

A RESEARCH PRACTICE REPORT
ON
ANALYSIS OF MIXED CRITICALITY SCHEDULERS FOR
RESOURCE CONSTRAINT SYSTEMS

BY

SHIMONI SINHA

2019H1030019G

Report completed in partial/complete fulfilment of the course

RESEARCH PRACTICE (BITS G540)



ACKNOWLEDGEMENT

Here, I would like to thank everyone who has helped or guided me throughout this research.

First and foremost, I would like to express my sincere gratitude to Prof. Biju K. Raveendran who has helped and guided me along in this research. Our discussions have taught me a lot on how to ask questions and how to work effectively towards addressing them. It was my honour and pleasure to work with him.

I would also like to thank my classmates who have provided support and from whom I have learned a lot. Their help and support will always be remembered.

Also, I would like to thank my parents for their unconditional support and guidance and for bearing with me for the last semester.

Thank you.

Shimoni Sinha

2019H1030019G

ABSTRACT

This research proposal initially gives an idea of what real time systems are and what was the motivation for creating such systems. It then introduces the idea of mixed criticality systems where tasks with different criticalities can exist concurrently in the same system.

However, this introduces the question of how the jobs for the tasks of different criticalities will be scheduled because it is hard to estimate any task's worst case execution time, and how to assign priorities to the task so that the high criticality are assured to complete within their deadline and low criticality tasks are arranged to execute such that they complete within the least time possible.

The problem is first addressed for uncore processor systems and the scheduling algorithm EDF-VD (Earliest Deadline First with Virtual Deadlines) is studied in great detail and implemented.

While the problem with uncore processors only addresses the question if how tasks will be scheduled on a core, systems with multicore processors have different types of problems that they must overcome before scheduling any mixed criticality system.

Two major problems faced by multicore processors include task allocation to cores and load balancing across all cores. This research proposal tries to address the problem of task allocation across cores by suggesting a solution by proposing a joint task partitioning and deadline shortening heuristic that uses Worst-Fit strategy for task partitioning and deadline shortening is done on the basis of demand bound functions.

Table of Contents

ACKNOWLEDGEMENT	2
ABSTRACT.....	3
List of Figures	6
List of Tables	6
1. Introduction.....	7
1.1 Motivation.....	7
2. Mixed Criticality System Model.....	9
2.1 Overview of Mixed Criticality Systems.....	9
2.2 Mixed Criticality Job Model	9
3. Related Work	11
3.1 Real Time Scheduling Theory	11
3.2 Mixed Criticality Scheduling Theory	12
4. Scheduling Mixed Criticality Implicit Deadline Tasks.....	14
4.1 EDF-VD Overview	14
4.2 Schedulability Test: Pre-Runtime Processing.....	15
4.3 Run time Scheduling Policy.....	16
4.4 Properties of EDF-VD Algorithm.....	17
4.4.1 Worst Case Reservation Scheduling Comparison.....	17
4.4.2 Task System τ with $U_2(1) = 0$	18
4.5 Speedup Factors of EDF-VD	18
4.6 Limitations of EDF-VD	20
5. Implementation of EDF-VD	21
5.1 Input File.....	21
5.2 Program Structure	21
5.2.1 Algorithm.....	22
5.3 Output Files.....	22
6. Further Research	26
6.1 Problems in multicore processor scheduling	26
6.2 The Isolation Scheduling (IS) Model.....	26
6.3 Partitioning Phase	26
6.3.1 Existing Approaches	27
6.3.2 MC Worst-Fit Decreasing and Worst-Fit Decreasing-Max	27
6.3.3 Per-Task Deadline Shortening Factor	28
6.4 Advantages.....	29
Conclusion	30

References.....	31
-----------------	----

List of Figures

Figure 4.1: EDF-VD: Pre-processing Phase	15
Figure 5.1: Sample Input File	23
Figure 5.2: Final Schedule generated for the sample input when no job exceeds its LO worst case execution time	23
Figure 5.3: Output statistics for the sample input when no job exceeds its LO worst case execution time	24
Figure 5.2: Final Schedule generated for the sample input when some job exceeds its LO worst case execution time	24
Figure 5.3: Output statistics for the sample input when some job exceeds its LO worst case execution time	25

List of Tables

Table 1.1: Criticality levels assigned to tasks categorized by effects on commercial aircraft ...	8
---	---

1. Introduction

Today, real-time embedded systems are widely used in safety-critical systems. The system faces two conflicting trends in development. First is that safety requirements are emphasized increasingly i.e. some real time tasks should never fail no matter the circumstance. Second is that since most functionalities today are implemented on integrated platform, therefore both critical and non-critical tasks have to share and compete for resources. Traditional real time systems using existing techniques fail to provide balance between the two requirements.

Real time systems are systems that provide temporal correctness. There are two types of algorithms on scheduling:

- Scheduling policies (scheduler) are algorithms that make scheduling decisions and control the run-time schedule.
- Schedulability tests are tests that are designed to execute before run-time to check if all deadlines will be met or not.

The scheduling algorithms must ensure that all deadlines are met, assuming that the tasks follow the specifications in the workload model.

1.1 Motivation

In the current scenario of real time systems, they must all pass certain safety requirements or certifications. These are provided by the certification authorities (CA). They are the ones who will verify all the standards of the system.

However, the CA tend to be very conservative. They require that the system must work correctly even under the most pessimistic and rigorous condition, even if the conditions are very unlikely to occur.

Traditional real-time scheduling techniques have a very hard time to satisfy and work efficiently on these systems. This is because the worst-case execution time (WCET) of a task, which must be known beforehand, is never required to be exceeded by the actual execution time. However, in practice knowing the exact value for WCET is very difficult and is an active area of research. Therefore, the WCET value that is used by CA to measure the system performance is a very tight upper bound. Also, in real case scenario there is always dependency between tasks and deadline miss of one can cause deadline miss of other tasks too. Therefore, we need a very pessimistic value for WCET for certification.

For example, in the avionics industry, tasks have 5 different criticality levels ranging from catastrophic to no effect. Catastrophic failure is a strict no, and should not occur no matter the condition or it can lead to disastrous results whereas no effect does not do any harm to the system. The levels have been described in table 1.1.

Level	Failure Condition	Interpretation
A	Catastrophic	Failure may cause a crash.
B	Hazardous	Failure has a large negative impact on safety or performance.
C	Major	Failure is significant, but has a lesser impact than a hazardous failure.
D	Minor	Failure is noticeable, but has lesser impact than a major failure.
E	No Effect	Failure has no impact on safety of system.

Table 1.1: Criticality levels assigned to tasks categorized by effects on commercial aircraft

2. Mixed Criticality System Model

In this chapter, we introduce the models used in mixed-criticality systems. In section 2.1 we provide an overview of mixed criticality systems. In section 2.2, an overview and some theorems related mixed criticality job model is discussed.

