Volume 2, Issue 4August 2013

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

Scheduling in Multi-core Systems: Minimizing Average Waiting Time by merging (Round-Robin with Shortest-Job-First Technique)

[1]Ahmad Mohsin, [2]Muhammad Imran Rafique, [3]Sumbul Aziz Khan, [4]Qurratulain Munir

[1]Department of Computer Science & Engineering, Air University Multan, Pakistan
[2]Department of Computer Science & Engineering, Air University Multan, Pakistan
[3]Department of Computer Science & Engineering, Air University Multan, Pakistan
[4]Department of Computer Science & Engineering, Air University Multan, Pakistan

E-mail: [1]ahmadspm@gmail.com, [2]mailtomirc@yahoo.com, [3]sumbulkhan2419@yahoo.com, [4]anniemunir57@yahoo.com

ABSTRACT

Since the emergence of Multi-core processors, operating systems have transformed altogether; trying to meet the resource capabilities and improve upon the overall performance of the systems. To date, many core processors have emerged, revolutionizing the computation capabilities. There are many performance related issues with the operating systems; one of them is proper and efficient scheduling of processes and threads which impacts heavily on overall Quality of the system. In past, much work has been done in devising new scheduling algorithms for multi-core processors but little attention has been given to merge classic scheduling algorithms for multi-core processor systems. In this paper, we shall focus how Round Robin (RR) algorithm can work with Shortest Job First (SJF) algorithm and First-Come-First-Serve (FCFS) for multi-core processor systems. We have tried to devise a new algorithm by using prioritization techniques for multi-core processors. Observations are made on the basis of these results and are revealed at the end. By this technique, overall waiting time can be reduced significantly which eventually leads to the better performance of the system.

Keywords: *multi-core processors, uni-processor, scheduling, FCFS, SJF, round robin (RR), throughput, waiting time*

1. INTRODUCTION

In the current age of multi-processors, every operating system is dreamed to be optimized to get maximum throughput in minimum processing time and waiting time for each process [1, 2]. Round Robin (RR) algorithm works on assigning the CPU to every available process in turn so that no process goes into starvation [3].

The CPU can be made more productive by reducing the idle time of the processors and switching and assigning it to any other process that is in waiting state. This idle time can be minimized by scheduling all processes using Round Robin (RR) such that all processes have equal access to the CPU but each process is assigned a certain time (i.e. time slice) to complete its execution. Previously, the Round Robin algorithm is working with FCFS algorithm, that is, the process arriving first is served or processed first by the processor; but a slight contrast to FCFS, i.e. the process arriving first is assigned higher priority and sent to the CPU first but given only a particular time to execute (RR-technique) after which it goes into waiting state and the CPU is given to the next process and so on [3, 4]. This context switching continues until all the available processes are completed one by one.

In the next section-2, we'll explain how the scheduling operations are performed by a process scheduler, enabling us to define a certain scheduling criteria in section-3. In section-4, the drawbacks of Round Robin (RR) with First-Come-First-Serve technique are discussed. Section-5 introduces our proposed idea of prioritizing the incoming processes using Round Robin with Shortest-Job-First technique. Section-6 shows our anticipated *Priority Algorithm* and the experiments relating to RR-FCFS and the RR-SJF using arbitrary values for Uni-processors and multi-processors separately. Finally section-7 & section-8 gives conclusion and the future work respectively related to the proposed idea.

2. PROCESS SCHEDULER

The process scheduler is the pre-emptive component of the operating system that is responsible for deciding whether the currently running process should continue running, move to ready or waiting queue and, which process should be sent to processor for execution [5]. The process scheduler selects an available process from a set of several available processes and sends it to the CPU. For a multiprocessor system, there may be more than one running process simultaneously. In multi-core processor systems scheduling is complex as compared to uni-processors. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

The following four steps may occur during the scheduler‘s decision making: [6]

- The current process that is running moves from the *running* to the *waiting* state because of an I/O request or some other kind of interrupt.
- The current process terminates.
- A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.
- An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the *ready* state. The scheduler may then decide to move this *ready* process into the *running* state.

