RTOS† Mini Project

† **Course Coordinator: Dr Layak Ali**

Y V Sai Dinesh
Department of Electronics & Communications Engineering
Central University of Karnataka
yvsdinesh@ieee.org

Compare the performance of Rate-Monotonic (RM) and Earliest-Deadline First (EDF) scheduling algorithm on a typical scenario

1) **Details explanation of both the algorithms**
2) **Advantages and disadvantages of each**
3) **Implementation Code**
4) **Comparison of both algorithms**
5) **Snapshots of execution**
6) **Any extra information as Appendix**

I. Rate-Monotonic Scheduling

A. Explanation

Rate monotonic scheduling is a priority algorithm that belongs to the static priority scheduling category of Real Time Operating Systems. It is preemptive in nature. The priority is decided according to the cycle time of the processes that are involved. If the process has a small job duration, then it has the highest priority. Thus if a process with highest priority starts execution, it will preempt the other running processes. The priority of a process is inversely proportional to the period it will run for.

$$\sum_{k=1}^{n} \frac{C_i}{T_i} \leq U = n(2^{1/n} - 1)$$

Where n is the number of processes in the process set, Ci is the computation time of the process, Ti is the Time period for the process to run and U is the processor utilization.

B. Example

Is the special types of functions that are used to implement the functionality of the operating system.

Processes	Execution Time	Time Period
P1	3	20
P2	2	5
P1	2	10

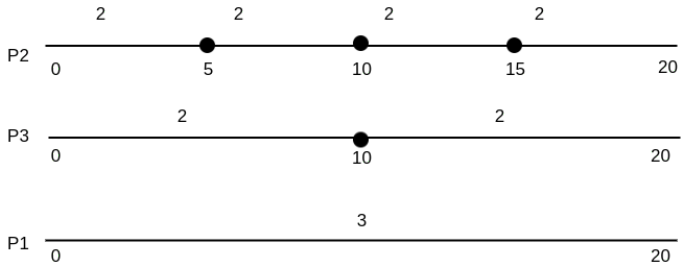
$$n(2^{1/n} - 1) = 3(2^{1/3} - 1) = 0.7797$$

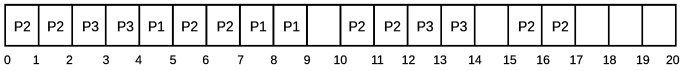
$$U = \frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75$$

It is less than 1 or 100% utilization. The combined utilization of three processes is less than the threshold of these processes that means the above set of processes are schedulable and thus satisfies the above equation of the algorithm.

- Scheduling time –
  For calculating the Scheduling time of algorithm we have to take the LCM of the Time period of all the processes. LCM (20, 5, 10) of the above example is 20. Thus we can schedule it by 20 time units.
- Priority –
  As discussed above, the priority will be the highest for the process which has the least running time period. Thus P2 will have the highest priority, after that P3 and lastly P1.
  $$P1 > P2 > P3$$
- Representation and flow –

  Above figure says that, Process P2 will execute two times for every 5 time units, Process P3 will execute two times for every 10 time units and Process P1 will execute three times in 20 time units.





- Process P2 will run first for 2 time units because it has the highest priority. After completing its two units, P3 will get the chance and thus it will run for 2 time units.

- As we know that process P2 will run 2 times in the interval of 5 time units and process P3 will run 2 times in the interval of 10 time units, they have fulfilled the criteria and thus now process P1 which has the least priority will get the chance and it will run for 1 time.
- And here the interval of five time units have completed. Because of its priority P2 will preempt P1 and thus will run 2 times.
- As P3 have completed its 2 time units for its interval of 10 time units, P1 will get chance and it will run for the remaining 2 times, completing its execution which was thrice in 20 time units.
- Now 9-10 interval remains idle as no process needs it. At 10 time units, process P2 will run for 2 times completing its criteria for the third interval (10-15).
- Process P3 will now run for two times completing its execution.
- Interval 14-15 will again remain idle for the same reason mentioned above. At 15 time unit, process P2 will execute for two times completing its execution.
- This is how the rate monotonic scheduling works.

C. Advantages and Disadvantages

Advantages:

- It is easy to implement.
- If any static priority assignment algorithm can meet the deadlines then rate monotonic scheduling can also do the same. It is optimal.
- It consists of calculated copy of the time periods unlike other time-sharing algorithms as Round robin which neglects the scheduling needs of the processes.

Disadvantages:

- It is very difficult to support aperiodic and sporadic tasks under RMA.
- RMA is not optimal when tasks period and deadline differ.

