Volume 4, Issue 2, February 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm

Abdulrazaq Abdulrahim*
Department of Mathematics, Ahmadu Bello University, Zaria, Nigeria

Salisu Aliyu
Department of Mathematics, Ahmadu Bello University, Zaria, Nigeria

Ahmad M Mustapha
Department of Computer Engineering, University of Maiduguri,Nigeria

Saleh E Abdullahi
Department of Mathematics, Ahmadu Bello University, Zaria, Nigeria

Abstract- Multiprogramming is an important aspect of operating systems (OS); it requires several processes to be kept simultaneously in the memory, the aim of which is maximum CPU utilization. Among other CPU scheduling algorithms, like the First Come First Serve (FCFS), Shortest Job First (SJF) and Priority Scheduling (PS); Round Robin is considered the most widely used scheduling algorithm in time sharing and real time OS for allocating the CPU to the processes in the memory in order to achieve the aim mentioned above. This paper proposed a more improvement in the Round Robin CPU scheduling algorithm by improving the algorithm by Manish and AbdulKadir. By experimental analysis, this proposed algorithm performs better than the simple Round Robin and the Improved Round Robin CPU scheduling algorithms in terms of minimizing average waiting time, average turnaround time and number of context switches.

Keywords: operating system, multiprogramming, CPU utilization, CPU scheduling algorithm, Round Robin.

I. INTRODUCTION

An operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the system hardware. One of the most important aspects of operating systems is the ability to multiprogram. Multiprogramming requires several processes to be kept simultaneously in the memory, the aim of which is maximum CPU utilization. If these several processes in the memory are ready to run at the same time, the system must choose among them, which process to run. Making this decision is CPU scheduling. CPU scheduling is the basis of multiprogramming systems. It refers to a set of policies and mechanisms to control the order of work to be performed by a computer system. It is made by the part of the operating system called the scheduler, using a CPU scheduling algorithm [7].

II. CPU SCHEDULING ALGORITHMS

There are many different CPU scheduling algorithms. Some basic CPU scheduling algorithms are listed below:

A. *First-Come First-Serve (FCFS)*

It is by far the simplest CPU scheduling algorithm. The implementation of this algorithm is easily managed with a First-In-First-Out (FIFO) queue. When a process enters the ready queue, it is inserted onto the tail (rear) of the ready queue and when the CPU is free, the process to be executed next is removed from the head (front) of the queue. The CPU is allocated to the processes on the basis of their arrival at the queue. It is simple and has low overhead. But long CPU-bound processes may dominate the CPU and may force shorter processes to wait for long periods, which minimizes the average CPU utilization or average throughput [4].

B. *Shortest-Job-First (SJF)*

This algorithm associates with each process the length of the process's next CPU burst time. When the CPU is available, it is assigned the process that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.
It gives the minimum average waiting time and minimum average turnaround time for a given set of processes [4]. But, it is difficult to know the length of the next CPU burst that's why it cannot be implemented. Long running jobs may starve, because the CPU has a steady supply of short jobs.

C. *Priority Scheduling (PS)*

Priority scheduling is a more general case of SJF. In which each process is assigned a priority and the process with the highest priority gets CPU allocated to it first. Equal-priority processes are scheduled in FCFS order. It has a good response for the highest priority processes. But, it suffers from a major problem known as indefinite blocking, or starvation, in which a low-priority task can wait forever because there are always some other jobs around that have higher priority.

D. *Round Robin Scheduling (RR)*

This is designed for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to the switch between processes. A small time unit called the time quantum or time slice is defined. The ready queue is maintained as a circular queue. The CPU scheduler goes round the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement the Round Robin scheduling, we keep the ready queue as a First-In-First-Out (FIFO) queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.
One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running burst is longer than 1 time quantum, the timer goes off, and will cause interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. This algorithm is effective in a general-purpose, times-sharing system or transaction-processing system. It gives fair treatment for all the processes and the CPU overhead is low. But, care must be taken in choosing quantum value because throughput will be low if time quantum is too small or high.

