Enhanced Maximum Urgency First Algorithm with Intelligent Laxity for Real Time Systems

Article *in* International Journal of Computer Applications · April 2012

DOI: 10.5120/6283-8460

Dr. H. S. Behera
Veer Surendra Sai University of Technology(VSSUT), Govt. of Odisha, India
162 PUBLICATIONS **1,720** CITATIONS
SEE PROFILE

International Journal of Computer Applications (0975 – 8887)
Volume 44– No.8, April 2012

Enhanced Maximum Urgency First Algorithm with Intelligent Laxity for Real Time Systems

H.S.Behera
Sr.Lecturer
Department Of Computer Science and Engineering,
Veer Surendra Sai University Of Technology(VSSUT), Burla
Sambalpur, Odisha, India

Naziya Raffat
Student
Department Of Computer Science and Engineering,
Veer Surendra Sai University Of Technology(VSSUT), Burla
Sambalpur, Odisha, India

Minarva Mallik
Student
Department Of Computer Science and Engineering,
Veer Surendra Sai University Of Technology(VSSUT), Burla
Sambalpur, Odisha, India

ABSTRACT

In this paper Enhanced Maximum Urgency First (EMUF) scheduling algorithm with intelligent laxity has been proposed. This algorithm is a further improvement in MMUF algorithm [1] and is a mixed priority scheduling algorithm which combines the advantages of both fixed and dynamic scheduling for better CPU utilization and throughput. The prime objective of this paper is to improve modified maximum urgency first scheduling (MMUF) using intelligent laxity as the dynamic priority. EMUF algorithm is mainly suited for real time systems where meeting of deadlines is an important criterion for scheduling. This proposed algorithm improves the Modified Maximum Urgency First scheduling algorithm for real time tasks proposed by V.Salmani et.al [1] and the experimental analysis shows that the proposed algorithm(EMUF algorithm) performs better than MMUF [1] and MUF[6] scheduling algorithm by minimizing average turnaround time, average waiting time and maximizing the throughput.

General Terms

Earliest Deadline First scheduling(EDF), Enhanced Maximum Urgency First scheduling (EMUF), Least Laxity First scheduling(LLF), Modified Least Laxity First scheduling(MLLF), Maximum Urgency First scheduling (MUF), Modified Maximum Urgency First scheduling (MMUF), Scheduling

Keywords

Context switches, intelligent laxity, laxity, process, real time system, real time system scheduling, turnaround time, throughput, waiting time

1. INTRODUCTION

Real time systems are designed to provide results within a specific time frame. Real time systems are used when correctness of the outputs as well as the time or the instants at which these results are produced affect the system's performance. In short, real time systems can say to have well defined, strict time constraints. A number of scheduling algorithms are available for scheduling of processes in a real time system and lot many scheduling algorithms have been proposed by researchers for real time systems.

Real time systems are basically divided into three types- hard, firm and soft.

Hard real time systems are also known as safety-critical systems [9]. These systems are very much particular about deadlines. The tasks must adhere to the specified deadlines very strictly, failing which it may result into a catastrophe. Here, the critical tasks must meet their deadlines. In soft real time systems timing constraints are provided but inefficiency to meet these deadlines wont result into system failures. Here, critical tasks receive higher priority over other available tasks and they need to be completed before other noncritical tasks. Linux supports soft real time system [9]. Some real-time operating systems have firm real-time requirements. Firm real time systems allow occasional deadline violations but those tasks which are not finished by their specified deadlines are rejected and not scheduled by the processors. [10].

Space research, video conferencing, weather forecasting, seismic detection, audio conferencing, money withdrawal, ATM, railways and flight reservations etc are some of the applications of real time systems.

Scheduling is the process of assigning jobs, processes or tasks to the various processors in a system in an efficient manner [9]. Scheduling can be classified into 2 types- static scheduling and dynamic scheduling [11]. In static scheduling, scheduling decision is made during the compile time before scheduling begins. Here, priorities in which jobs will be scheduled, is assigned before the process execution. It improves the objective function and searches for consistent schedule. Static scheduling algorithm is often associated with assignment of fixed priorities and is a subclass of dynamic priority algorithm as in static scheduling priorities of task doesn't change. Examples are First come first serve, shortest remaining time next etc. Dynamic scheduling makes scheduling decision at the time of execution. Various schedulability tests are available for both uniprocessors as well as multiprocessors to decide whether a set of tasks meet their deadlines or not. To check the successful execution of a process beforehand, schedulability tests are performed. If the schedulabilty test is successful, then the scheduler can guarantee the successful execution of the process. Earliest deadline first algorithm, least laxity first algorithm, modified least laxity first are few examples of dynamic scheduling.

