RTOS System Scheduling With WCET Estimation Using EDF-VD Algorithm

Sethupathi.M[1], Sivaramakrishnan.N[2], Theeijitha.S[3], G.Naveen Balaji[4]
[1,2,3]*UG Student, Department of ECE, SNS College of Technology, Coimbatore-641035, Tamil Nadu, India*
[4]*Assistant Professor, Department of ECE, SNS College of Technology, Coimbatore-641035, Tamil Nadu, India*

I. INTRODUCTION

System is a way of working, organizing or doing one or more tasks according to the fixed plan, program or set of rules. A system is also an arrangement in which all its units assemble and work together according to the plan or program. An embedded system is combined working of hardware and software or additional mechanical or technical component to perform desired function. Any sort of device which includes programmable computer but itself is not intended to be general purpose computer is said to be embedded system.

The lower layer of an embedded system is printed circuit board that includes busses and semiconductor devices. The upper layer is mainly application layer in between these two layers there are another two essential layers called device drivers and communication protocols. These features enable embedded systems to be relatively static and simple in functionality. However, there is a requirement for low cost, small physical footprint and negligible electrical or electronic radiation and energy consumption. Simultaneously they need to be physically rugged and impervious to external electrical and electronic interference.

Therefore, embedded systems invariably are limited resources available in terms of memory, CPU, screen size, a limited set (or absence) of key inputs, diskless operations-these parameters play a crucial part during the design, development and testing of such systems.

1.1 Embedded System Characteristics

In general, embedded systems are designed to perform any particular predefined task that must meet any real-time constraint. The main difference between a computer and an embedded system is a computer is used to perform a specific task that is pre-defined by the manufacturers.

Here, meeting all the real-time system constraints is a very important characteristic of an embedded system. A real-time constraint is divided into two parts.one is hard real-time system and the other is soft real-time system. Hard real-time system means it must meet all its deadlines with a zero degree of flexibility and it is acceptable to be little flexible in the soft real-time system. It is not necessary to be standalone always for the embedded devices. Actually, most of the embedded systems are integrated within a large computerised device. Devices such as MP3s, cameras and TV remotes are the examples of standalone embedded devices.

The term 'firmware' is used to refer the program instructions written for embedded systems. It is stored in ROM (read only memory) or in a flash memory chip. Resources like computer hardware do not need much time to run. Another important characteristic is the dedicated user interface.It may range from no user interface to complex graphical user interface. Hand held device such as joystick which needs to be pointed with the screen is a good example of user interface systems. Size and weight should be less for an embedded device. For that reason, microcontrollers are used in embedded devices to deliver the best performance on demand. Manufacturer companies try to keep the lowest price of their products. Using sensors and actuators it may be also connected to physical environment.

1.2 Embedded Architecture

An embedded system is built around a processor. The central processing unit does the necessary computation based on the input it receives from various external devices. The functionality of the CPU is an embedded system is same as the functionality of the CPU in a desktop, except that the CPU in an embedded system is less powerful.

The processor has limited internal memory, and if this internal memory is not sufficient for a given application external memory devices are used. The hardware also includes any components that facilitates the user-application interaction, such as display units, keypads etc.

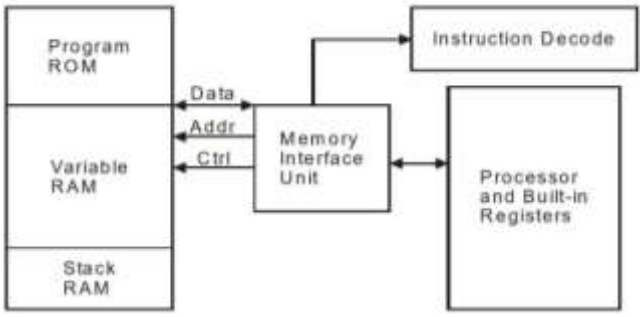


Figure 1. System Concept of Embedded Architecture

A number of 16-bit and 32-bit microprocessors are available from ARM, Atmel, Intel, Motorola, National Semiconductors, etc. In order to develop an embedded system with these processors, you great deal of peripheral circuitry. However, microprocessors higher clock speeds and word-length, so they are capable of addressing higher memory. These processors are used for high-end applications such as handheld computers, Internet access Devices, etc.

The memory used in embedded systems can be either internal or external. The internal memory of a processor is very limited. For small applications, if this memory is sufficient, there is need to used external memory.

II. BACKGROUND

2.1 Cyber Physical Systems

Unlike more traditional embedded systems, a full-fledged CPS is typically designed as a network of interacting elements with physical input and output instead of as standalone devices. The notion is closely tied to concepts of robotics and sensor networks with intelligence mechanisms proper of computational intelligence leading the pathway.

