Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EFFICIENT SCHEDULER AND MULTI THREADING FOR RESOURCE AWARE EMBEDDED SYSTEM

[1]KARTHIKEYAN.V, [2]Dr.S.RAVI

Asst.Professor, ECE Department, Dr.M.G.R.Educational & Research Institute, Chennai
Prof&Head, ECE Department, Dr.M.G.R.Educational & Research Institute, Chennai
keyansethu@gmail.com

ABSTRACT

In embedded system applications the processor is connected to different segments of the hardware and each hardware is associated with software called task. The scheduler is the part of the software that determines which task will run next and its implementation is a difficult process in RTOS. Existing scheduling methods fail to suit real time system requirements. The purpose of this research work is to design a efficient scheduler in real time. In the proposed method a new round robin scheduling algorithm approach is presented in µC/OS-II operating system and LPC 1768 environment. µC/OS-II provides two API's that lets user to enable and disable scheduler within a task. Although it is not advised to use this API's as it is against the concept of Real time priority scheduling, it may be required in certain situations. The scheduler designed handles scheduling of multiple tasks efficiently and an analysis of scheduler performance using threading and non threading approach is made. The experimental result shows improved performance over the existing methods in terms of various performance criteria. The implication of this research work is that the scheduler designed can execute tasks based on priorities within the allotted time.
Keywords: *Scheduler, multi threading, Alphabet and number sorting, LPC 1768*

1. INTRODUCTION

Real time operating systems are a segment or a part of the whole program that decides the next task, task priority, handles the task messages and coordinates all of the tasks. The scheduler, also called the dispatcher, is the part of the kernel responsible for determining which task will run next. Most real-time kernels are priority based. Each task is assigned a priority based on its importance. The priority for each task is application specific. In a priority based kernel, control of the CPU will always be given to the highest priority task ready-to-run. When the highest-priority task gets the CPU, its execution is determined by the type of kernel used. There are two types of priority-based kernels: non-preemptive and preemptive. When a task starts and runs until termination or giving control willingly are called non preemptive scheduling and those tasks which are forced to give the resource are called preemptive scheduling. The scheduler needs to save the status of the tasks and after going through the list of tasks having different priority levels, it switches control to the task having highest priority. In general the tasks have three states .The state before running state is called ready to run state, when the task is in execution is called Running state and when the task doesn't have resources to run then it is sent to a blocked state. The scheduler function is affected when the task switches from running to waiting state, switches from running to ready state ,switches from waiting to ready state or when the task terminates as illustrated in fig.1. In sections III & IV different performance criteria and various algorithms for Scheduler is discussed. In section V a presentation is made on the various RTOS functions used and about development cycle. The obtained experimental results are shown in section VI and a detailed discussion on the comparison of performance of scheduler designed with the existing methods is presented in chapter VII.

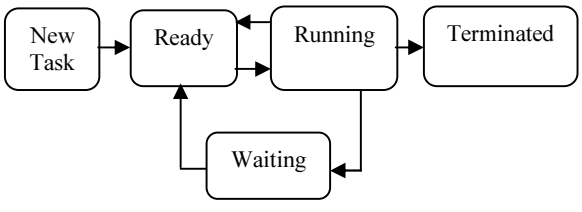

Figure 1: Various states of a task

2. RELATED WORKS

Task scheduling for multiprocessor system is presented in [12] .In this scheduling algorithm is

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

task scheduling based and it is queue based approach to schedule parallel tasks. High energy consumption is a biggest problem in cloud computing and it can be minimized by energy aware task scheduling by considering resource usage of different nodes [4]. Depending on the service reliability of grid system an effective resource scheduling method having fault tolerant feature is presented in [14]. In the systems that consists of both critical and non critical sections the scheduling algorithm needs to minimize average waiting time of the non critical tasks in the queue and also it should avoid context switching of critical tasks [7].In [13] a novel CPU scheduling algorithm for both pre emptive and non pre emptive is presented and the results are compared with existing algorithms and an effort is made to improve the CPU efficiency in multi programming operating system. A detailed survey of different CPU scheduling algorithms and their merits and demerits are presented in [9].In order to improve the efficiency of CPU there is a need to manage all the processes properly and a new round robin scheduling algorithm for CPU scheduling is presented in [11].