III. SCHEDULING CRITERIA

The various CPU scheduling algorithms have different properties as mentioned above. The choice of a particular algorithm may favor one class of processes over another. For selection of an algorithm for a particular situation, we must consider properties of various algorithms [2].
Many criteria have been suggested for comparing CPU scheduling algorithms. Whose characteristics are used for comparison and can make a substantial difference in which algorithm is judged to be the best. The criteria include the following:

1. Context Switch: This is process of storing and restoring context (state) of a preempted process, so that execution can be resumed from same point at a later time. It is usually computationally intensive, lead to wastage of time and memory, which in turn increases the overhead of scheduler, so the design of operating system is to optimize only these switches, the goal is to minimize it.
2. Throughput: This is defined as number of processes completed per unit time. Context switching and throughput are inversely proportional to each other.
3. CPU Utilization: This is a measure of how much busy the CPU is. Usually, the goal is to maximize the CPU utilization.
4. Turnaround Time: This refers to the total time which is spend to complete the process and is how long it takes the CPU to execute that process. The time interval from the time of submission of a process to the time of completion is the turnaround time.
5. Waiting Time: This is the total time a process has been waiting in ready queue. The CPU scheduling algorithm does not affect the amount of time during which a process executes or does input-output; it affects only the amount of time that a process spends waiting in ready queue.
6. Response Time: It is the time from the submission of a request until the first response is produced. So the response time should be low for best scheduling.

So we can conclude that a good scheduling algorithm for real time and time sharing system must possess following characteristics [1]:

- Minimum context switches.
- Maximum CPU utilization.
- Maximum throughput.
- Minimum turnaround time.
- Minimum waiting time.

Due to a number of disadvantages these scheduling algorithms have, they are severely used except Round Robin scheduling in timesharing and real time operating system; and considered most widely used CPU scheduling algorithm [1] [2].

IV. RELATED WORKS

Over a period of time, researchers have developed a number of CPU scheduling mechanisms which have been used for predictable allocation of CPU. Some of the important works are listed below.

[1] Developed an algorithm and proved the experimental results of its performance over simple Round Robin scheduling algorithm. This algorithm reduces the number of context switching, average waiting time and average turnaround time. The algorithm performs by allocating the CPU to every process, at the same time by applying Round Robin scheduling with an initial time quantum (say k units). After completing first cycle, it doubles the initial time quantum (2k units); selects the shortest process from the waiting queue and assign the CPU to it and after that, it selects the next shortest process for execution by excluding the already executed process. After completing this cycle, if any process remains in the ready queue after doubling the time quantum, it will half the doubled time quantum (i.e. the initial time quantum) and apply it to the processes in the ready queue. And again, doubles the time quantum if any process remains in the ready queue. This algorithm assumes that all processes arrive at the time in the ready queue.
[2] developed, analyzed the operation and performance of the Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems over the simple Round Robin scheduling algorithm and simple priority scheduling algorithm. The algorithm is more efficient because it has less average waiting time, average turnaround time and number of context switches as compared to simple Round Robin, in turn reducing the operating system overhead and dispatch latency. Also, it reduces the problem of starvation as the processes with less remaining CPU burst time are assigned with the higher priorities and are executed first in the second round of algorithm. [3] Developed an algorithm that describes an improvement in Round Robin scheduling algorithm. After improvement in Round Robin scheduling algorithm, it has been found that the waiting time and turnaround time have been reduced drastically. It works by reallocating the process to CPU if the remaining CPU burst of the process is less than the time quantum.

V. PROPOSED ALGORITHM

The proposed algorithm (AAIRR) focuses on improving more on the improved Round Robin CPU scheduling algorithm by [3]. The algorithm by [3] reduces the waiting time and turnaround time drastically compared to the simple Round Robin scheduling algorithm. This proposed algorithm works in a similar way as [3] but with some modification. It works in three stages:

Stage 1: It picks the first process that arrives to the ready queue and allocates the CPU to it for a time interval of up to 1 time quantum. After completion of process's time quantum, it checks the remaining CPU burst time of the currently running process. If the remaining CPU burst time of the currently running process is less or equal to 1 time quantum, the CPU is again allocated to the currently running process for remaining CPU burst time. In this case this process will finish execution and it will be removed from the ready queue. The scheduler then proceeds to the next shortest process in the ready queue. Otherwise, if the remaining CPU burst time of the currently running process is longer than 1 time quantum, the process will be put at the tail of the ready queue.

Stage 2: The CPU scheduler will then select the next shortest process in the ready queue, and do the process in stage 1.

Stage 3: For the complete execution of all the processes, stage 1 and Stage 2 have to be repeated.