There are many algorithms available for process scheduling, some of them discussed in this paper are:

- Round Robin (RR)
- First-Come, First-Served (FCFS)
- Shortest Job First (SJF)

3. THE SCHEDULING CRITERIA

Scheduling criteria is one of the most important factors determining which CPU-Scheduling Algorithm is best to use in a particular situation meeting certain properties. There are many scheduling criteria suggested but following is the mostly used one. [3]

Sr. #	Criterion Factor	Description
1	CPU Utilization	Ideally 0- 100 % In reality 40%
2	Throughput	Maximum Amount of CPU- being busy for executing processing.
3	Turnaround Time	Total time from time of submission to the time of completion of a process. Tr = Ts + Tw (Ts = Execution time, Tw = Waiting Time)
4	Waiting Time	Amount of time Process spends in waiting in Ready Queue
5	Response Time	Measure of time from submission of request to the first response received. It is variable depending upon environment.

Table-1: Scheduling criteria

3.1. DESIRABLE FACTORS EFFECTING SCHEDULING CRITERIA

Some of the characteristics of the desirable factors effecting scheduling criteria are:

Throughput	→ Higher is desirable
Turnaround time	→ lower is desirable
Response Time	→ Lower is desirable

These factors are affected by following two secondary criteria:

- CPU Utilization
- Waiting Time

These are multiple factors which are useful to determine the performance of a scheduling algorithm. Here in our case, we shall be focusing on Average Waiting Time, as this factor is most important for processes / threads when they are in ready queue waiting for their turn to be assigned to the processor(s).

4. HOW ROUND ROBIN (RR) WORKS WITH FIRST-COME-FIRST-SERVE (FCFS)?

In FCFS algorithm, processes are prioritized and dispatched to the processor according to their arrival time in the ready queue, regardless of their size [7, 8]. So, if the short jobs arrive after the large ones, then this short job has to wait a long time to get its first response and may even go into starvation if the larger process never ends. This drawback is overcome by using Round Robin and FCFS together as the algorithm for RR and FCFS are same except for the presence of time quantum or time slice. Therefore, RR which is preemption of processes based on a clock (called time slice) is used as it is one of the oldest, simplest and easiest scheduling algorithm. The time slice interrupts at periodic intervals. When the interrupt occurs, the currently running process is placed in the ready queue, and the next ready job is selected on a FCFS basis.

5. ROUND ROBIN (RR) WITH SHORTEST-JOB-FIRST (SJF)

According to the proposed idea, the Round Robin is merged with Shortest Job First (SJF) instead of FCFS, i.e. when the time slice assigned to a process end, the next available SHORTEST sized process is selected for execution. In this technique, the process having the shortest CPU burst time will be assigned to the CPU first, i.e. the available processes are arranged on the basis of their required service time, the smallest process or job is assigned with priority-1, the next available shortest process as priority-2, and so on in Shortest job First (SJF) algorithm

Volume 2, Issue 4August 2013

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

[9] and these prioritized processes execute using Round Robin technique. Hence this technique minimizes average waiting time and reduces response time for short processes. For the sake of simplicity this paper works on processes rather than threads.

6. EXPERIMENTS & RESULTS

6.1. Working on Uni-Processor Systems

To evaluate the performance of our proposed idea, we have used some arbitrary values of five processes namely; P1, P2, P3, P4 and P5 arriving at time 0 in the uni-processor system. The table-2 shows the processes and their burst time:

Process	P1	P2	P3	P4	P5
Burst Time (msec)	20	12	8	16	4

Table-2: Five processes along-with their burst time

The time quantum or Time slice assigned in this example will be 4 msec.

a. RR with FCFS:

In normal Round Robin algorithm working with FCFS, the processes given in Table-1 will have access to the CPU according to their arrival priority (First-Come-First-Serve) and the CPU will switch between all processes giving each process a defined time slice of 4 milliseconds (Round-Robin). The situation is shown in figure-1.