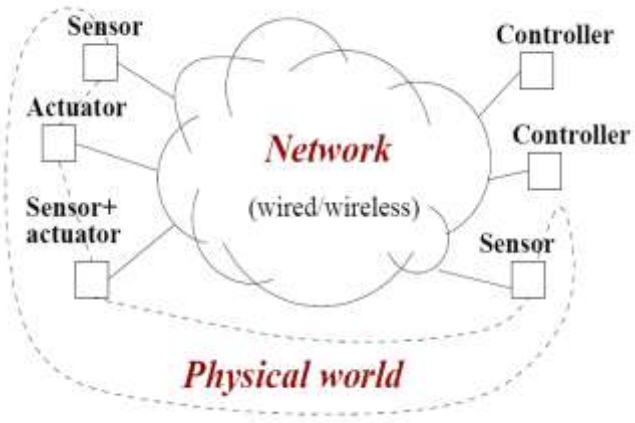


Figure 2. Architecture of Cyber Physical Systems

Ongoing advances in science and engineering will improve the link between computational and physical elements by means of intelligent mechanisms, dramatically increasing the adaptability, autonomy, efficiency, functionality, reliability, safety, and usability of cyber-physical systems.This will broaden the potential of cyber-physical systems in several dimensions, including: intervention (e.g., collision avoidance); precision (e.g., robotic surgery and nano-level manufacturing); operation in dangerous or inaccessible environments (e.g., search and rescue, firefighting, and deep-sea exploration; coordination (e.g., air traffic control, war fighting); efficiency (e.g., zero-net energy buildings); and augmentation of human capabilities (e.g., healthcare monitoring and delivery).

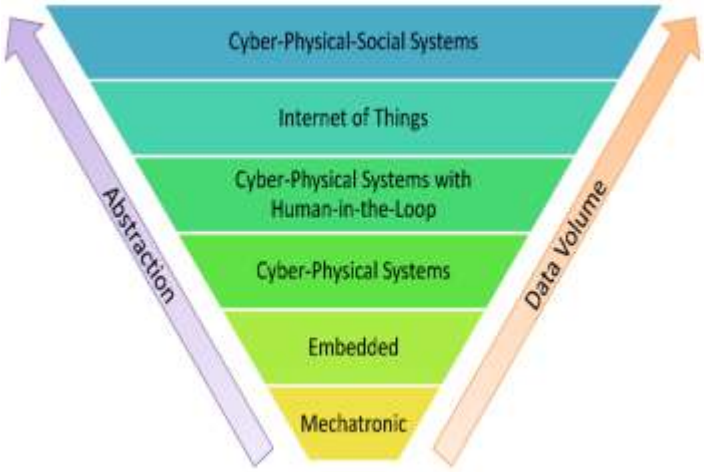


Figure 3. Work Flow Diagram of CPS

2.1.1 Definition

A cyber-physicalsystem (CPS) integratescomputing, communication and storage capabilities with monitoring and / or control of entities in the physical world, and must do so dependably, safely, securely, efficientlyand in real-time.Cyber-physical systems will transform how we interact with the physical world just like the Internet transformed how we interact with one another.

III. SCHEDULING

In computing, scheduling is the method by which work specified by some means is assigned to resources that complete the work. The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

A scheduler is what carries out the scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as in load balancing), allow multiple users to share system resources effectively, or to achieve a target quality of service. Scheduling is fundamental to computation itself, and an intrinsic part of the execution model of a computer system; the concept of scheduling makes it possible to have computer multitasking with a single central processing unit (CPU).

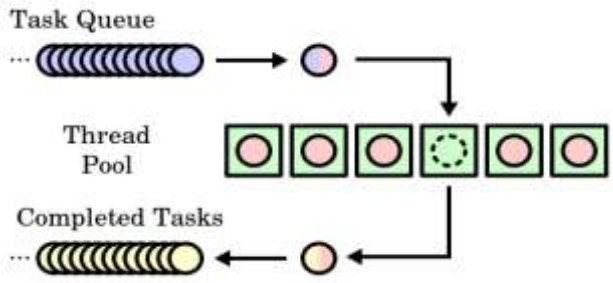


Figure 4. Task Arrival and Task Completed in a Scheduler

A scheduler may aim at one or more of many goals, for example: maximizing throughput (the total amount of work completed per time unit); minimizing wait time (time from work becoming enabled until the first point it begins execution on resources); minimizing latency or response time (time from work becoming enabled until it is finished in case of batch activity or until the system responds and hands the first output to the user in case of interactive activity) or maximizing fairness (equal CPU time to each process, or more generally appropriate times according to the priority and workload of each process). In practice, these goals often conflict (e.g. throughput versus latency), thus a scheduler will implement a suitable compromise. Preference is measured by any one of the concerns mentioned above, depending upon the user's needs and objectives.

In real-time environments, such as embedded systems for automatic control in industry (for example robotics), the scheduler also must ensure that processes can meet deadlines; this is crucial for keeping the system stable. Scheduled tasks can also be distributed to remote devices across a network and managed through an administrative back end.

3.1 Real Time Operating System

A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays. RT systems require specific support from OS. The operating system processing time requirements are measured in tenths of seconds or the shorter increments of time. A real time system is a time bound system which has well defined fixed time constraints. Processing must be done within the defined constraints or the system will fail. They either are event driven or time sharing. Event driven systems switch between tasks based on their priorities while time sharing systems switch the task based on clock interrupts. Most RTOS's use a pre-emptive scheduling algorithm. An RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more frequently dedicated to a narrow set of applications.

