AN IMPLEMENTATION OF MULTITASK SCHEDULING ALGORITHM FOR VXWORKS

A dissertation submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY
In
VLSI DESIGN AND EMBEDDED SYSTEM
By

CHILAKALA VENKATA KRISHNA REDDY
ROLL NO: 213EC2199

Department of Electronics and Communication Engineering
National Institute of Technology
Rourkela, Orissa, India-769008.
May 2015

AN IMPLEMENTATION OF MULTITASK SCHEDULING ALGORITHM FOR VXWORKS

A dissertation submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY
In
VLSI DESIGN AND EMBEDDED SYSTEM
By

CHILAKALA VENKATA KRISHNA REDDY
ROLL NO: 213EC2199

Under the Guidance of

PROF. KAMALA KANTA MAHAPATRA

Department of Electronics and Communication Engineering
National Institute of Technology
Rourkela, Orissa, India-769008.
May 2015

Dedicated to…

My Friends

My Dear Parents and Grand Parents

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA
ROURKELA, ODISHA, INDIA- 769008

CERTIFICATE

This is to certify that the dissertation report entitled

AN IMPLEMENTATION OF MULTITASK SCHEDULING ALGORITHM FOR VXWORKS

Submitted by

Mr. Ch Venkata Krishna Reddy

bearing roll no. **213EC2199**
in partial fulfillment of the requirements for the award of

MASTER OF TECHNOLOGY
in
VLSI DESIGN AND EMBEDDED SYSTEM

During session 2012-14 at National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/institute for the award of any degree or diploma.

Prof. Kamala Kanta Mahapatra
Department of ECE,
National Institute of Technology,
Rourkela.

Date: ----------

Table of Contents

ACKNOWLEDGEMENT

The work postured in this proposal is by a wide margin the most significant achievement in my life and it would be unbelievable without individuals who asserted me and had faith in me. Above all else I manifest my significant respect and profound respects to my Guide Prof. (Dr.) **K. K. Mahapatra** for model direction, for promoting learning the topic absolutely new for me, directing and steady support throughout this theory. A man of his word typified, in genuine shape and soul, I consider it my great success to have consisted with him. I express my true appreciation to Prof. Ayas Kanta Swain, Prof.(Dr.) P.K.Tiwari, Prof.(Dr.) D.P Acharya, Prof.(Dr.) Nurul Islam, who had presented the universe of VLSI and Embedded System and helped me in snatching learning in different spaces of my specialization. I might likewise want to say thanks to all different resources and staff of ECE Department, NIT Rourkela for their help and backing to finish my project work.

My special thanks to Ph.D. scholar Mr. Sudeendra Kumar, of Department of Electronics and Communication Engineering for their consistent motivation and support amid my research and all other research scholars of ECE Department, NIT Rourkela, Who has been always ready to share their knowledge throughout our course.

Being an M.Tech scholar of NIT Rourkela, I had a privilege of getting and working together with my classmates. I am truly thankful to my close friends Anil Kumar Sarika, Rakesh Chanamala, Hanumanth Rao Gorrepati, Siddharth Pottepalem, Sailaja Saragadam, Sri Kanya Dasari, my roommates and my classmates.

Ultimately, I thank my family whose constant support and encouragement, always help me move forward in life even during hard times.

CH Venkata Krishna Reddy

ABSTRACT

The application of real time multitasking on a single processor increases day by day and its implementation complexity also increases. The real time systems have raised many scheduling issues, Such as Hierarchical scheduling and resource sharing. While hypothesis turns out to be more developed, but the implementation of real time multitasking systems still to be a challenge for designers.

In embedded real-time operating systems (RTOS), As the number of tasks increases the complexity of Scheduling also increases, it will lead missing of task deadline. To overcome this type of problems we have designed and simulated Round-robin scheduling (RR), Rate Monotonic scheduling (RMS), and Earliest Deadline First scheduling (EDF) algorithms by using single Portable Operating System Interface (POSIX) timer in Wind River Vxworks 6.9.The communication among the tasks could be established by using semaphores. The purpose of the implementation of these scheduling algorithms is: We would like to verify the various task scheduling schemes and develop a novel task scheduling scheme if required for proposed Vxworks based DAC system. During implementation of Multitask Scheduler, we have also implemented some basic scheduling algorithms. We present details of the implementation of Round-robin scheduling (RR), Rate Monotonic scheduling (RMS), and Earliest Deadline First scheduling (EDF) algorithms.

List of Figures

List of Tables

LIST OF ACRONYMS

The following is the list of Acronyms that are encountered in this thesis.

CPU Central Processing Unit

DAC Data Acquisition and Control

DKM Downloadable Kernel Module

EDF Earliest Deadline First

FPS Fixed priority Scheduling Policy

FTP File Transfer Protocol

HSF Hierarchical Scheduling Framework

ID IDentifier

IIP Immediate Inheritance Protocol

IOT Internet Of Things

ISR Interrupt Service Routine

PCP Priority Ceiling Protocol

PIP Priority Inheritance Protocol

POSIX Portable Operating System Interface

RMS Rate Monotonic Scheduler

RR Round Robin

RTOS Real-Time Operating Systems

RTP Real-Time Process

SBC Single Board Computer

SIRP Subsystem Integration Resource Allocation

SMP Symmetric MultiProcessing

SPDA Second Power Distribution Accouterment

SRP Stack Resource Policy

TCB Task Control Block

TEQ Time Event Queue

USR User Service Routine

WDB Workforce Development Board

1. Introduction

In today's point of view real time operating systems (RTOS) are facing various difficulties in scheduling. Hard real time applications are strictly bounded by time. The mathematical analysis of these real time applications simpler, but the implementations of these applications on real time systems is difficult. The timing properties of these applications are determined by available system resources by corresponding application. Multitasking is a process of executing a large number of tasks on a single processor on a time basis. One of the difficulties is getting of extra functional correctness, i.e. such as hard real time timing constraints. The integration of temporal isolation of components involves more difficulty. Most recent hard real time systems are interdependent on inter task communication and multitasking. To address the above mentioned challenges, we have chosen the Power PC based Single Board Computer (SBC) MPC834X as hardware and Vxworks as Real Time Operating System (RTOS).

1.1 Motivation:

Hard Realtime schedulers on a single processor got many applications in embedded systems, Like Data Acquisition and Control (DAC) systems, IoT networks, In Auto Mobile Industry etc. This has introduced many system resources to the real time schedulers which increases the complexity of scheduling and decreases the processor Utilization.

It is a good area to research and develop an innovative scheduling algorithm. Yet implementation of Earliest deadline first Scheduler of real time embedded applications are challengeable task. Currently, many researchers are working on development of feasible schedulers. Hence the current research area is chosen.

1.2 Objective:

Hierarchical multitasking scheduling has many advantages when it comes to integrating real-time applications on a single CPU, The major challenge design Hierarchical multitasking Schedulers are.

- Developing of Hard Real- time schedulers without changing of its kernel in a Vxworks operating system.
- Providing effective utilization of all CPU resources and complete isolation among subsystems.
- By increasing the CPU utilization, to maximize the Throughput and to decrease the USR execution time.
- Design a for Hard Real- time task Scheduler for getting low response time.

1.3 Literature review:

- Ren Peng and Xiang Zheng were designed a second power distribution accouterment (SPDA) by using a hybrid algorithm in Tornado. To schedule periodic tasks Ren Peng used Round robin scheduling algorithms; to schedule aperiodic tasks Ren Peng used static priority based preemptive scheduling [1].
- Moris Behnam, Thomas Nolte and Insik Shin were developed hierarchical scheduling framework with the help of Rate Monotonic scheduling and also implemented Rate Monotonic and Earliest Deadline first scheduling algorithms [2].

- Raffia Inam, Jukka Maki-Turja and Mikael Sjodin have implemented a hierarchical scheduling framework in the FreeRTOS (an open source environment). In this paper they used static priority preemptive scheduling algorithm in local as well as in global level [5].
- Mikael Asberg and Thomas Nolte were proposed an ExSched; it can support to develop several schedulers on different operating systems without any modifications of its kernel space [3].
- Junking Li and Yuting Wang have developed a hybrid genetic algorithm to schedule in the internet of things. They have processed different tasks on different devices based on the Radio frequency identification [6].
- Reinder J. Bril and Mikael A sberg have proposed a solution for resource sharing in hierarchical scheduling framework systems, I.e. skipping and overrun mechanism [4].

1.4 Outline of Thesis:

This thesis gives overview of basic Vxworks functions and implementation of basic scheduling algorithms by using Board support package libraries and Implementation of advanced scheduling techniques in Vxworks and its simulations. The management of thesis as follows.

- **Chapter 1** narrates the importance of task scheduling, motivation of this thesis and various goals achieved during this project time.
- **Chapter 2** explains about Real time tasks, Real time tasks classifications and important basic and advanced scheduling algorithms.
- **Chapter 3** discusses most successful Real time embedded operating system Vxworks, usage of Wind River Workbench 3.3 and connection of target hardware.

- **Chapter 4** gives an overview of building periodic tasks with multiple system timers and with only one timer. Design and simulation of user service routine and scheduling algorithms with one shot timer.
- **Chapter 5** explains hierarchical scheduling framework and its implementation and simulation without any resource sharing among subsystems.
- **Chapter 6** gives an overview of conclusion and scope of future work of the dissertation.

2. Overview of task scheduling

2.1 Introduction to Task Scheduling

Scheduling is a process of CPU or systems parameters processing threads and data in a specific sequence. Task scheduling is nothing but assigning of specific time to each task to complete its operation. Real time task scheduling must be a useful system in supporting seclusion of ongoing assignment scheduling programming by giving temporal dividing among applications. Real time task schedulers or processors would be classified depending on time criticality[18].

- **Soft real time tasks:**

  In these types of task scheduling, the time constraints are expressed in their average time requirements. The time criticality is very less. The time bound generally from few ms seek to couple of Suez [18].

- **Firm real time tasks:**

  In these types of task scheduling, each and every task associated with some predefined task deadline time before get into its execution state. If these tasks would meet its deadline time system does not fail. After its deadline time its results will become zero [18].

- **Hard real time task:**

  In these types of task scheduling, each task is obliged to create its outcomes sure predefined time limits. If these tasks would not meet its deadline time system certainly fails. For practical, hard real time systems, the time bound generally range from few microseconds to a couple of milliseconds [18].

2.2 Important Terminologies

Task Instance: every time a task is initiated when some particular event comes. Generally real_time tasks a repeat maximum number of times at distinctive moments of time relying upon the event times. It is conceivable that real-time tasks repeat indiscriminately times. Although Hard realtime task repeat with a constant amount period. So each occurrence of the task is known as task instance [18].

Arrival Time: It is the time at which task came into its execution state. I.e. the task will take staring of its execution time.

Response Time: it is defined as the time from the task entry time to deliver its outcomes. As previously discussed, task instances get access because of happen of events. These processes could be belongs to a system, for example process interrupts, clock interrupts or outside to a system, for example peripheral interrupts [1].

Absolute Deadline: Absolute deadline is the period in which task gets executed. The absolute deadline of a task is a duration measured from its arrival instance to the next arrival instance. I.e. The absolute deadline is equivalent to the scope of time between the zeroth time of task and expected completion of task [18].

Relative Deadline: This deadline time calculated from the beginning of the undertaking to the moment at which deadline happens. In the different manner, relative deadline is the time interim between the beginning execution of an assignment and its relating deadline [2].

Preemptive Scheduler: in this preemptive scheduler, suppose a high priority task wants to process, before it is being processed first it suspends it's below priority task and then it begins its

execution. I.e. a priority preemptive scheduler, always higher priority task completes its first execution; it cannot wait for any type of resources. If a preempted lower priority job can restart its process only when no other above priority job is alive [1].

Utilization: it can be defined as the meantime, for which it executes every unit time interim.

$$U = \sum_{i=1}^{n} (e_i / p_i)$$

Here p_i, e_i are the corresponding T_i s absolute deadline or period, the time required to execute respectively. Here n is represented by number of task number.

The processor utilization $U \leq 1$

If any scheduler would be the best one, it should be feasible schedule its every task with a very large process utilization factor [18].

Jitter: It is the divergence of a periodic task from its actual response. It is measured from actual arrival to expected arrival time. It may be occurring due to imprecise clocks, or network congestions. Jitters are unavoidable for some applications [18].