Following is the pseudo code of the proposed AAIRR CPU scheduling algorithm

```
Step 1: START

Step 2: Make a ready queue of the Processes say REQUEST.

Step 3: Do

Step 4: Pick the first process that arrives to the ready queue and allocate the CPU to it for a time interval of up to 1 time quantum.

Step 5: If the remaining CPU burst time of the currently running process is less or equal to 1 time quantum
    • Reallocate the CPU again to the currently running process for the remaining CPU burst time. After completing the execution, remove it from the ready queue.
    • Otherwise, remove the currently running process from the ready queue REQUEST and put it at the tail of the ready queue.

Step 6: Pick the next shortest process from the ready queue and allocate the CPU to it for a time interval of up to 1 time quantum then go to step 5.

Step 7: WHILE queue REQUEST in not empty

Step 8: Calculate Average Waiting Time, Average Turnaround Time and Number of Context Switch.

Step 9: END
```

The flowchart for proposed algorithm is shown in Figure 1 below.

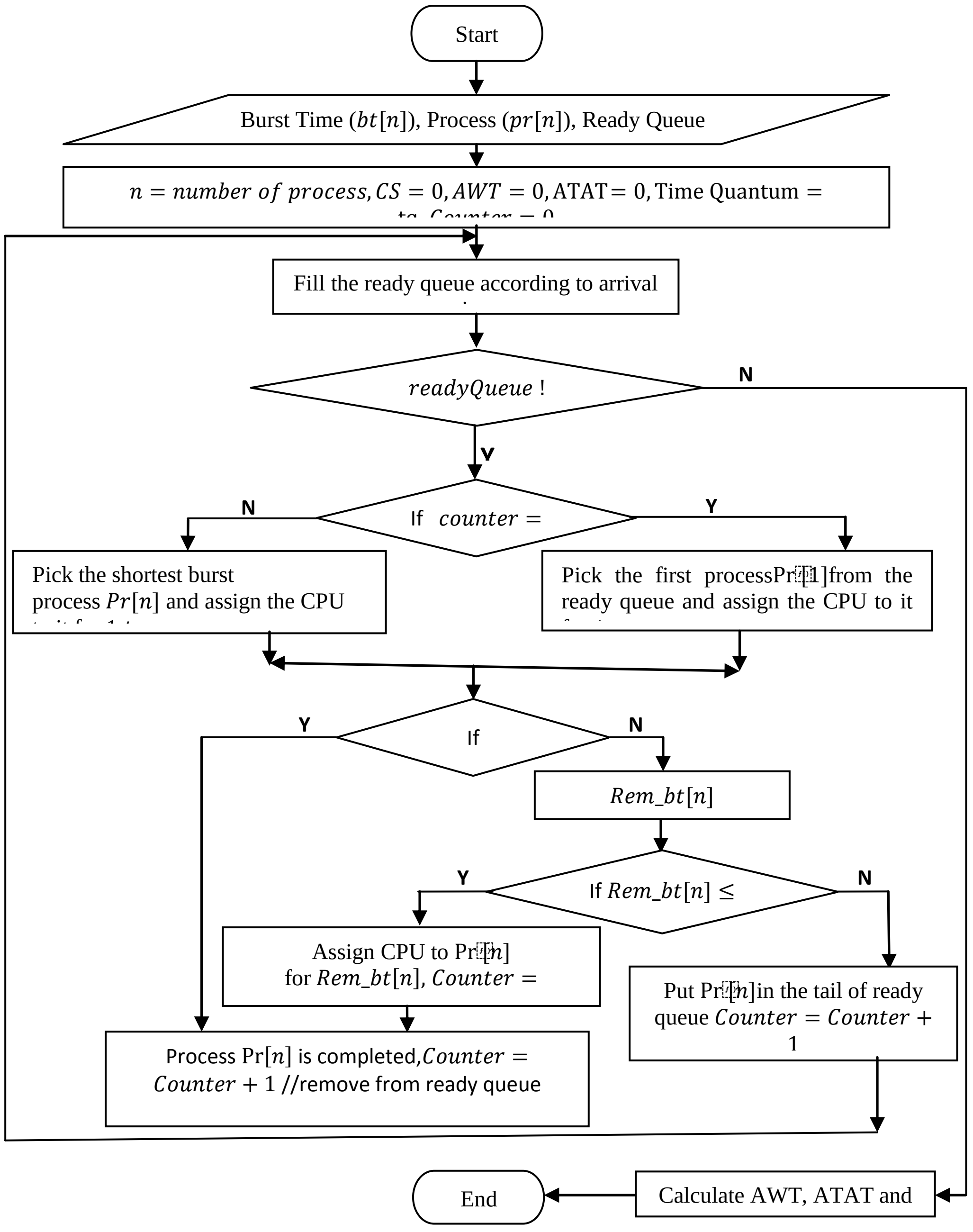
Start
Burst Time ($bt[n]$), Process ($pr[n]$), Ready Queue
$n = number\ of\ process, CS = 0, AWT = 0,$ ATAT= 0, Time Quantum =
Fill the ready queue according to arrival
$readyQueue$!
N
Y
If $counter =$
N
Y
Pick the shortest burst process $Pr[n]$ and assign the CPU
Pick the first process Pr[1] from the ready queue and assign the CPU to it
Y
If
N
$Rem\_bt[n]$
Y
If $Rem\_bt[n] \leq$
N
Assign CPU to Pr[n] for $Rem\_bt[n]$, $Counter =$
Put Pr[n] in the tail of ready queue $Counter = Counter + 1$
Process Pr[n] is completed, $Counter = Counter + 1$ //remove from ready queue
End
Calculate AWT, ATAT and