Modified Maximum Urgency First scheduling algorithm has been proposed by V.Salmani et.al [1]. It combines the advantages of fixed and dynamic scheduling to provide dynamically changing systems with flexible scheduling. Here in this paper we are proposing Enhanced Maximum Urgency first (EMUF) scheduling algorithm for real time systems where intelligent laxity is the dynamic priority and it has been calculated for each process of the system. EMUF performs better than the algorithm proposed by V.Salmani et.al (MMUF) [1] and MUF [6].

1.1 Preliminaries

An instant of a computer program in execution containing program code and its activity is called a process. It is made of multiple threads and these threads run concurrently. The processes assigned to the processor are organized into a queue known as ready queue. CPU utilization is the process of keeping the CPU busy with the useful work. Burst time of a process is defined as the time required by the processor to execute that particular process. In simple words it is the execution time of a process. Arrival time is the time at which the process arrives at the ready queue. Turnaround time is the time between submissions of a process to its completion. Waiting time is the amount of time a process spends in the ready queue. Laxity is the remaining time required to complete a process. Laxity is calculated by subtracting the burst time from the deadline. Intelligent laxity is the laxity which is calculated at a particular instant. It is calculated by subtracting the current time from the laxity. Response time is the amount of time it takes from when a request was submitted until the first response is produced. Throughput is the number of processes that completes their execution per unit time. The number of time a CPU switches from one process to another is the number of context switches [9].

1.2. Organisation of the Paper

Section 2 presents the related work done in this area, various scheduling policies and the loopholes with each algorithm which has motivated us towards the development of EMUF. Section 3 describes the algorithm, pseudo code and flow chart of the proposed algorithm. In section 4 experimental analysis of the proposed algorithm (EMUF) and its comparison with MUF [6] algorithm and MMUF algorithm [1] is presented. Section 5 and section 6 contain the conclusion and the references respectively.

2. PRIOR WORK

The EDF [3] and LLF [4] algorithms are treated as optimal dynamic priority algorithm. LLF [4] reduces the system performance as it increases the number of context switches and hence increases overhead of the system. Hence, modified least laxity first (MLLF) algorithm is proposed by Oh and Yang to resolve the drawbacks of LLF algorithm by reducing the no of context switches [5].

But again, whether it is modified least laxity first [5], EDF[3] or LLF[4] a transient overload in the system may cause a critical task to fail. Stewart and Khosla [6] have designed a mixed priority urgency based scheduling algorithm which defines a critical set of tasks as critical tasks which is guaranteed to meet its deadline during a transient overload [6]. Critical task set is a set of tasks for which CPU utilisation or CPU load factor is less than 100%. Transient overload in a system occurs when this CPU load factor exceeds 100%. MUF as proposed by Stewart et.al [6] uses least laxity as its dynamic priority. But with MUF algorithm sometimes a situation may be there where a critical task may fail at the expense of a non-critical task. This has been explained by V.Salmani et.al Least laxity first is a dynamic priority scheduling policy but here we have seen if remaining execution time of any task 't1' is greater than the laxity time of another task 't2' then even t1 is scheduled first since here the concept applied is that the scheduling should be produced in such a way that the task having the highest priority should always be running. This increases the number of deadline miss for a particular number of processes.

To overcome the deficiencies of MUF a mixed priority based scheduling algorithm modified maximum urgency first is proposed by Salmani and Zargar [1]. Here they have introduced a unique importance parameter to create the critical set.

2.1 Scheduling policies

2.1.1 Earliest Deadline First scheduling algorithm

EDF is a dynamic priority scheduling algorithm which uses the deadline of a task as priority while scheduling the tasks. The task with the earliest or smallest deadline gets higher priority over other tasks while the task with the latest/longest deadline has the lowest priority. This algorithm has the schedulability bound of 100% for all task sets. Schedulabilty bound is defined by a parameter known as CPU load factor or CPU utilisation.

CPU utilisation of a task is computed as the ratio of its worst case computation time C_i to its relative deadline T_i [1] where relative deadline is obtained by deducting arrival time from the absolute deadlines of the task.