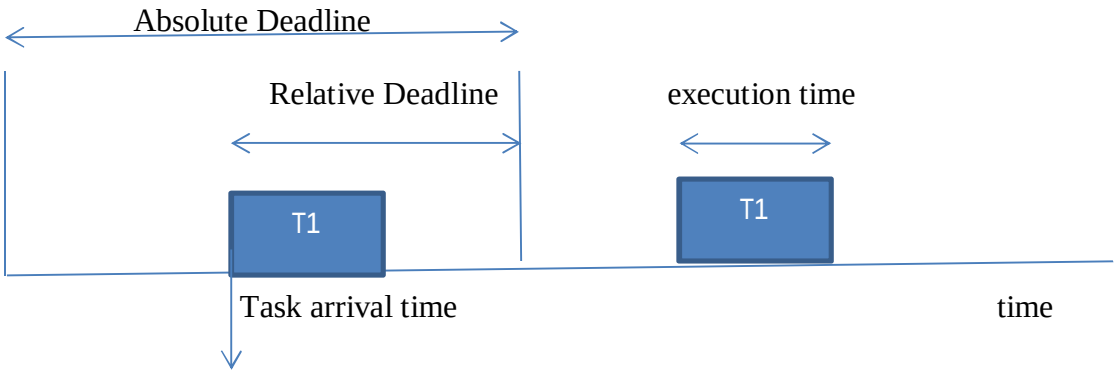

Figure 2-1 Task deadlines

2.3 Classifications of Real time tasks

Based upon how they are repeated over a period of time, real time task are categories in three different ways.

Periodic tasks:

A task that repeats for every fixed amount of time is known as periodic tasks. Usually these interrupts generated by clock time interrupts. So, normally we can say clock-driven tasks are periodic tasks. Each and every task repeats after a particular time, these times we can represent period or absolute deadline of corresponding task. Let a periodic task $T_{i,}$ then the phase of task is defined as, the time from zeroth point to the Ti first instance occurrence is noted as φ_i. The second instance (i.e. $T_i[2]$) comes at a time of $\varphi_i + p_i$. And next instance (i.e. $T_i[3]$) comes at a time of $\varphi_i + 2 * p_i$ and etc. Generally, $(\varphi_i, p_i, e_i, d_i)$, is a four tuple representation of periodic job T_i. In this representation phase shift, period or absolute deadline, worst case execution time, and relative deadline of a job task are defined above sequentially [1].

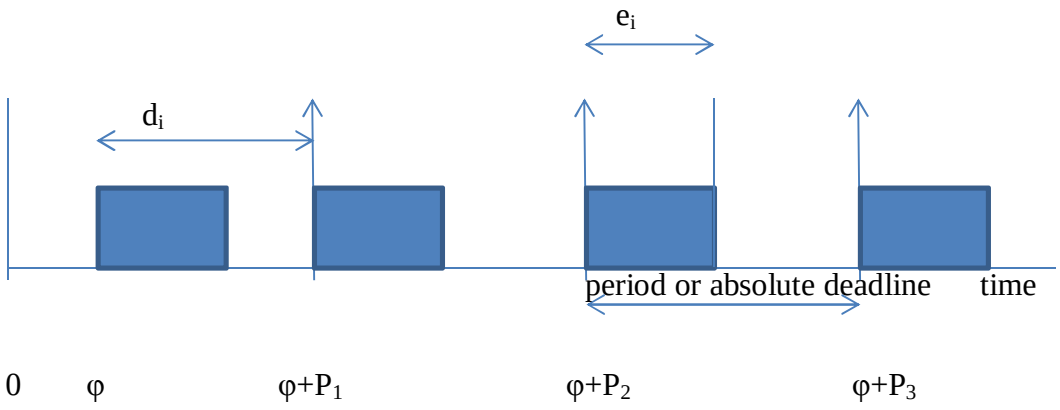

Figure 2-2 periodic tasks

Sporadic Task:

In these tasks that repeats at arbitrary moments. Generally the below representation is three tuple representation [18].

$$T_i = (e_i, g_i, d_i)$$

Where tasks, poor case execution time is e_i, least detachment between two back to back task moments is g_i, tasks relative deadline is d_i. In a Sporadic task, there should be at least detachment between two back to back task moments. That is, g_i limits the rate at which sporadic tasks can emerge.

For example, in a car, handling of braking conditions. The complexity prediction of sporadic event occurrence is very high. The time of occurrence of these tasks cannot be predicted. Most sporadic task occurrences are highly critical in nature. The critical nature of sporadic tasks fluctuates from moderately critical to excessively critical.

Aperiodic Task:

An aperiodic task is one that repeats at random moments. An aperiodic, a sporadic task is almost same, but the last detachment between two back to back task moments could be zero. I.e., at any particular moment of time one or more jobs may interrupt the execution. Generally aperiodic tasks are soft real-time tasks [18].

Aperiodic tasks would repeat in rapid progression. So, it is very hard to reach deadlines of all aperiodic tasks at all instances. Suppose, a small group of aperiodic task repeats in a swift time, there should be a group of the task moments and most of tasks cannot meet its deadline. So soft real-time events can accept a couple of deadline misses. So, the examples are mouse movements and clicks, keyboard presses, etc.

2.4 Characterization of Hard Real-Time schedulers

Based on type of scheduling parameters, these real time scheduling tasks are characterized in many ways. Among these some of schedulers are explained below [2].

Clock Driven Scheduler

On the clock driven scheduler, the scheduling tasks are calculated by interrupts got from a clock (Timer interrupts). These types of schedulers are also known static type schedulers, because these schedulers fix their scheduling algorithm before coming to its execution [18]. Clock Driven scheduling algorithms suffers a bit of runtime overhead. These are

1. Table Driven Scheduler

Table-driven schedulers generally pre-calculated set of tasks. These tasks could collect task sequence from the table, execute it in the same manner. For soft real time application these schedules are regularly used.

2. Cyclic Schedulers

In the industrial applications cyclic schedulers are mostly used, and a greater number of embedded OS applications are used by cyclic schedulers. A cyclic scheduler is also a pre calculate type scheduler. In this kind of scheduler every task will be available in a major cycle. The integration of a few couples of minor cycles (also called as frames) forms a Major cycle. These frames or minor cycle time duration is defined by timer or clock interrupts.

Event Driven Scheduler

In the event-driven schedulers, assign priorities to each job or task. These jobs or tasks are scheduled based on its priorities. I.e. which job executes next is decided by task priorities. Event driven schedulers can easily deal with aperiodic and sporadic tasks. These classes are generally primitive based schedulers [3].

Fixed Priority Preemptive Scheduler:

In fixed priority preemptive scheduling, the scheduler always keeps highest priority task in the ready queue. Scheduler every time search for the highest priority task, if it founds highest priority task, it suspends current task execution and the spawns highest priority task. The preemptive scheduler owns a clock interrupt task that will execute after expiration of task duration. The advantages of these types of schedulers very less number of context switches exist and the implementations of these types of schedulers a bit simpler. The disadvantages are, if we increase processor utilization, it would lead deadline misses and since top priority tasks always executes, the bottom-priority tasks could hold up an inconclusive measure of time [1].

Rate Monotonic Scheduler:

The Rate Monotonic Scheduler (RMS) is a static priority based preemptive scheduling algorithm, and popularly famous for real time embedded applications [2]. RMA assigns the priorities to the tasks based on the rate of occurrence. I.e. it will give highest priority to the task which occurs regularly. RMS proved the optimized scheduler. According to the Liu and Layland the necessary condition to feasibly schedule the RMS is

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

Where U is the CPU utilization, C_i , T_i ,n are time of computation, absolute deadline or tasks period and count of tasks respectively.

For example: utilization should be $(U) \leq 0.77976$ for three processes.

The schedulability of RM scheduling model analysis of each task is defined as

W(i) = Ci +

The above equation shows the worst case execution time of each task, here Ci is the execution time of its task, HP (I) is the higher priority task set. Tk is absolute activation time.

The advantages of RMS are, it has optimal static priority based algorithm, and the design complexity of the user service routine is lesser, etc. And the disadvantage are, handling of critical tasks when long periods and etc [2].

Earliest deadline first Scheduler:

We can say it is a dynamic or active Priority based preemptive Scheduling Algorithm. In this algorithm least the relative deadline time of a job having maximum priority. It can assigns highest priority to the lowest deadline time. This could be implemented by changing the ready queue dynamically with respect to its deadline time [2].

The feasibility of the EDF scheduling algorithm is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

Where U is the CPU utilization time, C_i is the worst-case processing-times of the n jobs and its absolute deadline is Ti.

The EDF schedulability of tasks is shown below, it considers task deadlines are less than or equal to task periods.

W(i)=

Here T_i is a tasks period, D_i is deadline of each task, C_i is the worst case execution time of each task, t is time instance.

The advantages of EDF are, it has optimal dynamic priority based algorithm, and having the less context switching times etc. And the disadvantage is the complex implementation.

Hybrid Scheduler

Hybrid type scheduler use event interrupts as well as clock interrupts to regulate their scheduling tasks. Clock interrupts are set time slice for hybrid scheduler and event interrupts arises every task or job completion [1].

Round Robin Scheduler:

The Round Robin Schedulers are very commonly used in real time system applications. The Round Robin Schedulers algorithm is preemptive scheduling base method. In Round Robin Scheduling, the ready tasks are held in circular Queue. The tasks are taken sequentially from the Queue. Once a task is taken up, it executes for a fixed interval of time called its Time Slice. If the executed task doesn't complete its work within its time slice it could be inserted back into the ready queue. In RR scheduling all tasks are treated as equals, in the sense all tasks having equal priority and equal time slice. It is possible to consider the task priorities in the time slice round robin scheduler, though a small extension to the basic round robin scheduler. The advantages of round robin Scheduler have advantages like, better mean response time, delay time, which is lying on the number of job tasks. The disadvantages are, it has more number of switching times and more response time [1].

3. Introduction to Vxworks and WindRiveR Workbench

3.1 Introduction to VxWorks

VxWorks is a business real-time operating framework grew by Wind River with an attendant on execution, adaptability and foot shaped impression. VxWorks with support for mutual exclusion, multi-tasking, easy to use inter-task communication. The wind River provides Workbench to simulate Vxworks functions. Wind River Workbench supports different types of projects that will execute in different areas, such as; Vxworks Image project, Boot Loader project, ROMFS File System projects, Real-time Process projects, Downloadable Kernel, etc. Module Projects Many intriguing highlights are given Vxworks, which make it broadly utilized as a part of industry, for example, Wind micro Kernel, systematic task management, prediction of context switching, systematic interrupt, multitasking and exemption taking care of, POSIX Pipes, binary semaphore, mutex semaphore, counting semaphores, message queues, signals, preemptive and round-robin scheduling, etc [8].

The Vxworks micro-kernel assists 256 diverse priority levels and a countless tasks in the static priority scheduling approach. It also assists the time slice scheduling approach. VxWorks supplies two unique modes for application assignments to execute; either user method or kernel method. In kernel method, application-jobs can get to the hardware assets straightforwardly. In user method, then again, undertakings can't straightforwardly get to hardware assets, which give more security (e.g., in user method, tasks cannot strike the kernel). Kernel method is given in all adaptations of Vxworks while a user method was given as a piece of the Real time process (RTP) prototype, and it will be available from Vxworks version 6.0 and beyond [8].

Wind River Workbench supports several different project types, but each one is used for a specific purpose.

VxWorks Kernel Image:

In the event that you are creating a Vxworks image to boot your target, utilize a Vxworks image project. By adding sub image projects or DKM projects, such as a Vxworks ROMFS file system project and kernel modules, applications, etc [9].

VxWorks ROMFS file system project: You can utilize this as a subproject of other projects; this entails targeting-side record system framework [9].

Real-Time Processes: In the event that you are creating an executable that keeps running in user method or exterior to the kernel area. You can independently construct, run, and debug the executable. At run time, the executable document is downloaded to a different methodology, location, space to keep running as an autonomous procedure [9].

Downloadable Kernel Module: You can independently construct the modules, run, and investigate them on a target running VxWorks, stacking, emptying, and reloading those prototypes if you need [9].

3.2 Task Execution in Vxworks

In VxWorks, the small portion of execution is the task, corresponding to a Unix/Linux process, Task has its own context registers, such as stack, memory space, and priority. VxWorks having task ID, which same as the process ID in reaming operating systems. A task can be created by using task spawn or tasks create. A task's precedence could be arranged when the task is done, and amid the execution priority of a task can be changed. At that point, amid runtime, the highest need prepared task will always execute. If a job of top priority than the current priority is would like to execute, first it suspends executing a job and the takes top priority job to

process. When a current job ends its execution and next to this task will go in to ready state. At the time of task creation, its corresponding Task Control Block (TCB) was generated to retain context switching of task. At that point, amid the life span of a task, the task could be in one or a blend of the accompanying states.