Real-Time Scheduling Algorithms are a special class of algorithms of which it is required that they can guarantee a process will be done before its deadline. The only way these algorithms can work is if they at least know when the deadline for a process is, and how much the process takes of the system. Only if the system is not overloaded (subjective term) can the threads be guaranteed to finish before their deadline.

Each task has to be scheduled Xt times a second, or every Yt milliseconds (Yt = 1000 / Xt). Each run of that task takes at most Zt milliseconds. This task then creates a load of Lt = Zt / Yt.

The system as a whole has a load L, which is the sum of all task-loads: L = E Lt. If the system load exceeds 0.7 (in some rare cases it can be slightly larger, but we don't count them) the system is unschedulable using Rate Monotonic Scheduling. If this system load exceeds 1.0 it is unschedulable for any real-time system. Note that for normal systems any load is possible, including the ones that are extremely large. They will make the system very unusable though.

3.1.1 Kernel

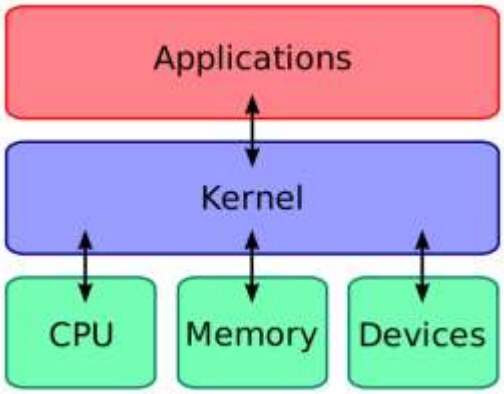


Figure 5. Architecture of Kernel

A kernel is the central part of the operating system that manages the operation of the computer and the hard ware most notably memory and CPU unit.There are two types of kernels. A microkernel, which only contains basic functionality.A monolithic kernel, which contains many device drivers.Itfunctions at a basic level, communicating with hardware and managing resources, such as RAM and the CPU. The kernel performs a system check and recognizes components, such as the processor, GPU, and memory. It also checks for any connected peripherals.Kernel is the software responsible for running programs and providing secure access to the machine's hardware.Since there are many programs, and resources are limited, the kernel also decides when and how long a program should run.

3.1.2 Process

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating systems. Operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing. Process Scheduling Queues. The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue. Job queue − This queue keeps all the processes in

the system. Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue .Device queuesis the processes which are blocked due to unavailability of an I/O device constitute this queue.

3.2 RTOS Scheduling Models

Cooperative multitasking

Preemptive multitasking

Rate monotonic scheduling

Round robin scheduling

3.2.1 Cooperative Multitasking

Cooperative multitasking, also known as non-preemptive multitasking, is a style of computer multitasking in which the operating system never initiates a context switch from a running process to another process. Instead, processes voluntarily yield control periodically or when idle or logically blocked in order to enable multiple applications to be run concurrently. This type of multitasking is called "cooperative" because all programs must cooperate for the entire scheduling scheme to work. In this scheme, the process schedulerof an operating system is known as a cooperative scheduler, having its role reduced down to starting the processes and letting them return control back to it voluntarily.

3.2.2 Preemptive Multitasking

The term preemptive multitasking is used to distinguish a multitasking operating system, which permits preemption of tasks, from a cooperative multitasking system wherein processes or tasks must be explicitly programmed to yield when they do not need system resources.In simple terms: Preemptive multitasking involves the use of an interrupt mechanism which suspends the currently executing process and invokes a scheduler to determine which process should execute next. Therefore, all processes will get some amount of CPU time at any given time.In preemptive multitasking, the operating system kernel can also initiate a context switch to satisfy the scheduling policy's priority constraint, thus preempting the active task. In general, preemption means "prior seizure of". When the high priority task at that instance seizes the currently running task, it is known as preemptive scheduling.

3.2.3 Rate Monotonic Scheduling

Rate Monotonic Scheduling is a way to schedule Real-Time threads in such a way that can be guaranteed that none of the threads will ever exceed their deadline.The load of the system may vary, but there is autilisation-based test that, if satisfied, guarantees that the system will always be schedulable. As an example the utilisation limit for a system with one process is 100% (as there is no need for preemption). The utilisation limit for a system with 3 processes is approximately 69%.

3.2.4 Round Robin Scheduling

Round-robin (RR) is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time slices (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an operating system concept.

IV. LITERATURE SURVEY

4.1 Introduction

The paper provides the overview of the previous research on several ways and the methodologies to estimate the WCET estimation.The below paper will estimate the WCET for periodic,aperiodic and sporadic task using EDF algorithm.

4.2 Summary

Mixed-Criticality Scheduling Theory: Scope, Promise, and Limitations SanjoyBaruah Washington University in St. Louis-2018

In this paper Safety-critical Systems are typically subject to stringent correctness requirements; these requirements may be considered from two distinct (though related) perspectives: a priori verification, and runtime robustness. The disadvantage in this paper is Misunderstandings arising from this unfortunate ambiguity in terminology seem to lie at the heart of many of the arguments that are made against the use of MCSh in practice. Although it is probably too late to change terminology, it is important that this ambiguity be highlighted by advocates of MCSh when they seek to convince practitioners to take a closer look.