CPU utilisation for n periodic task is computed as

$U=\sum_{i=0}^{n} C_i/T_i <=1$

If $U>1$, then almost no algorithm can successfully schedule the task set. If $U<1$ just like EDF, many algorithms are there to schedule the tasks set successfully. EDF is an optimal algorithm which attempts to fully utilize the processor and has less idle time. Even, context switches are less in EDF and hence system overhead reduces.

The disadvantage with this algorithm is that often it is seen that with EDF a critical task may fail at the expense of a lesser important task in transient overloaded system. Thus, it is very unpredictable. To keep track of absolute deadlines in long data structures it requires additional hardware resulting in implementation overhead. EDF has less control over the execution of a process that is priority of a process can't be changed by EDF in order to reduce the response time.

2.1.2 Least laxity First algorithm

LLF is also a dynamic priority scheduling algorithm. It computes laxity of each task in the system and then selects the task with the minimum laxity. Laxity is defined as the difference between the deadlines by which the task must be finished to the amount of computation time remaining to finish the task [6]. Deadline of a task is often referred as latest useful completion time of a process. Laxity of a process changes over time whereas as pointed earlier deadline of a process doesn't change over time. A task having zero laxity must be scheduled first and executed without pre-emption or else it will fail to meet its specified deadline. If the laxity of a task comes out to be negative then at any cost the task will miss its deadline. If a process waits for a longer time for execution, it has the smallest laxity.

A significant shortcoming is related to laxity ties. Laxity tie is defined as a condition in which two or more tasks have the same laxity. Laxity ties results in frequent context switches among the corresponding task. This increases the system overhead and ultimately degrades the system's performance. Like EDF, LLF has also a schedulability bound of 100% and there is no guarantee that all the critical tasks will get executed in a transient overload situation.

2.1.3 Modified Least Laxity First algorithm

MLLF [5] scheduling algorithm solves the problem of LLF algorithm by significantly reducing the number of context switches. The performance of LLF algorithm is challenged mainly due to laxity ties because when laxity ties occur, context switching increases. MLLF algorithm is an optimization of LLF which reduces the context switches and improves the system's performance. If there is no laxity tie, MLLF schedules the task same as the LLF scheduling. If the laxity tie occurs, the running task continues to run with no preemption as far as the deadlines of other tasks are not missed. This not only decreases context switching but also improves performance. But again, like EDF and LLF [1], MLLF has a schedulability of 100% and hence it can't ensure that no critical tasks miss its deadline

2.1.4 Maximum urgency first algorithm

Maximum Urgency First (MUF) [6] scheduling algorithm resolves the problem of unpredictability of the system during transient overload that is when CPU load factor exceeds 100%. This algorithm is urgency based scheduling algorithm. It is a mixed priority scheduling algorithm and employs both fixed as well as dynamic priority for efficient scheduling of tasks. With this algorithm, each task is given an urgency which is defined as a combination of two fixed priorities (criticality and user priority) and a dynamic priority that is inversely proportional to the laxity. The critical priority is set to 1 if tasks are present in the critical set and the CPU load factor for these tasks is less than 100%.
Critical priority > dynamic priority > user priority

The MUF algorithm assigns priorities in two phases. Phase one is concerned with the assignment of static priorities to tasks. Static priorities are assigned once and do not change after the system . the MUF scheduling algorithm as mentioned in V.Salmani et.al paper is as follows[1].

In phase 1, fixed priorities are defined and the scheduler sorts the task in increasing order of their periods. First N tasks having CPU utilization<100% are taken in critical tasks and the remaining tasks are considered in non-critical task set. Every task is given a optimal user priority that depends entirely on the user.

In phase 2, dynamic priorities are set and in case of MUF [6], it is MLLF [5]. If there is only 1 critical task the task is executed. If more than 1 critical task is there, the task with the minimum laxity is picked up for execution. If there are more than 1 tasks with the same laxity then the task with the highest user priority is considered and scheduled.

Once all the tasks present in the critical set are finished, the same set of steps are repeated for the tasks in the non-critical task set.

The disadvantage with this algorithm has been discussed by V.Salmani et.al [1]. Whenever a task arrives at the ready queue, rescheduling occurs [6]. Hence there is a possibility of failing of a critical task in many situations.