An efficient memory management using buddy allocator algorithm in the LPC 1768 and µC/OS-II environment is presented in [6].Here the author highlights advanced features of µC/OS-II and LPC 1768 and a effective memory management algorithm is designed using them. In [3] the author presented a priority based round robin algorithm. The method integrates the features of round robin and priority scheduling algorithm thereby reducing various performance parameters. A new scheduling algorithm based on the integration of round robin and shortest job first is presented in [8].In this priority is calculated by SJF and starvation is reduced by round robin algorithm.

3. PERFORMANCE EVALUATION OF SCHEDULER ALGORITHMS

The following parameters are the performance indicators of scheduler.

Utilization : The term describes the fraction of time a device is in use.
Throughput : It is the number of jobs completed per second
Service time : It is the time taken by the device to complete the execution of a request.
Queuing Time : It is the amount of time a request waits on a queue to get service from the device.
Residence Time : It is equal to the sum of service time and queuing time
Response time : It is the time used by a system to respond to a submission of request
Think Time : It is the time taken for figuring out the next request.

The function of the scheduler is to allocate CPU to a specific task for a period of time. Schedulers are basically classified into two basic types Long term and short term schedulers [1]. The functionality of schedulers is shown in fig.2. Long term scheduler selects the required tasks and bring them into the ready queue. It consumes much time for the creation of task and eventually slow in nature. Whereas short term scheduler selects particular task among the tasks that are ready to execute and allocate CPU or core to that particular task. Short term scheduler decides the task that is going to be executed next and it is fast compared to long term scheduler.

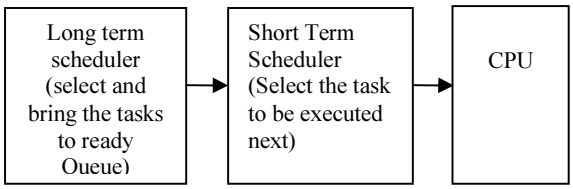

Figure 2: Functionality of Schedulers

4. TYPES OF SCHEDULING ALGORITHMS

The different types of scheduling algorithms are explained below

4.1 First Come First Served

This method is same as that of FIFO and it is a very simple algorithm. The main drawback in this is average queuing time is long and residence time depend on ordering of processes in queue.

4.2 Shortest Job First

It is suitable for situations where minimizing queuing time is of prime importance.

Here scheduling is based on logged history and is hard to implement.

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.3 Preemptive Algorithms

When a Higher priority task is ready for execution, the execution of current task is stopped and execution of higher priority task starts. The algorithm gives first priority to short jobs.

4.4 Priority Based Scheduling

Here priority is assigned for each and every task to be executed and highest priority task is scheduled first. Tasks with same priority are scheduled using FCFS.

4.5 Round Robin Scheduling

The task need to complete its execution within the fixed time slot otherwise the task may lose its flow, and data generated or it has to wait for its next turn. In this algorithm choosing the time quantum is very important.

4.6 Multilevel Queues

In this each queue has a scheduling algorithm and a priority based algorithm is used to arbitrate between the queues..A feedback method is used to move between queues. The method is complex but flexible.

4.7 Real Time Scheduling

This type of scheduling is used when a critical task is to be executed within a fixed time in hard real time systems and where critical processes need higher priority over lower priority tasks in soft real time systems [10].

5. PROPOSED METHOD

In the proposed method two tasks are created task 1 with priority 57(high) and Task2 with priority of 58(low).Task1 will read raw rgb file from the sd card and display it on the lcd. Task 2 will lock the scheduler and wait for key to be pressed and then unlocks the scheduler. The flowchart for task1 is shown in fig.3 and flow chart for task 2 is shown in fig. 4.