2.1 Overview of Mixed Criticality Systems

In this section, mixed-criticality system model is introduced by taking an example.

Example 2.1. Consider a system with two jobs: J_1 that has higher criticality and J_2 with lower criticality. Both jobs arrive at time instant 0 and have deadline equal to 10. Let us consider $C_i(1)$ to denote the worst case execution time of job J_i . This value has been assigned to the system by the system designer. Let us consider another value $C_i(2)$ which denotes the worst case execution time of job J_i . This value has been assigned to the system by the certification authority. Let us assign the values to the worst-case execution values of the system with the values $C_1(1) = 3$, $C_1(2) = 5$ and $C_2(1) = C_2(2) = 6$. It is assumed that J_1 executes first and then J_2 executes.

Let us consider the system from the point of view of the certification authority. According to them, job J_1 will execute and will complete at time instant 5 (the worst case execution time assigned to job J_1 by the certification authority is 5). Because the deadline of the job J_1 is 10, the job has completed its execution without deadline miss. The certification authority has no interest in whether job J_2 completed before its deadline or not because job J_2 is a non-critical task. Therefore, the certification authority will deem the system to be safety critical and will certify the system as such because the critical task was able to complete before its deadline.

From the system designer point of view, job J_1 will first start executing and will complete its execution at time instant 3. After that, job J_2 will start executing and will complete its execution by time instant 9. Because the deadline of both the jobs J_1 and J_2 is at time instant 10, both the jobs have executed successfully without any deadline miss. Therefore, the system designer also deems the system to be correct.

2.2 Mixed Criticality Job Model

In the example in section 2.1, the system we have considered has only two criticality levels (dual critical systems). However, in real life scenarios, there exists many systems who work with more than two criticality levels.

Let us describe the formal model and the parameters that are to be used to describe any real time job. Let us consider a value L that tells how many criticality levels are there in the system. The value L will always be a natural number.

Definition 2.1. Any mixed criticality system job is defined using a 4-tuple parameter: $J_j = (a_j, d_j, \chi_j, C_j)$, where

- a_j denotes the release time. Its value can be either an integer or a decimal number;

- d_j denotes the deadline of the task. The value of deadline can be either an integer value or a decimal value. The value of deadline is always greater than or equal to the arrival time of the job;
- χ_j denotes the criticality of the job. It always takes an integer number as its value;
- C_j denotes the worst-case execution time of the job. However, it is a collection of values. The total number of worst-case execution time for every job is equal to the number of criticality levels in the system. Each value corresponds to the worst-case execution time of that job at that criticality level. It can also take either an integer value or decimal value.

Definition 2.2. Every job is required to execute for some time within the time interval $[a_j, d_j]$ where a_j is the arrival time and d_j is the deadline of the job. This value will not be known beforehand. It is represented by c . It is known once that job finishes execution. The collection of these values (c_1, c_2, \dots, c_n) is called the scenario of the task system.

Definition 2.3. The criticality level of the system is the smallest value k such that for all values of c in the system is less than the corresponding worst case execution time of that job at the criticality level k .

Definition 2.4. A feasible schedule is one in which all the jobs having criticality level greater than or equal to k have received their time to execute within the time interval $[a_j, d_j]$.

Definition 2.5. Consider any mixed-criticality system, then the utilization factor of the system at criticality level k is given by the formula

$$U_k = \sum_{\tau_j: \chi_j \geq k} \frac{C_j(k)}{T_j}.$$

The utilization factor of all the tasks that have the criticality level k is given by the formula

$$U_k(i) = \sum_{\tau_j: \chi_j = i} \frac{C_j(k)}{T_j}.$$

3. Related Work

In this chapter, we review some works related to mixed criticality scheduling. In the first section, real time scheduling theory is introduced. In the second section, important results on mixed criticality scheduling is discussed.

3.1 Real Time Scheduling Theory

Here, the Earliest Deadline First (EDF) scheduling policy has been introduced. We have also included and proved the optimality and schedulability of the scheduling policy.

Definition 3.1. Scheduling decisions of the EDF policy is based on the next smallest value of the deadline from all the jobs that are present in the ready queue.

Theorem 3.1. EDF is an optimal scheduling algorithm because if any real time task system is schedulable for any algorithm on a uniprocessor system, then we can say that it is also schedulable by the EDF policy.

Example 3.1. Consider a dual criticality system with two jobs J_1 and J_2 with their parameter values given as:

$$J_1 = (0, 4, 2, (2, 4))$$

$$J_2 = (1, 3, 1, (1, 1)).$$

Consider EDF scheduling policy in low-criticality behaviour for the task set then: (1) at time 0, J_1 starts executing. (2) at $t = 1$, suspend J_1 and start J_2 because it has lowest deadline of the two. (3) at $t = 2$, J_2 completes so J_1 starts again. (4) at $t = 3$, J_1 also completes execution. Therefore, both J_1 and J_2 are able to meet their deadlines.

However, in high criticality behavior of the task set where criticality change occurs, EDF policy doesn't give a correct answer and thus isn't optimal anymore. To simulate the behavior, then: (1) at $t = 0$, J_1 starts executing. (2) at $t = 1$, suspend J_1 and start J_2 . (3) at $t = 2$, J_2 finishes so restart J_1 . (4) at $t = 5$, J_1 completes but misses its deadline which was $t = 4$.

Instead of EDF, we can use priority-based scheduling and assign J_1 a higher priority. So, J_1 will keep on executing even when J_2 arrives and will complete at $t = 2$ and will meet its deadline. If J_1 is still not completed, then we can discard J_2 and keep on executing J_1 till it finishes.

Theorem 3.2. (Baruah et al., 1990) Any real time task set can be schedulable by EDF only if its total utilization is less than or equal to 1.

Theorem 3.3. (Baruah et al., 1990) Deciding whether a real time system is schedulable or not is a NP-hard problem.

Theorem 3.3 tells that there is no polynomial time algorithm to test the schedulability of any arbitrary deadline real time task set, even though EDF is an optimal scheduling algorithm.

However, if the task set has an implicit deadline, then the schedulability test becomes much more efficient as compared to arbitrary deadline task set.

Theorem 3.4. (Liu and Layland, 1973) An implicit-deadline real time task set can be scheduled by EDF only if its utilization is less than 1.

3.2 Mixed Criticality Scheduling Theory

One of the main problems in scheduling of mixed criticality systems occurs due its requirement of different certifications for different criticality levels. Vestal presented the idea of applying a more conservative WCET parameter for more critical tasks in preemptive uniprocessor real-time systems, so that they can achieve more timing assurance for higher criticality level. Also, in the paper, fixed priority response time analysis was conducted using three techniques: partitioned criticality (PC), static mixed criticality (SMC) and adaptive mixed criticality (AMC).