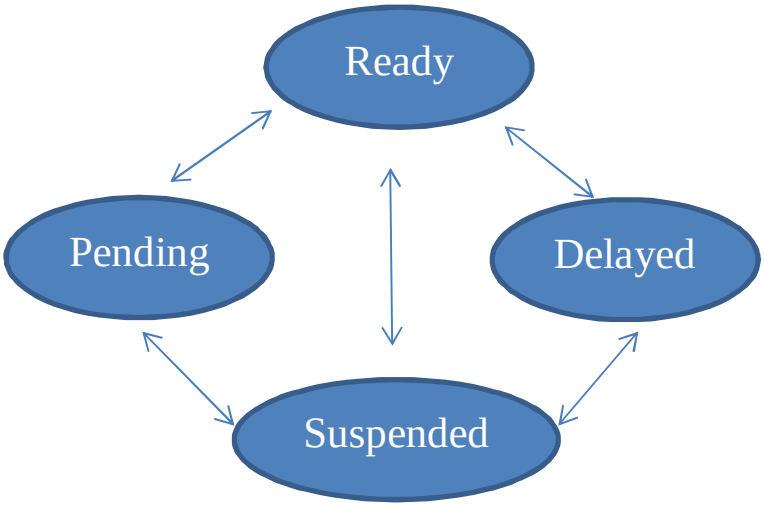

Figure 3-1 Vxworks Task State Diagram [8].

• **Ready state**, Task is sitting tight for CPU assets.

• **Suspended state**, Task is distracted for execution, yet not postponed or pending.

• **Pending state**, Task is blocked sitting tight for some assets apart from the CPU assets.

• **Delayed state**, Task is resting for quite a while.

Note: in a kennel area, scheduler arranges all available tasks in a particular sequence to execute [8].

3.3 Inter task and Interprocess Communications

VxWorks inter task and interprocess is used for synchronization of the activity of different tasks as for communication between them. Their accesses should be synchronized using a facility designed to provide mutually exclusive to get a shared resource, using binary semaphores [8].

VxWorks provides the following alternatives for inter task and interprocess communication:

- **Semaphores**

  Basically, provided for task synchronization and mutually exclusion, semaphores allow interprocess communication.

  - Binary semaphore

    The most general and fastest semaphore.

  - Mutual exclusion semaphore

    A regular binary semaphore streamlined for problems intrinsic in erasure safety, mutual exclusion, priority inversion and recursion.

  - Counting semaphore

    Like a binary semaphore, whoever keeps track of the quantity of times a semaphore is given. Streamlined for various resources.

  - read/write semaphore

    A peculiar kind of semaphore used for mutual exclusion of tasks that need compose to get an item, and simultaneous access for tasks that just need read access to the article. This sort of semaphore is especially convenient for SMP systems [8].

- **Message queues**

  Provide a high level direct communication of messages among the tasks and interlocking of tasks. These message queues allow full duplex communication between tasks.

- **Pipes**

  Pipes Provides an alternative for the messaging queues. Pipes operates through the input, output system, which allows for use of standard input output routines. These are virtual input, output devices organized by pipedrv driver. By using pipes we can also implement inter task communication [7].

- **VxWorks events**

  Provide a method of correspondence and synchronization between tasks; interrupt service routines (ISR's) and tasks, semaphores and tasks, and message queues and tasks. Tasks only receives, interrupt service routines, semaphores, message queues will send events [7].

3.4 Introduction to Workbench and Target

Executing programs on developing systems is known as Host, execution of programs on other systems is known as Target [7].

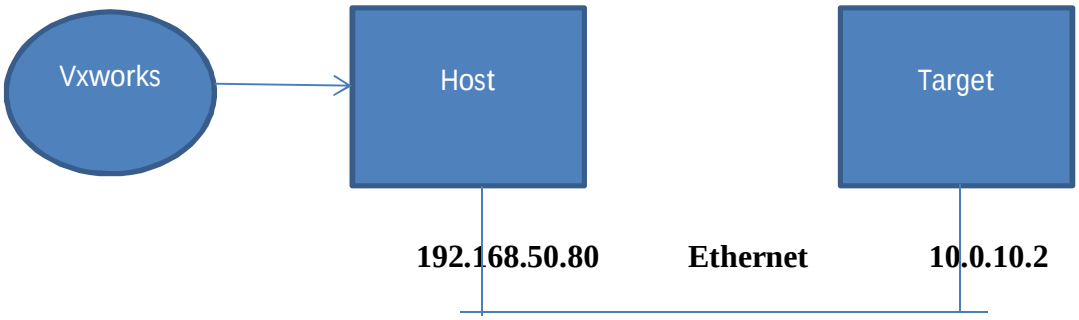

Figure 3-2 communication between Host and Target

Creation of image project and connecting to Hardware:

1. Open workbench (Start-> all programs ->WindRiver ->Workbench 3.3) ,the following window will appears on windows screen.

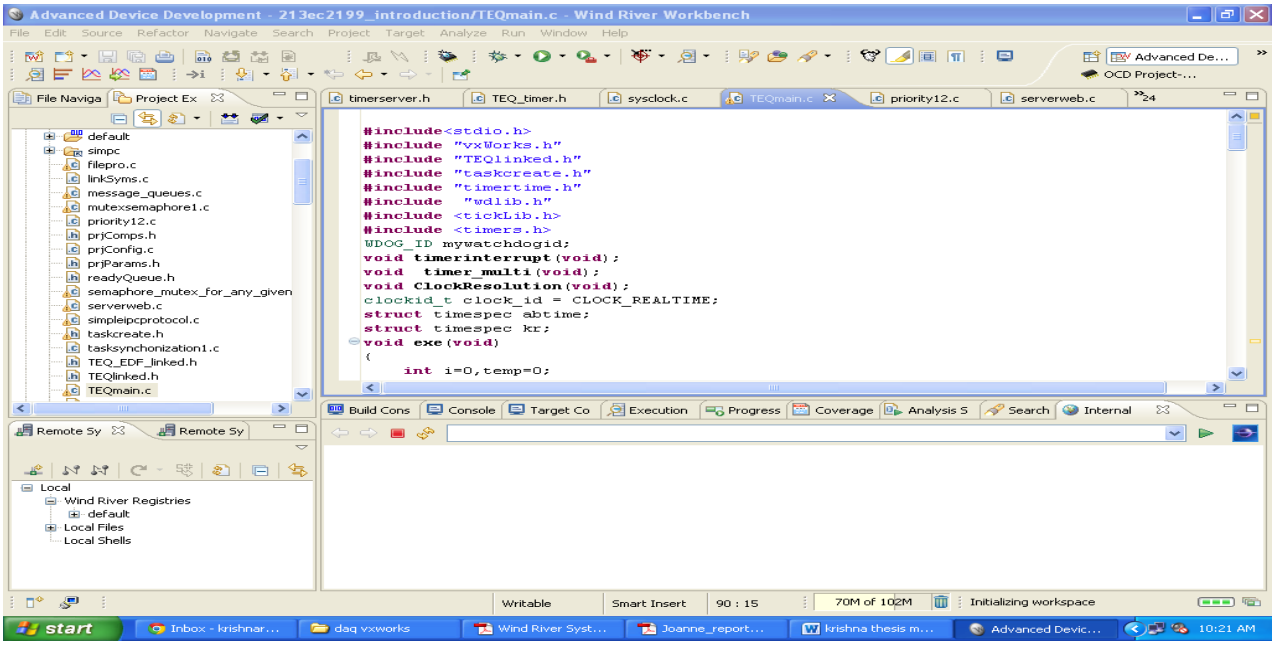

Figure 3-3 Windriver workbench 3.3

For Creating new project follows (File-> New->Vxworks Image project), give a name to your project. Make sure that select appropriate Board support package (simpc), Address mode (32 bit Kernel), tool chain (gnu). After clicking Finish [7].

2. After creating your project, your project name will be appearing in the project explorer window. Build your project by right clicking on your project.
3. To connect external target server, Target-> New connection -> Wind River Vxworks 6.x Target Server Connection->Finish.

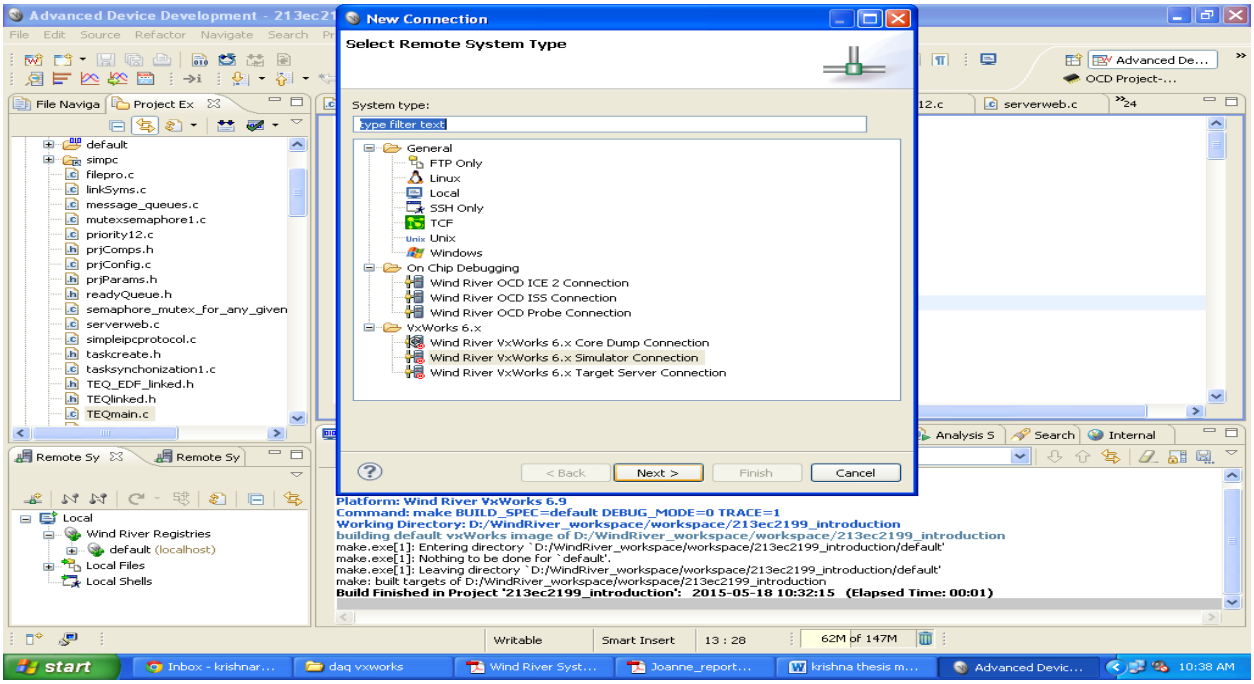

Figure 3-4 Wind River Target server connection 6. x

This target server will appear in Remote Systems. Program execution will be done at Target console window.

Wind River System Viewer:

In Real time embedded applications System viewer is used as a logic analyzer, by using this can troubleshoot and visualize target activities.it will help in visualizing behavior of multicore system, i.e. identifies CPU bound problems, detect deadlocks, race conditions, task interaction problems. Determine application performance, data for delay analysis [8].

Accessing the System Viewer Tools

If you want to utilize System Viewer Triggering and Configuration editors, you should connect to an internal simulator or external target via FTP. These tools collect information from the simulator or target.