Robustness Results Concerning EDF Scheduling upon Uniform Multiprocessors SanjoyBaruah, Member, IEEE, Shelby Funk, Student Member, IEEE, and Joe Goossens-2003

In this paper, we study the scheduling of hard-real-time systems upon uniform multiprocessor platforms. In contrast to identical multiprocessors, in which it is assumed that all processors are equally powerful, each processor in a uniform multiprocessor machine is characterized by a speed or computing capacity, with the interpretation that a job executing on a processor with speed s for t time units completes units of execution. We consider the scheduling of hard-real-time systems on uniform multiprocessor platforms which allow for preemptions and interprocessor migrations (i.e., a job executing on a processor may be interrupted at any instant and its execution resumed later on the same or a

different processor with no cost or penalty), but forbid job-level parallelism—at any instant in time, each job may be executing upon at most one processor. The disadvantage in this paper is the total time complexity of the implementation given in Fig. 1 is Oðm3Þ, where m denotes the number of processors in the platform 1 upon which EDF-feasibility is being tested. This follows from the observations that the implementation of algorithm for determining whether a real-time instance, known to be feasible upon a uniform multiprocessor with fastest processor having speed and total computing capacity b, is EDF-feasible upon a uniform multiprocessor platform.

Scheduling Real-Time Mixed-Criticality Jobs SanjoyBaruah, Vincenzo Bonifaci, GianlorenzoD'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie-2012

Many safety-critical embedded systems are subject to certification requirements; some systems may be required to meet multiple sets of certification requirements, from different certification authorities. Certification requirements in such "mixed-criticality" systems give rise to interesting scheduling problems,that cannot be satisfactorily addressed using techniques from conventional scheduling theory. In this paper, we study a formal model for representing such mixed-criticality workloads. We demonstrate first the intractability of determining whether a system specified in this model can be scheduled to meet all its certification requirements, even for systems subject to merely two sets of certification requirements. Then we quantify, via the metric of processor speedup factor, the effectiveness of two techniques, reservation-based scheduling and priority-based scheduling, that are widely used in scheduling such mixed-criticality systems, showing that the latter of the two is superior to the former. We also show that the speedup factors we obtain are tight for these two techniques.The disadvantage in this paper is assigning priorities according to criticality may result in very poor processor utilization. An innovative slack-aware approach is proposed that builds atop priority-based scheduling (with priorities not necessarily assigned according to criticality).

V. EXISTING SYSTEM

5.1 EDF

Earliest deadline first (EDF) or least time to go is a dynamic priority scheduling algorithm used in real-time operating systems to place processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.EDF is an optimal scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent jobs, each characterized by an arrival time, an execution requirement and a deadline, can be scheduled (by any algorithm) in a way that ensures all the jobs complete by their deadline, the EDF will schedule this collection of jobs so they all complete by their deadline.

There is a significant body of research dealing with EDF scheduling in real-time computing; it is possible to calculate worst case response times of processes in EDF, to deal with other types of processes than periodic processes and to use servers to regulate overloads.EARLIEST DEADLINE FIRST

Each task in an EDF scheduler is assigned a _deadline_ (e.g. a moment in the future at which the task must be completed). Every time a task is inserted in the system or completed, the scheduler looks for the task which has the closest deadline and selects it for execution. In order to ensure that the scheduler is still able to meet each deadline, a 'monitor must evaluate if each new task doesn't overload the system and deny execution if it will do so.

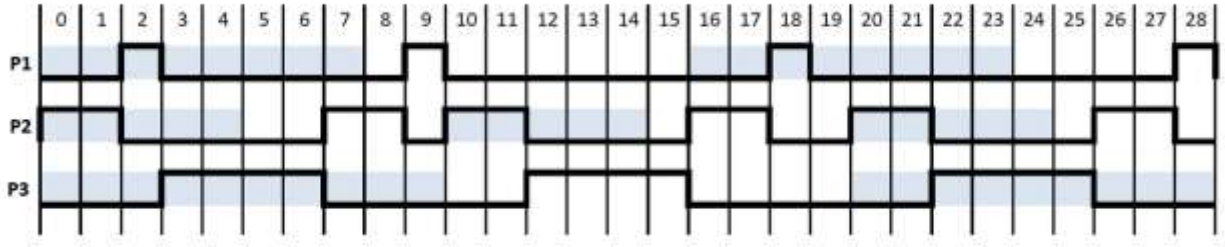


Figure 6. Earliest Deadline First

In order to implement EDF-based system, one will have to know both the _deadline_ of the task (which could optionally be computed as "no more than X ms in the future") and the expected time needed to perform the task (required by the monitor). QoS network routers usually implement variants of EDF scheduling.

Again, there is autilisation based test for EDF systems. The limit is simpler however - it is always 100%, no matter how many processes are in the set. This makes dynamic analysis of schedulability easier. Not only that, but the EDF utilisation test is both sufficient and necessary, so can be relied on to provide an accurate indication of schedulability.

For more information, see "Real time systems and programming languages" by Burns &Wellings.