In partitioned criticality (PC) scheme, priorities are assigned according to criticality. Within the same criticality, priority is assigned according to some standard optimal scheme (like monotonic priority assignment).

In static mixed criticality (SMC) scheme, no job is allowed to execute for more than its own C_i time after which they are either aborted or if error recovery is required, then they are descheduled until they can be fit to be scheduled again. Its main advantage over partitioned criticality scheme is that schedulability is obtained by optimizing temporal characteristics of tasks rather than priority.

In adaptive mixed criticality (AMC) scheme, if any job with criticality level L executes beyond its $C(L)$ time, then all the jobs with criticality level L will be aborted.

Later on, Baruah and Vestal proposed a more precise schedulability test based on a hybrid approach of combining fixed task priority scheduling and EDF scheduling. It improves the multi-criticality scheduling. In this paper, it has been suggested that during runtime it is allowed to prevent low criticality tasks to execute once the system goes into high criticality behavior unlike in the previous papers where low criticality tasks can run even when the system shifts into high criticality behavior.

Theorem 3.5. (Baruah et al., 2010c, 2012b) Mixed criticality problems are NP-hard even when if the task set has implicit deadline and is a dual-criticality system.

Therefore, we can say that there exists no polynomial or pseudo-polynomial time algorithm that decides weather a particular task set can be schedulable or not. Therefore, research mainly focuses on finding an approximation algorithm.

The OCBP (Own Criticality Based Priority) algorithm was proposed and analysed by Baruah in 2010. Park and Kim proposed a dynamic programming-based algorithm that works much more efficiently for dual criticality system when compared to OCBP. Baruah and Fohler proposed a time-triggered algorithm for dual criticality system with 1.618 speedup. This was

the first algorithm that allowed the CPU to remain idle even when a job existed in the ready queue (called non-work conserving mixed criticality scheduling algorithm).

The EDF algorithm is an optimal scheduling policy for uniprocessor real time systems and thus has been modified so that it can support mixed criticality task set also. This modified algorithm called EDF-VD proposed by Baruah, was first proposed for implicit deadline real time systems. The algorithm reduces the deadline for high criticality jobs by introducing a virtual deadline so that the deadline for high criticality jobs can be smaller than that of low criticality job. This enables the EDF scheduler to schedule high criticality job first. The speedup factor for EDF-VD was calculated as 1.618 and was improved to 1.33. It is also shown that EDF-VD is optimal for dual criticality implicit deadline system.

4. Scheduling Mixed Criticality Implicit Deadline Tasks

In this section, the algorithm EDF-VD suggested by Baruah for implicit deadline task systems with mixed criticality has been studied in detail. Section 4.1 provides a brief overview of what EDF-VD algorithm does. Section 4.2 discusses the pre-runtime processing in detail. Section 4.3 discusses the runtime scheduling policy of EDF-VD. Section 4.4 discusses some properties of EDF-VD and what occurs when it is run in some corner cases. Section 4.5 discusses the speedup bound of EDF-VD. Section 4.6 lists some drawbacks of EDF-VD.

4.1 EDF-VD Overview

We focus on implicit deadline task systems (systems in which the deadline and the period of a task are always the same for all the tasks in the system, i.e. $D_i = T_i$). Also, all systems are assumed to have a dual criticality i.e. they have either execute in LO criticality behavior or HI criticality behavior. So, a new algorithm EDF-VD was created. EDF-VD has the speedup guarantee of $(1 + \sqrt{5})/2$ times that of an optimal clairvoyant algorithm when both EDF-VD and the optimal clairvoyant algorithm run on the same task set.

The schedulability test for EDF-VD has polynomial run-time complexity, and the run-time complexity per scheduling decision is logarithmic in the number of tasks. Therefore, we can say that EDF-VD, in contrast to the algorithms discussed above, can be implemented in an actual system.

Let τ denote any mixed criticality implicit-deadline task system that has to be scheduled on a unit-speed preemptive processor. The EDF-VD algorithm performs the following actions:

Before run-time, EDF-VD will perform a schedulability test to check whether τ can be successfully scheduled or not. If τ is deemed to be schedulable, then we create an additional parameter, called modified period denoted by T_i^* , which is computed for all tasks that have been assigned HI criticality. The algorithm to compute the modified period is shown in the pseudo-code of figure 4.1 and is proved correct in section 4.2. It is to be noted that it is always the case that $T_i^* \leq T_i$.

Run-time scheduling is done according to EDF-VD algorithm, with modified period (called virtual deadlines) T_i^* . This virtual deadline is calculated for all HI-criticality tasks and assigned to them in the pre-processing phase that occurs before runtime. EDF-VD then uses this virtual deadline to assign priority to all the tasks with the lowest deadline getting the highest priority and so on.

Task system $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ to be scheduled on a unit-speed preemptive processor.

1. Compute x as follows:

$$x \leftarrow \frac{U_2(1)}{1 - U_1(1)}$$

2. If $(x U_1(1) + U_2(2) \leq 1)$ **then**

$\hat{T}_i \leftarrow x T_i$ for each HI-criticality task τ_i

declare success and **return**

else declare failure and **return**

Figure 4.1: EDF-VD: Pre-processing Phase

The virtual deadlines are assigned to all tasks in the system as follows. Consider a job of task τ_i arrives at a time instant t ;

- If $\chi_i = \text{LO}$, then the job is assigned a virtual deadline equal to $t + T_i$.
- If $\chi_i = \text{HI}$, then the job is assigned a virtual deadline equal to $t + \hat{T}_i$.

If any job exceeds its LO-criticality worst case execution time and still has not signalled that it has completed, then the system does the following changes to all the tasks in the system:

1. All the LO-criticality jobs that are currently in the ready queue are discarded and no LO-criticality jobs are further admitted into the ready queue.
2. All the jobs that have their criticality as HI have their deadlines modified to arrival time plus the unmodified deadline. Any new HI-criticality job that will be admitted to the ready queue will also have their deadlines modified and will be scheduled according to EDF algorithm.

4.2 Schedulability Test: Pre-Runtime Processing

In this sub-section, a detailed explanation and working of pre-runtime processing done by EDF-VD is provided. Also described is the strategy used to check whether a system is schedulable or not, and also for computing the virtual deadlines (the \hat{T}_k 's) of the tasks for systems that are found to be schedulable. This has also been shown in the form of pseudo-code in figure 4.1.