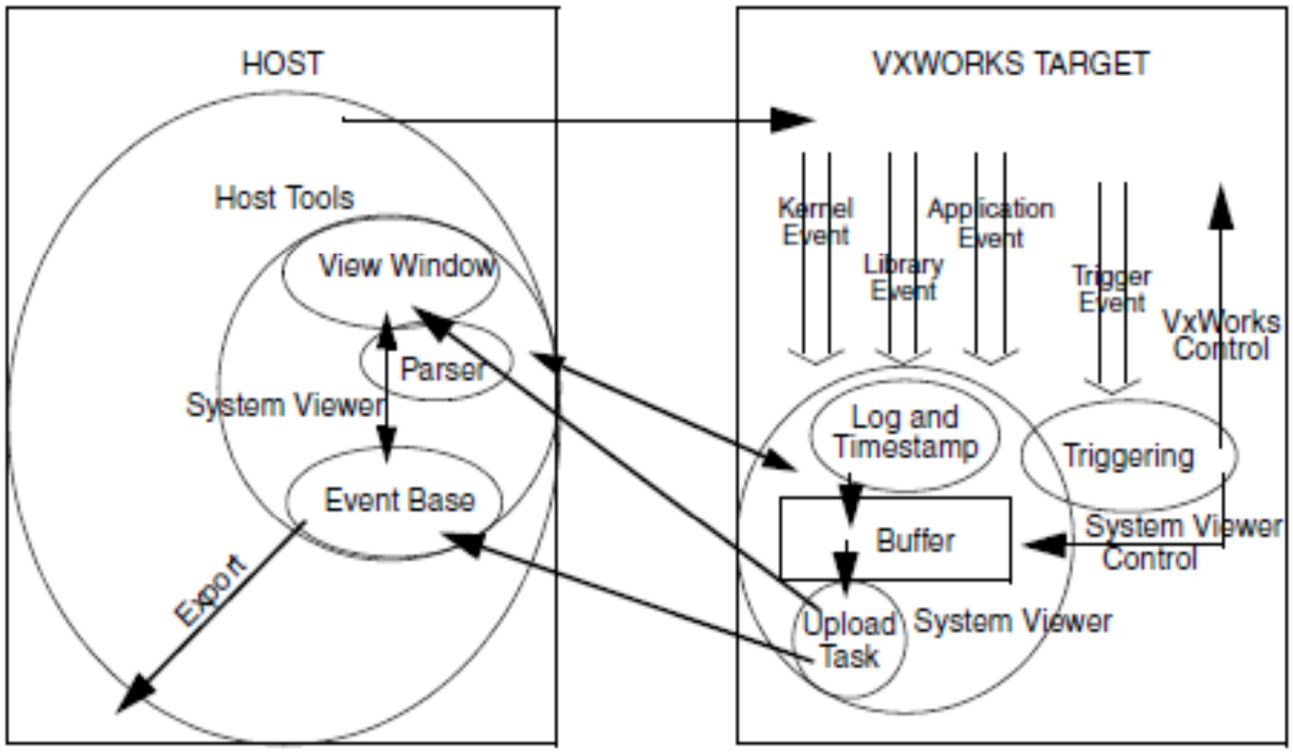

Figure 3-5 Wind River System Viewer Architecture [9].

The above figure represents both target and host activities. The target side activities are data uploading, event logging, time stamping, etc. The communication between target and host includes the WDB protocol over a network line or serial communication.

After establishing connection we can access the tools of system viewer by clicking the Remote Systems view and picking **Launch System Viewer**.

On the main menu, click on **System viewer configuration -> Start logging.**

To open **Event Receive**,

Select **Analyze -> Event Receive** on the main Workbench 3.3 menu.

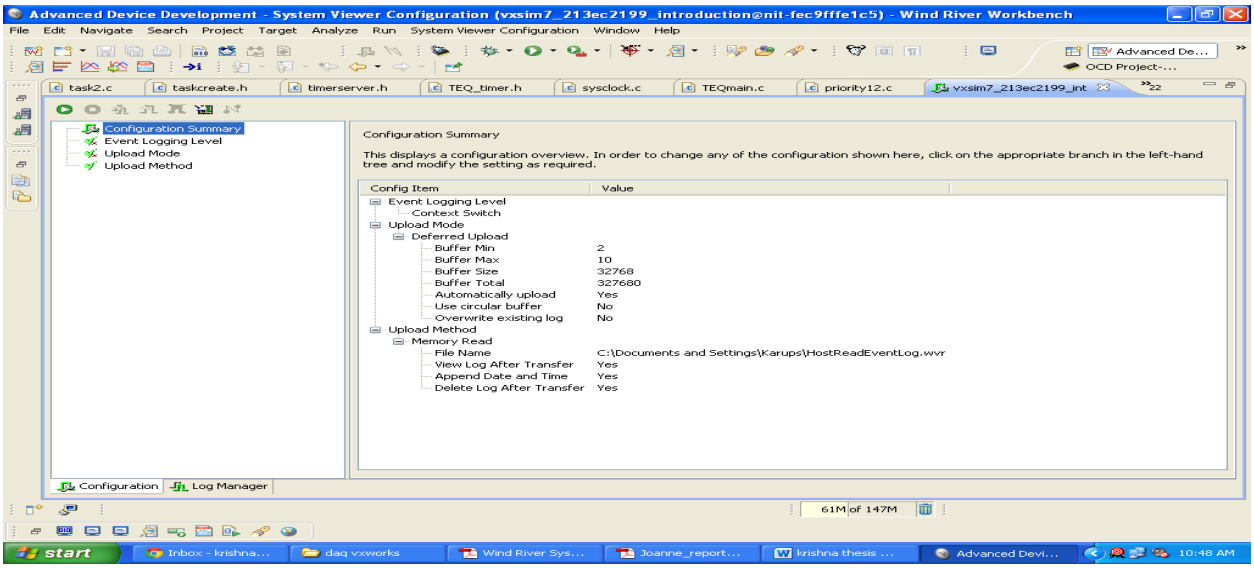

Figure 3-6 Windriver system viewer

The above figure represents launching configurations of System viewer, it is having event logging level, upload mode, and upload method.

4. Development and execution of custom uSR SchedulerS

This part presents the scheduling of periodic tasks using single timer and user service routine.

4.1 USR and Periodic tasks

After completion of its execution, periodic tasks change its state form ready state to suspend state by calling suspends function explicitly. So to design a periodic task a POSIX timer or watchdog timer could be used to trigger the user service routine before activation of the new task. Sometimes we could trigger USR after completion running task with independent on POSIX timer or watchdog timer. In this scenario, a processor having n number of periodic tasks to process, we need n number of timers or watchdog timers, which could be lengthy or sometimes not even possible. So in this Thesis we have used only one timer to schedule periodic tasks. I.e. By multiplexing single timer. The USR consists of the Time Event Queue (TEQ), these TEQ stores each and every identifier regarding User Service Routine. Then every time TEQ set shortest time period to the timer, after timer time expires, the timer will call timer interrupt function to reactivate timer. The USR checks every identifier of TEQ list. These TEQ lists are implemented by using Double Linked lists. Double linked lists consist of two fields one is Data field, second one is Address field. TEQ will consumes (number of tasks)*(4* 2 integer bytes + 4 long integer bytes) of memory [2].

Task id: it is a long data type, it can store task id when a new task is created or spawned.

Ex: task id is 272525744

Task period: each task has its own period, before the expiration of its period, task has to complete its execution time, and otherwise it could not meet its deadline.

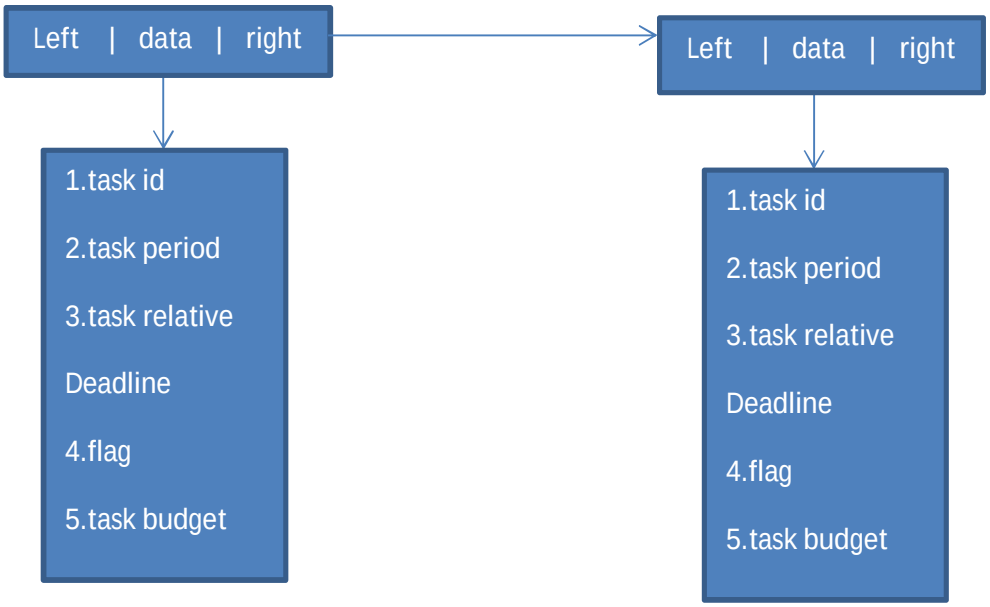

Figure 4-1 Time event queue identifiers list [3]

Relative Deadline: it is an integer data type. It is always holds remaining worst scenario execution time of tasks. If the relative deadline of a task equals to null, it refills relative deadline time by taking its execution time after completion of its absolute deadline.

Flag: it is an integer data type. The flag contains only two values; it indicates the status of tasks in the event queue. If a task has complete its execution, it will go to the last position of the queue, and a flag will becomes one, if it is not completed its execution the flag is zero.

Budget: it is also an integer data type, it always calculates the remaining time of each task and saves it. One remaining time becomes zero, it could raise an interrupt to Time Event Queue to update the budget.

4.2 User Service Routine Execution

USR is used to schedule tasks in some particular sequence, i.e. depending on what type of scheduling algorithms (Rate monotonic, or EDF) us are used [5].

User Service Routine has called in the following situations

1. After completion of task execution.
2. Expiration of timer budget.

After activation of USR, first it will get the clock time stamp by using clock_gettime () function. This time stamp will be helpful for calculation of USR worst case execution time and also this time stamp is used to update relative deadline, task budget of each and every task of USR.in this paper, we have used some macro functions to update USR, those macro functions are updateTEQ(), flagset(), sortTEQbudget(), sortTEQflag(),and allcompleted().

UpdateTEQ (): This macro updates relative deadline and task budget [3].

Task new budget=task old budget - timer current set time or completion task execution time. Where timer current set time or completion task execution time = previous clock time- current clock time. These two clock time stamps are collected from clock_gettime () functions. flagset(): this macro updates flag of TEQ. It consists of only those two values are either zero or one. Where zero means the task has created and it is waiting for the processor to execute. I.e. currently some higher priority task is executing. One means task had completed its execution.

SortTEQbudget (), sortTEQflag () these two macros sorts all available tasks on budget bases and flag basis respectively. Allcompleted () macro is checks all task status, if all tasks complete its execution, it will call some delay or sleep macros. The following flowchart will shows the complete execution of USR.

