeNumber = atoi(help);
	// getting processes from the file input.txt;
	for(i = 0; i<employeeNumber; i++)
	{
		fgets(help, 255, InputFile);
		pptGenerationTime[i]= atoi(help);

		fgets(help, 255, InputFile);
		pptDeadline[i]= atoi(help);

		fgets(help, 255, InputFile);
		pptPeriod[i]= atoi(help);
	}
	fclose(InputFile);
	// showing the processes;
	printf("Ei| Wi, Dli, pi\n");
	printf("----------------\n");
	for(i=0; i<employeeNumber ; i++)
	{
		printf("E%d | %d, %d, %d\n",i,pptGenerationTime[i], pptDeadline[i],
pptPeriod[i]);
	}


}
void getPeriodsLCM()
```

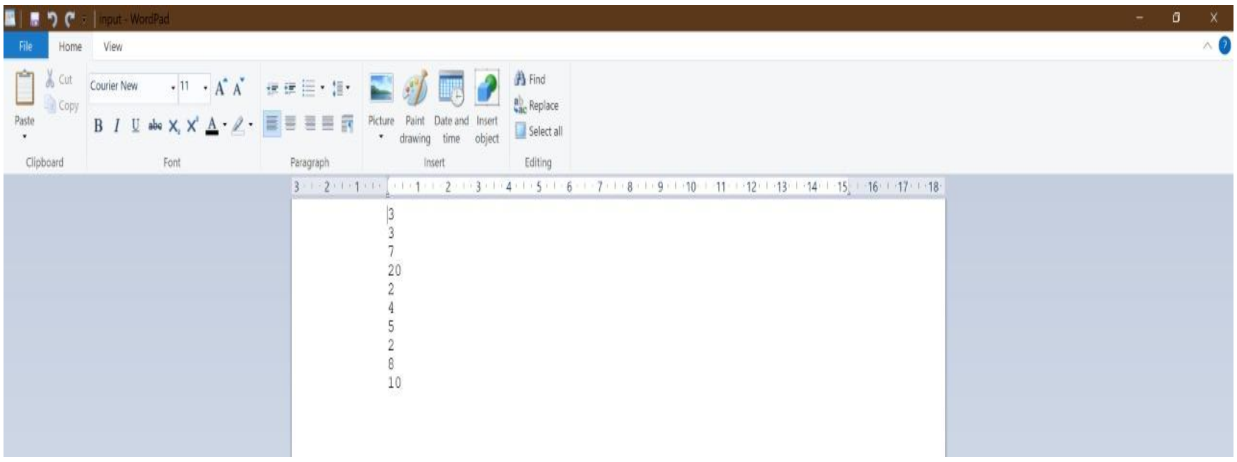
```
{
	int help[1000];
	for(i=0 ; i<employeeNumber; i++)
	{
		help[i] = pptPeriod[i];
	}
	int h = 0;
	int theLowestValueInHelp;
	int theLowestValueIndex;
	while(h == 0)
	{
		h = 1;
		theLowestValueInHelp = help[0];
		theLowestValueIndex = 0;

		for(i=0 ; i < employeeNumber; i++)
		{
			if(theLowestValueInHelp != help[i])
			{
				h = 0;
				if(theLowestValueInHelp>help[i])
				{
					theLowestValueInHelp = help[i];
					theLowestValueIndex = i;

				}

			}
		}
		if(h == 0)
		{
			help[theLowestValueIndex]        =        theLowestValueInHelp        +
pptPeriod[theLowestValueIndex];
		}
	}
	PeriodsLCM = help[0];
	printf("Employee Can Complete the Given Job In %d Hours(%d is the Lcm of our Job
periods)",PeriodsLCM,PeriodsLCM);

}

float schedulability()
{
	float Condition = 0;
	float x,y;
	printf("\n The Sum Of Work/Period of Each Employee:");
	for(i=0 ; i<employeeNumber ; i++)
	{
		x = pptGenerationTime[i];
		y = pptPeriod[i];
```