5.2 *Estimation of* WCET

The worst case response time of $t_i$is equal to the smallest t satisfying the following equality.

$$t = Ci + \sum_{j<i} \left\lceil \frac{t}{Tj} \right\rceil Cj.$$

C_i-worst case execution time

D_i-Relative deadline parameter

T_i-Inter arrival separation time

The worst case workload of the i highest priority tasks over an interval of length t,

$$\sum_{i} \left\lceil \frac{t}{Tj} \right\rceil Ci$$

The worst case idle time of the I highest priority tasks over an interval of length t,

$$Hi(t) = t - W(t)$$

The pseudo inverse function $X_i(c)$ of $H_i(t)$ is the smallest,

$$Xi(c) = \min_{t}\{: Hi(t) \geq c\}$$

The worst case response time R_i of task is given by,

$$Ri = \max_{k=1,2,\ldots} \{Xi - 1(kCi) - (k-1)Ti\} \ldots$$

VI. PROPOSED SYSTEM

6.1 Shortest Remaining Time

Shortest remaining time, also known as shortest remaining time first (SRTF), is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

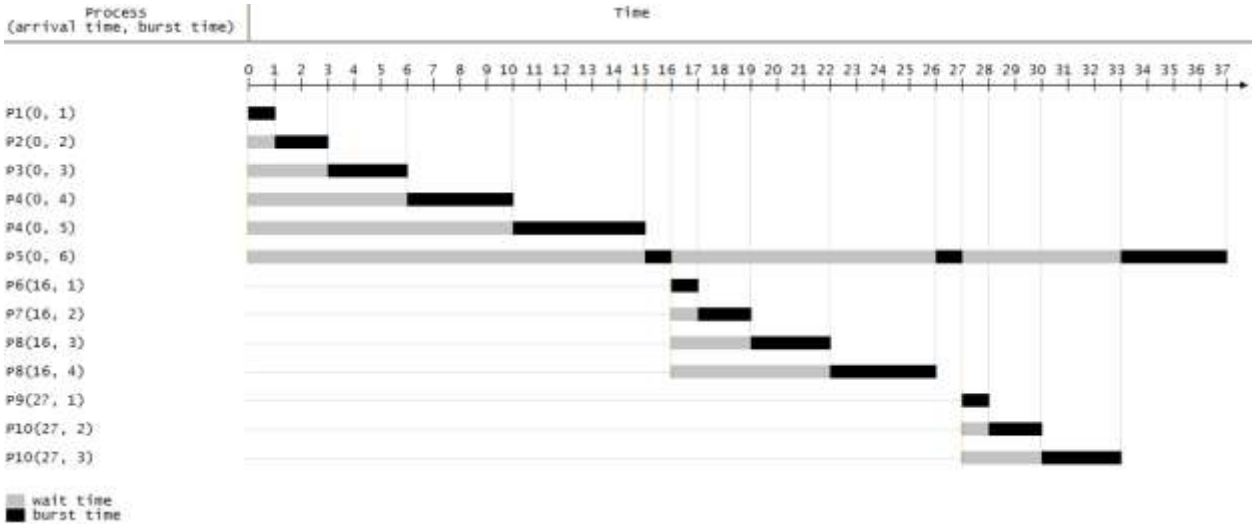


Figure 7. Shortest Remaining Time First

6.2 EDF-VD

In the existing system they have determined the WCET estimation using the EDF algorithm for sporadic task. In the proposed system they have determined the WCET estimation using the EDF-VD algorithm. The EDF-VD scheduling algorithm has implicit deadline. Implicit deadline is defined as a task is said to be constrained if the relative deadline is less or equal to its period. The task becomes an implicit deadline task if the relative deadline is exactly equal to the period. If the task is neither constrained nor implicit, then it is arbitrary. This thesis considers the scheduling of implicit deadline periodic task systems.EDF-VD is the maximum amount of time a job may take to execute after it becomes eligible for execution. This time must be less than the relative deadline of the task or in the worse case be equal to it. It is a run time algorithm which has virtual dead line and the scaling factor. The speedup bound for EDF-VD is a first attempt which has better bound and the pragmatic enhancements to EDF-VD. The speed optimality of the EDF-VD is high.

6.2.1 Task Response Time

The task response time has been estimated for the processor,

K^{th} job in task T_i

$$Ri = \max_{k=1,\ldots k}\{Xi - 1(kCi) - (k-1)Ti\}$$

K*<+∞

$$Rik = Xi - 1(kCi + I) - (k-1)Ti \geq Xi - 1(kCi) - (k-1)Ti$$

It is non decreasing function,

$$Ri \geq \max_{k>k*}\{Rik\} \geq \max_{k>K*}\{Xi - 1(kCi) - (k-1)Ti\}$$

We assume that,

$$\forall x :: flb(x) \leq f(x) \leq fub(x)$$

For any upper bound on the workload, there is a corresponding upper bound on the worst case response time R_i,

$$W_i^{ub}(t) \geq W_i(t)$$

Relationship for idle time,

$$H_i^{lb}(t) = t - W_i^{ub}(t) \leq t - W_i(t) = H_i(t)$$

The relationship between pseudo inverse function,

$$X_i^{ub}(c) = \min_t\{t: H_i^{lb}(t) \geq c\} \geq \min_t\{t: H_i(t) \geq c\} = X_i(c)$$

