International Journal of Smart Home
Vol. 9, No. 3 (2015), pp. 101-112
http://dx.doi.org/10.14257/ijsh.2015.9.3.10

An Effective Self-test Scheduling for Realtime Processor based System

J. V. N. Ramesh, B. Naresh Kumar Reddy, V. V. Murali Krishna, B. M. Kumar Gandhi, V. Shiva and M. Dronika Devi

Dept. of Electronics and computer Engineering
K. L. University
jvnramesh@gmail.com, naresh.klu@gmail.com

Abstract

Now a days Jobs are Scheduled in a single processor or more than one processor, a real time job is scheduled or executed based on requirements, An Successful task in embedded system ought to have constrained asset necessities: Memory, execution time and power utilization, these necessity are not generally simple to fulfil in real-time embedded system with hard task deadlines. In this paper we explore the effective time utilization, without influencing the deadline requirements of typical hard real time task ,there are no limitation on the new results each task can be periodic or sporadic ,with relative deadline which can be less than ,equivalent to or greater than its period, it is too fast , best-effort ,effective real time scheduling algorithm for a wide variety of job parameters.

Keywords: *Real Time Systems, Multi-Processor, Resource Parameters*

1. Introduction

Many papers have been published in the field of Real time scheduling. These include surveys of published literature. In scheduling, at what time the jobs are to be scheduled is given by the system. This means the jobs scheduled should have certain Release time and deadline [1]. Suppose the two jobs are having same release time and different deadlines, then the jobs are executed based on Earliest Deadline First (EDF) scheduling [2]. At that time if jobs are scheduled based on time driven or clock driven, jobs are not executed at a time in a single processor. If jobs have different release times and different deadlines in a single processor [3-5], some jobs are scheduled based on EDF [6] and some are scheduled based on Least Release Time (LRT) [7].

In this paper we are discussing the basic scheduling techniques. These scheduling techniques are used to schedule the jobs in a simple manner.

There are two types of real time systems.

Hard real time systems
Soft real time systems

Hard real time systems are based on hard deadline and soft real time systems have soft deadlines. Hard deadline means the jobs complete its execution exactly on its deadline. Soft deadline means the jobs complete its execution on or before the deadline. Release time is defined as the time or the instance at which the job becomes ready for the execution. We can start its execution on or after its release time but not before the release time.

Deadline is defined as the time or the instance at which the job execution must and should be complete its execution. The job may be completed before the deadline but not after the deadline. Jobs are scheduled different processor either preemptive method or non-preemptive method, in preemptive [8-10] method when interrupt occurs that job will execute another processor In Non-preemptive [11] method if any interrupt occurs job will execute one processor from starting to ending without any delay.

2. Basics

2.1. Clock Driven Scheduling

In clock driven scheduling the job is having its own release time and deadline time. It should be completed within this time. Clock-driven scheduling is also called time driven scheduling. In this scheduling schedules the job at what time which job is executed by the system or specified system [12-14].

In a system that uses clock driven scheduling the parameters of hard real time jobs are fixed and known. The time is set to expire periodically without the intervention of the scheduler.

EXAMPLE:

J1 2[1, 3]
J2 3[3, 6]
J3 4[6, 10]

In this example, the jobs are scheduled based on system priority. The system priority [2] is J1>J2>J3. Figure 1 shows the scheduling of jobs based on system priority.J1 has the highest priority and J3 has the lowest priority. So initially J1 is executed. The release time of J1 is 1. So J1 starts execution on or after the release time. J1 Dead line is 3. The job is must and should be completed by the time 3. J1 has execution time of 2. So it is scheduled from 1 to 3 on the processor. Our next priority is J2. Its release time is 3 and dead line is 6. J2 has execution time 3. So J2 is scheduled in the processor from 3 to 6. The last job is J3. Its release time is 6 and deadline is 10. Now J3 is scheduled in the processor from 6 to 10.