Figure 1: Flowchart for the proposed algorithm

VI. ILLUSTRATION:

This uses the example in [3]. Considering a ready queue with four processes P1, P2, P3 and P4 arriving at time 0 with burst times 15, 7, 28 and 20 respectively. Time quantum (TQ) been assumed was 10milliseconds (*ms*). Our proposed AAIRR CPU scheduling picks the first process P1 from the ready queue and allocates the CPU to it for a time interval of 10*ms*. After executing P1 for 10 ms, the remaining CPU burst of P1 is 5*ms*. Since the remaining CPU burst time of P1 is less or equals to the TQ, CPU will be reallocated to P1 for a time interval of 5*ms*. P1 has finished execution; it will be removed from the ready queue. The shortest process in the ready queue will be the next to be allocated the CPU, and that is P2 with 7*ms* CPU burst time. CPU will be allocated to P2 for a time interval of 7*ms*. P2 will finish execution and it will be removed from the ready queue. Next shortest process in the ready queue is P4 with 20*ms* CPU burst time. CPU will be allocated to P4 for a time interval of 10*ms*. Since the remaining CPU burst time of P4 is less or equal to the TQ, CPU will be reallocated to P4 for the remaining burst time i.e. 10*ms*. P4 will finish execution and it will be removed from the ready queue.

Next shortest process in the ready queue is P3 with 28 ms CPU burst time. CPU will be allocated to P3 for a time interval of 10*ms*. Since the remaining CPU burst time of P3 is not less or equal to the TQ, the CPU is supposed to be allocated to the next shortest process in the ready queue, but no other process remains in the ready queue. So, the CPU will be reallocated to P3 for the remaining burst time. P3 will finish its execution and removed from the ready queue.

The waiting time is 0*ms* for P1, 15*ms* for P2, 42*ms* for P3 and 22*ms* for P4, The average waiting time is 19.75*ms*. Using the same set of process with same arrival and CPU burst times, the average waiting time is 30.25 and 24.25*ms* in RR and IRR respectively. The average turnaround time is 37.25*ms* in AAIRR, 41.75*ms* in IRR and 47.75*ms* in RR.

VII. EXPERIMENTAL ANALYSIS

The experimental analysis that will be adopted in this paper uses all the assumptions and experiments performed in [3] and then compared the results in the original paper along with the results by the proposed algorithm.

The performance evaluation was taken in two different cases. In the first case, arrival time has been considered zero and CPU burst time has been taken in increasing, decreasing and random orders. In the second case, arrival time has been considered non zero and CPU burst time has been taken in increasing, decreasing and random orders.

A. *CASE 1 - Zero Arrival Time*

In this case arrival time has been considered zero and CPU burst time has been taken in increasing, decreasing and random orders. Time quantum is 10*ms*.

1) ***CPU Burst Time in Increasing Order:*** The ready queue with five processes P1, P2, P3, P4 and P5 arriving at time 0 with burst time 5, 12, 20, 26 and 34*ms* respectively was considered. The comparative results of RR, IRR and proposed AAIRR are shown in Table 1. Figures 2-4 show the Gantt chart representation of RR, IRR and AAIRR respectively. Figure 5 shows the bar chart of the comparison.