II. Earliest Deadline First Scheduling

A. Explanation

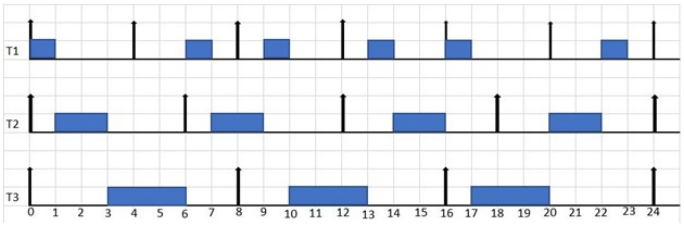
Earliest deadline first (EDF) is **dynamic priority scheduling algorithm** for real time embedded systems. Earliest deadline first selects a task according to its deadline such that a task with earliest deadline has higher priority than others. A task which has a higher priority due to earliest deadline at one instant it may have low priority at next instant due to early deadline of another task. EDF typically executes in **preemptive** mode i.e. currently executing task is preempted whenever another task with earliest deadline becomes active.

B. Example

Task	Execution Time	Deadline	Time Period
T1	1	4	4
T2	2	6	6
T3	3	8	8

$$U = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = 0.25 + 0.333 + 0.375 = 0.95 = 95\%$$

As processor utilization is less than 1 or 100% so task set is surely schedulable by EDF.



1) At t=0 all the tasks are released, but priorities are decided according to their absolute deadlines so T1 has higher priority as its deadline is 4 earlier than T2 whose deadline is 6 and T3 whose deadline is 8, that's why it executes first.
2) At t=1 again absolute deadlines are compared and T2 has shorter deadline so it executes and after that T3 starts execution but at t=4 T1 comes in the system and deadlines are compared, at this instant both T1 and T3 has same deadlines so ties are broken randomly so we continue to execute T3.
3) At t=6 T2 is released, now deadline of T1 is earliest than T2 so it starts execution and after that T2 begins to execute. At t=8 again T1 and T2 have same deadlines i.e. t=16, so ties are broken randomly an T2 continues its execution and then T1 completes. Now at t=12 T1 and T2 come in the system simultaneously so by comparing absolute deadlines, T1 and T2 has same deadlines therefore ties broken randomly and we continue to execute T3.
4) At t=13 T1 begins it execution and ends at t=14. Now T2 is the only task in the system so it completes it execution.
5) At t=16 T1 and T2 are released together, priorities are decided according to absolute deadlines so T1 execute first as its deadline is t=20 and T3's deadline is t=24.After T1 completion T3 starts and reaches at t=17 where T2 comes in the system now by deadline comparison both have same deadline t=24 so ties broken randomly ant we T continue to execute T3.
6) At t=20 both T1 and T2 are in the system and both have same deadline t=24 so again ties broken randomly and T2 executes. After that T1 completes it execution. In the same way system continue to run without any

problem by following EDF algorithm.

C. Advantages and Disadvantages

Advantages:

- No need to define priorities offline
- It has less context switching than rate monotonic
- It utilize the processor maximum up to 100% utilization factor as compared to rate monotonic

Disadvantages:

- It is less predictable. Because response time of tasks are variable and response time of tasks are constant in case of rate monotonic or fixed priority algorithm.
- EDF provided less control over the execution
- It has high overheads

III. RMS vs EDF

With EDF a higher utilization is possible than with RMS

EDF has a maximum utilization of 100%; for RMS there are tasksets with a lower utilization that are not feasible. Hence, the statement is true, though the actual utilization depends on the taskset. The argument that RMS has a lower utilization bound is not acceptable as such, since this bound is a pessimistic one.

RMS is more efficient than EDF

- When used on-line, EDF needs more advanced data structures (logarithmic access). RMS is then more efficient.
- EDF assigns different priorities to jobs and therefore needs dynamic priority assigment which is generally more costly. Again, RMS is then more efficient.
- EDF can have a higher utilization. It makes more efficient use of the resources.

RMS is more general than EDF (i.e., applicable in more situations)

- RMS admits to separate importance and schedulability in a systematic way.
- RMS performs more predictable under overload conditions. An argument against it would be to refer to the utilization once again.

Key difference between rate monotonic scheduling and earliest deadline first

The first is a fixed priority policy to the activity with the shortest length. The second is a dynamic priority policy that refers to the absolute deadline.