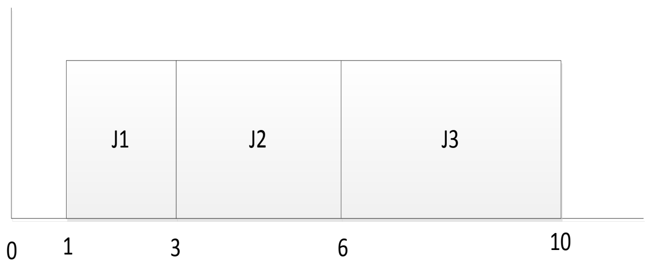

Figure 1. An Example of Illustrating Clock Driven Scheduling

2.2. Round Robin Scheduling

It is based on FIFO.If the execution time is less, then it will be continued after the all the jobs completed again round will execute continuously is the round robin approach. Round robin [15] approach is also called processor sharing algorithm.

EXAMPLE:

In single processor 4 jobs want to schedule using round robin. Total time is 10.

J1 Execution time is 4
J2 Execution time is 5
J3 Execution time is 4

J4 Execution time is 5

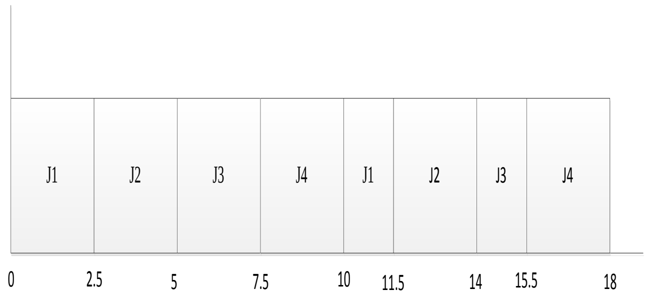

Figure 2. An Example of Illustrating Round Robin Scheduling of Different Jobs

The system priority is J1>J2>J3>J4.Initially we schedule J1 as per the system priority. So J1 is scheduled from 0 to 4 in the processor. Our next priority is J2. Its execution time is 5. So it is scheduled from 4 to 9 in the processor. Here we have only 1 is left. But still two jobs J3, J4 are to be scheduled. The Figure 2 shows the scheduling of jobs using round robin scheduling.

So to overcome this problem we go for the new technique called round robin scheduling. Here the total time is 10. Based on time in the single processor only two jobs are perfectly executed but in round robin we want to execute all jobs at that time we will take extra round. Based on round total time partitioned into 4 parts each job schedule 2.5 times. It is scheduled as shown in fig. After remaining time schedule next round.

J1 total execution time is 4. In the processor first we execute for 2.5 in first round. The remaining 1.5 we will execute in second round. Our next priority is J2 whose execution time is 5. In the first round after the execution of J1 we execute J2 for 2.5. The rest of the 2.5 we execute in second round. Next priority is J3. The total execution time is 4. In the first round we execute 2.5 and the remaining time 2.5 we execute in second round. Finally J4 is executed 2.5 in first round and rest of the 2.5 we execute in second round. Now in second round we schedule the remaining jobs as per the priority.

2.3. Weighted Round Robin

It is like as round robin but the difference is round robin method is based on FIFO, weighted round robin method based on weight. Here the weight is in terms of execution time.

Total method explained in following example.

EXAMPLE:

4 jobs are to be scheduled in a single processor. Total time is 10.

J1 Execution time is 3
J2 Execution time is 5
J3 Execution time is 4
J4 Execution time is 6

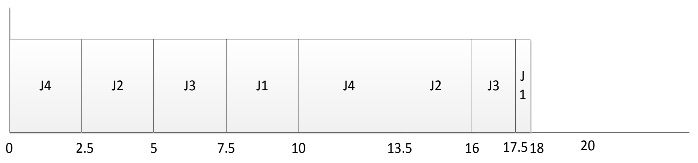

Figure 3. An Example of Weighted Round Robin Scheduling

Here we schedule the jobs based on their weights. The priority is J4>J2>J3>J1.Based on the priority we schedule J4 whose execution time is 3. If we schedule each job at a time we can't complete all 4 jobs before the time limit (16). So we schedule these jobs as we did in the above method. The scheduling of different jobs using weighted round robin is shown in Figure 3.

The total time is partitioned into 4 parts with each job executes for 2.5.As per the priority we schedule J4 from 0 to 2.5. The remaining execution time is scheduled in second round. Next we go for J2 scheduled from 2.5 to 5. The remaining 2.5 we will execute in second round. Our next priority is J3. The execution time of J3 is 4. J3 is scheduled from 5 to 7.5 and the remaining to be scheduled in second round. Similarly we execute J1 from 7.5 to 10.