As shown the pseudo-code of figure 4.1, we first compute the x parameter and then assign value to the modified parameter (virtual deadline) \hat{T}_i as follows:

$$\hat{T}_i \leftarrow x \times T_i \tag{4.1}$$

Theorem 4.1. The condition sufficient for ensuring that EDF-VD is successfully able to schedule all LO-criticality behaviour of the task system τ is shown below:

$$x \geq \frac{U_2(1)}{1 - U_1(1)} \quad (4.2)$$

Proof. If, after replacing the deadlines of all HI-criticality tasks with virtual deadline and are still able to schedule all the tasks in the task system without missing any deadline, then from sustainability theorem, we can say that EDF can also schedule all LO-criticality tasks of τ . Because scaling the deadline of all HI-criticality tasks by factor x is equal to increasing the utilization of the system by $1/x$, we can conclude that

$$\begin{aligned} U_1(1) + \frac{U_2(1)}{x} &\leq 1 \\ \Leftrightarrow \frac{U_2(1)}{x} &\leq 1 - U_1(1) \\ \Leftrightarrow x &\geq \frac{U_2(1)}{1 - U_1(1)} \end{aligned}$$

is a sufficient condition to ensure that EDF-VD can schedule all LO-criticality behaviour of τ successfully.

Thus, EDF-VD chooses the smallest value of x such that theorem 4.1 is satisfied:

$$x \leftarrow \frac{U_2(1)}{1 - U_1(1)} \quad (4.3)$$

With the above calculated value of x , we can now find sufficient condition to ensure that EDF-VD can successfully meet all deadlines of HI-criticality tasks when the system is running in HI-criticality.

Theorem 4.2. The condition sufficient for ensuring that EDF-VD is able to successfully schedule in HI-criticality behavior of τ :

$$xU_1(1) + U_2(2) \leq 1 \quad (4.5)$$

Proof. Consider that τ is able to satisfy Condition 4.2 but EDF-VD is unable to meet all deadlines when executing in HI-criticality. Let us consider I as a minimal instance of job that was released by τ , on which deadline has been missed (i.e. EDF-VD will meet all the deadline when we schedule any proper subset of I). We can assume without loss of generality that at time = 0, the first job release in I occurs and let t_f denote the instant at which first deadline miss occurs. This deadline miss must be of a HI-criticality job because as proved in theorem 4.1 and according to condition 4.2, task system is assumed to satisfy LO-criticality job deadlines.

4.3 Run time Scheduling Policy

The jobs, during execution, are selected by following the below steps:

1. A criticality level indicator Γ is created and is initially assigned to LO.
2. While Γ has value denoting LO-criticality behaviour,
 - a. Consider a job belonging to the task τ_i arrives at time instant t , then
 - if $\chi_i \equiv \text{LO}$, the virtual deadline (or the modified parameter) of the job is assigned a value equivalent to $t + T_i$.
 - if $\chi_i \equiv \text{HI}$, the virtual deadline (or the modified parameter) of the job is assigned a value equivalent to $t + T_i^*$.
 - b. At every decision point, the job which has the lowest virtual deadline is chosen to be executed. Ties are broken by randomly any job.
 - c. If the job that is currently running in the CPU exceeds its LO-criticality worst case execution time and has not signalled for completion, then the behavior of the system is changed to HI-criticality and value of Γ is changed to HI.
3. Once Γ has value denoting HI-criticality behavior,
 - a. The virtual deadline of all HI-criticality jobs that are currently in ready queue is changed to its release time plus the original deadline.
 - b. Whenever any future HI-criticality job arrives, then it is also given a virtual deadline equal to $t + T_i$.
 - c. All LO-criticality jobs that are still in ready queue will be discarded and no LO-criticality job will be allowed to execute from that time instant.

4.4 Properties of EDF-VD Algorithm

Here, two properties of EDF-VD are discussed. The first that EDF-VD is better than worst case reservation (WCR) scheduling and the second case that EDF-VD can schedule successfully in the border case $U_2(1) = 0$.

4.4.1 Worst Case Reservation Scheduling Comparison

In worst case reservation scheduling, every task is provided some computing capacity at its own criticality level. Therefore, it follows from the utilizations of EDF that the condition

$$U_1(1) + U_2(2) \leq 1 \quad (4.8)$$

is necessary and sufficient to ensure that EDF can meet all the deadlines of the task set, with the condition that all the LO-criticality task execute upto its LO-criticality worst case execution time and all the HI-criticality task execute upto its HI-criticality worst case execution time.

Theorem 4.3. Any task set that can be scheduled by worst case reservation scheduling can also be scheduled by EDF-VD.

Proof. Any task system that can be scheduled using worst-case reservations satisfies Condition 4.8 above

$$\begin{aligned}
U_1(1) + U_2(2) &\leq 1 \\
\Rightarrow (\text{Since } U_2(1) &\leq U_2(2)) \\
U_1(1) + U_2(1) &\leq 1 \\
\Rightarrow \frac{U_2(1)}{1 - U_1(1)} &\leq 1
\end{aligned}$$

From the above equation and from equation 4.3, we come to the conclusion that the value for x factor will always be less than or equal to 1.

4.4.2 Task System τ with $U_2(1) = 0$

This situation occurs when the LO-criticality worst case execution time of all HI-criticality tasks is 0.

In such a situation, $U_2(1) = 0$. Therefore, the value for the x parameter becomes equal to 0 and the test condition in the step 2 of the pre-processing phase will evaluate to true for all tasks in the task set and $U_2(2) \leq 1$. Thus, we can say that EDF-VD can schedule any task system when

$$U_1(1) \leq 1 \text{ and } U_2(2) \leq 1.$$

is satisfied. The above equation denotes that EDF-VD can schedule any task system provided all the LO and HI-criticality tasks are separately schedulable.

Because the x parameter in the pseudo-code evaluates to 0, the modified deadline (virtual deadline) assigned to every HI-criticality task also evaluates to 0. Therefore, all the HI-criticality tasks become the earliest deadline task when they arrive in the ready queue and the system directly shifts to HI-criticality behaviour whenever a HI-criticality task with non-zero execution time is encountered.

4.5 Speedup Factors of EDF-VD

The smallest value f for which a task set that can be scheduled by an optimal clairvoyant algorithm on a unit speed processor can also be scheduled on a speed f processor by another algorithm is called the speed up factor of that algorithm. It is a metric that can be used to study the worst case behavior of different algorithms when they all try to solve the same problem.

Theorem 4.4. The speedup factor of EDF-VD is $\leq 4/3$.