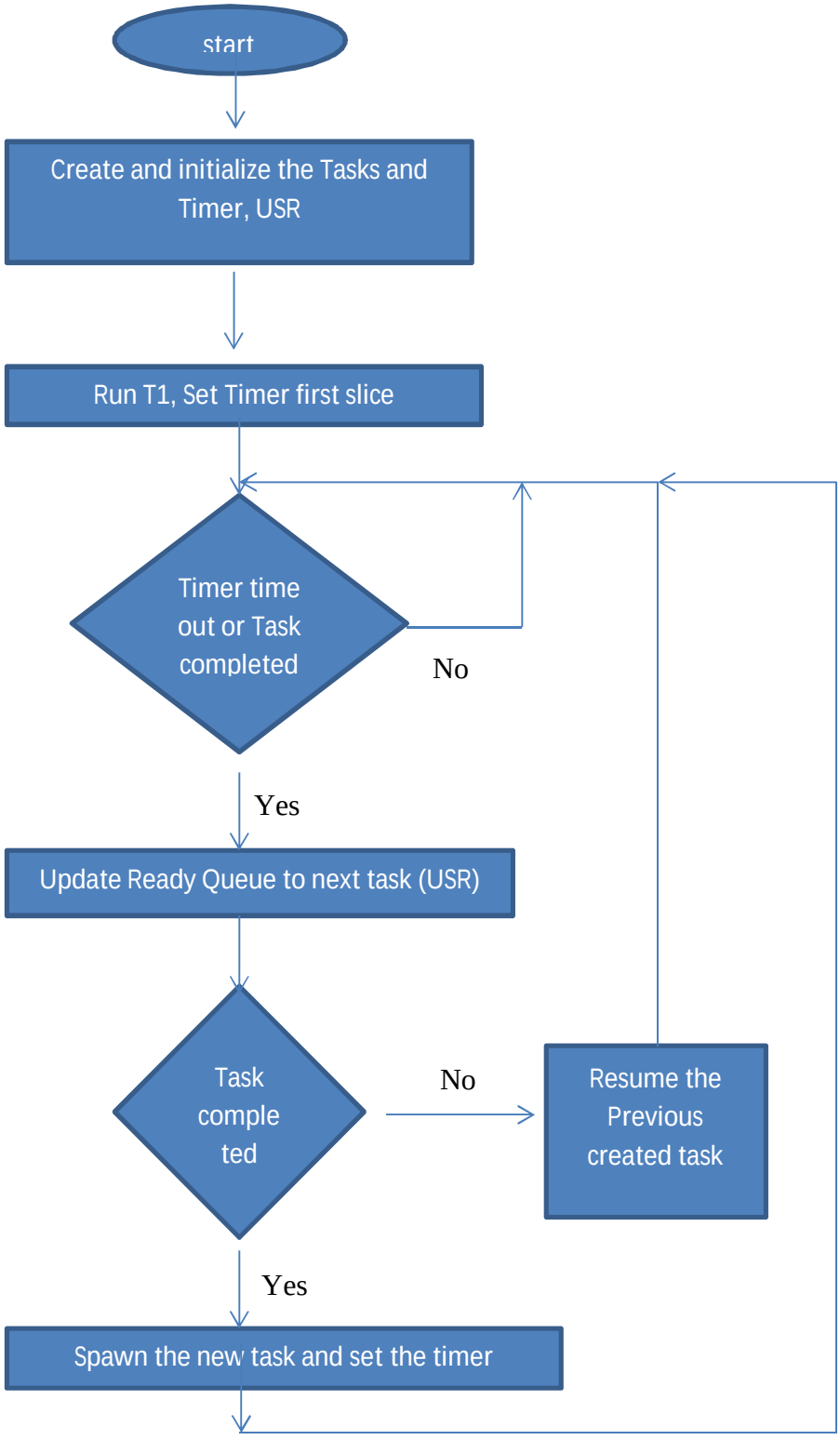

Figure 4-2 The Algorithm for running of Task Scheduling using USR

The flow chart defines the execution of USR and its corresponding macro execution

Like C-language the execution of macros is a sequential process, but in c-language the function is main, but in Vxworks we can run our execution from any function onwards.

4.3 Inbuilt Functions and its usage

To design above macros we have used Vxworks functions, among those we are specifying some important functions [9].

- taskCreate or taskSpwan: these functions could be used to create user defined tasks, with task name, task id, Stack size, priority.
- taskPrioritySet: this macro could be used to change the priority of each task during run time.
- taskDelay: this macro will helpful to sleep or delay the required tasks.
- taskActivate: this macro is useful to activate tasks, those have already been created or initialized.
- timerCreate: this macro could be useful to create a user defined timer in a POSIX environment.
- timerGettime: it takes the count value from system clock before expiration time and these values is reloaded again.

4.4 Simulation results and discussion

4.4.1 Round Robin Scheduling Algorithm:

Let us consider the four tasks T1, T2, T3, T4 having equal priorities of 100 and its worst case execution times E1, E2, E3, E4.Time slice 720 ms.

Table 4-1 tasks and its periods for RR

Task	Priority	Execution time
T1	100	1s
T2	100	2s
T4	100	3s
T4	100	3s

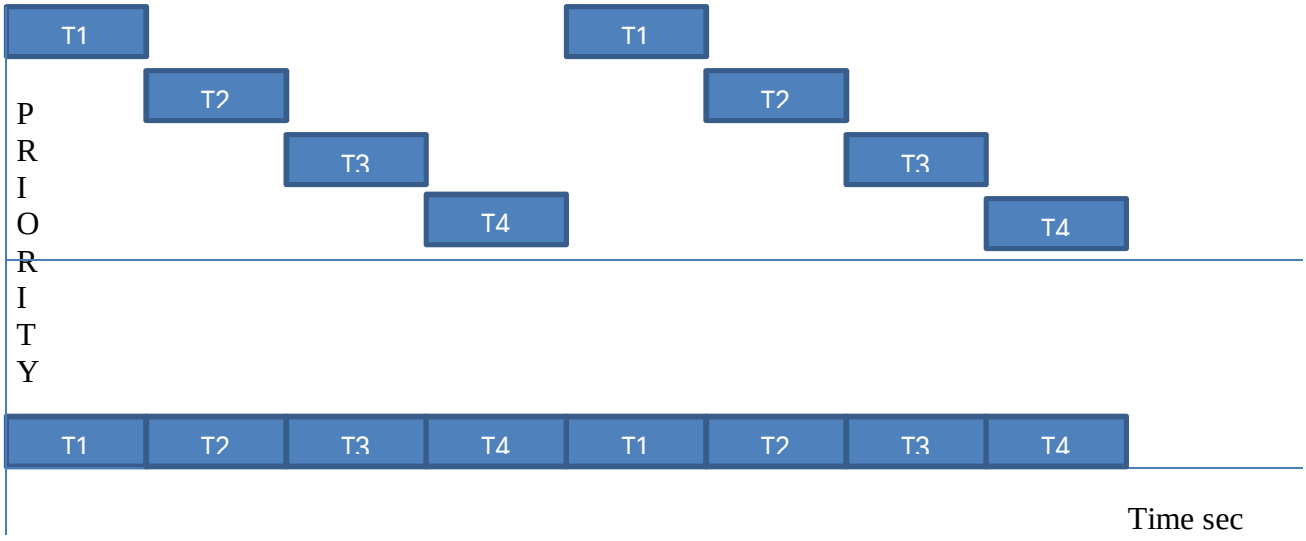

Figure 4-3 Round robin scheduling with equal periods and equal priority.

In the Above Table4.1 Show Four Task are scheduled in RR Algorithm, Initially, all tasks are created and activated, During the slot1 task1 is executed, After completion of task1 or completion of time slot1 the controller shifts to task2,and task2 executed during time slot2, this way tasks are executed sequentially one by one . The task Synchronization is done through Binary semaphores.

Simulation Results for Round Robin Scheduler**:**

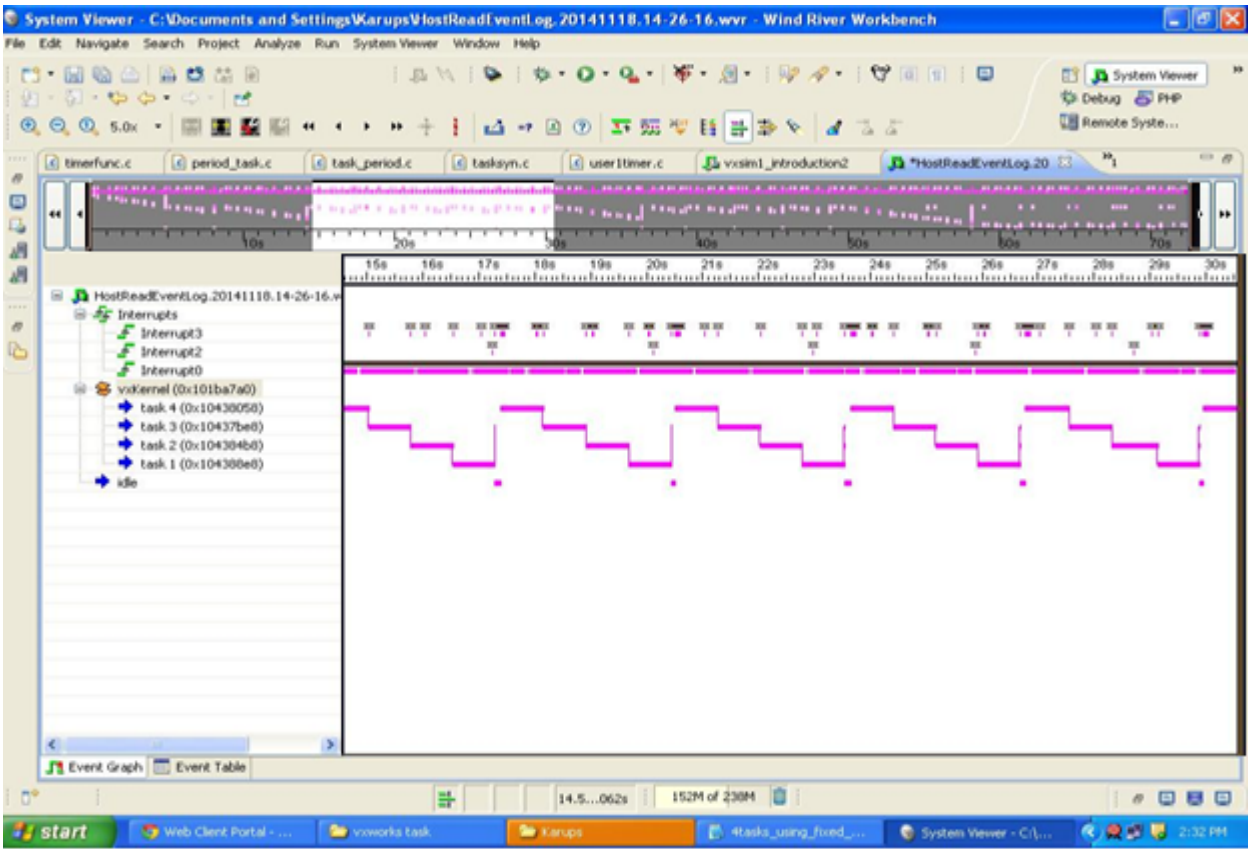

Figure 4-4 Scheduling of periodic tasks using Four Timers in Vxworks

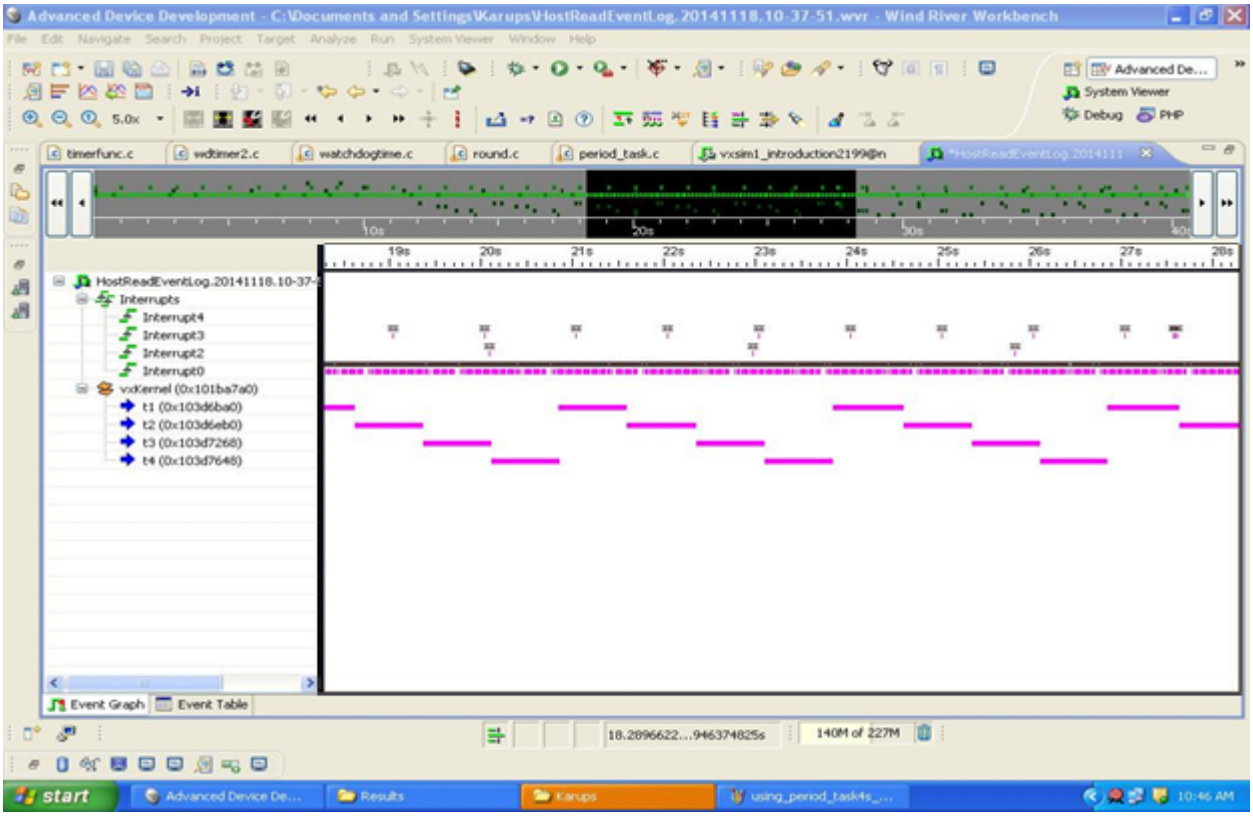

Figure 4-5 Scheduling of periodic tasks using USR and single Timer in Vxworks

4.4.2 Rate Monotonic Scheduling Algorithm:

Let us consider the three tasks T1, T2, T3 having different priorities of P1,P2,P3 and its worst case execution times E1, E2, E3.