$$R_i^{ub} = \max_{k=1,2,\ldots..}\{X_{i-1}^{ub}(kCi) - (k-1)Ti\} \geq R_i$$

The maximum amount of time that the processor execute a task,

$$W_i(t) = \sum_{j=1}^{t} w_j(t)$$

$W_j^0(t)$ is the maximum amount of time that the processor executes the task at any time interval,

$$\forall_j \forall t \quad w_j^o(t) \geq w_j(t)$$

The equation of linear bound is given by,

$$w_j^o(t) \leq U_j t + Cj(1 - Uj)$$

The linear upper bound function,

$$W_i(t) = \sum_{j=1}^{i} wj(t) \leq \sum_{j=1}^{i} w_j^o(t)$$

$$\leq \sum_{j=1}^{i}\left(U_j t + C_j\left(1 - U_j\right)\right) = W_i^{ub}(t)$$

The worst case response time for upper bound,

$$R_i \leq \frac{C_{i+\sum_{j<i} Cj(1-U_j)}}{1 - \sum_{j<i} Uj} = R_i^{ub}$$

$$W_i^{ub}(t) = \sum_{j=1}^{i}(U_j t + Cj(1 - Uj))$$

$$H_i^{lb}(t) = t(1 - \sum_{j=1}^{t} U_j) - \sum_{j=1}^{t}(C_j\left(1 - U_j\right))$$

It is invertible,

$$X_i^{ub}(h) = \frac{h + \sum_{j=1}^{i} C_j(1 - U_j)}{1 - \sum_{j=1}^{i} U_j}$$

6.3 Predictability

Predictability in real-time systems has been defined in many ways. For static real-time systems we can predict the overall system performance over large time frames (even over the life of the system) as well as predict the performance of individual tasks. If the prediction is that 100% of all tasks over the entire life of the system will meet their deadlines, then the system is predictable without resorting to any stochastic evaluation. In dynamic real-time systems we must resort to a stochastic evaluation for part of the performance evaluation. Predictability for these systems should mean that we are able to satisfy the timing requirements of critical tasks with 100% guarantee over the life of the system, be able to assess overall system performance over various time frames (a stochastic evaluation), and be able to assess individual task and task group performance at different times and as a function of the current system state. If all these assessments meet the timing requirements, then the system is predictable with respect to its timing requirements.

6.3.1 Types of Predictability

Functionalpredictability

Timingpredictability

6.3.2 Functional Predictability

The arefacts of computing are designed for functional predictability are

Example

Input (ftoatx, ftoaty, time duration t)

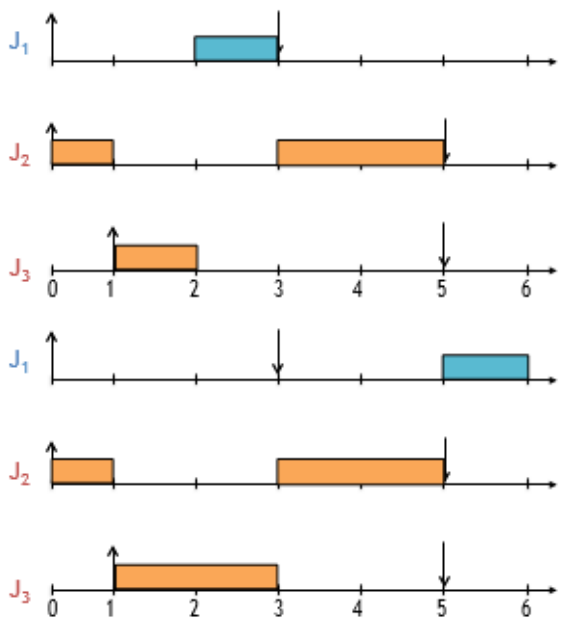
Compute x*y within t time units

The functional correctness is the constraint and the timing behavior the optimization object and the formalisms of computing abstract away the concept of physical time.

6.3.3 *Timing Predictability*

All run time properties are not equally important behaviour emerges from three interacting models validation of properties is done under assumptions that depend upon the semantics of the property. The timing predictability is a deterministic programs executing on non-deterministic platforms, interacting with non deterministic platform.Threereal-time“jobs”onasharedpreemptiveprocessor