Here least laxity is considered as the dynamic priority. A task with minimum laxity may be selected whose remaining execution time is greater than the remaining execution time to another task's laxity. According to [7], the task having the highest priority should always be running. We are here citing an example taken from [1]

Table 1. Table taken from [1] to show disadvantage of [6]

Tasks	**Remaining Execution Time**	**Deadline**	**Remaining Laxity Time**
T_1	6	8	2
T_2	3	6	3

Here, t1 will be selected first having minimum laxity time and it will run till its execution [7]. Remaining execution time of t1 is greater than laxity time of t2. As a result t2 will miss the deadline. MUF orders the task from shortest period to longest period and then defines the critical task set. It is not mandatory that the task with the shortest period is always critical and more important for the system

2.1.5 Modified maximum urgency first algorithm

To overcome the drawbacks of MUF[6], MMUF has been proposed. Modified maximum urgency first scheduling algorithm as proposed by V.Salmani et.al [1] is basically a slight modification in maximum urgency first (MUF)[6] scheduling algorithm. User priority is set in the beginning according to the importance of the tasks. Task with the highest importance are given user priority as 1 and task with the second highest priority is assigned user priority 2 and so on. After the user priority has been set first n tasks with CPU utilization less than 100% are allotted to the critical set and assigned critical priority as 1. Remaining tasks are allotted to the non-critical set and critical priority is 0 for these tasks. Unlike, MUF it is not always that the task with the shortest period is the most important one. Here EDF is used as the dynamic priority. Here number of context switches is reduced to a great extent resulting in an enhanced system performance.
User priority>critical priority> dynamic priority
The MMUF scheduling algorithm as proposed by V.Salmani et.al is as follows:
The MMUF algorithm consists of two phases with the following details:

In phase 1, fixed priorities are defined and the tasks are arranged in the decreasing order of their user priorities. First N tasks having CPU utilization<100% are taken in critical tasks and the remaining tasks are considered in non-critical task set.

In phase 2, dynamic priorities are calculated and accordingly the tasks are selected for execution. If there is only 1 critical task the task is executed. If more than 1 critical task is there, the task with the earliest deadline is picked up for execution. If there is more than 1 task with the same deadline then the task with the highest importance is considered and scheduled.

Once all the tasks present in the critical set are finished, the same set of steps are repeated for the tasks in the non-critical task set.

3. PROPOSED APPROACH– ENHANCED MAXIMUM URGENCY FIRST ALGORITHM (EMUF)

3.1Motivation

Urgency based scheduling is a very effective scheduling policy. In real time systems, achieving predictability is equally important as abiding by the time constraints. Predictability affects the overall system efficiency, leading to the successful completion of tasks which are critical for the system. Predictability in a system is achieved by using urgency based algorithm like MUF [6], MMUF [1] etc. Thus, this importance of urgency based scheduling has motivated us towards the development of EMUF.

3.2 Uniqueness Of The Proposed Algorithm

In modified maximum urgency first scheduling MMUF [1], always a process having earliest deadline is scheduled first although that process may has a chance to miss the deadline. It results into poor utilization of CPU.
With EDF task with earliest deadline is scheduled first but if a task with earliest deadline and higher execution time is scheduled it misses its deadline because of the higher execution time and eventually the task fails. This condition increases the response time of the remaining processes. But for an optimal scheduling algorithm response time should be minimal.
In MUF [6] least laxity first is used as the dynamic priority.

But, LLF is not an optimal scheduling algorithm since it decreases the overall system performance by frequent context switching. In the proposed algorithm we are overcoming the disadvantages associated with both MUF [1] and MMUF [6]. Hence we introduce the concept of intelligent-laxity to enhance CPU utilization and to present a better and unique algorithm for scheduling of tasks in real time systems. In EMUF intelligent Laxity is considered as dynamic priority. Intelligent laxity of each process is calculated for all the remaining processes at every scheduling event by the scheduler. This means, intelligent laxity for the remaining processes is calculated whenever a process arrives at the ready queue and also at its completion. In case intelligent laxity is negative for a process, the process is not at all scheduled as it will definitely miss its deadline. Thus, it prevents unwanted process from being scheduled and improves CPU utilisation.
Two fixed priorities are assigned to the processes which are the same priorities as used in MMUF [1] and better than the priorities of MUF [6]. The dynamic priority used in the proposed algorithm is intelligent laxity which outperforms dynamic priorities used in both the previous algorithms.