Table 4-2 List of available tasks and its priorities

Task	Period	Priority	Execution time
T1	5	50	1S
T2	7	60	2S
T3	11	70	3S

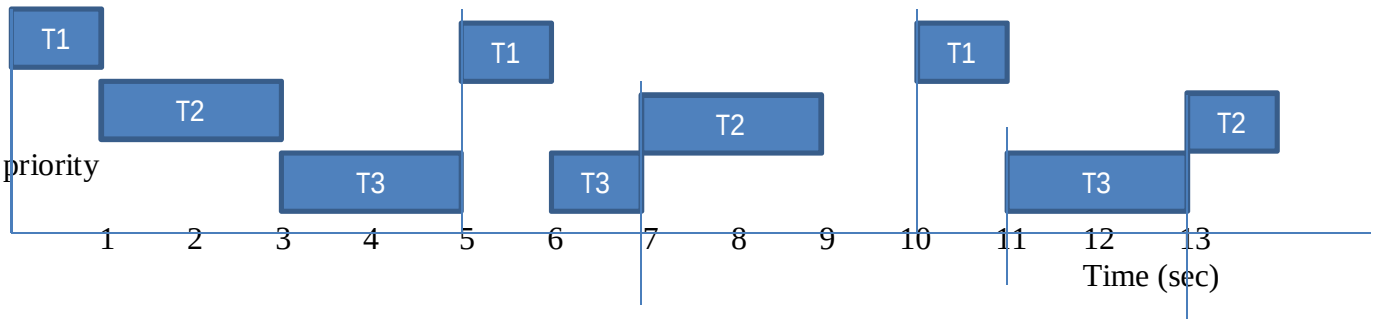

Figure 4-6 RMS Task scheduling with three Tasks with Different Priority and execution time

CPU utilization for feasibility of tasks is given by [18]

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

U=1/5+2/7+3/11=0.2+0.286+0.272=0.758<0.779

In the Above Table 4.2 Shows 3 Task are scheduled in RMS Fashion, Initially all task are created and activated with Different Priorities Depending on its period, Now the Highest priority task1 is executed, After completion of task1 ,controller shifts to task2 because task2 is next highest priority, and task2 executed during this period if highest priority task comes, it just suspends current execution task, activate the highest priority task after completion of highest priority task lowest priority task resumes, this way tasks are executed sequentially one by one . The task Synchronization done through Binary semaphore. The simulation Results are shown below.

Simulation Results for Rate Monotonic Scheduler:

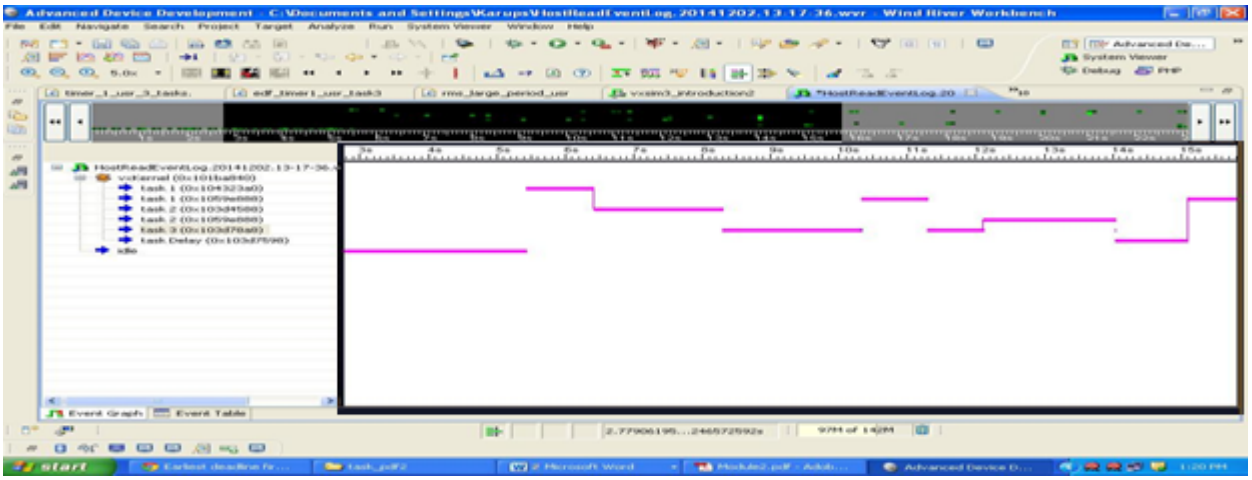

Figure 4-7 feasible Task Scheduling in Rate Monotonic Scheduling

CPU utility for feasibility of tasks is given by [18]

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

U=1/5+2/6+3/7=0.2+0.333+0.428=0.961>0.779

In this case feasibility of RMS Scheduling fails.

Table 4-3 List of available tasks and its priorities for RMS Scheduling

Task	Period	Priority	Execution time
T1	5	50	1S
T2	6	60	2S
T3	7	70	3S

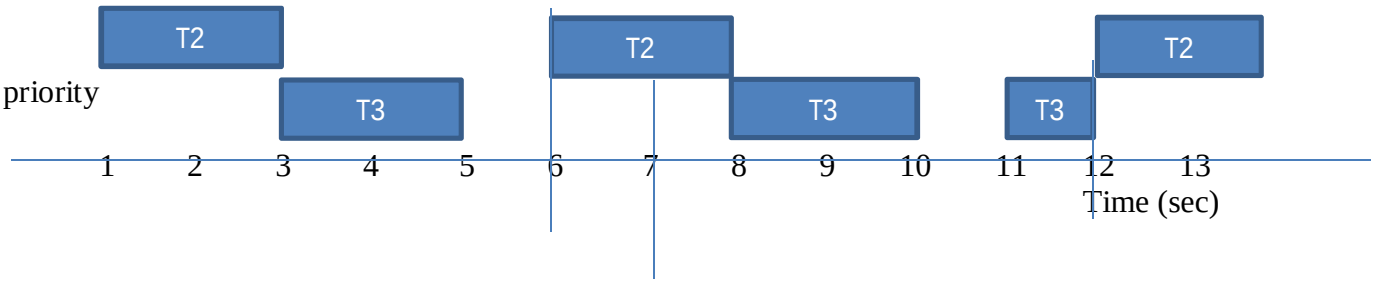

Figure 4-8 Task Scheduling Algorithm with over load tasks

Simulation Result for RMS Scheduling

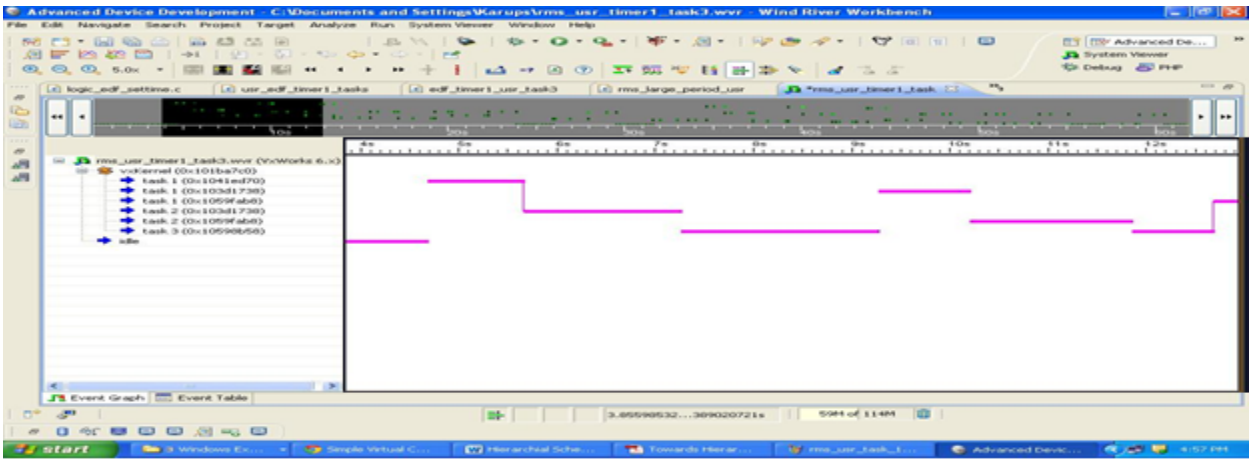

Figure 4-9 Task scheduling deadline miss.

System clock with resolution: Proposed 60 ticks/sec
: Conventional 4500 ticks/sec

Table 4-4 USR RMS execution times

No of tasks	USR (RMS) (us)			USR(RMS) Proposed (ms)		
	max	avg	min	max	avg	min
3				2.579	2.413	2.30
10	71	65	63	4.468	4.251	4.035
20	119	110	106	8.144	8.048	7.97
30	127	158	155	13.451	11.781	11.303

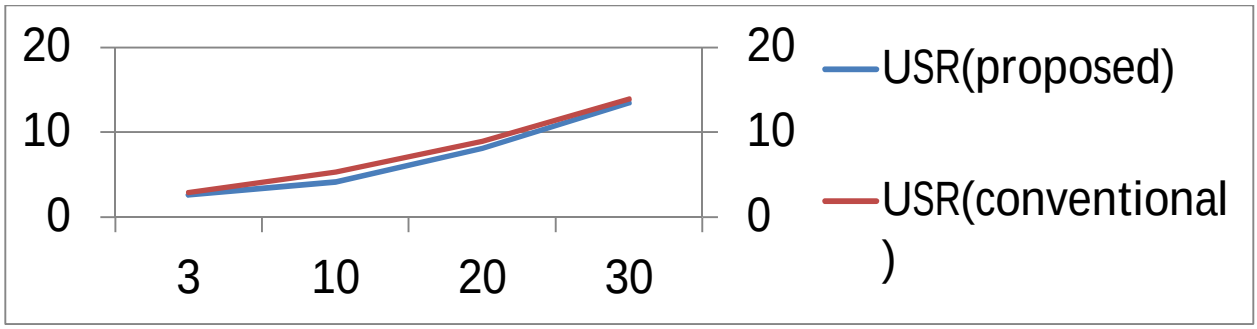

Figure 4-10 Maximum USR execution time in ms

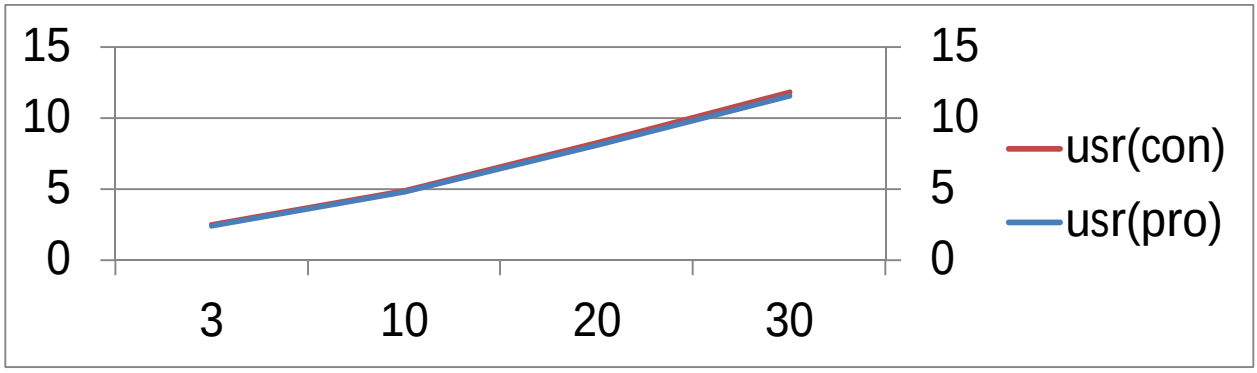

Figure 4-11 Average USR execution times in ms

The above table 4.4 will shows comparisons proposed algorithm and conventional algorithm. We have executed these tasks some 50 to 60 minutes. If we increase the number of tasks the buffer size of system viewer is not sufficient so we are confined execution time to 60 minutes.