Proof. Let b denote the upper bound on the LO and HI-criticality utilization for task set τ .

$$b \geq \max(U_1(1) + U_2(1), U_2(2))$$

From theorem 4.1 and 4.2, we can say that if a value of x factor exists that satisfies the below equations, then no deadline miss will occur. Theorem 4.1 states that

$$\frac{U_2(1)}{1 - U_1(1)} \leq x$$

and Theorem 4.2 states that

$$x \leq \frac{1 - U_2(2)}{U_1(1)},$$

Using both the above equations, we can substitute the value of x and solve it further as shown below:

$$\begin{aligned}
 & \frac{U_2(1)}{1 - U_1(1)} \leq \frac{1 - U_2(2)}{U_1(1)} \\
 \Leftrightarrow & \text{ (Since } U_1(1) + U_2(1) \leq b \Rightarrow U_2(1) \leq b - U_1(1) \text{)} \\
 & \frac{b - U_1(1)}{1 - U_1(1)} \leq \frac{1 - U_2(2)}{U_1(1)} \\
 \Leftrightarrow & \text{ (Since } U_2(2) \leq b \text{)} \\
 & \frac{b - U_1(1)}{1 - U_1(1)} \leq \frac{1 - b}{U_1(1)} \\
 \Leftrightarrow & \\
 & (U_1(1))^2 - U_1(1) + (1 - b) \geq 0 \tag{4.10}
 \end{aligned}$$

If we substitute $b = \frac{3}{4}$, then equation 4.10 become

$$\begin{aligned}
 & (U_1(1))^2 - U_1(1) + \frac{1}{4} \geq 0 \\
 \Leftrightarrow & \\
 & (U_1(1) - \frac{1}{2})^2 \geq 0
 \end{aligned}$$

Thus, we have proved that any clairvoyant schedulable task system is also schedulable on a processor with $\frac{3}{4}$ times the speed by the EDF-VD algorithm.

Theorem 4.5. There exists no clairvoyant algorithm which can provide speedup better than $\frac{4}{3}$ on an implicit deadline system.

4.6 Limitations of EDF-VD

Once any high criticality task overruns its low criticality WCET, following are the behaviours that are assumed about the task model as a whole:

1. All the low criticality tasks are to be discarded. However, this is a pessimistic assumption to make because the low criticality tasks also have some timing performance.
2. If any one job of a single task overruns its WCET for low criticality, all the high criticality jobs of all tasks are naturally assumed to exhibit high criticality behaviour (mode switch). This assumption is also overly pessimistic as mode switches of all tasks are independent of each other.
3. High criticality tasks are directly transitioned to exhibit the WCET for high critical level whenever a mode switch occurs. However, tasks will rarely ever reach their most pessimistic WCET during runtime and therefore assuming as such will lead to unnecessary resource allocation to that task.

All the above behaviours assumed are very pessimistic for the task execution behaviour.

5. Implementation of EDF-VD

This section gives the algorithm based on which EDF-VD scheduling has been implemented. The first sub-section describes the input file format of the program. The second sub-section describes the program structure and algorithm in detail. The third sub-section shows the output of the code.

5.1 Input File

The first line of the input line contains the number of tasks (N) in the task set. The next N lines contains tasks with parameters as,

1. Phase – It denotes the initial arrival of the job of the task. It takes only integer value.
2. Period – It denotes the time after which new jobs of the task will be released. It can take only integer value.
3. Criticality – It denotes the criticality of the task. Value 1 denotes LO criticality task and value 2 denotes HI criticality task. It can take only integer value.
4. Worst Case Execution Time – It shows the worst execution time a job of that task will take when it runs at that criticality value. For LO criticality tasks, the value of worst-case execution time at HI criticality will be 0. It takes two values, one at LO criticality and the other at HI criticality. It can take integer and float values.
5. Relative Deadline – It denotes the relative deadline of the task. It can take integer and float values.

5.2 Program Structure

Take command line argument for input file. If file doesn't exist, then throw error and exit the program.

1. Load the periodic task set from the input file and load them into struct 'taskSet'.
2. Calculate utilizations of all tasks. Check if U11 and U22 values are less than or equal to 1. Otherwise, return error and terminate the program. There are three types of utilizations calculated:
 - a. U11 – Utilization of all LO criticality tasks at their LO worst case execution time.
 - b. U22 – Utilization of all HI criticality tasks at their HI worst case execution time.
 - c. U21 – Utilization of all HI criticality tasks at their LO worst case execution time.
3. Calculate hyper period using LCM method.
4. Calculate virtual deadlines of all HI criticality tasks by calculating x factor and multiplying it with the relative deadline of that task. If the task has LO criticality, then its virtual deadline and relative deadline are the same.

$$x = U21 / (1 - U11)$$

5. Generate final schedule and print all output parameters by running 'EDFSchedule' function.

5.2.1 Algorithm

1. Initially system is in LO criticality.
2. Create a new array of struct of type job that contains all job parameters for each task.
3. Initialize all job struct with initial values for every task.
4. Create a queue that contains all jobs that are present at a particular instant t and are sorted based on their virtual deadlines.
5. Calculate the minimum next job arrival time.
6. Let 'curr' contain the job that contains job at the top of the queue. Let t be the current time instant.
7. If ($\text{curr.executionTime} + t \leq \text{minimum next job arrival time}$), then
 - a. execute the current job completely and remove it from the queue.
 - b. Goto step 5.
8. If ($\text{curr.executionTime} + t > \text{minimum next job arrival time}$), then
 - a. execute current job upto minimum next job arrival time,
 - b. reduce current job remaining execution time,
 - c. insert new job into queue.
 - d. If $\text{priority}(\text{curr}) > \text{priority}(\text{new job})$, then continue curr.
Else, goto step 5.
9. If ($\text{curr.executionTime} + t > \text{curr.worstCaseExecutionTime(LO)}$), then
 - a. Change criticality of system to HI.
 - b. Delete all LO criticality jobs from the queue.
 - c. Change the virtual deadlines of all HI criticality jobs to original absolute deadline.
 - d. Goto step 5.
10. Exit

5.3 Output Files

The above program is run for two cases.