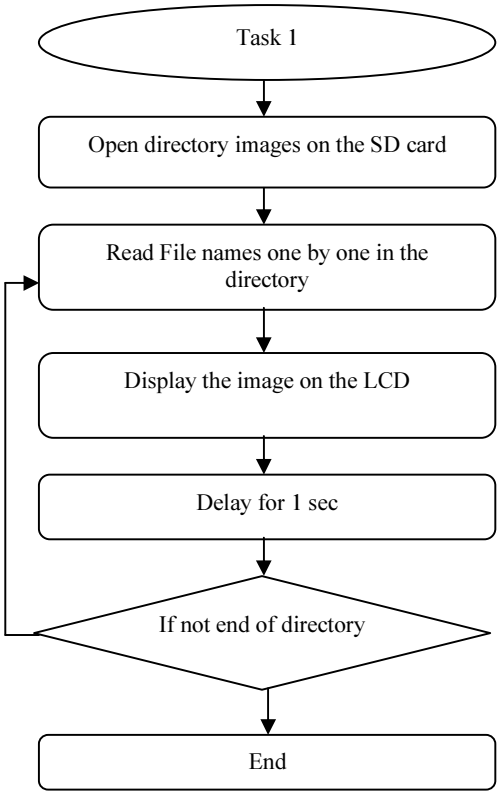

Figure 3: Flow sequence for task 1

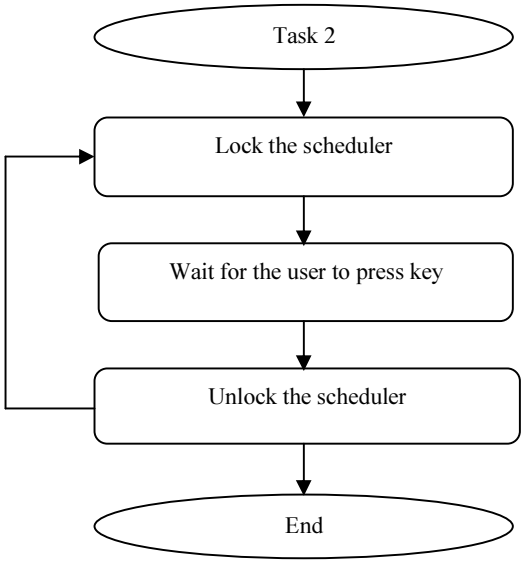

Figure 4: Flow sequence for Task 2

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5.1 µCOS API's and its functions

The various RTOS functions used in this work is given in table 1.

Table 1 : RTOS functions

µC/OS-II API	Functions
App_Task1Create()	Create the Application Task
OSTaskCreate()	Creating an Task
OS_TASK_Name_EN	Enabling Task Name
OSSchedLock()	Prevents tasks rescheduling
UART0_Send string()	Send the string through UART 0
Static OS_STK align	Aligning the state
Static void uctsk_Task()	Using the stack in tasks
UART0_Init()	Initialize the UART Port
OSTaskChangePrio()	Changes the priority of task
OSSchedUnlock()	Re enables tasks scheduling

5.2 Development Cycle

The main objective of task scheduler is to meet the deadline of execution of tasks and also the real time constraints [5]. The following algorithms are used for the scheduler design. Two tasks are chosen to be scheduled. Task 1 consists of opening different stored images and displaying them sequentially with a delay of 1 sec. Task 2 is to lock the scheduler and release the scheduler upon a key press.

Task 1

1. Open different images one by one in the directory
2. Display the image on the LCD
3. Delay for 1 sec
4. Go back to step2
5. End

Task 2

1. Lock the scheduler
2. Wait for the user to press a key
3. Unlock the scheduler
4. Go back to step 1
5. End

In order to implement the above two algorithms RTOS functions App_Task1Create() and App_Task2Create are used. Stack of sufficient size is created using OSTaskCreate API in order to run the task properly. SDCard_Init() to perform the SD card initialization where the images are stored.

5.3 Code Implementation for Scheduler

The following are the codes for Task creation, Initialization and ISR Code, Task 1 and Task 2.