3.3 Detailed Structure of the Proposed Algorithm

EMUF is also a mixed priority scheduling algorithm. Urgency is defined as a combination of two fixed priorities user priority and critical priority. User priority is a fixed priority which is generally set by user. In our proposed algorithm we have considered the user priority according to the importance of the tasks. More important processes are given the higher user priorities. Critical priority is defined as the priority given to the critical tasks, the tasks which are included in the critical set. Critical set includes those tasks which are really critical for the system and they need to be executed for better system performance. With the MMUF [1] algorithm either EDF [3] or MLLF [1] can be used to define the dynamic priority but in EMUF we are considering intelligent laxity (laxity which is calculated at every scheduling event). And the process with the minimum intelligent laxity is scheduled first.
Critical priority > dynamic priority > user priority.

Enhanced MUF algorithm consists of two phases with the following details.

Phase 1: in this phase fixed priorities are defined. These priorities remain constant throughout the scheduling.

1) Order the task from most important to least important.
2) Add the critical tasks as defined before to the critical set where CPU utilisation factor is less than 100%.

Phase 2: This phase calculates the dynamic priority.

1)If there is only 1 critical task it will be executed without any pre-emption
2) If there is more than 1 critical task, select the task which has least intelligent laxity.
   a) If there is a tie in intelligent laxity then select the task with the highest user priority.
   b) After the completion of each task, again intelligent laxity is calculated for all the remaining processes and process with least intelligent laxity is selected for execution.
3) If there is no critical task in the ready queue select the task from non-critical set which has the least intelligent laxity.
   a) If there is a tie in intelligent laxity then select the task with the highest user priority.
   b) After the completion of each task, again intelligent laxity is calculated for all the remaining processes and process with least intelligent laxity is selected for execution.

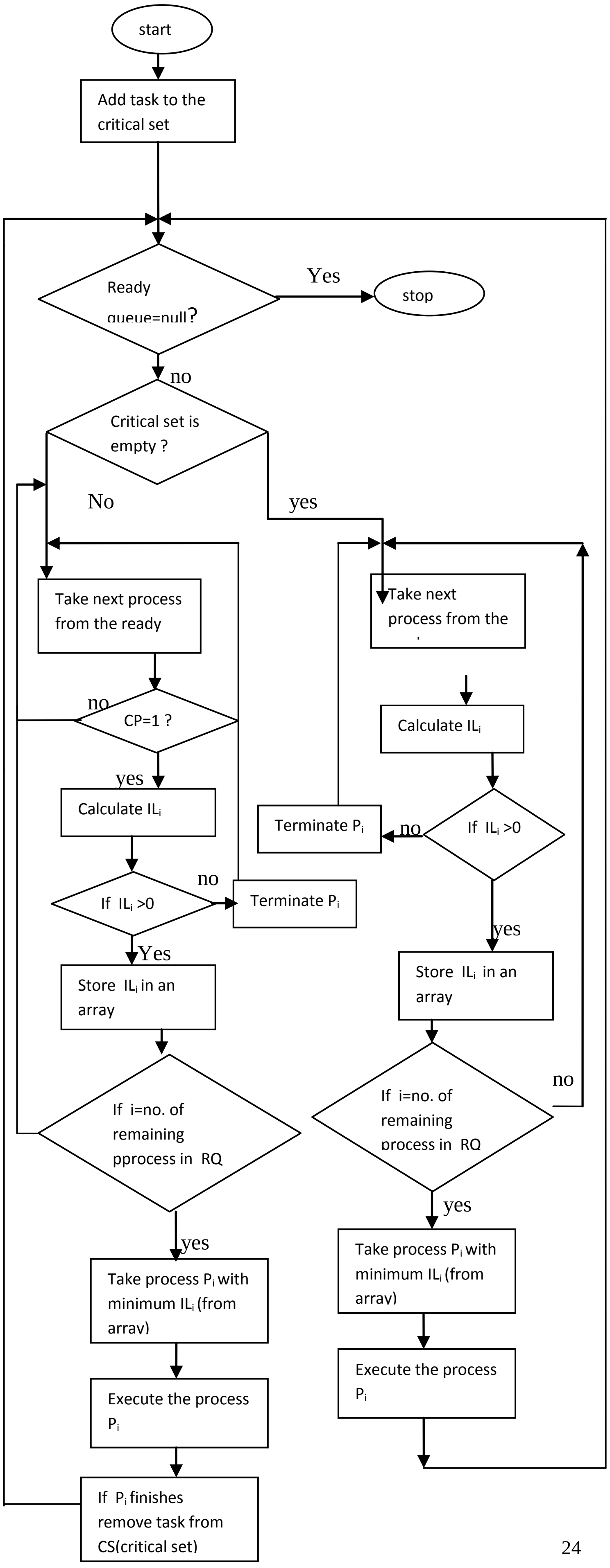
Calculate average turnaround time, average waiting time and throughput of the processes.