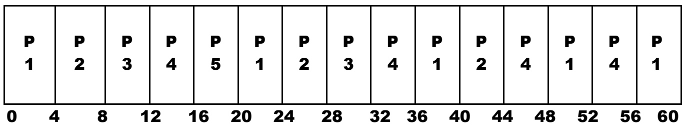


Figure-1: Gantt chart showing waiting time for each process using RR-FCFS

The average waiting time according to Round-Robin with FCFS algorithm will be calculated as follows:

P1 = 16 + 12 + 8 + 4 = 40
P2 = 4 + 16 + 12 = 32
P3 = 8 + 16 = 24
P4 = 12 + 16 + 8 = 36
P5 = 16

Therefore, the Average waiting time with RR-FCFS will be:

40 + 32 + 24 + 36 + 16 / 5 = **29.6 msec.**

b. RR with SJF

According to our proposed idea of merging Round Robin with SJF algorithm, the smallest process will have the highest priority and will be assigned the CPU first; then the next smallest process will be assigned the CPU and so on i.e. the sequence of processes in table-1 will be P5, P3, P2, P4, P1 for round-1. After giving response to all processes, the scheduler will then selects the same processes sequence as in round-1 as shown in figure-2. In general, the scheduler performs SJF to arrange the processes and performs RR for the entire processor queue.

The Gantt chart of all processes discussed in the above scenario is shown figure-2. All 5 processes are given processor queue according to their burst time in round-1 and in round-2 the same sequence is preserved as in round-1. The time quantum or Time slice assigned in this example is 4 milliseconds for the processes shown in table-1.

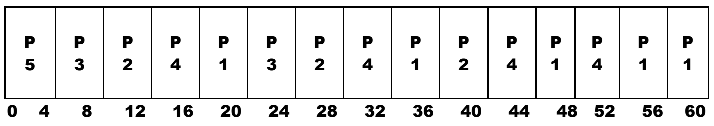


Figure-2: Gantt chart showing waiting time for each process using RR-SJF

The average waiting time according to Round-Robin with SJF algorithm will be calculated as follows:

$P_1 = 16 + 12 + 8 + 4 = 40$
$P_2 = 8 + 12 + 8 = 28$
$P_3 = 4 + 12 = 16$
$P_4 = 12 + 12 + 12 + 4 = 40$
$P_5 = 0$

Now, the Average waiting time with RR-SJF will be:

40 + 28 + 16 + 40 + 0 / 5 = **24.8 msec.**

This is obvious from both the results that the average waiting time for RR-FCFS is 29.6 milliseconds and for RR-SJF, it is 24.8 milliseconds which is a smaller value according to our proposed idea. Comparison of both techniques is shown in figure-3.

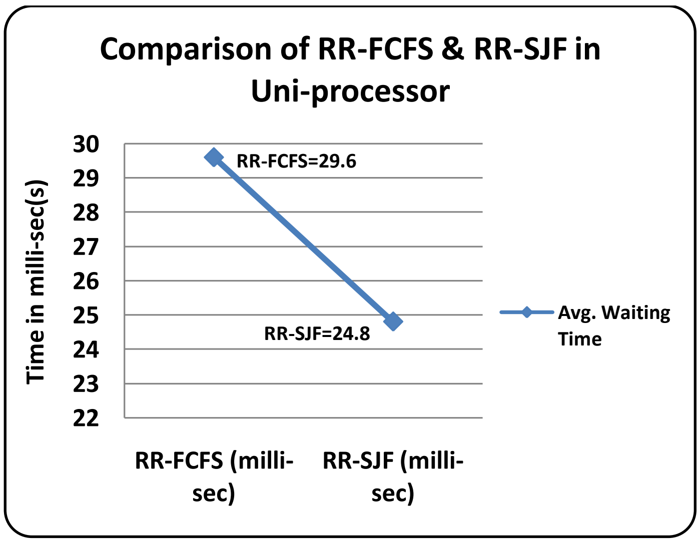


Figure-3: Graph showing the comparison between RR-FCFS & RR-SJF for Uni-processor