IV. Implementation

A. *Rate Monotonic Scheduling*

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

int main()
{
//arrival time of process A and B
int A,B;
//period and execution for A and B processes
int cycA,cycB,serveA,serveB;
float m,util_bound;
int i,j,a=0,b=0,ka=0,kb=0;

int numa=0,numb=0;
int T;

printf("please input period and",
"execution for A process\n");

scanf("%d%d",&cycA,&serveA);

printf("please input period and",
"execution for B process\n");

scanf("%d%d",&cycB,&serveB);

m=(float)serveA/cycA+(float)serveB/cycB;
util_bound = (float)2*(pow(2,0.5)-1);

for(T=0;T<=100;T++)
{
/* to check if CPU can schedule*/
if(m > 1)
{
printf("Utilization is greater than 100%",
"\nCPU cannot schedule these tasks.");
exit(1);
}
else if(m > util_bound) {
printf("CPU cannot schedule these",
"tasks.\n");
exit(1);
}

/*process A has been done*/
/*but process B has not been done yet*/
if(numa==serveA)      //process A is done
{
numa=serveA+1;
printf("when T=%d, ",T);
printf("process A%d is done\n",a);
if(numb<serveB)
{
printf("run process B%d\n",b);
kb=1;
}
ka=0;
}

/*process B has been done*/
/* but process A has not been done yet*/
if(numb==serveB)
{
numb=serveB+1;
printf("when T=%d, ",T);
printf("process B%d is done\n",b);
if(numa<serveA)
{
printf("run process A%d\n",a);
ka=1;
}
kb=0;
}

/* start running the process*/
if(T%cycA==0 && T%cycB==0)
{
A=B=T;
j=++a;
i=++b;
printf("when T=%d, process A%d and",
"process B%d are generated together\n",T,j,i);
if(cycA<=cycB) // deadline is cycle
{
printf("run process A%d and",
"suspend process B%d\n",j,i);
printf("-------------------------------\n");
ka=1;
kb=0;
}
else
{
printf("run process B%d and",
"suspend process A%d\n",i,j);
ka=0;
kb=1;
}
numa=numb=0;
}

/* initializes A another new period*/
if(T%cycA==0&&T%cycB!=0)
{
A=T;
printf("when T=%d, ",T);
printf("process A%d is generated\n",++a);
```

```
numa=0;
//process B is unfinished yet
if(numb<serveB)
if(cycB>cycA)
{
printf("process B%d was preempted",
"by process A%d\n", b, a);
printf("run process A%d\n",a);
ka=1;
kb=0;
}
else
printf("process B%d is moving",
"forward\n",b);
else  //process B is done, just run A
{
printf("process A%d is run\n",a);
ka=1;
}
}



/* initializes B another new period*/
if(T%cycA!=0&&T%cycB==0)
{
B=T;
printf("when T=%d, ",T);
printf("process B%d is generated\n",++b);
numb=0;
if(numa<serveA)  //process A is undone yet
if(cycB>=cycA)
printf("process A%d is on run\n",a);
else
{
printf("process A%d was preempted by",
"process B%d\n", a, b);
printf("process B%d is to run\n",b);
kb=1;
ka=0;
}
else                //process A is done
{
printf("process B%d is on run\n",b);
kb=1;
}
}



/*accumulate running time*/
if(ka)
numa++;
if(kb)
numb++;
}
return 1;
}
```

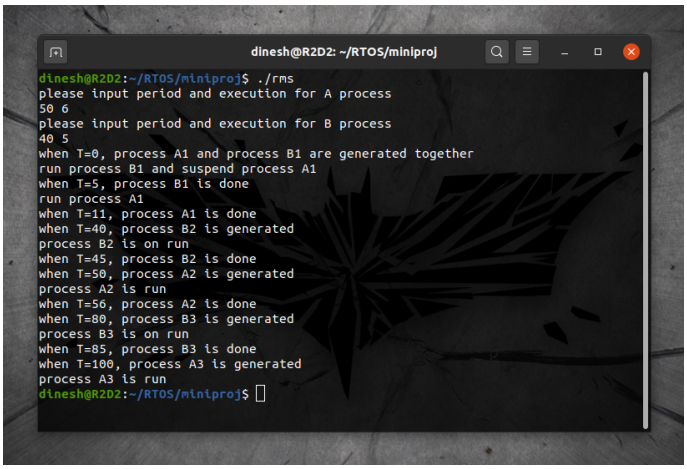
B. *Earliest Deadline First*

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

int main()
{
int A,B;
int cycA,cycB,serveA,serveB;
float m;
int i,j,a=0,b=0,ka=0,kb=0;

int numa=0,numb=0;
int T;
int deadlineA, deadlineB;

printf("please input period and ");
printf("execution for A process\n");
scanf("%d%d",&cycA,&serveA);
printf("please input period and ");
printf("execution for B process\n");
scanf("%d%d",&cycB,&serveB);
m=(float)serveA/cycA+(float)serveB/cycB;

for(T=0;T<=100;T++)
{

if(m > 1)
{
printf("U_t is greater than 100%");
printf("\nCPU cannot be scheduled.");
exit(1);
}


/* process A has been done*/
/* but process B has not been done yet*/
if(numa==serveA)    //process A is done
{
numa=serveA+1;
printf("when T=%d, ",T);
printf("process A%d is done\n",a);
if(numb<serveB)
{
printf("run process B%d\n",b);
kb=1;
}
ka=0;
}

/* process B has been done*/
/* but process A has not been done yet*/
if(numb==serveB)
{
numb=serveB+1;
printf("when T=%d, ",T);
```