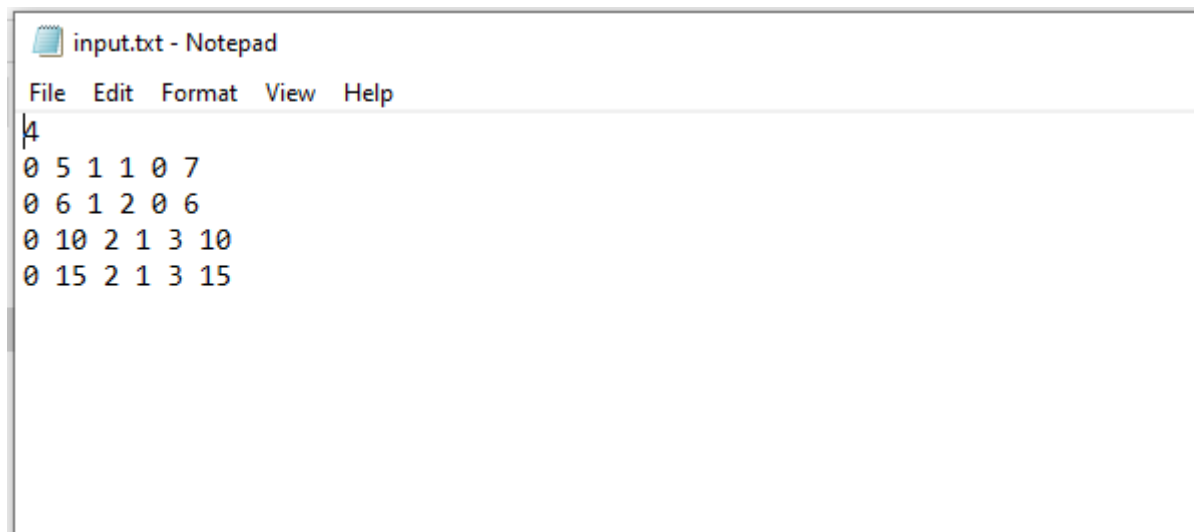
1. When all tasks are able to execute completely without the system changing its criticality.
2. One of the jobs exceeds its worst-case execution time for LO criticality and thus sends the system to HI criticality.

For the first case, the final schedule and output statistics are stored in the file 'schedule1.txt'. For the second case, the final schedule and output statistics are stored in the file 'schedule2.txt'.

The output statistics that are displayed in the output file include

1. Total number of context switches
2. Total number of preemptions.
3. Maximum, minimum and average response time for all tasks.
4. Absolute and relative jitter time for all tasks.
5. Maximum, minimum and average waiting time for all tasks.

Below figures show a sample input task set and the corresponding output generated.

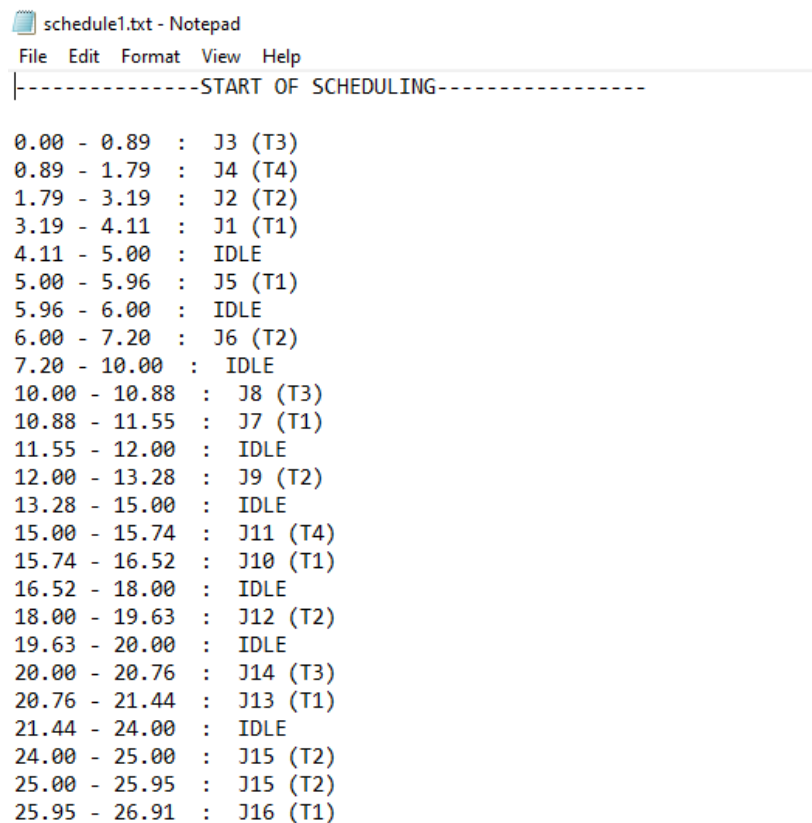


```

input.txt - Notepad
File Edit Format View Help
4
0 5 1 1 0 7
0 6 1 2 0 6
0 10 2 1 3 10
0 15 2 1 3 15

```

Figure 5.1: Sample Input File



```

schedule1.txt - Notepad
File Edit Format View Help
-----START OF SCHEDULING-----

0.00 - 0.89 : J3 (T3)
0.89 - 1.79 : J4 (T4)
1.79 - 3.19 : J2 (T2)
3.19 - 4.11 : J1 (T1)
4.11 - 5.00 : IDLE
5.00 - 5.96 : J5 (T1)
5.96 - 6.00 : IDLE
6.00 - 7.20 : J6 (T2)
7.20 - 10.00 : IDLE
10.00 - 10.88 : J8 (T3)
10.88 - 11.55 : J7 (T1)
11.55 - 12.00 : IDLE
12.00 - 13.28 : J9 (T2)
13.28 - 15.00 : IDLE
15.00 - 15.74 : J11 (T4)
15.74 - 16.52 : J10 (T1)
16.52 - 18.00 : IDLE
18.00 - 19.63 : J12 (T2)
19.63 - 20.00 : IDLE
20.00 - 20.76 : J14 (T3)
20.76 - 21.44 : J13 (T1)
21.44 - 24.00 : IDLE
24.00 - 25.00 : J15 (T2)
25.00 - 25.95 : J15 (T2)
25.95 - 26.91 : J16 (T1)

```

Figure 5.2: Final schedule generated for the sample input when no job exceeds its LO worst case execution time

```

-----OUTPUT STATISTICS-----

1. TOTAL CONTEXT SWITCHES = 17
2. TOTAL PREEMPTION COUNT = 1

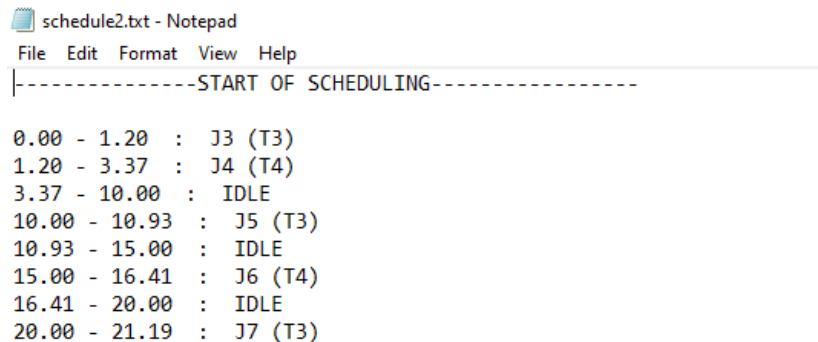
3. RESPONSE TIMES
TASK ID MAX RESPONSE TIME      MIN RESPONSE TIME      AVG RESPONSE TIME
1         4.11                0.96                  1.91
2         3.19                1.20                  1.85
3         0.89                0.76                  0.84
4         1.79                0.74                  1.26

4. ABSOLUTE AND RELATIVE RESPONSE TIME JITTER
TASK ID ABS. JITTER      REL. JITTER
1         3.15           26.91
2         1.99           25.95
3         0.13           20.76
4         1.05           15.74

5. WAITING TIMES
TASK ID MAX WAITING TIME      MIN WAITING TIME      AVG WAITING TIME
1         3.00                0.00                  0.50
2         1.00                0.00                  0.20
3         0.00                0.00                  0.00
4         0.00                0.00                  0.00