4.2.3 Earlier Deadline First Scheduling Algorithm:

Let us consider the three tasks T1, T2, T3 having different priorities of P1,P2,P3 and its worst case execution times E1, E2, E3.

Table 4-5 List of available tasks and its priorities for EDF

Task	Period	Priority(initially)	Execution time
T1	5	50	1S
T2	6	60	2S
T3	7	70	3S

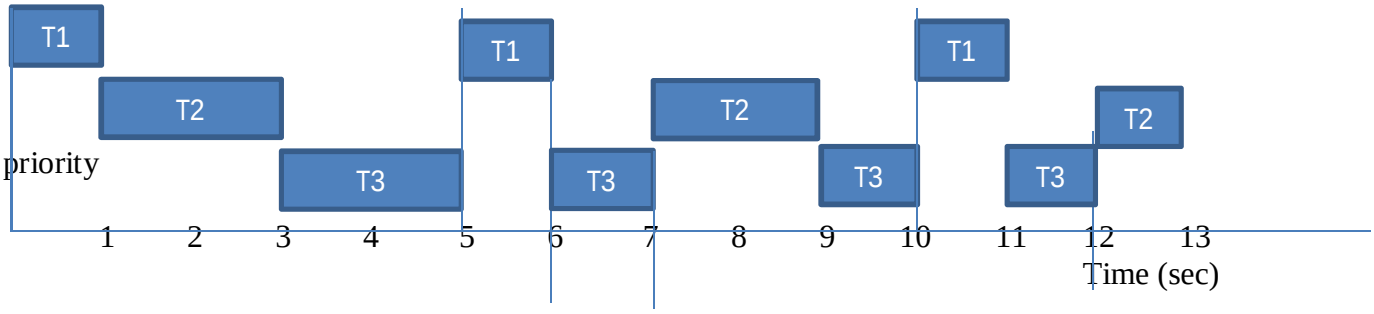

Figure 4-12 EDF Task Scheduling with Different Priority and execution time

An implementation of EDF would be maintained that all tasks that are ready for execution queue. Every time USR has to ravine the absolute deadline of the each task. At every time USR Scans the ready queue which one is the shortest deadline task. Depending on the shortest deadline USR has to activate the highest priority task.

Simulation Result for EDF Scheduling

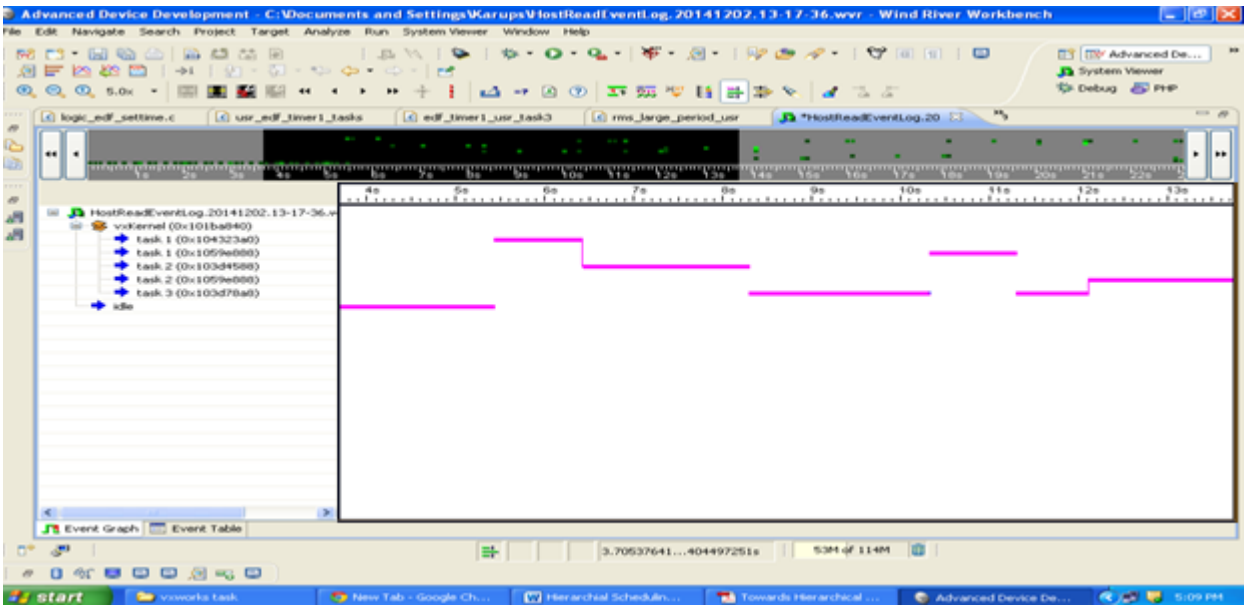

Figure 4-13 Task Scheduling using EDF Algorithm.

System clock with resolution: Proposed 60 ticks/sec
: Conventional 4500 ticks/sec

Table 4-6 USR EDF execution times

No of tasks	USR (EDF) (us)			USR(EDF) Proposed (ms)		
	max	avg	min	max	avg	min
3				2.59	1.422	1.18
10	74	70	68	4.72	4.46	4.10
20	131	118	115	8.94	8.21	7.28
30	187	172	169	14.13	12.37	10.94

5. Design of HierarcHical Scheduling structure

5.1 Introduction to Hierarchical Scheduling structure

Hierarchical scheduling structure is a functional process in carrying portable of real time tool by considering time dividing among different applications. In this, each system can be divided into a number of subsystems those can be scheduled by an external scheduler (global scheduler). Every subsystem includes a bunch of those tasks are scheduled by internal scheduler (local scheduler). In this framework, each and every subsystem can be isolated. This could be helpful for building and analysis of each subsystem individually. It could support the integration of all subsystems into a system. This mainly involves all timing needs are fulfilled. The main aim is to develop a cost effective Hierarchical scheduling structure. In this paper a 2-level hierarchical scheduling structure has been implemented [3].

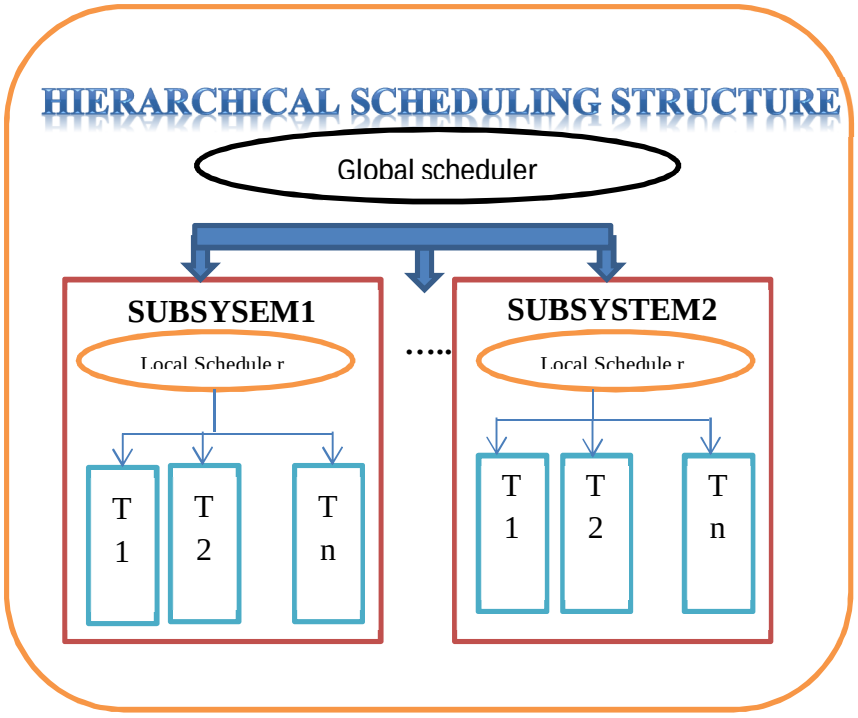

Figure 5-1 hierarchical scheduling framework [13].

5.2 Design of Hierarchical Scheduling Frameworks without Resource sharing

Let all the available tasks are unrelated to one another, logical resource sharing among the tasks is zero and tasks are not suspend itself. The hierarchical scheduling structure includes subsystems S_i belongs S. Here the integrated whole System is S, the number of subsystems are denotes I. Every subsystem S_i assists a bunch of tasks and an internal Scheduler (Either EDF or RMS), and the Whole system assists a bunch of subsystems and an external Scheduler (Either EDF or RMS) [5].

Global or External Scheduler:

Like tasks, subsystems are also periodic in nature. To schedule subsystems need to implement periodic servers, these servers similar to periodic tasks [3]. Each and every server includes the following server parameters.

1. Server_period
2. Sever_budget
3. Server_remainig_budget
4. Server_task_count
5. Scheduler_type_status

Server Ready Queue:

This is implemented using double linked list, it can store valid reaming budget of all servers. Every time server starts it tasks budget value equivalent of its period. The server would be attached to Ready Queue. If the server takes execution, it would execute its internal tasks for its time, after execution of the above tasks the server budget would be decreased by same time t, now the server remaining budget is Server_period – t. Once Server_budget becomes zero, the

timer generates Server interrupt, the Server ready queue will give the control to the next highest priority server.

Server TEQ:

Server Event queue maintains all Server related time events. Every time it calculates next event time, these times will be used to set a timer interrupt.

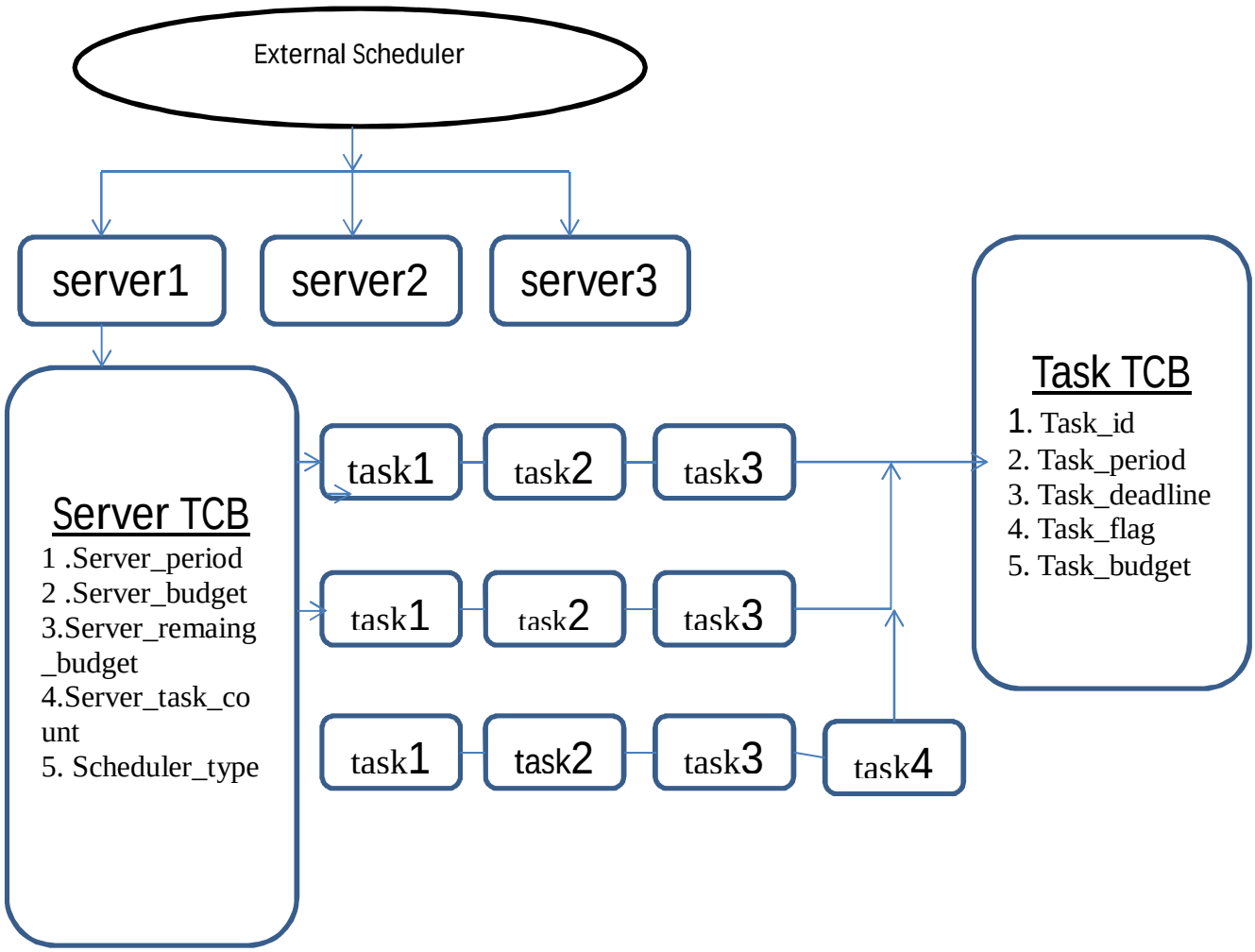

Figure 5-2 implementation of HSF scheduling algorithm [3].