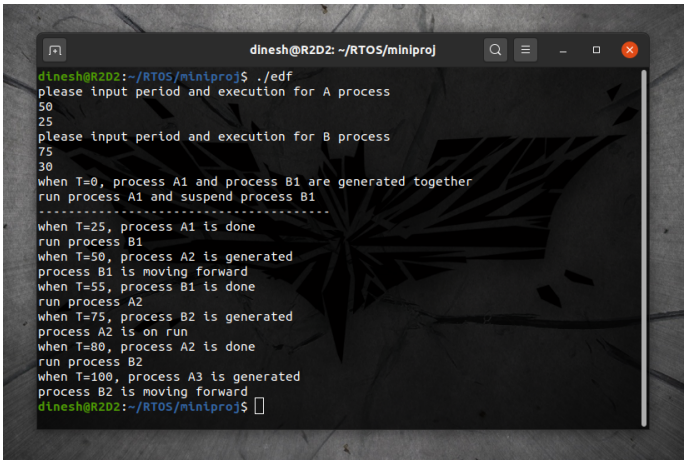
```
printf("process B%d is done\n",b);
if(numa<serveA)
{
printf("run process A%d\n",a);
ka=1;
}
kb=0;
}

/* start running the process*/
if(T%cycA==0 && T%cycB==0)
{
deadlineA = serveA;
deadlineB = serveB;
A=B=T;
j=++a;
i=++b;
printf("when T=%d, process A%d and",
"B%d are generated together\n",T,j,i);
if(cycA<=cycB) // deadline is cycle
{
printf("run process A%d and ",
"suspend process B%d\n",j,i);
printf("------------------------\n");
ka=1;
kb=0;
}
else
{
printf("run process B%d and ",
"suspend process A%d\n",i,j);
ka=0;
kb=1;
}
numa=numb=0;
}


 /* Next period of A*/
if(T%cycA==0&&T%cycB!=0)
{
A=T;
printf("when T=%d, ",T);
printf("process A%d is generated\n",++a);
numa=0;
deadlineA += serveA;
if(numb<serveB)  //B is unfinished yet
if(deadlineB>deadlineA)
{
printf("run process A%d\n",a);
ka=1;
kb=0;
}
else    //period of B is earlier
//than period of A
printf("B%d is moving forward\n",b);
else //process B is done, just run A
{
printf("process A%d is run\n",a);
ka=1;
}
}


/* Next period of B */
if(T%cycA!=0&&T%cycB==0)
{
B=T;
printf("when T=%d, ",T);
printf("process B%d is generated\n",++b);
numb=0;
deadlineB += serveB;

if(numa<serveA) //process A is undone yet
if(deadlineB>=deadlineA)
printf("process A%d is on run\n",a);
else
{
printf("process B%d is to run\n",b);
kb=1;
ka=0;
}
else            //process A is done
{
printf("process B%d is on run\n",b);
kb=1;
}
}


/* running time*/
if(ka)
numa++;
if(kb)
numb++;

 }
 return 1;
}
```

V. Snapshots

A. *RMS*



B. *EDF*



VI. References

- https://www.researchgate.net/publication/226597712_Rate_Monotonic_vs_EDF_Judgment_day
- https://nptel.ac.in/content/storage2/courses/108105057/Pdf/Lesson-30.pdf
- https://www.geeksforgeeks.org/rate-monotonic-scheduling/
- https://www.geeksforgeeks.org/earliest-deadline-first-edf-cpu-scheduling-algorithm/
- https://microcontrollerslab.com/rate-monotonic-scheduling-algorithm/