Intelligent laxity for the remaining processes is calculated whenever a process arrives at the ready queue and also at its completion. Here in EMUF intelligent least laxity is calculated every time a process is completed, until the ready queue is empty. In EMUF tasks with negative intelligent laxity are not at all executed by the processor as a result it improves the throughput of the system and prevents the processor from doing unwanted work.
Intelligent laxity is mathematically calculated by subtracting remaining execution time and current clock cycle from relative deadline.

3.4 Pseudo Code of the Proposed Algorithm

```
1)let n=no of process

    $P_i$ = process i.
    $BT_i$ = Burst time of process i
    $D_i$ = deadline of process i
    $RBT_i$ = remaining burs time of process i
    CP = critical priority
    UP = user priority
    $IL_i$ = intelligent laxity of process i
  Initialise i=0,avg.TAT=0, avg.WT=0

2)  Set CP=1 for process which are in critical set

3)  Set user priority according to the importance of the task

4)  // Intelligent -Laxity calculation
      $IL_i=D_i – RBT_i$ - current time

5)  while(ready queue!=NULL)
    {
     x=0,y=0;
     for(i=0 to n)
     Arr[i]=100;
     For(i=0 to n)
        {
        If ($CP_i$= =1)
          {
            Calculate $IL_i$
             If($IL_i$<0)
            Terminate the process from ready queue;
             Else
               {
               Arr[x]=$IL_i$;
                x++;
                }
          }
         Else if($CP_i$=0 && all critical task completed)
           {
          Calculate $IL_i$;
                If ($IL_i$<0)
                 Terminate the process from the ready queue;
              Else
               {
                Arr[y]=$IL_i$
                 y++;
               }
            }// else if closed
        }//for loop closed
    }//while loop closed

 6)  if (x= =1)
     Execute the process $P_i$ having CP=1;

 7)  if(y= =1)
     Execute the process $P_i$ having CP=0;

 8)  if (x>1||y>1)
     min(arr[i]) = minimum of the array arr[i];
     if($IL_i$ ==  min(arr[i]))
      {
        Execute the process $P_i$
      }
     Repeat 5,6, 7,8;

 9)  //calculation of throughput
      Throughput= no of task completed successfully
      /totalTAT

 10)  //calculation of avg.TAT
      Avg.TAT=$\sum TAT_i$/no of task completed
     successfully

 11)  //calculation of avg.WT
      Avg.WT=$\sum WT_i$/no of task completed successfully
```

3.5 Flow Chart of the Proposed Algorithm


start
Add task to the critical set
Ready queue=null?
Yes
stop
no
Critical set is empty ?
No
yes
Take next process from the ready
no
CP=1 ?
yes
Calculate IL_i
If IL_i >0
no
Terminate P_i
Yes
Store IL_i in an array
If i=no. of remaining pprocess in RQ
yes
Take process P_i with minimum IL_i (from array)
Execute the process P_i
If P_i finishes remove task from CS(critical set)
Take next process from the
Calculate IL_i
If IL_i >0
no
Terminate P_i
yes
Store IL_i in an array
If i=no. of remaining process in RQ
no
yes
Take process P_i with minimum IL_i (from array)
Execute the process P_i

4. EXPERIMENTAL ANALYSIS

4.1Assumptions

All the experiments are performed is a single processor environment and all the processes are independent. Attributes like burst time, priority, numbers of processes are known before submitting the processes to the processor. All processes are CPU bound.

4.2 Experimental Frame work

The experiment consists of several input and output parameters. The input parameters consist of burst time, deadline, critical task priority, user priority and the number of processes. The output parameters consist of average waiting time, average turnaround time and throughput.

4.3 Data Sets

Several experiments have been performed for evaluating performance of the new proposed algorithm but only two cases are shown here. The data set have been considered for different processes with variable burst time and deadlines.

4.4 Performance Metrics

The significance of our performance metrics for experimental analysis is as follows:

1) *Turnaround time (TAT)*: For the better performance of the algorithm, average turnaround time should be less.
2) *Waiting time (WT)*: For the better performance of the algorithm, average waiting time should be less.
3) *Throughput*: throughput of the system should be high to improve CPU utilization.