```

Figure 5.3: Output statistics for the sample input when no job exceeds its LO worst case execution time



```

schedule2.txt - Notepad
File Edit Format View Help
|-----START OF SCHEDULING-----

0.00 - 1.20 : J3 (T3)
1.20 - 3.37 : J4 (T4)
3.37 - 10.00 : IDLE
10.00 - 10.93 : J5 (T3)
10.93 - 15.00 : IDLE
15.00 - 16.41 : J6 (T4)
16.41 - 20.00 : IDLE
20.00 - 21.19 : J7 (T3)

```

Figure 5.4: Final schedule generated for the sample input when some job exceeds its LO worst case execution time


```

-----OUTPUT STATISTICS-----

1. TOTAL CONTEXT SWITCHES = 5
2. TOTAL PREEMPTION COUNT = 0

3. RESPONSE TIMES
TASK ID MAX RESPONSE TIME      MIN RESPONSE TIME      AVG RESPONSE TIME
3         1.20                  0.93                   1.11
4         3.37                  1.41                   2.39

4. ABSOLUTE AND RELATIVE RESPONSE TIME JITTER
TASK ID ABS. JITTER      REL. JITTER
3         0.26            21.19
4         1.96            16.41

5. WAITING TIMES
TASK ID MAX WAITING TIME      MIN WAITING TIME      AVG WAITING TIME
3         0.00                0.00                 0.00
4         1.00                0.00                 0.50

```

Figure 5.5: Output statistics for the sample input when no job exceeds its LO worst case execution time

6. Further Research

6.1 Problems in multicore processor scheduling

The major obstacles that one faces while scheduling in multicore systems are:

1. Task partitioning – How to partition the task so that all the cores are equally utilized.
2. Load Balancing – If any core is over utilized, then what are the jobs that can be sent to some other core so that the load on this core gets reduced while also checking that the cost of sending the job to other core is less than the cost required to execute the job on the same core.

In this section we will try to solve the problem of task partitioning in multicore systems.

6.2 The Isolation Scheduling (IS) Model

The isolation scheduling (IS) model will partition a hardware platform among different task classes so that, at any particular instant of time, the jobs of only one task class is able to utilize the platform resources. This model is used for homogeneous multicore processors with m cores and which share all the on-chip resources.

For any task set, it is assumed that tasks are partitioned into K task classes $S = \{S_k \mid 1 \leq k \leq K\}$. If the tasks need to satisfy timing constraints, then the set of task classes are defined to be IS-schedulable if the timing constraints are met, and all the task classes are mutually exclusive. The formal definition of task class is as follows:

Definition 6.1. IS constraint is being enforced by a scheduling policy if at any time instant t , all the tasks executing across all cores belong to the same task class. Mathematically, we can say that for any pair of tasks $\tau_i \in S_a$ and $\tau_j \in S_b$ that are executing simultaneously in different cores at time t , then $S_a = S_b$.

6.3 Partitioning Phase

The partitioning phase has to assign tasks to cores so that it can guarantee the tasks are schedulable in both LO and HI criticality modes. To guarantee schedulability, two conditions given below

$$x \geq U_2(1) / (1 - U_1(1))$$

and

$$\sum_{\tau_i \in S_{HI}} \frac{u_i(HI)}{u_i(LO) + (1 - x)} \leq 1.$$

should hold for each core.

6.3.1 Existing Approaches

Burns et al. has proposed a scheme for partitioning the tasks using cyclic executives. The processor executes in frames where every frame contains one or more tasks. The schedule derived is a static schedule and is thus restrictive because deadline of all tasks has to be multiple of frame size.

Kelly et al. has proposed and compared many bin packing algorithms to assign mixed-criticality tasks to multicore systems. The disadvantage of this approach is that it has restricted application.

Three different partition schemes are evaluated: First Fit (FF), Worst Fit (WF), First Fit with Branch and Bound (FFBB). First Fit has been shown to not be a good heuristic because it always tries to fill up the first core as much as possible before moving onto the second core. This leads to over-utilization of some cores and under-utilization of other cores. Worst Fit and First Fit with Branch and Bound have almost equal performance with First Fit with Branch and Bound being better when utilizations are high.

Gu et al. has proposed a different approach called Mixed criticality Partitioning with Virtual Deadlines (MPVD) for partitioning. In this approach, the HI-criticality tasks are first assigned to the cores using Worst Fit algorithm. All HI-criticality tasks are assigned virtual deadline based on deadline shortening approach. After all HI-criticality tasks have been assigned, LO-criticality tasks are assigned using First Fit.

Baruah et al. has proposed a partitioned EDF-VD scheme. First HI-criticality tasks are assigned using First Fit. Maximum utilization of any core can be $\frac{3}{4}$. After that LO-criticality tasks are also assigned using First Fit. The main disadvantage of this approach is that it uses First Fit approach.

6.3.2 MC Worst-Fit Decreasing and Worst-Fit Decreasing-Max

Two heuristic algorithms Worst Fit Decreasing and Worst Fit Decreasing Max are evaluated for partitioning tasks.

In the Worst Fit Decreasing heuristic, all the tasks are first sorted in their criticality level in descending order of their LO mode density. After this, all HI-criticality tasks are assigned to cores.

The aim is to assign a HI-criticality task to each of the m cores. For any given core j , we first calculate the maximum value of deadline shortening factor x^*_j which causes minimum increase in LO mode density, so that transitions from LO to HI mode is feasible for all HI criticality tasks.

We then calculate the LO mode density of all cores using the new deadline shortening factor. By using the greedy approach, we choose the assignment that has the minimum mean squared error. This is because the mean squared error causes the minimum deviation in the LO mode density across all cores.

This leads to minimum schedulability loss as all the load is balanced across all cores. After all the HI-criticality tasks are assigned, we then assign the LO-criticality tasks to cores with the lowest LO mode density.

In the Worst Fit Decreasing Max heuristic, after all the HI-criticality tasks have been assigned to the cores using the same method as in Worst Fit Decreasing heuristic, the LO mode density of all the cores are set to the maximum value of the LO mode density across all the cores.