In the second round we execute the rest of the jobs as per the priority. Initially we execute J4 from 10 to 12.5 here the remaining part is executed in next round. Next we go for J2 which is executed from 12.5 to 15. Our next priority is J3 scheduled from 15 to 16.5. We execute the job J1 from 16.5 to 17. Now in the final round we execute the job J4 from 17 to 18.

2.4. Priority Driven Approach

In this approach the jobs are scheduled based on priorities. There are 4 priorities listed as below

First in First out (FIFO)
Last In First Out (LIFO)
Earliest Deadline (EDL)
Least Release Time (LRT).

EXAMPLE:

The following jobs are to be scheduled in two processors. Release time of J5 is 5

$$\begin{tikzcd}
\text{J1 4} & & \\
\text{J2 2} \arrow[r] & \text{J3 1} \arrow[r] & \text{J4 2} \\
\text{J5 3} \arrow[r] \arrow[dr] & \text{J6 4} & \\
\text{J7 4} \arrow[r] \arrow[ur] & \text{J8 1} &
\end{tikzcd}$$

Here the jobs J1, J2, J5, and J7 are independent jobs. That is the execution of that job is independent of other jobs. The jobs J3, J4, J6, and J8 are dependent jobs. These jobs are executed only after the previous jobs are executed. The job J3 is executed only after the job J2 is executed and J4 is scheduled only after the job J3 is completed. Similarly the job J6 is executed after the jobs J5 and J7 are finished. J8 is scheduled after the completion of J5 and J7.

Non Pre-emptive Case:

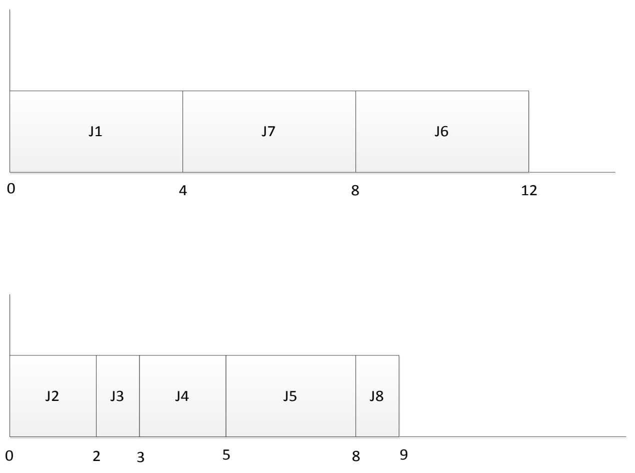

Figure 4. An Example of Priority Driven Scheduling using Non Pre-emptive

In this case each job is completed from the start to the end. The job is scheduled in a processor based on the priority. System priority is J1>J2>J3>J4>J5>J6>J7>J8.But if follow this scheduling priority, after the execution of J4 we cannot schedule the job J5 in the processor1 since J5 has the release time 5. So we execute the job J7 in processor1 and then J5 in processor2.Now the priority is J1>J2>J3>J4>J7>J5>J6>J8. Initially J1 is scheduled in processor1 from 0 to 4. Now our priority is J2. The processor1 is busy in executing the job J1 and processor2 is free. So we schedule J2 in processor2 from 0 to 2.

Here we execute J3 in processor2 from 2 to 3. Next J4 is scheduled in processor2 from 3 to 5. At this moment our job priority is J7 and processor priority is processor1. So we execute J7 in processor1 from 4 to 8. Now as per our priority we schedule J5 in processor2 since it is free. So J5 is scheduled from 5 to 8 in processor2.

Next at the time of 8 the two processors 1& 2 are free. So we execute the job J6 in processor1 from 8 to 12 and J8 in processor2 from 8 to 9.The figure 4 shows the effective scheduling using non preemptive[17] scheduling.