Volume 2, Issue 4August 2013

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

6.2. Working on Multi-Processor Systems

If we are working on multi-processors then there is a need of changing the technique and also a process can arrive at any time and there is a need to define its position with respect to its burst time in a queue in order to mimic SJF algorithm. In this proposed idea, the Operating System first waits for some time, say 1-second, and makes a batch of those processes in queue Q_1 (figure-4) --- say, maximum process in a queue Q1 would be 5. This batch would be assigned to another queue Q2 (figure-4) which will mark priority no. according to their burst time, i.e. shortest process will have the highest priority and these processes will be placed in the ready queue PQ (figure-4) of those processor who is running with less no. of processes and the processor queue PQ will be executed in the same fashion as discussed previously in working with uni-processor in part b of this section. These are shifted to Q2 where they are given priority no. and are finally assigned to processor's ready queue PQ.

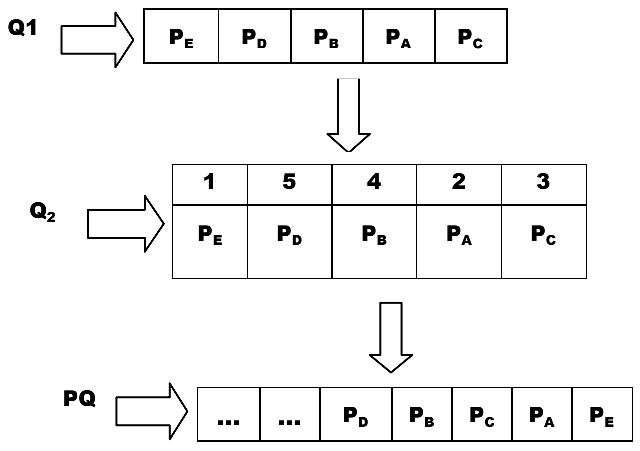


Figure-4: Multiple Queues with multi-core processors

6.3. Priority Algorithm:

A *Priority* algorithm can be designed to mark the priority of each new process entering into the system. Every time a new process arrives, its burst time is compared with the processes already residing in the queue; if the new process is found to have a smaller burst time than the older ones then it is given the higher priority. The following *Priority Algorithm* is given to mimic the whole scenario discussed before. In this algorithm, there are two arrays one of which is **B**, which is burst array that contains the burst time of processes and the other is **P,** which is priority array that contains priority numbers of corresponding burst time of respective processes. Both the arrays have length 5. The burst array is initially set to 0, 1, 2, 3, 4, which shows that process P1 has priority 0 and so on. And after executing the algorithm, the priority array i.e. P is updated with respect to burst time in the burst array i.e. B in such a way that the process which has smaller burst time is assigned with the highest priority and so on. For example if we have five processes in the burst array with burst time as 25, 20, 10, 15, 30, then its corresponding priority array will be 2, 3, 1, 0, 4.

The proposed Priority Algorithm is shown below:

```
Priority (B, P) /* B is Burst Array showing processes
Burst time and P is Priority Array initialized with
(0, 1, 2, 3, 4) default priorities */
    1.  k ← 0
    2.  for i = 0,3
    3.  do k ← i+1
    4.        for j = 1 down to 0
    5.        do if (B[k] ≤ B[j])
    6.      then P[k] ← P[k] – 1; P[j] ← P[j] + 1
```

After analysing the algorithm, it is obvious to see that line no.6 depends upon the condition given in line no.5, which compares the burst time of an arriving process with the rest of the processes. If the burst time of an arriving process is greater than the burst time of an already arrived process being compared then line no.6 is not executed.

Also, the flowchart explaining the algorithm is given in figure-5.