5.3.1. Task creation

```
Void App_Task1Create(void)
{
        CPU_INT08U os_err;

Os_err = os_err;
UART0_SendString("Creating Task 1 with priority
57---\r\n");
Os_err =
OSTaskCreate((void(*)(void*))uctsk_Task1,
        (void     *)0,
        (OS_STK
        *)&App_Task1Stk[APP_TASK_STK_SIZ
E-1],
        (INT8U )APP_TASK1_PRIO );

#if OS_TASK_NAME_EN>0
```

5.3.2. Initialization and ISR code

```
Void BUTTON_init(void)
{

        LPC_GPIO2->FIODIR     &= -(1<<10);
        LPC_GPIOINT->IO2intEnf l = (1<<10);

        NVIC_EnablelRQ(EINT3_IRQn);
}
```

5.3.3. Task 1

```
Static void uctsk_Task1(void*pdata)
{

Pdata = pdata;
res = f_opendir(&Dir,"0/images");

if(res! = FR_OK)
{UART0_SendString("Unable to open dir\r\n");
While(1);
}
```

5.3.4. Task 2

```
Static void uctsk_Task2(void*pdata)
{
        While(1)
```

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
{
        UART0_SendString("Task2 is Running\r\n");

        OSSchedLOCK();

        While(UserStatus! = 1)
        {
                Delay();
                }
```

5.4. Experimental setup

In this work an LPC 1768 board and µC/OS-II operating system are used. The LandTiger V2.0 NXP LPC1768 ARM development board is a 32 bit Microprocessor and it has the features of 512KB on chip flash program memory,64KB SRAM for high performance CPU, Standard JTAG test/debug interface, Two RS 232 serial interfaces, Two CAN bus communication interfaces, RS 485 communication interface,RJ45-10/100M Ethernet network interface, DAC o/p interface and ADC i/p interface, USB 2.0 interface, SD/MMC card (SPI) interface, Color LCD display interface etc.

µC/OS-II is a real time multi tasking operating system kernel version 2. It is used for inter task communication and synchronization. It has the features like Portability, preemptive, multitasking kernel, can handle 64 tasks, supports processors up to 64 bit and has deterministic execution times. µC/OS-II is simple to use and simple to implement KERNEL. The experimental setup is shown in fig. 5.

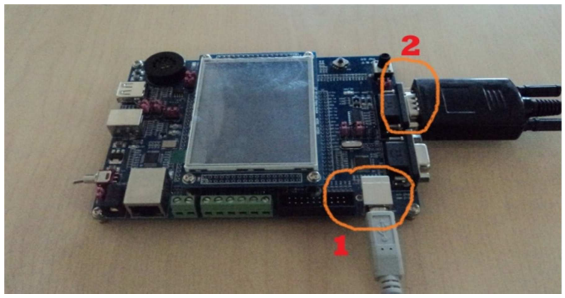

Figure 5: Experimental Setup (1) USB cable of the board connected to USB port of PC (2) Serial port of the RTOS board connected to Serial port of the PC

6. IMPLEMENTATION RESULTS

The implementation of multithreading consists of two case studies