4.5 Results Obtained

EXAMPLE 1:

We assume five processes arriving at time=0, with burst time (P_1=18,P_2=6,P_3=23,P_4=8,P_5=20) and critical task set={P_1,P_2} and deadlines {P_1=35,P_2=20,P_3=42,P_4=42,P_5=80}. Table 2 contains data to be used by MUF [6], MMUF [1] and our proposed algorithm. Giant charts are drawn for all the three algorithms. Table 3 shows the comparison among the three algorithms.

Table 2. Contains data for example1

P_I	BT_I	D_I	CRITICAL PRIORITY	USER PRIORITY	LAXITY
P_1	18	35	1	1	17
P_2	6	20	1	2	14
P_3	23	42	0	3	19
P_4	8	42	0	4	34
P_5	20	80	0	5	60

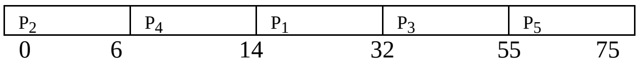


Fig 1: Gantt chart for MUF

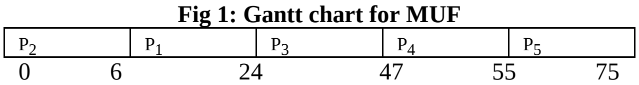


Fig 2: Gantt chart for MMUF

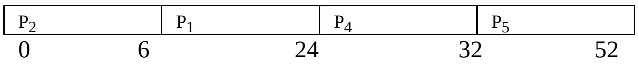


Fig 3: Gantt chart for EMUF

Table 3. Comparison between MUF, MMUF,EMUF

ALGORITHMS	AVG.TAT	AVG.WT	THROUGHPUT
MUF	36.4	21.4	.0533
MMUF	41.4	26.4	.04
EMUF	28.5	15	.0769

EXAMPLE 2:

We assume five processes arriving at time=0, with burst time (P_1=36,P_2=30,P_3=25,P_4=24,P_5=18) and critical task set={P_4,P_5 } and deadlines {P_1=140,P_2=90,P_3=62,P_4=65,P_5=30}.Table 4 contains data to be used by MUF [6], MMUF [1] and our proposed algorithm(EMUF). Giant charts are drawn for all the three algorithms. Table 5 shows the comparison among the three algorithms.

Table 4. Contains data for example2

P_I	BT_I	D_I	CRITICAL PRIORITY	USER PRIORITY	LAXITY
P_1	36	140	0	5	104
P_2	30	90	0	4	60
P_3	25	62	0	3	37
P_4	24	65	1	2	41
P_5	18	30	1	1	12