Earliest Deadline First(EDF) scheduling is



VII. RESULTS AND DISCUSSION

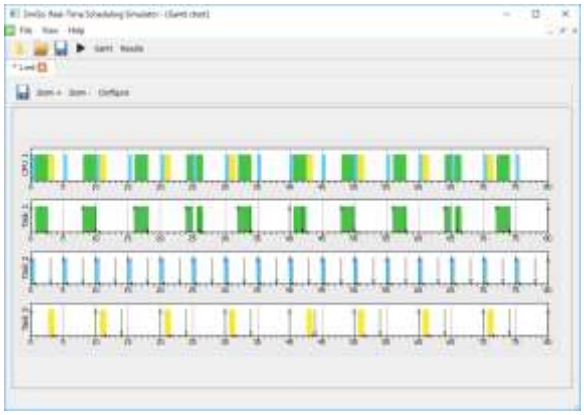


Figure 7.1.Gantt Chart for Uniprocessor Scheduler under EDF

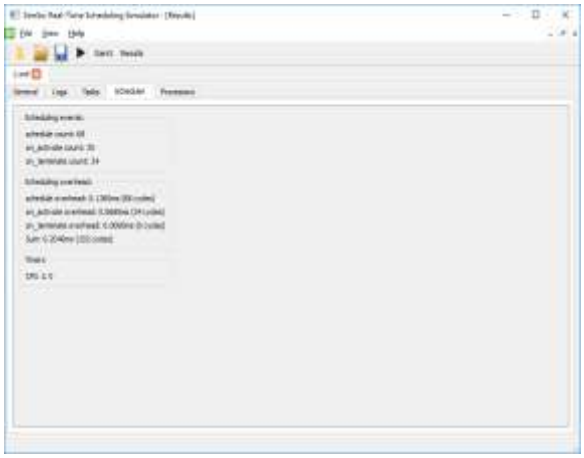


Figure 7.2. Timing Overhead for Uniprocessor Scheduling under EDF

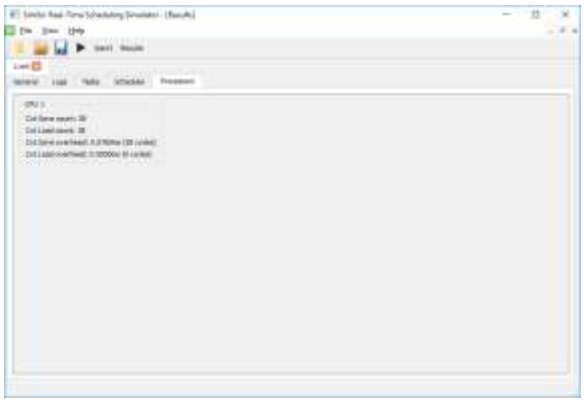


Figure 7.3. CPU Cycles (Save and Load Count)