Pre-emptive case:

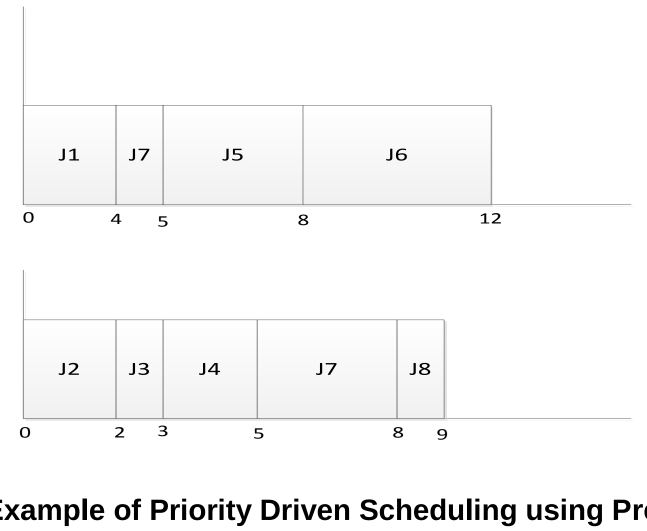

Figure 5. An Example of Priority Driven Scheduling using Preemptive Scheduling

In this case the job is not executed continuously. A part of the job is scheduled in one processor and rest is executed in another processor. Without partitioning the job we cannot write the shortest algorithm.

The job J1 is scheduled in processor1 from 0 to 4. J2 is executed in processor2 from 0 to 2 and J3 from 2 to3. Now J4 is scheduled in processor2 from 3 to 4 and remaining part is scheduled in processor1 from 4 to5. Here we used the pre-emption. Now J5 is executed in processor1 from 5 to 8 and J7 is executed in processor2 from 4 to 8. Here our job priority is J6 scheduled in processor1 from 8 to 12 and J8 executed in processor2 from 8 to 9. The figure 5 shows the scheduling of different jobs using preemptive [17-18] scheduling.

3. Scheduling Algorithm

3.1. In A Single System Based On Minimum 2 To 3 Priority Schedule:

In this example we schedule based on two priorities. Here J2 and J3 are scheduled based on Least Release Time (LRT) and J1 is scheduled based on FIFO.

J1 2[6, 10]
J2 4[0, 8]
J3 4[0, 8]

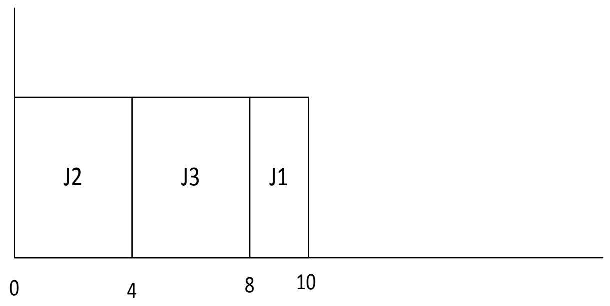

Figure 6(a). Scheduling of Jobs using LRT and FIFO

Now at the initial time two jobs J2 and J3 are ready to execute so as per the system priority we schedule J2 from 0 to 4. Next we schedule J3 from 4 to 8. Finally J1 is executed from 8 to 10. The figure 6(a) shows the scheduling of different jobs using two different algorithms.

As per FIFO, priority is J1>J2>J3
According to EDF, the priority is J2>J3>J1
And by the LRT, the priority is J2>J3>J1

EXAMPLE:

Schedule the jobs in the processor as per the priority

J1 2[0, 8]
J2 4[0, 5]
J3 2[0, 8]

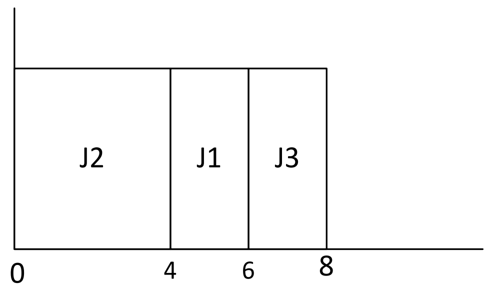

Figure 6(b). Scheduling of Jobs using EDF and FIFO

Here in this example the jobs J1 and J2 are scheduled as per the EDF priority and J3 is scheduled to FIFO.As per system priority, J1 is scheduled and next J2 is scheduled. But if we execute J1 first, we can't execute J2 within the deadline. So J2 is first executed and then we go for J1.J2 is executed in the processor from 0 to 4 as per EDF and J1 is scheduled from 4 to 6. Now J3 is executed in processor as per FIFO priority. The figure 6(b) shows the scheduling of different jobs using EDF and FIFO algorithms [19].