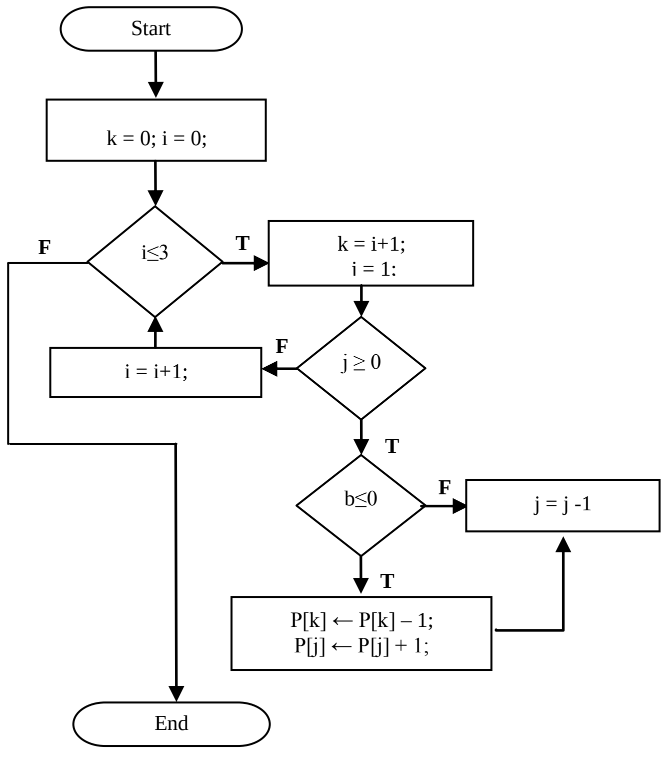


Figure-5: Flowchart for the Priority Algorithm

Now we look at the following two cases:

a. RR with FCFS:

Let suppose we are working with a system comprising of two cores; processor-1 and processor-2, the processes P1

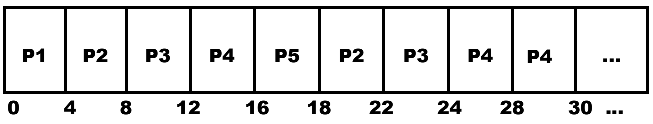
Volume 2, Issue 4August 2013

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

to P5 are assigned to PQ1 and P6 to P10 are assigned to PQ2 (Figure-6).

PQ1 in Processor-1:



PQ2 in Processor-2:

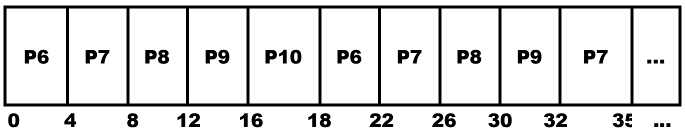


Figure-6: Gantt chart showing waiting time for each processor PQ1 & PQ2 using RR-FCFS in Multi-processors

For example, the burst time of 10 arriving processes in the system is given below:

Process	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Burst time (msec)	4	8	6	10	2	8	12	8	6	2

Table-3: Processes arrival time

Processes P1 to P5 are assigned to processor-1's queue PQ1 and processes P6 to P10 are assigned to processor-2's queue PQ2. The time slice assigned is again 4 milliseconds. Now, the average waiting time of PQ1 of processor-1 according to RR-FCFS technique for processes P1 to P5 will be:

P1 = 0
P2 = 4 + 10 = 14
P3 = 8 + 10 = 18
P4 = 12 + 8 = 20
P5 = 16

Average Waiting time = 0 + 14 + 18 + 20 + 16 / 5
= **13.6msec.**

Similarly, the average waiting time of each process in PQ2 of processor-2 according to RR-FCFS technique for processes P6 to P10 will be:

P6 = 14
P7 = 4 + 14 + 6 = 24
P8 = 8 + 14 = 22
P9 = 12 + 14 = 26
P10 = 16

Average Waiting time = 14 + 24 + 22 + 26 + 16 / 5
= **20.4 msec.**

b. RR with SJF

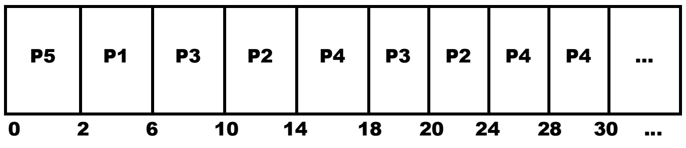
Now the same arbitrary values used above to test RR-FCFS are now considered to check RR-SJF according to our ***Priority Algorithm***. Again, processes P1 to P5 arrives in queue Q1, say, in 1 second and are shifted to Q2 to mark priority no. according to their burst time and are then transferred to PQ1of processor-1 in such a way that the process having the highest priority will be placed first in the queue. After assigning processes P1 to P5 to PQ1, processes P6 to P10 come in Q1 and are then shifted to Q2 to mark priority no. and then finally moved to PQ2 of processor-2. Remember that priority number is just to define the position of processes in processor's queue and after placing processes in PQ according to their priority no. the processor queue PQ operates sequentially. Figure-7 shows the complete scenario.

The average waiting time of each process in PQ1 of processor-1 according to RR-SJF technique will be:

P1 = 2
P2 = 10+ 6 = 16
P3 = 6 + 8 = 14
P4 = 14 + 6 = 20
P5 = 0

Average Waiting time = 2 + 16 + 14 + 20 + 0 / 5
=**10.4msec.**

PQ1 in Processor –1:



PQ2 in Processor –2:

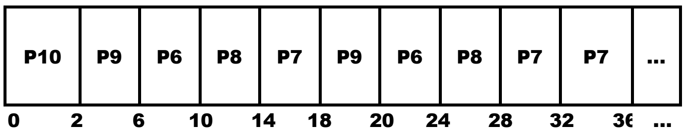


Figure-7: Gantt chart showing waiting time for each processor PQ1 &PQ2 using RR-SJF in Multi-processors

Similarly, the average waiting time of each process in PQ2 of processor-2 according to RR-SJF technique will be:

P6 = 6 + 10 = 16
P7 = 14 + 10 = 24
P8 = 10 + 10 = 20
P9 = 2 + 12 = 14
P10 = 0

Average Waiting time = 16 + 24 + 20 + 14 + 0 / 5
= **14.8msec.**

Volume 2, Issue 4August 2013

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

After computation, it is obvious from the technique that RR-SJF is giving a less waiting time in comparison with RR-FCFS, which is given below:

Processors	**RR-FCFS**	**RR-SJF**
Processor-1	13.6 msec	10.4 msec
Processor-2	20.4 msec	14.8 msec

Table-4: Average Waiting Time (RR-FCFS & RR-SJF)

The comparison of all these results obtained with RR-FCFS and RR-SJF in multi-processors is shown with the help of a graph in figure-8.

7. CONCLUSION

The experimental results clearly show that the value of average waiting time for our proposed Shortest-Job-First has always a less value with RR algorithm as compared to First-Come-First-Serve with RR in both cases of uni-processor and multi-processors.

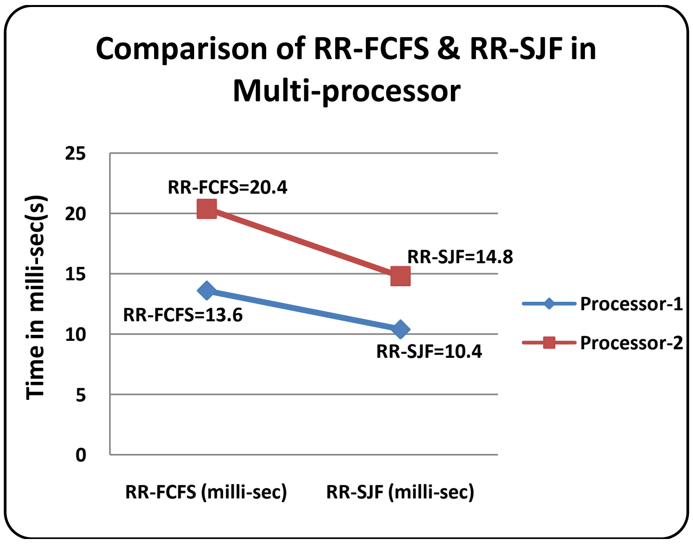


Figure-8: Graph showing the comparison between RR-FCFS & RR-SJF for Multi-processors

Thus, by using the proposed scheduling technique, the average waiting time of the processes will be decreased. The proposed system will contain a scheduling algorithm that will prioritize processes residing in the memory. The setup speed of processes can be slow because of operating priority algorithm but after the setup the execution speed of processes will increase because of having less average waiting time. As we had discussed earlier that average waiting has a greater impact on other scheduling criterion factors these results support our new algorithm.