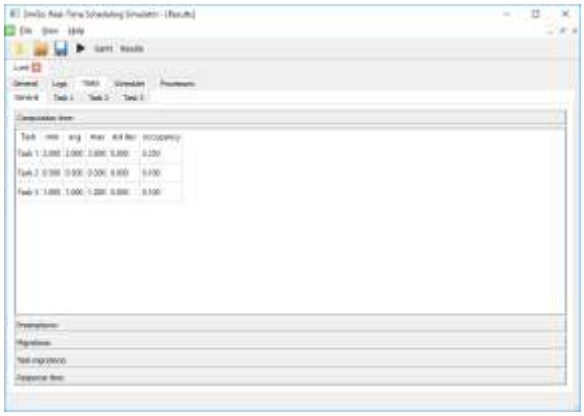


Figure 7.4. Task distribution under EDF Scheduler

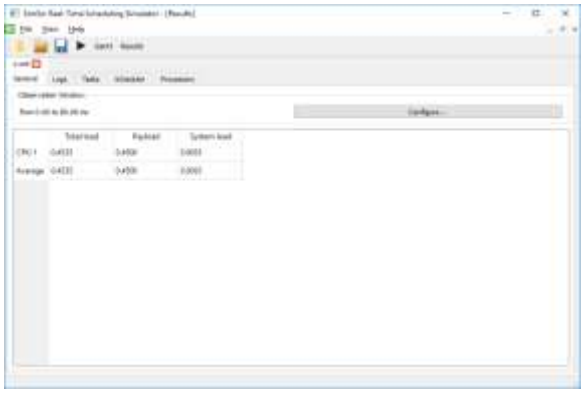


Figure 7.5. Load by CPU under Simulation

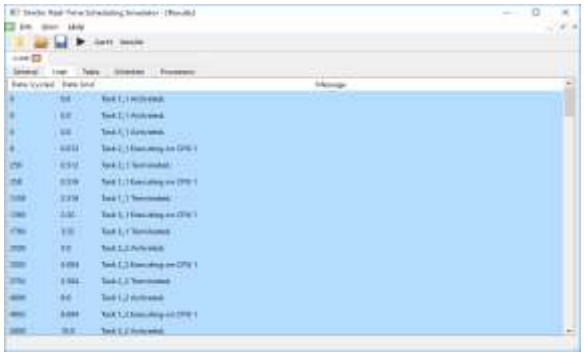


Figure 7.6. Log file of the scheduler under EDF algorithm

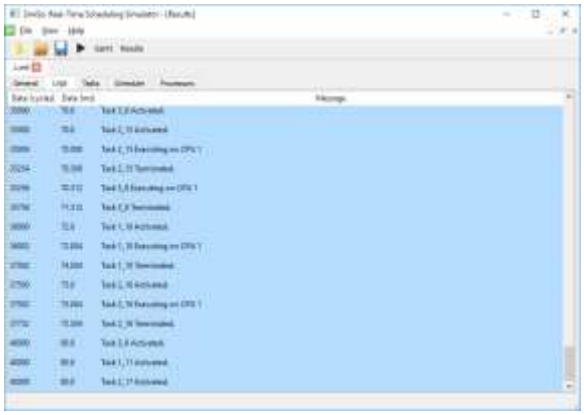


Figure 7.7. Log file of the scheduler under EDF algorithm

VIII. CONCLUSIONS AND FUTURE WORK

The project contributions can be summarized as follows: the test for a WCET estimation with EDF scheduling algorithm that adopts a global approach to task allocationupon uniprocessors. That is, the behaviorof Algorithm EDF—one such previously defined [10], [17] static-priority global scheduling algorithm upon uniprocessor platforms. The simple sufficient conditions fordetermining whether any given periodic task system will besuccesfully scheduled by Algorithm EDF-VD upon a given uniprocessor platform. The condition of the response time have been calculated for further scheduling of tasks under EDF-VD algorithm. The outputs were visualized using the Gantt charts. The deadlines and miss were clearly visible in the output.

The efficient condition of Earliest Deadline First along with Virtual Deadline by estimating the shortest remaining time algorithm, the system model can be designed and outputs may express sufficient schedulable task over uniprocessor platforms. The uniprocessor algorithm may be enhanced further to create a platform on multiprocessor environments.

REFERENCES

[1]. J. N. Buxton and B. Randell, Eds., Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27–31 Oct. 1969, Brussels, Scientific Affairs Division, NATO (1970).

[2]. S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in Proc. Real-Time Systems Symp., Tucson, AZ, USA, Dec. 2007, pp. 239–243.

[3]. Burns and R. Davis, “Mixed-criticality systems: A review (7th edition),” 2016; Available: http:// www-users.cs.york.ac.uk/~burns/review.pdf 37 March/April 2018

[4]. J. Fenn and M. Raskino, Mastering the Hype Cycle: How to Choose the Right Innovation at the Right Time, Harvard Business School Press, 2008.

[5]. R.K.Sharmaetal. “Balance of Power: DynamicThermalManagement ofInternet Data Centers”. Jan.2005

[6]. Esper, G. Nelissen, V. Nélis, and E. Tovar, “How realistic is the mixed-criticality real-time system model?” in Proc. 23rd Int. Conf. Real Time Networks Sys. (RTNS ’15), New York, NY, USA, 2015, pp. 139–148.

[7]. R. Ernst and M. Di Natale, “Mixed criticality systems – A history of misconceptions?” IEEE Des. Test, vol. 33, no. 5, pp. 65–74, 2016.

[8]. S. Baruah, “Schedulability analysis of mixed-criticality systems with multiple frequency specifications,” in Proc. 16th Int. Conf. Embed. Software (EMSOFT), Oct. 2016, Pittsburgh, PA, USA.

[9]. S. Baruah and B. Chattopadhyay, “Response-time analysis of mixed criticality systems with pessimistic frequency specification,” in Proc. IEEE Int. Conf. Embed. Real-Time Comput. Syst. Appl. (RTCSA), Taipei, Taiwan, 2013.

[10]. Burns and R. Davis, “Mixed criticality on controller area network,” in Proc. 2013 25th Euromicro Int. Conf. Real-Time Systems (ECRTS ’13), Paris, France,Jul. 2013, pp. 125–134.

[11]. S. Baruah and A. Burns, “Implementing mixedcriticality systems in Ada,” in Reliable SoftwareTechnology – Ada Europe 2011, A. Romanovsky andT. Vardanega, Eds., Edinburgh, UK: Springer, 2011,pp. 174–188.

[12]. S. Baruah, “Schedulability analysis for a general modelof mixed criticality recurrent real-time tasks,” in Proc.2016 IEEE Real-Time Syst. Symp., Dec. 2016.

[13]. L. Sha, R. Rajkumar, J. Lehoczky, andK. Ramamritham, “Mode change protocols for priority-driven preemptive scheduling,” J. Real-Time Syst.,vol. 1, no. 3, pp. 243–264, 1988.

[14]. S. Davari and S.K. Dhall, “On a Real-Time Task Allocation Problem,” Proc.19th Hawaii Int’l Conf. System Science, Jan. 1985.

[15]. S. Davari and S.K. Dhall, “An On-Line Algorithm for Real-Time TasksAllocation,” Proc. Real-Time Systems Symp., pp. 194-200, 1986.

[16]. M. Dertouzos, “Control Robotics: The Procedural Control of PhysicalProcessors,” Proc. IFIP Congress, pp. 807-813, 1974.

[17]. S.K. Dhall and C.L. Liu, “On a Real-Time Scheduling Problem,” OperationsResearch, vol. 26, pp. 127-140, 1978.

[18]. S. Funk, J. Goossens, and S. Baruah, “On-Line Scheduling on Uniform Multiprocessors,” Proc. IEEE Real-Time Systems Symp., pp. 183-192, Dec.2001.

[19]. J. Leung and J. Whitehead, “On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks,” Performance Evaluation, vol. 2,pp. 237-250, 1982.

[20]. C. Liu and J. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” J. ACM, vol. 20, no. 1, pp. 46-61, 1973.

[21]. D.I. Oh and T.P. Baker, “Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment,” Real-Time Systems: The Int’lJ. Time-Critical Computing, vol. 15, pp. 183-192, 1998.