3.2. Multiple Jobs in 2 Processors:

The following jobs are to be executed by using 2 processors.

$$\begin{tikzcd}
J1\ 2[0, 4] \arrow[r] \arrow[dr] & J2\ 3[2, 20] \arrow[r] & J3\ 3[0, 15] \\
J4\ 4[0, 10] \arrow[ur] & J5\ 4[0, 20] &
\end{tikzcd}$$

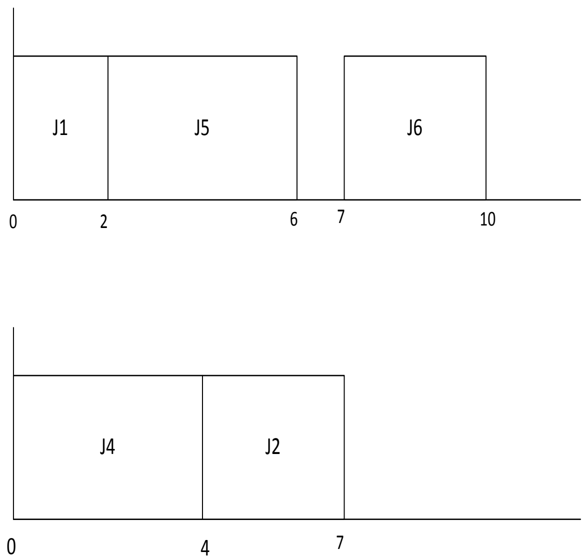

Figure 7(a). An Example of Preemptive Scheduling using Two Processors

Here J1 and J4 are independent jobs and J2, J3 and J5 are dependent jobs. System priority is J1>J2>J3>J4>J5. But since the jobs are dependent we execute the jabs as J1>J4>J5>J2>J3.So J1 is executed in the processor from 0 to 2. Next as per the priority we schedule J4 in second processor from 0 to 4. At this moment the processor1 is free so we execute J5 in first processor from 2 to 6. Now we execute J2 in the processor 2 from 4 to 7.

Now at this point in the processor1 no job is scheduled from 6 to 7 since J3 is scheduled only after the completion of J2. So J2 is executed in processor1 from 7 to 10. The Figure 7(a) shows the example of multiprocessor scheduling.

EXAMPLE:

Schedule the following jobs using 2 processors.

J1 4[2, 10]
J2 3[0, 4]
J3 2[2, 6]
J4 4[4, 10]
J5 3[4, 10]
J6 2[0, 2]

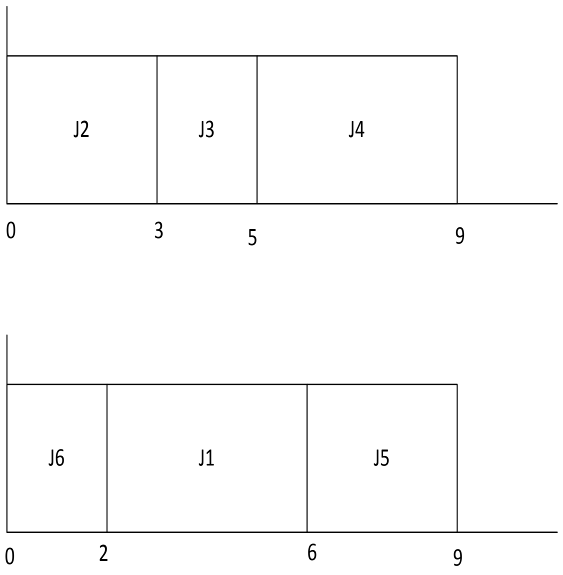

Figure 7(b). An Example of LRT Scheduling using Two Processors

The jobs are scheduled based on LRT priority J2>J6>J1>J3>J4>J5

Initially we schedule J2 in the first processor from 0 to 3. Next second processor is free so we schedule J6 in processor2 from 0 to 2. Now processor1 is busy in executing J2 so we execute J1 in processor2 from 2 to 6. Next we go for J3 execute in processor1 from 3 to 5. At this point of time second processor is busy. So we execute J4 in processor1 from 5 to 9. Finally we schedule J5 in processor2 from 6 to 9. The Figure 7(b) shows scheduling of different jobs in multiprocessor systems.