```
                Condition = Condition + (x/y);
                printf(" (%d/%d) ",pptGenerationTime[i],pptPeriod[i]);
        }
        printf("is equal : %f", Condition);
        return Condition;
}

void schedule()
{
        int earliestDeadline;
        int earliestDeadlineIndex;
        int schedulingTable[PeriodsLCM];
        int remainsCapacity[1000];
        int nextDeadline[1000];
        int processOnlineNewPeriod[1000];
        for(i=0 ; i<employeeNumber ; i++)
        {
                nextDeadline[i] = pptDeadline[i];
                remainsCapacity[i] = pptGenerationTime[i];
                processOnlineNewPeriod[i] = 0;
        }
        // scheduling time milestone...
        FILE *OutputtFile;
        OutputtFile = fopen("output.txt", "w");
        fprintf(OutputtFile, "Periods LCM = %d\n",PeriodsLCM);
        for(i=0; i<PeriodsLCM; i++)
        {
                printf("\n(%d,%d) : ",i,i+1);

                        //getting the earliest deadline
                        earliestDeadline = PeriodsLCM;
                        earliestDeadlineIndex = -1;
                        for(j=0 ; j<employeeNumber ; j++)
                        {
                                if(remainsCapacity[j] > 0)
                                {

                                        if(earliestDeadline > nextDeadline[j])
                                        {

                                                earliestDeadline = nextDeadline[j];
                                                earliestDeadlineIndex = j;
                                         }
                                }
                        }
                        printf("    [exc = %d] ",earliestDeadlineIndex);
                        fprintf(OutputtFile,         "(%d,%d)         :                        [exc         =
%d]\n",i,i+1,earliestDeadlineIndex);
                        remainsCapacity[earliestDeadlineIndex]--;
```

```
                    //get the next deadline distance
                    for(j=0 ; j<employeeNumber ; j++)
                    {

                            if(processOnlineNewPeriod[j] == (pptPeriod[j] - 1 ) )
                            {
                                    nextDeadline[j] =  pptDeadline[j];
                                    remainsCapacity[j] = pptGenerationTime[j];
                                    processOnlineNewPeriod[j] = 0;
                            }
                            else
                            {
                                    if(nextDeadline[j] >0)
                                    {
                                            nextDeadline[j]--;
                                    }
                                    else
                                    {
                                            if(remainsCapacity[j] > 0)
                                            {
                                                    printf("\nthe Employee %d has no chance to
complete its Job",j);
                                                    fprintf(OutputtFile,  "\nthe  Employee  %d
has no chance to complete its Job",j);
                                            }
                                    }
                                    processOnlineNewPeriod[j]++;
                            }


                    }
        }
        fclose(OutputtFile);


}

int main()
{


        printf("Employee Job Capability Check \n");
        for(i=0; i<1024000; i++);
        printf("done \n");


        printf("\n\nOur System :\n");
        getJob();
```

```
	printf("\n\nPeriods LCM :\n");
	getPeriodsLCM();