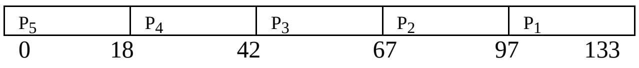


Fig 4: Gantt chart for MUF

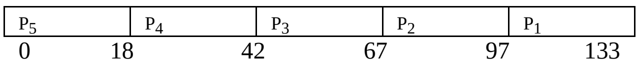


FIG 5

Fig 5: Gantt chart for MMUF

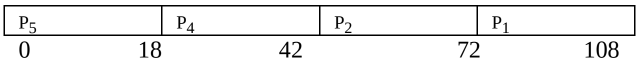


Fig 6: Gantt chart for EMUF

Table 5. Comparison between MUF, MMUF, EMUF

ALGORITHMS	AVG.TAT	AVG.WT	THROUGHPUT
MUF	71.4	44.8	.0150
MMUF	71.4	44.8	.022
EMUF	60	33	.0370

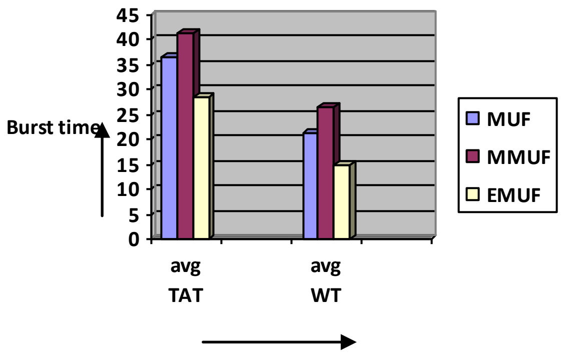


Fig 7 : comparison between avg WT and avg TAT of MUF, MMUF, EMUF in EX-1

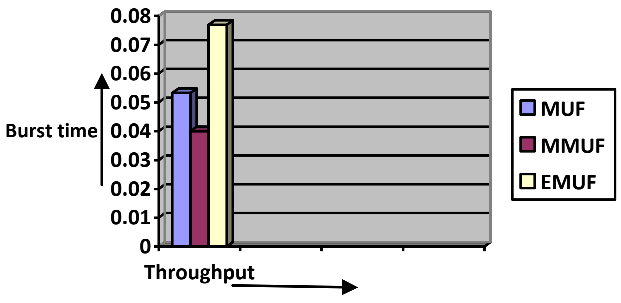


Fig 8 : comparison between throughput of MUF, MMUF, EMUF in EX-1

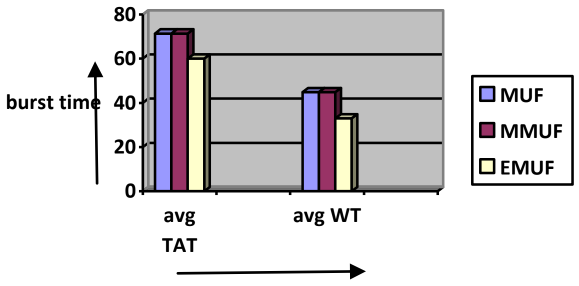


Fig 7 : comparison between avg WT and avg TAT of MUF, MMUF, EMUF in EX-2

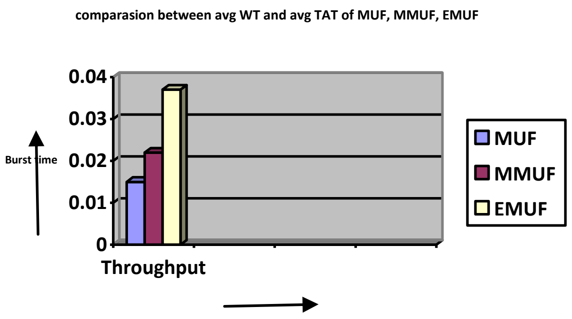


Fig 8 : comparison between throughput of MUF, MMUF, EMUF in EX-2

5. CONCLUSION

It is concluded from the above experiments that the proposed algorithm (EMUF) performs better than the MUF [6] algorithm and the algorithm proposed by V.Salmani et.al [1] MMUF, in terms of performance metrics such as average waiting time, average turnaround time and throughput. Our proposed algorithm can be further investigated to be useful in providing more and more task-oriented results in future.

6. REFERENCES

[1] V.Salmani, S.zargar and M. Naghibzadeh- *a modified maximum urgency first scheduling algorithm for real time tasks*- world academy of science and technology 9 2005

[2] S.Baskiyar and N. Meghanathan: *A Survey Of Contemporary Real Time Operating Systems, Informatica* S. 29 pp 233-240, 2005.

[3] C. L. Liu, and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard real Time Environment," *Journal of the Association for Computing Machinery*, vol.20, no.1, pp. 44-61, January 1973.

[4] A. Mok. "Fundamental Design Problems of Distributed Systems for Hard Real-time Environments". *PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1983.*

[5] S. H. Oh, and S. M. Yang, "A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks", in *Proc. Fifth International Conference On Real Time Computing System And Application*,October1998,pp.31-36

[6] D.B.Stewart , and P.K.Khosla," Real Time Scheduling Of

Dynamically Reconfigurable Systems," in *proc. IEEE International Conference on Systems Engineering*, Dayton Ohio, August 1991, pp. 139-142

[7] J. Goossens, and P. Richard, "Overview of real-time scheduling problems", in *Proc. the ninth international workshop on project management and scheduling*, Nancy, France, April 2004.

[8] Yaashuwanth .C IEEE Member, Dr.R. Rames *Department of Electrical and Electronics Engineering, Anna University Chennai*, "Design of Real Time scheduler simulator and Development of Modified Round Robin architecture "

[9] Silberschatz, A., P.B. Galvin and G. Gagne, *Operating Systems Concepts.* 7th Edn. John Wiley and Sons, USA

[10] J.Goossens, Universit´e Libre de Bruxelles and P. Richard, Laboratoire d'Informatique Scientifiq et Industrielle ENSMA (France), "Overview of real-time scheduling problems"

[11]S.Chengs, J.A Stankovic and K.Ramamritham: *Scheduling Algorithms for Hard Real Time Systems- A Brief Survey* –'Tutorial: hard real time systems'(IEEE, 1988)pp.150-173