The proposed idea of using RR with SJF can be implemented practically in future on all kind of systems including Linux, Windows, etc. to reduce the average waiting time of the processes and sometimes getting a maximum throughput. And in multi-level queue systems, a separate queue can also be maintained using this technique.

8. FUTURE RESEARCH WORK

In our work we have tried to give a new mechanism for improving the overall performance of the operating system by reducing average waiting time of the processes (s) in the ready queue. We have done this both for uni and multi- core processors. Work can further be extended by applying this basic technique to threads in uni- processors as well multi-core processors We have check our algorithm in C++ compiler for measuring average waiting time. Moreover, it may further be extended to POSIX API for simulation and results for multi-core processors with multithreading evaluation.

REFERENCES

[1] I. Informatyka, P. Poznanska, Poznan and Poland, “Scheduling Multiprocessor tasks- An overview”, European Journal of Operational Research 94, page 215—230, 1996.

[2] G. Levin, C. Sadowski, I. Pye, S. Brandt, “A Simple Model for Understanding Optimal Hard Real-Time Multiprocessor Scheduling”. Technical Report UCSC-SOE-11-09, May 26, 2009.

[3] A. Silberschatz, P. B. Galvin and G. Gagne, 2009, Operating Systems Concepts, 8th ed., John Wiley & Sons, Inc. 183-222 p.

[4] W. Stallings, Operating Systems – Internals and design Principles, 7th ed., Pearson Education, Inc., Prentice Hall, 1 Lake Street, Upper Saddle River, New Jersey, America.

[5] S. Siddha, V. Pallipadi, A. Mallick, “Process Scheduling Challenges in the Era of Multi-core Processors”, Intel Technology Journal, vol. 11, issue 4, 2007.

[6] Pk.org website. Available: http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html, 2010.

[7] C. Faisstnauer, D. Schmalstieg, W. Purgathofer, “Priority Round-Robin Scheduling for Very large

Virtual Environments", Vienna University of Technology, Austria, May 30, 2000.

[8] D. Nayak, S. K. Malla, D. Debadarshini, "Improved Round Robin Scheduling using Dynamic Time Quantum", International Journal of Computer Applications (0975 – 8887), vol. 38 - No. 5, January, 2012.

[9] H. S. Behera, B. K. Swain, "A New Proposed Precedence based Round Robin with Dynamic Time Quantum (PRRDTQ) Scheduling Algorithm For Soft Real Time Systems", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, issue 6, June, 2012.

AUTHORS' PROFILES

Ahmad Mohsin is a faculty Member of Department of Computer Sciences and Engineering, Air University Multan campus. He has done MS from FAST NUCES Lahore with distinction and BS (Hons.) Degree in Computer Sciences from BZU Multan. His Research interests are Software Requirements Engineering, Software Design and Cloud-based applications.

Muhammad Imran Rafique received BS degree in Computer Science from Bahauddin Zakariya University, Multan Pakistan, in 2007. He is MS-CS student of Air University, Multan campus. Currently, he is Computer Science teacher in Secondary School Education department. His research interests are Operating system, Management Information System.

Sumbul Aziz Khan received Masters Degree in computer science from Bahauddin Zakariya University, Multan Pakistan, in 2008. She is MS-CS student of Air University, Multan campus. Currently, she is Computer Science teacher in Secondary School Education department. Her research interests are Data mining, Operating system and Software Engineering.

Qurratulain Munir received Masters Degree in computer science from Bahauddin Zakariya University, Multan Pakistan, in 2008. She is MS-CS student of Air University, Multan campus. Currently, she is Computer Science teacher in Secondary School Education department. Her research area is Software Engineering.