	printf("\n\nJob Schedulability Test :\n");
	f = schedulability();
	if(f <= 1)
	{
		printf("\n The Given Job of employee are Schedulable because %f <= 1", f);
		schedule();
	}
	else
	{
		printf("\nThe Given Job of employee are Schedulable because %f > 1",f);
		FILE *OutputtFile;
		OutputtFile = fopen("output.txt", "w");
		fprintf(OutputtFile, "The Given Job of employee are Schedulable because %f >
1",f);
		fclose(OutputtFile);
		exit(0);
	}
}
```

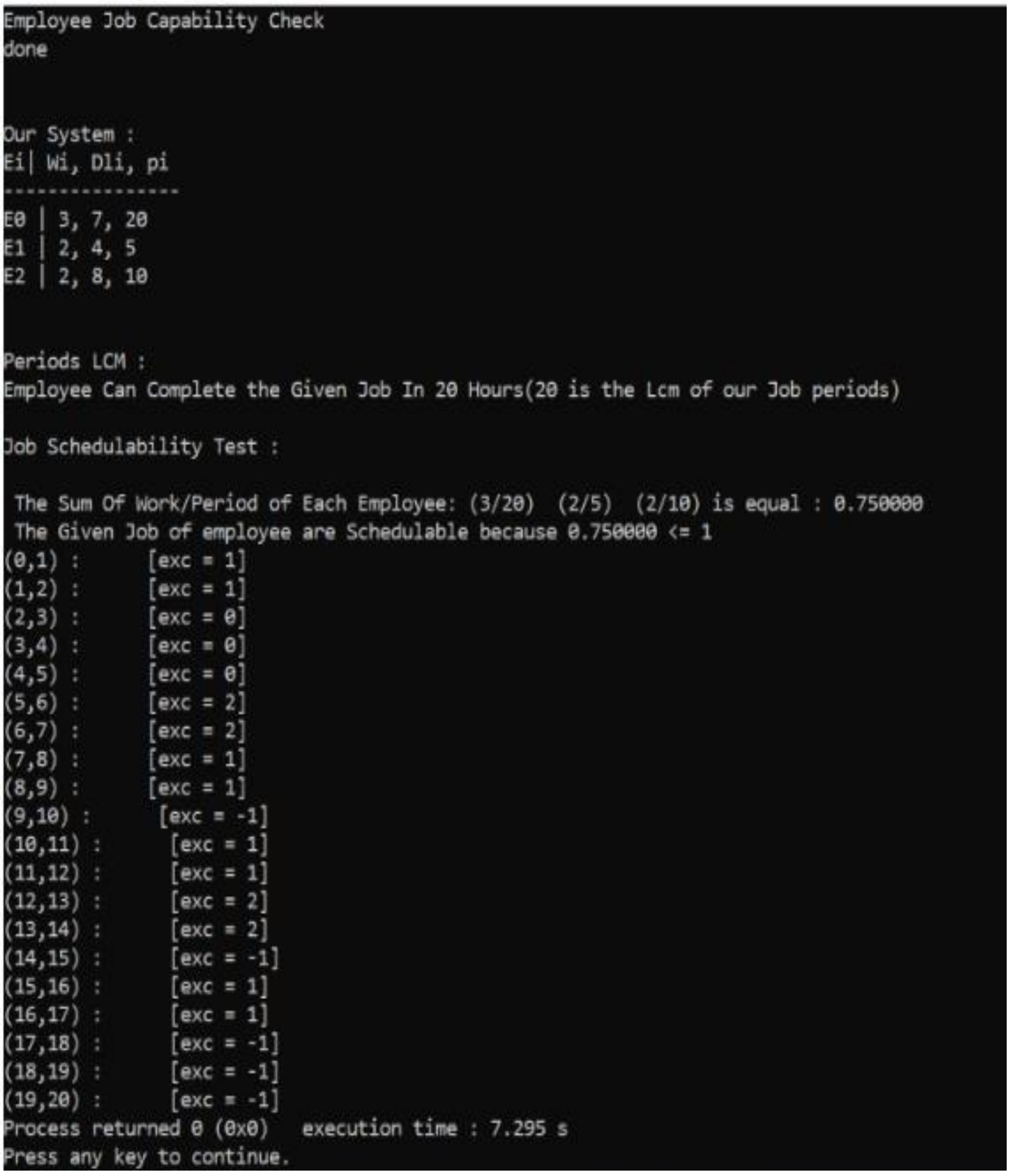
SCREENSHOT OF INPUT FILE



```
3
3
7
20
2
4
5
2
8
10
```

First value is the number of tasks. And the remaining values are execution time, deadline and period of processes in sequence.

SCREENSHOT OF OUTPUT



```
Employee Job Capability Check
done


Our System :
Ei| Wi, Dli, pi
---------------
E0 | 3, 7, 20
E1 | 2, 4, 5
E2 | 2, 8, 10


Periods LCM :
Employee Can Complete the Given Job In 20 Hours(20 is the Lcm of our Job periods)

Job Schedulability Test :