4. Evaluation

4.1. Theorem1: when pre-emption is allowed and jobs do not contend for resources. The EDF&LRT algorithm can produce a feasible schedule of a set j of jobs with arbitrary release times and deadlines on a processor if and only if j has feasible schedules.

Proof: suppose in a processor, three jobs Ji, Jk and Jt are to be executed and their release times are ri, rk and rt and are scheduled in intervals ei, ek and et furthermore their deadlines are di, dk and dt.

Any feasible schedulable J can be systematically transformed into an EDF schedulable. To see why, consider two cases; in the first case release time rk is earlier than ri and rt. But job

Jk cannot possible to schedule in execution time ei; because all the three jobs already scheduled in EDF basis. The Figure 8(a) shows the scheduling of different jobs using system priority.

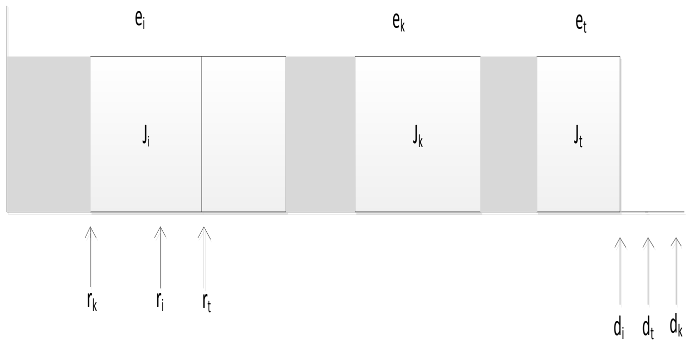

Figure 8(a). An Example of Non EDF Algorithm for Different Jobs

Given that the execution time of ek is double that of ei which is equal to et.

i.e., ei =et and ek=2ei.

as the release time of Jk is low it will be executed first and at the same time based on EDF algorithm as the dead line of dt is low it can also be executed. So, based on system priority Jk will be executed then it will be executed. Here the shaped area is waste of time. To avoid these delays the jobs will be executed first allowing the free time to be given next. Hence a feasible schedule is obtained using EDF and LRT. The Figure 8(b) shows how the system schedules the EDF algorithm.

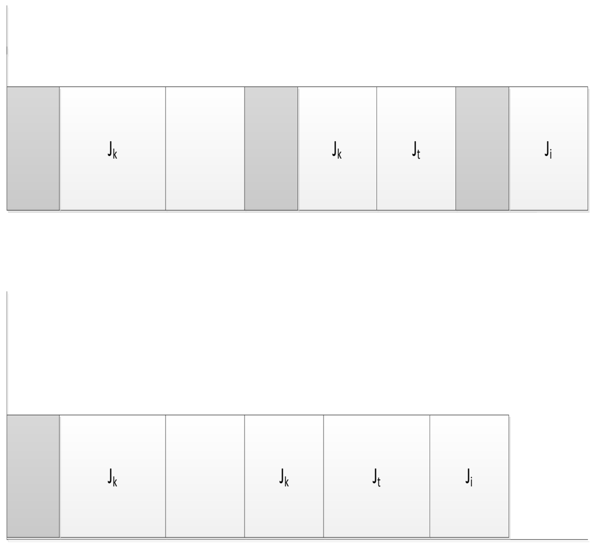

Figure 8(b). An Example of Converting a Non EDF into EDF Algorithm

Lemma1: consider the set of jobs j1 and j2 uses an earliest deadline first scheduling to schedule the jobs. At time 0, two independent jobs j1 and j2, with execution time 4 and 6 respectively, are released. Their deadlines are 18 and 7 respectively.

Proof: Initially the system priority is j1>j2, if we schedule like that it can’t produce feasible schedule. So, we adopted EDF algorithm to schedule these jobs feasible. The J1 job has deadline (d) is 18 and completion time(c) is 4. Similarly job j2 has deadline 7 and execution time is 7. Based on EDF algorithm, least deadline will be executed first. The figure below shows scheduling of different jobs using EDF algorithm. The Figure 8(c) shows the scheduling of different jobs using EDF algorithm.