6.3.3 Per-Task Deadline Shortening Factor

In the previous sub-section, the partitioning of tasks, a common deadline shortening factor was considered for all HI-criticality tasks executing on a core. However, it is a restrictive assumption and the schedulability of tasks can be increased by considering deadline shortening factor for each task instead of each core. Here, we first present a deadline shortening factor proposed by Ekberg and Yi and is referred to as EY algorithm. Then, a new algorithm is proposed for partitioning the task set.

EY Heuristic. EY is a heuristic approach for assigning per-task virtual deadlines to HI criticality tasks in sporadic, constrained-deadline dual-criticality systems. EY is designed to improve dual-criticality schedulability on single cores. Its basic principle lies in the construction of separate task demand bound functions for tasks in LO mode and HI mode. We use $D_i(\text{LO})$ and $D_i(\text{HI})$ to denote the LO and HI mode relative deadlines of task τ_i . For τ_i , we use $\text{dbf}_{\text{LO}}(\tau_i, t)$ and $\text{dbf}_{\text{HI}}(\tau_i, t)$ to denote the LO and HI mode demand bound functions, respectively. These functions are defined as follows:

$$\text{dbf}_{\text{LO}}(\tau_i, t) = \max \left\{ \left\lfloor \frac{t - D_i(\text{LO})}{T_i} \right\rfloor + 1, 0 \right\} \cdot C_i(\text{LO}),$$

where $D_i(\text{LO}) = D_i$ if $\tau_i \in S_{\text{LO}}$ and $D_i(\text{LO}) \leq D_i(\text{HI}) = D_i$ if $\tau_i \in S_{\text{HI}}$. For $\tau_i \in S_{\text{HI}}$:

$$\text{dbf}_{\text{HI}}(\tau_i, t) = \text{full}(\tau_i, t) - \text{done}(\tau_i, t),$$

where,

$$\begin{aligned} \text{full}(\tau_i, t) &= \max \left\{ \left\lfloor \frac{t - (D_i(\text{HI}) - D_i(\text{LO}))}{T_i} \right\rfloor + 1, 0 \right\} \cdot C_i(\text{HI}), \\ \text{done}(\tau_i, t) &= \begin{cases} \max\{C_i(\text{LO}) - n & \text{if } D_i(\text{HI}) > n \geq D_i(\text{HI}) - D_i(\text{LO}) \\ + D_i(\text{HI}) - D_i(\text{LO}), 0\}, & \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

and $n = t \bmod T_i$. For $\tau_i \in S_{\text{LO}}$: $\text{dbf}_{\text{HI}}(\tau_i, t) = 0, \forall t$. For all tasks executing on a unit-speed core, we have the following schedulability conditions:

$$\forall t \geq 0 : \sum_{\tau_i \in S} dbf_{LO}(\tau_i, t) \leq t \quad \text{and} \quad \sum_{\tau_i \in S} dbf_{HI}(\tau_i, t) \leq t .$$

Proposed Solution: For dual criticality system, the solution containing the combination of task partitioning using worst fit decreasing max heuristic and using EY algorithm as deadline shortening heuristic is proposed for isolation scheduling model. Initially, all the tasks in the system are sorted in non-decreasing order based on their LO mode density. The order of assignment of tasks to cores is first all HI-criticality tasks and then all LO-criticality tasks. The HI-criticality tasks are assigned to the core with the smallest LO-mode density. After every assignment, LO mode density is recalculated for all HI-criticality tasks so that both LO and HI mode densities are reschedulable. Because EY is a greedy solution, if EY heuristic is unable to find a feasible assignment, then the next smallest LO mode density core is chosen and it goes until either a feasible assignment is found or we exhaust all the possibilities and thus deem the task set to be unschedulable. When all the HI-criticality tasks have been assigned to their cores, the value of LO mode density of all cores is set to the maximum value of the LO mode density across all cores. Then, the LO-criticality tasks are assigned to the cores based on worst fit heuristic.

6.4 Advantages

1. The proposed solution has a lot more schedulability due to its freedom to select individual deadline shortening factor for all tasks individually.
2. The Worst Fit Max heuristic is a good candidate for partitioning mainly because:
 - a. It is most effective in finding any unschedulable task set in the partitioning phase itself.
 - b. It also creates server schedules with larger cycle lengths. This leads to less frequent class switches and also causes very less runtime overhead.

Conclusion

1. Mixed criticality tasks in a single processor system can be scheduled if they satisfy theorem (4.1) and (4.2) respectively.
2. EDF-VD is a schedulability algorithm that can increase the probability of scheduling HI criticality tasks over LO criticality tasks by introducing virtual deadlines.
3. However, EDF-VD provides a very pessimistic execution time for tasks that have HI criticality and therefore have unnecessary overbooking of resources.
4. In multi-core systems, task partitioning and load balancing are two main issues that are faced by the systems.
5. One solution for task partitioning that is proposed is to provide task partitioning along with deadline shortening. Assign HI criticality tasks first to cores with lowest LO mode density. Then assign LO criticality tasks to the cores using worst-fit strategy.
6. The proposed solution enables maximal schedulability due to its freedom to select individual deadline shortening factors per task.

References

- [1].S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," 28th IEEE International Real-Time Systems Symposium (RTSS 2007), Tucson, AZ, 2007, pp. 239-243.
- [2].Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2012a). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems, ECRTS '12, Pisa (Italy). IEEE Computer Society.
- [3].Baruah, S., Burns, A., and Davis, R. (2011b). Response-time analysis for mixed criticality systems. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Vienna, Austria. IEEE Computer Society Press.
- [4].F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems Journal*, 46(3):305–331, 2010. 9
- [5].S.K. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In ECRTS, pages 147–155, 2008. 5, 15.
- [6].G. Chen, N. Guan, B. Hu and W. Yi, "EDF-VD Scheduling of Flexible Mixed-Criticality System with Multiple-Shot Transitions," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2393-2403, Nov. 2018.
- [7].O. R. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pages 1051–1059, 2011.
- [8].C. Gu, N. Guan, Q. Deng, and W. Yi. Partitioned mixed-criticality scheduling on multiprocessor platforms. In Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1–6, 2014.
- [9].P. Ekberg and W. Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems*, 50(1):48–86, 2013.
- [10]. Burns, T. Fleming, and S. Baruah. Cyclic executives, multi-core platforms and mixed criticality applications. In 27th Euromicro Conference on Real-Time Systems (ECRTS), pages 3–12, 2015.
- [11]. S. Baruah, B. Chattopadhyay, H. Li, and I. Shin. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems*, 50(1):142–177, 2014.