i) Sorting of large set of numbers using threading and normal approach

ii) Dictionary sorting of alphabetical words using threading and normal approach

In both the applications, the number of threads is varied and time metric is evaluated. It is established that thread based implementation is fast particularly in sorting of alphabets. The thread is implemented in python and openMP platform. In case (i) multithread implementation of sorting of numbers is presented and menu to perform selection of threading or non threading for sorting application is shown in fig. 6.In fig. 7 two threads are selected to sort the input numbers in ascending order. The time taken by the merged output is also computed and displayed as a metric.

Figure 6: Menu to perform selection of threading or non threading for sorting application

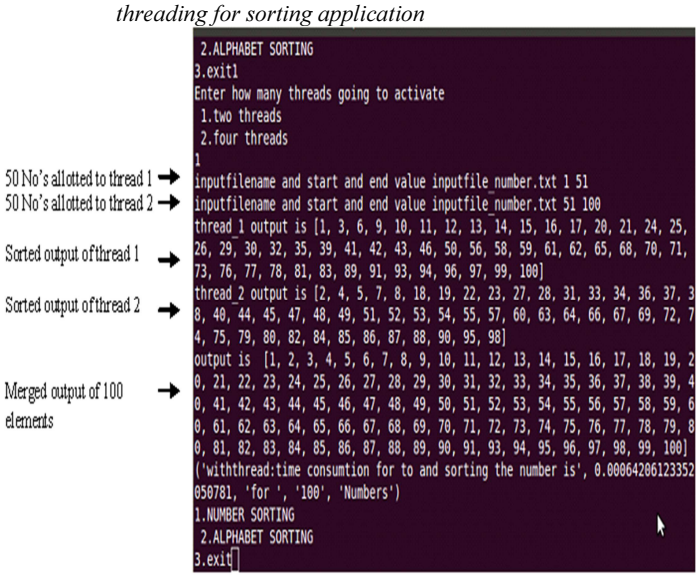

Figure 7: Sorting of numbers in ascending order using two threads

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In case (ii) Dictionary sorting of alphabetical words using threading is presented .In figure. 8 & 9 sorting of alphabets for dictionary is implemented using four threads.

Figure 8: Sorted Alphabet in progress for four threads

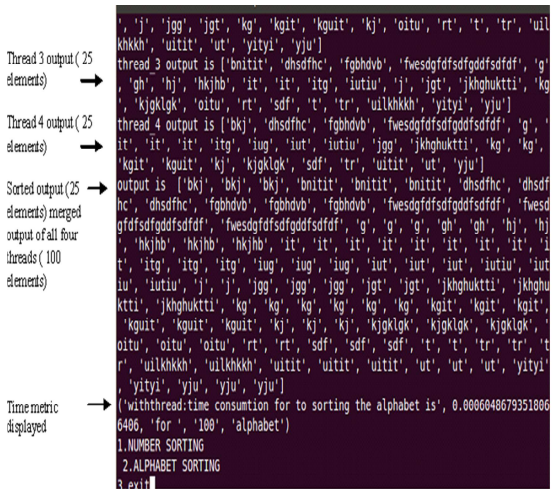

Figure 9: Continued output of Fig.8(Four thread case)

7. DISCUSSION

In [2] the author presented a optimized round robin scheduling algorithm for CPU scheduling to solve the problem encountered in simple round robin scheduler(i.e., giving equal priority to all tasks) by decreasing the performance parameters to a maximum possible extent. The results of designed scheduler performance shows improvement in reducing average waiting time and the time taken for completing the task execution is very small.

The scheduler performance in sorting of numbers using normal approach and threading is presented in fig. 10.

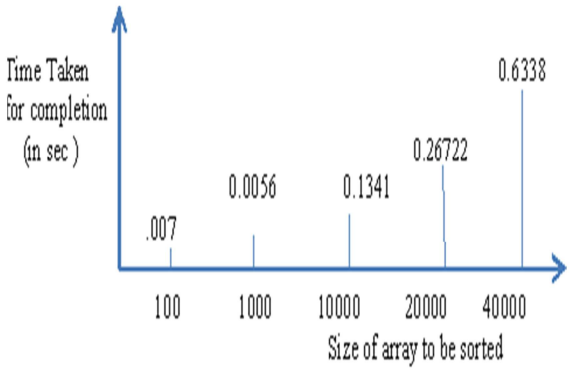

Figure 10a: Non-Threading (Non Multitasking) based implementation (sorting of numbers)

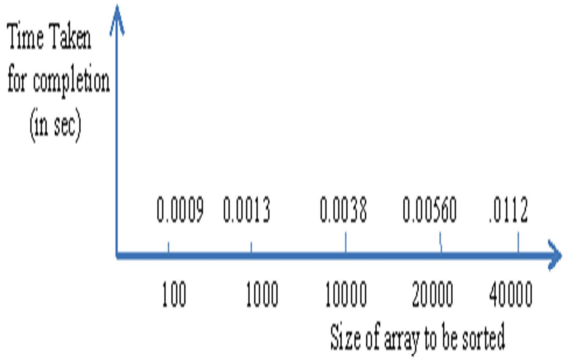

Figure 10b: Threading based implementation - Two threads (sorting of numbers)

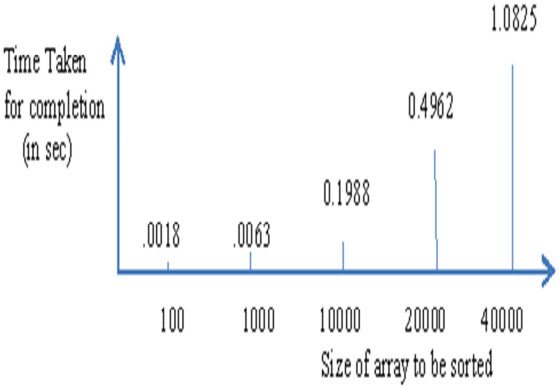

Figure 10c: Threading based implementation -Four Threads (sorting of numbers)