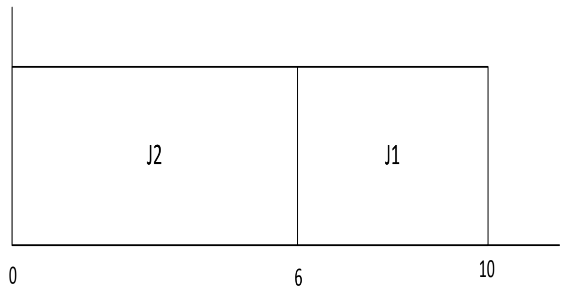

Figure 8(c). An Example of Illustrating the EDF Algorithm

At time 0, job j2 starts execution because it has least deadline than job j1, it completes execution at time 4 and hence, j2 becomes ready. Starts execution and completes by at time 10. In this way EDF produce feasible schedule.

Lemma2: consider the set of jobs j1 and j2 uses a LRT scheduling to schedule the jobs. At time 2, job j1, with execution time 4 and deadline 10. At time 0, job j2 with execution time is 6 and dead line is 7.

Proof: Initially the system priority is j1>j2, if we schedule like that it can't produce feasible schedule. So, we adopted LRT algorithm to schedule these jobs feasible. The figure below shows the scheduling of jobs using LRT algorithm.

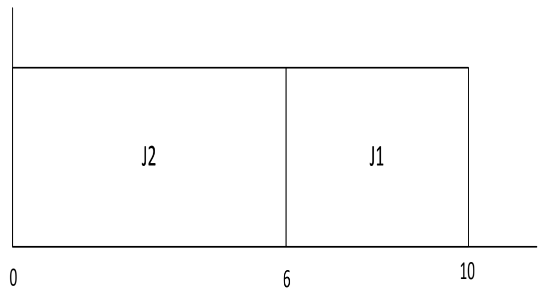

Figure 8(d). An Example of Illustrating the LRT Algorithm

At time 0, job j2 starts execution based on LRT algorithm and it completes the job at time 6. At that time job j1becomes ready and completes by time 10. In this way LRT produce feasible schedule. The Figure 8(d) shows the scheduling of multi jobs using LRT algorithm.

Corollary: The schedulbility of EDF&LRT in a single processor.

Proof: The proof directly comes from lemma1 and lemma2. Combination of these two scheduling mechanism in a single processor.

4.2. Theorem 2: when pre-emption is not allowed the LST algorithm can produce feasible schedule of a set j of jobs with arbitrary release times and deadline on a processor. If and only if j has feasible schedule.

Proof: In a processor jobs are scheduled based on priority first giving 'FIFO' priority. If it misses then it goes to EDF algorithm.

When EDF does not require knowledge of the execution times of jobs then prefer LST algorithm. We don't know exact completion time of all jobs. It is impossible for us to calculate the actual amount of slack under this circumstance; slack time is difference between dead line and completion time. Suppose execution time 'ei' .The completion time may be ei (or) ei+, then based on slack we will be schedule.

Lemma3: consider the set of jobs j1 and j2, uses a least slack time scheduling to schedule the jobs. At time 0, two independent jobs j1 and j2, with execution time 4 and 2 respectively, are released. Their deadlines are 8 and 4 respectively.

Proof: here, slack time of a job is the difference between deadline (d) and completion time(c). J1 job has deadline (d) is 8 and completion time(c) is 4, and hence the slack time (d-c) is 4. Similarly job j2 slack time is 2. Based on LST algorithm, least slack time will be executed first. The figure below shows the scheduling of given jobs. The Figure 9 shows the scheduling of different jobs using LST algorithm.

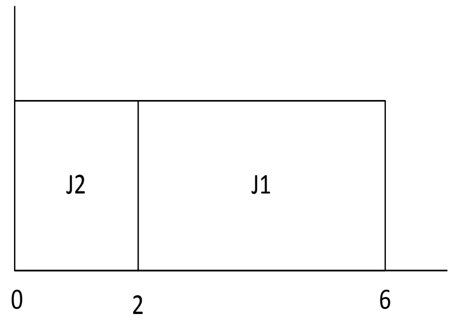

Figure 9. An Example of Illustrating the LST Algorithm

5. Conclusion

We presented too fast, best effort effective real time scheduling algorithm, it achieves high resource utilization scheduling in real time processor -based systems, both pre-emptive & Nonpreemptive Algorithms are available, we scheduled jobs easily based on priority even it is hard deadline.