 The Sum Of Work/Period of Each Employee: (3/20)  (2/5)  (2/10) is equal : 0.750000
 The Given Job of employee are Schedulable because 0.750000 <= 1
(0,1) :      [exc = 1]
(1,2) :      [exc = 1]
(2,3) :      [exc = 0]
(3,4) :      [exc = 0]
(4,5) :      [exc = 0]
(5,6) :      [exc = 2]
(6,7) :      [exc = 2]
(7,8) :      [exc = 1]
(8,9) :      [exc = 1]
(9,10) :      [exc = -1]
(10,11) :      [exc = 1]
(11,12) :      [exc = 1]
(12,13) :      [exc = 2]
(13,14) :      [exc = 2]
(14,15) :      [exc = -1]
(15,16) :      [exc = 1]
(16,17) :      [exc = 1]
(17,18) :      [exc = -1]
(18,19) :      [exc = -1]
(19,20) :      [exc = -1]
Process returned 0 (0x0)   execution time : 7.295 s
Press any key to continue.
```

CONCLUSION:

From the relative study and comparisons, we may establish that the idea of "time" is vital in real-time systems. These system for the most part include various procedures that challenge each other. The need of scheduling is particularly imperative for constant programming design and assessment. From the assessment of the real-time scheduling algorithm, we can determine that the earliest deadline first (EDF) is one of the proficient scheduling algorithm if the CPU utilization is not more than 100% but instead scales well when the system is over-burden or occupied. Our algorithm places no restrictions on the total system utilization, but requires per-task utilizations to be at most one-half of a processor's capacity. This restriction is very liberal, and hence, our algorithm can be expected to be adequate for scheduling a large percentage of soft real-time applications. The genuine advantage of EDF is its less difficult implementation in commercial kernels that don't give unequivocal support to timing requirements such as periods and deadline. EDF permits a full processor use, which suggests a more proficient exploitation of computational assets and a greatly improved responsiveness of aperiodic activities.
In future, new algorithm ought to be created which is a blend of fixed and dynamic priority or a scheduling algorithm that switch consequently between EDF algorithm and fixed based scheduling algorithm to deal over-loaded and under loaded conditions.

REFERENCES

[1] W. Zhao and K. Ramamritham, “Distributed scheduling using bidding and focused addressing”, in Proc. Real-Time Systems. San Diego, CA, Dec. 3-6, 1985, pp. 112-122.

[2] K. Ramamritham and J. Stankovic, “Dynamic task scheduling in hard real-time distributed systems, IEEE Software. Vol. 1. No. 3. pp. 65-75, July 1984.

[3] O. Serlin, “Scheduling of time critical processes” in Proc. Spring Joint Computer Conf., 1972, pp. 925-932.

[4] C. L. Liu and J. W. Layland, “Scheduling algorithm for multiprogramming in a hard real-time environment.” J. ACM, vol. 20, no. 1, pp. 46-61, 1973.

[5] D. Towsley and S. S. Panwar. On the Optimality of Minimum Laxity and Earliest Deadline Scheduling for Real-Time Multiprocessors. In Proceedings of IEEE EUROMICRO-90 Workshop on Real-Time, 17–24, 1990.

[6] D. Towsley and S. S. Panwar. Optimality of the Stochastic Earliest Deadline Policy for the G/M/c Queue Serving Customers with Deadlines. In Proceedings of the Second ORSA Telecommunications Conference, 1992.

[7] S. S. Panwar, D. Towsley, J. K. Wolf. Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Beginning of Service. J. Assoc. Compute., 35(4):832–844, 1988.

[8] K. Kant. Introduction to Computer System Performance Evaluation, McGraw-Hill, 1992.

[9] A. Movaghar. On queueing with customer impatience until the end of service. Stochastic Models, 22:149–173, 2006.

[10] J. P. Lehoczky. Using Real-Time Queueing Theory to Control Lateness in Real-Time Systems. Performance Evaluation Review, 25(1):158–168, 1997.

[11] A. Leulseged and N. Nissanke. Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameters. In Proceedings of the 9t h
International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2003), LNCS 2968:103–122, Springer-Verlag, 2004.

[12] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. J. Assoc. Compu. Machinary, 20(1):46–61, 1973.

[13] N. Nissanke, A. Leulseged, and S. Chillara. Probabilistic Performance Analysis in Multiprocessor Scheduling. J. Computing and Control Engineering, 13(4):171–179, 2002.

[14] Su-Lim TAN and Tran Nguyen Bao Anh, "Real-time operating system (RTOS) for small (16-bit) Microcontroller "proceedings The 13th IEEE International Symposium on Consumer Electronics (ISCE2009), pp. 1007-1011.