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The scheduler performance in sorting of alphabets using normal approach and threading is presented in fig. 11.

.

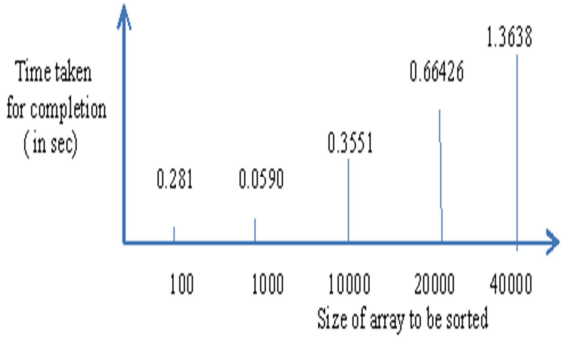

Figure 11a: Non-Threading (Non Multitasking) based implementation (sorting of alphabets)

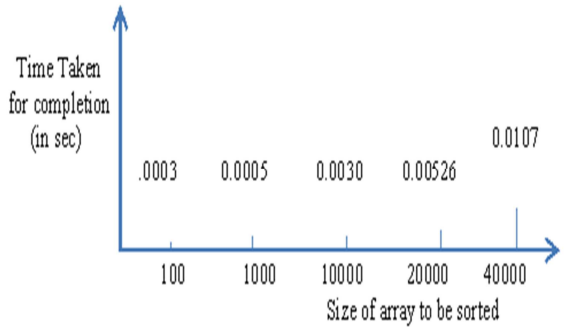

Figure 11b: Threading based implementation-Two Threads (sorting of Alphabets)

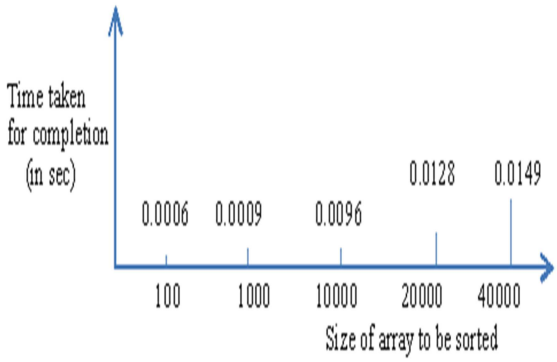

Figure 11c : Threading based implementation-Four Threads (Sorting of Alphabets)

The performance analysis of scheduler for non threading, two threads and four threads for the two cases is shown in fig. 12.

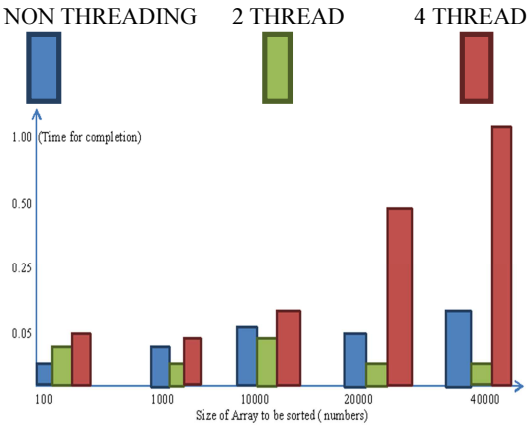

Figure 12a: Performance analysis of schedulers for sorting of numbers

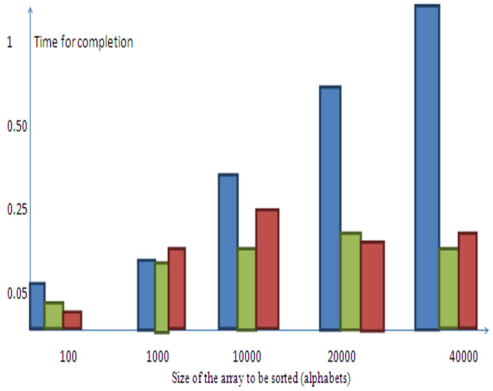

Figure 12b: Performance analysis of schedulers for sorting of Alphabets

8. CONCLUSION

Improving the CPU efficiency in real time and time sharing operating system is very important. In this study an effort is made to design a scheduler using round robin algorithm for resource awareness embedded system in real time environment. Two case studies are taken one for sorting of numbers and one for sorting of alphabets using non threading and threading concepts and the scheduler performance is obtained. The scheduler presented shows improved performance over existing methods in terms of performance criteria like Utilization, Throughput, Queuing Time, Response time etc. The main limitation of the research work is that its practical implementation requires multicore units. Future research shall include use of openMP tools to implement parallel threads more effectively and to improve the performance of the scheduler.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] Abbas noon, Ali kalakech,and Seifedine kadry, "A new round robin based scheduling algorithm for operating systems:Dynamic quantum using the mean average",*International Journal of Computer Science Issues*,Vol. 8,No. 3, 2011, pp. 224-229.

[2] Ajit singh, Priyanka goyal and Sahil batra, "An optimized round robin scheduling algorithm for CPU Scheduling",*International Journal on Computer Science and Engineering*,Vol. 2,No. 7, 2010, pp. 2383-2385.