References

[1] G. Manimaran and C. S. R. Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems". IEEE Transactions on Parallel and Distributed System, vol. 9, no. 3, **(1998)** March.

[2] T.-W. Kuo, W.-R. Yang and K.-J. Lin, "A Class of Rate-Based Real-Time Scheduling Algorithms", IEEE Transactions on Computers, vol. 51, no. 6, **(2002)** June.

[3] G. Martinovic, L. Budin and Z. Hocenski, "Undergraduate Teaching of Real-Time Scheduling Algorithms by Developed Software Tool", IEEE Transactions on Education, vol. 46, no. 1, **(2003)** February.

[4] P. Li and B. Ravindran, "Fast, Best-Effort Real-Time Scheduling Algorithms", IEEE Transactions on Computers, vol. 53, no. 9, **(2004)** September.

[5] P. Li, H. Wu, B. Ravindran and E. D. Jensen, "A Utility Accrual Scheduling Algorithm for Real-Time Activities with Mutual Exclusion Resource Constraints", IEEE Transactions on Computers, vol. 55, no. 4, **(2006)** April.
[6] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters", IEEE Transactions on Computers, vol. 55, no. 7, **(2006)** July.
[7] T. Chantem, X. S. Hu and M. D. Lemmon, "Generalized Elastic Scheduling for Real-Time Tasks", IEEE Transactions on Computers, vol. 58, no. 4, **(2009)** April.
[8] D. Gizopoulos, "Online Periodic Self-Test Scheduling for Real-Time Processor-Based Systems Dependability Enhancement", IEEE Transactions on Dependable and Secure Computing, vol. 6, no. 2, **(2009)** April-June.
[9] F. Zhang and A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling", IEEE Transactions on Computers, vol. 58, no. 9, **(2009)** September.
[10] B. D. Bui, R. Pellizzoni and M. Caccamo, "Real-Time Scheduling of Concurrent Transactions in Multi domain Ring Buses", IEEE Transactions on Computers, vol. 61, no. 9, **(2012)** September.
[11] S. Li, S. Ren, Y. Yu, X. Wang, L. Wang and G. Quan, "Profit and Penalty Aware Scheduling for Real-Time Online Services", IEEE Transactions on Industrial Informatics, vol. 8, no. 1, **(2012)** February.
[12] Y. Qiao, N. Wu and M. C. Zhou, "Real-Time Scheduling of Single-Arm Cluster Tools Subject to Residency Time Constraints and Bounded Activity Time Variation", IEEE Transactions on Automation Science and Engineering, vol. 9, no. 3, **(2012)** July.
[13] F. Mastrogiovanni, A. Paikan and A. Sgorbissa, "Semantic-Aware Real-Time Scheduling in Robotics", IEEE Transactions on Robotics, vol. 29, no. 1, **(2013)** February.
[14] M. Saleh and L. Dong, "Real-Time Scheduling with Security Enhancement for Packet Switched Networks", IEEE Transactions on Network and Service Management, vol. 10, no. 3, **(2013)** September.
[15] A. Subramanian, M. J. Garcia, D. S. Callaway, K. Poolla and P. Varaiya, "Real-Time Scheduling of Distributed Resources", IEEE Transactions on Smart Grid, vol. 4, no. 4, **(2013)** December.
[16] M. Chetto, "Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems", IEEE Transactions on Emerging Topics in Computing.
[17] Z. Wu, S. Zhou, J. Li and X.-P. Zhang, "Real-Time Scheduling of Residential Appliances via Conditional Risk-at-Value", IEEE Transactions on Smart Grid, vol. 5, no. 3, **(2014)** May.
[18] M. Hu, J. Luo, Y. Wang, M. Lukasiewycz and Z. Zeng, "Holistic Scheduling of Real-Time Applications in Time-Triggered In-Vehicle Networks", IEEE Transactions on Industrial Informatics, vol. 10, no. 3, **(2014)** August.
[19] N. Audsley and A. Burns, "Real-Time System Scheduling", IEEE Transactions on Industrial Informatics, vol. 10, no. 2, **(2011)** August.