The above diagram 5.2 describes hierarchical scheduling framework implementation in Vxworks. Ready queue server is responsible for Scheduling of servers. Calculation of Server_remaining_budget involves getting of physical clock time, setting of absolute time to outdo its server budget. When a higher priority server is running, the lower priority server subtracts higher priority server execution time of its budget. After completion of server execution, it will go to bottom of server ready queue [4].

Local or internal scheduler:

The CPU resources on receiving the server will execute the ready tasks that belong to the server. Tasks in the ready queue can be dealt using the following two approaches [12].

- Swapping of severs occurs which means the previously running tasks of server will be pushed to the bottom of the ready queue and ready tasks of new executing server will be added up. In order to push the tasks to the bottom, the state of the tasks is altered to suspend the state. This in turn suspends the position of the tasks that finish their their work which is undesired. This can be overcome by modifying the same flag in the task control block.
- A lower priority will be assigned to the tasks of preempting server and those tasks that related to the new executing server will have more priority so that they will execute exclusively on the processor, based on its scheduling policy subsystem scheduling is done.

An example of simple Server execution is shown table 5.1.

Table 5-1 list of servers and its resources

Server	S1	S2
Priority	P2	P1
Period	20(sec)	40(sec)
Budget	10(sec)	15(sec)

Where S1, S2 are the couple Servers, P1 is the lowest priority and P2 is the highest priority. T1, T2 are the couple of tasks corresponding Server S1. T3 is task corresponding Server S2.

Table 5-2 list of tasks and its resources

Tasks	T1	T2	T3
Serves	S1	S1	S2
Priority	P1	P2	P2
Period	20(sec)	15(sec)	30(sec)
Execution	4(sec)	2(sec)	1(sec)

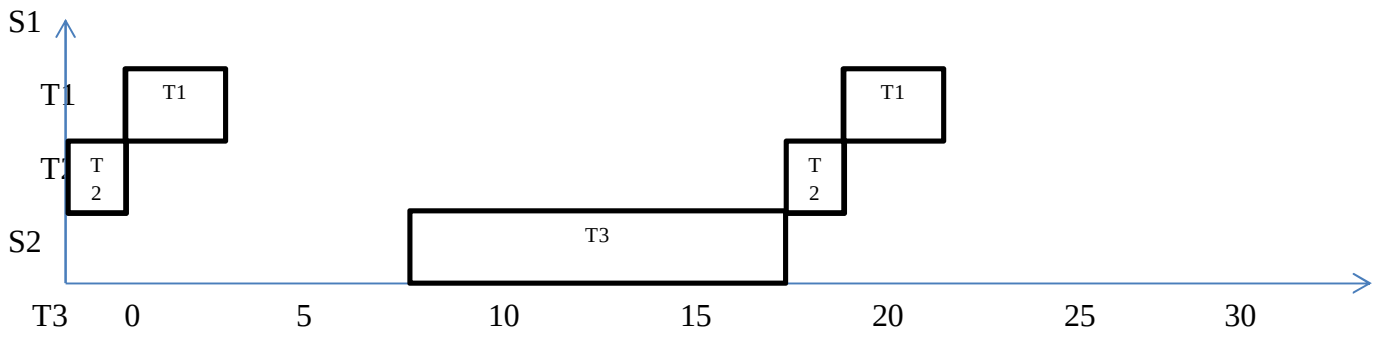

Figure 5-3 implementation of HSF scheduling algorithm [13].

5.3 Simulation results and discussion

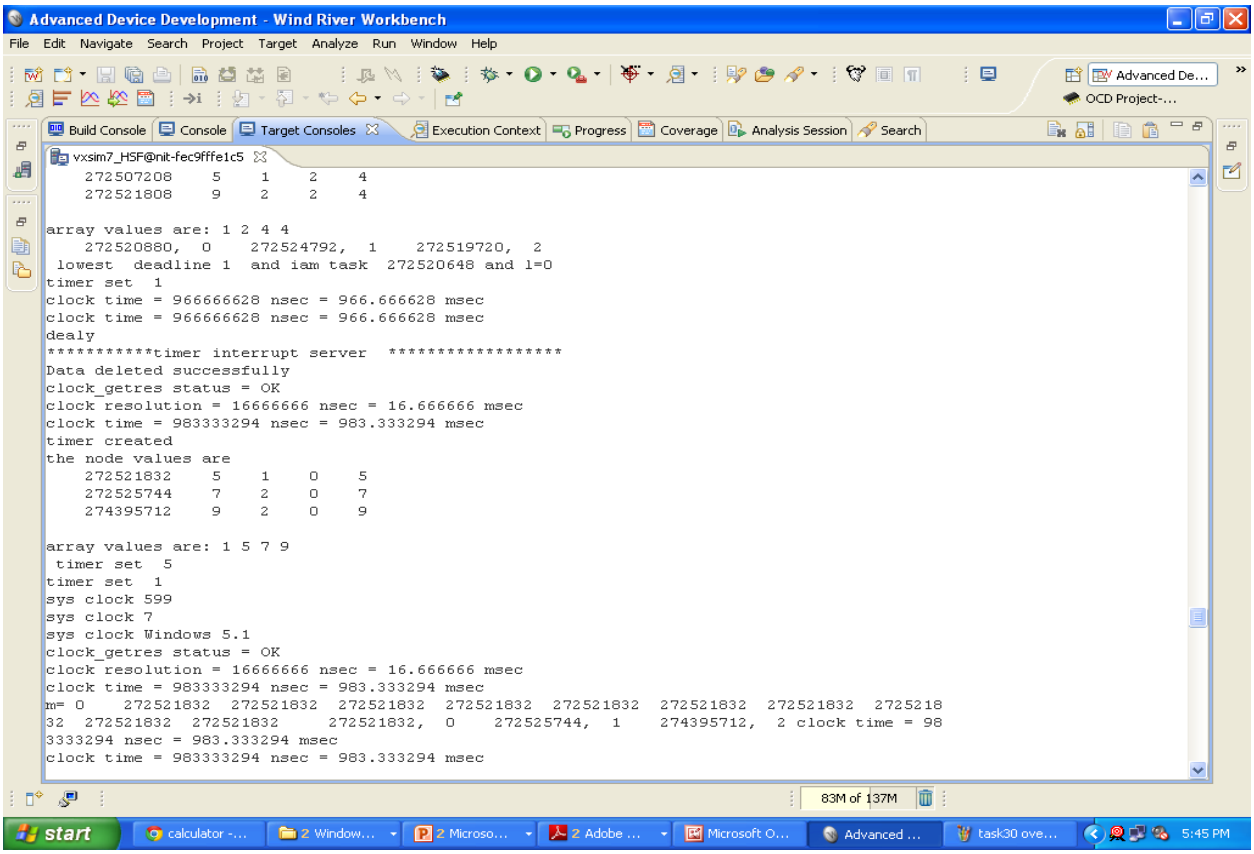

Figure 5-4 executions of servers and its corresponding tasks

In the figure there are two servers, these are interrupted by using a single timer. The server1 having highest priority compared to server 2. When the server came to execution first it will go top of server ready queue and then it starts execution. This figure is target console window of Windriver workbench.

6. Conclusion and scope of future work

6.1 Conclusion

In this thesis, Learning WindRiver Workbench 3.3, developing and simulating the programs related to Task management, memory testing and file systems in workbench. Also ported to, SBC hardware for validation. The Basic Task Scheduling algorithms along with First in, first out, Round Robin, Rate Monotonic and Earliest Deadline First Scheduling Algorithms in Vxworks are implemented by using internal Vxworks functions and user defined macros. A User Service Routine is designed with single timer. Extensive study of Rate monotonic and Earliest Deadline First Scheduling Algorithms and its implementation is done without any modifications of the kernel. A two level Hierarchical scheduling framework (Rate monotonic and Earliest Deadline First Scheduling is used to schedule local and global level tasks) has been implemented without any logical sharing of system resources among the subsystems and tasks in Vxworks.

6.2 Scope of future work

The future work will includes aperiodic and sporadic tasks along with more system resources will increase system towards more real time. Implementation of these schedulers in real time Data Acquisition and Control (DAC) systems is possible. In the era of Internet of Things (IoT) these schedulers will play a crucial role in the success of IoT networks.

Bibliography

[1] Ren Peng, Xiang Zheng, "A Multitasking Scheduling For Vxworks: Design Task Simulation". In Proc. Of International Conference on Artificial Intelligence, 2009.

[2] Moris Behnam, Thomas Nolte, Insik Shin. Towards Hierarchical Scheduling in VxWorks. in Proceedings of the International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'08), July 2008,

[3] Mikael Asberg, Thomas Nolte, Shinpei Kato,"ExSched: An Externel CPU Scheduler FrameWork for Real-Time Systems". In 2012 IEEE International Conference On Embedded and Real-Time Computing Systems and Applications.

[4] Reinder J. Bril and Thomas Nolte," Implementation of Overrun and Skipping in VxWorks". In 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications.

[5] Martijn M.H.P. van den Heuvel, Reinder J. Bril and Johan J. Lukkien," Extending a HSF-enabled Open-Source Real-Time Operating System with Resource Sharing". In 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications.

[6] Junqing Li, Yuting Wang, Tao Sun, "A Hybrid genetic algorithm for task scheduling in internet of things". In ICIT 2013 the 6th International Conference on Information Technology.

[7] Joanne Sirois, "VxWorks Real-Time Kernel Connectivity: Cumulative Report", Florida Gulf Coast University, May 4, 2009 .

[8] Wind River VxWorks. " Wind River. Jan. 2008. Wind River". 21 Sept. 2008.

[9] Wind River. VxWorks Application Programmer Guide 6.x [10] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In Proc. of the Fourth ACM International Conference on Embedded Software, September 2004.

[11]S.Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Journal of Real-Time Systems,2:301–324, 1990.

[12]M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems. In Proc. of the 7th ACM and IEEE International Conference on Embedded Software (EMSOFT'07), 2007.

[13] Rafia Inam, Jukka Maki-Turja, Mikael Sjodin, Moris Behnam, "Hard Real-time Support for Hierarchical Scheduling in FreeRTOS".

[14]Alex Gantman, Pei-Ning Guo, James Lewis, Fakhruddin Rashid, Scheduling Real-Time Tasks in Distributed Systems: A Survey: University of California, San Diego Department of Computer Science.

[15]Junqing Li, Yuting Wang, Tao Sun, A Hybrid Genetic Algorithm For Task Scheduling In Internet Of Things, ICIT 2013 The 6th International Conference on Information Technology May 8.

[16]G. C. Buttazzo. "Rate Monotonic vs. EDF: Judgement day.Real-Time Systems", 29(1):5–26, January 2005.

[17]C. Diederichs, U. Margull, F. Slomka, G. Wirrer. “An application-based EDF scheduler for osek/vdx”. In DATE’08: Proc. of the conference on Design, automation and test in Europe, 2008.

[18] “ Real Time Task Scheduling “, NPTEL, IIT Kharagpur.

[19]R. I. Davis and A. Burns. “Hierarchical fixed priority preemptive scheduling”. In Proc. of the 26th IEEE International Real-Time Systems Symposium (RTSS’05), December 2005.

[20]M. M. H. P. van den Heuvel, M. Holenderski, W. Cools, R. J. Bril, and J. J. Lukkien, “Virtual timers in hierarchical real-time systems,” Proc. WiP session of the RTSS, pp. 37–40, Dec. 2009.

[21]I. Shin and I. Lee, “Periodic resource model for compositional real-time guarantees,” in Proc. RTSS, Dec. 2003, pp. 2–13.

[22]X. Feng and A. Mok, “A model of hierarchical real-time virtual resources,” in 23th IEEE Int. Real-Time Systems Symposium (RTSS’02), Dec. 2002.

[23]M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonz´alez Harbour, A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Mono-tonic Analysis for Real-Time Systems. Kluwer Academic Publishers, 1993.