[3] Ishwari singh rajput and Deepa gupta, "A Priority based round robin CPU scheduling algorithm for real time systems",*International Journal of Innovations in Engineering and Technology*, Vol. 1,No. 3, 2012, pp. 01-11.

[4] Jie song, Xuebing Liu,Zhiliang zhu,Dazhe Zhao and Ge Yu, "A novel task scheduling approach for reducing energy consumption of Map reduce cluster",*IET Technical review*,Vol. 3,No. 1, 2014, pp. 65-74.

[5] M.Kaladevi and Dr.S.Sathiyabama, "A comparative study of scheduling algorithms for real time task", *International journal of advances in science and technology*, Vol. 1, No. 4, 2010, pp. 8-14.

[6] Karthikeyan.V and Dr.S.Ravi, "Robust memory management using real time concepts",*Journal of computer science*, Vol. 10,No. 9, 2014, pp. 1480-1487.

[7] G.Muneeswari and K.L.Shunmuganathan, "A novel hard soft processor affinity scheduling for multicore architecture using multiagents",*European journal of scientific research*, Vol. 55, No. 3, 2011, pp. 419-429.

[8] Neeraj kumar and Nirvikar, "Performance improvement using CPU Scheduling Algorithm",*International Journal of Emerging Trends of Technology in Computer Science*,Vol. 2,No. 2, 2013, pp. 110-113.

[9] Neetu goel and Dr.R.B.Garg, "A comparative study of CPU scheduling algorithms",*International journal of graphics & image processing*, Vol. 2, No. 4, 2012, pp. 245-251.

[10]Rajkamal, "Embedded system architecture,programming and design",*Tata Mcgraw hill publishing company limited*,2003.

[11) Rakesh kumar yadav,Abhishek k mishra,Navin prakash and Himanshu Sharma, "An improved round robin scheduling algorithm for cpu scheduling",*International journal on computer science and engineering*, Vol. 02, No. 04, 2010, pp. 1064-1066.

[12] Ranjit Rajak, "A novel approach for task scheduling in multiprocessor system", *International journal of computer applications*,Vol. 44,No. 11,2013,pp. 12-16.

[13] Sukumar babu bandarupalli and Neelima priyanka nutulapati, "A novel CPU scheduling algorithm preemptive & non preemptive", *International journal of modern engineering research (IJMER)*, Vol. 2, No. 6, 2012, pp. 4484-4490.

[14] U.Syed Abudhagir and Dr.S.Shanmugavel, "A novel dynamic reliability optimized resource scheduling algorithm with fault tolerant approach for grid computing system", *Journal of Theoretical and applied information technology*,Vol.58,No.1, 2013, pp.81-89.