Table 1: Comparative results of RR, IRR and AAIRR

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	38.4	57.8	10
IRR	30.4	49.8	7
AAIRR	26.4	45.8	6

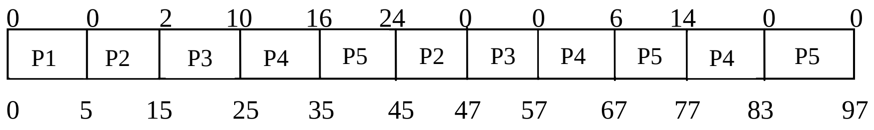

Figure 2: Gantt chart representation of RR

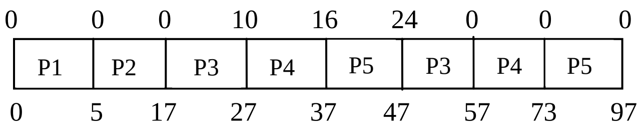

Figure 3: Gantt chart representation of IRR

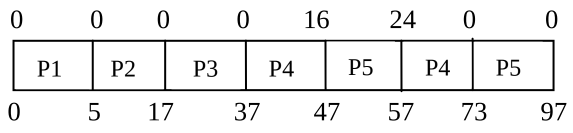

Figure 4: Gantt chart representation of AAIRR

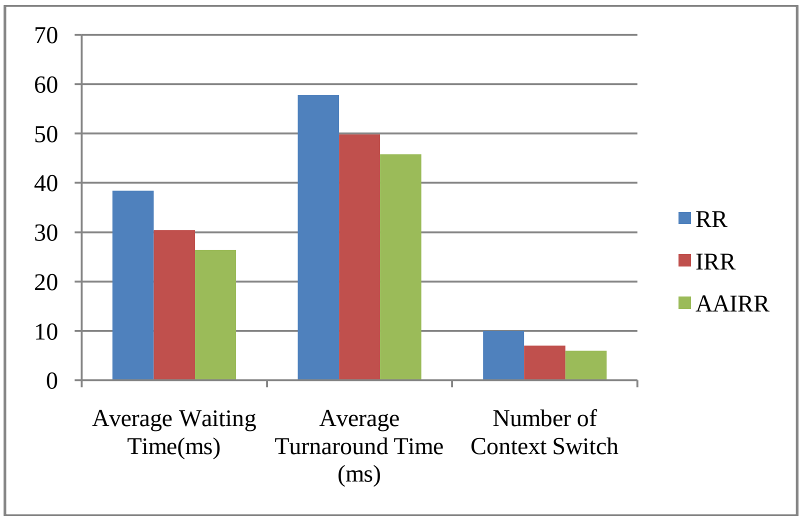

Figure 5: Bar chart of Case 1 (Increasing order)

2) ***CPU Burst Time in Decreasing Order:*** The ready queue with five processes P1, P2, P3, P4 and P5 arriving at time 0 with burst time 34, 26, 20, 12 and 5*ms* respectively was considered. The comparative results RR, IRR and proposed AAIRR are shown in Table 2. Figures 6-8 show the Gantt chart representation of RR, IRR and AAIRR respectively. Figure 9 shows the bar chart of the comparison.

Table 2: Comparative results of RR, IRR and AAIRR

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	58	77.4	11
IRR	49	68.4	8
AAIRR	34.4	53.8	7

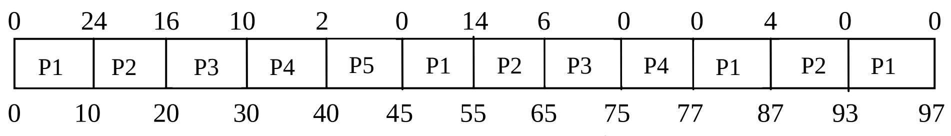

Figure 6: Gantt chart representation of RR

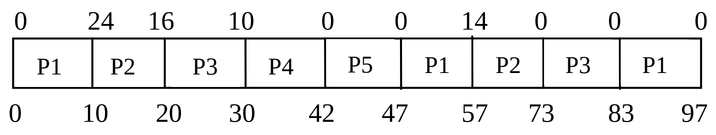

Figure 7: Gantt chart representation of IRR

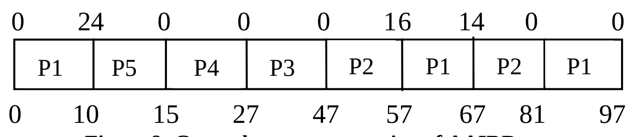

Figure 8: Gantt chart representation of AAIRR

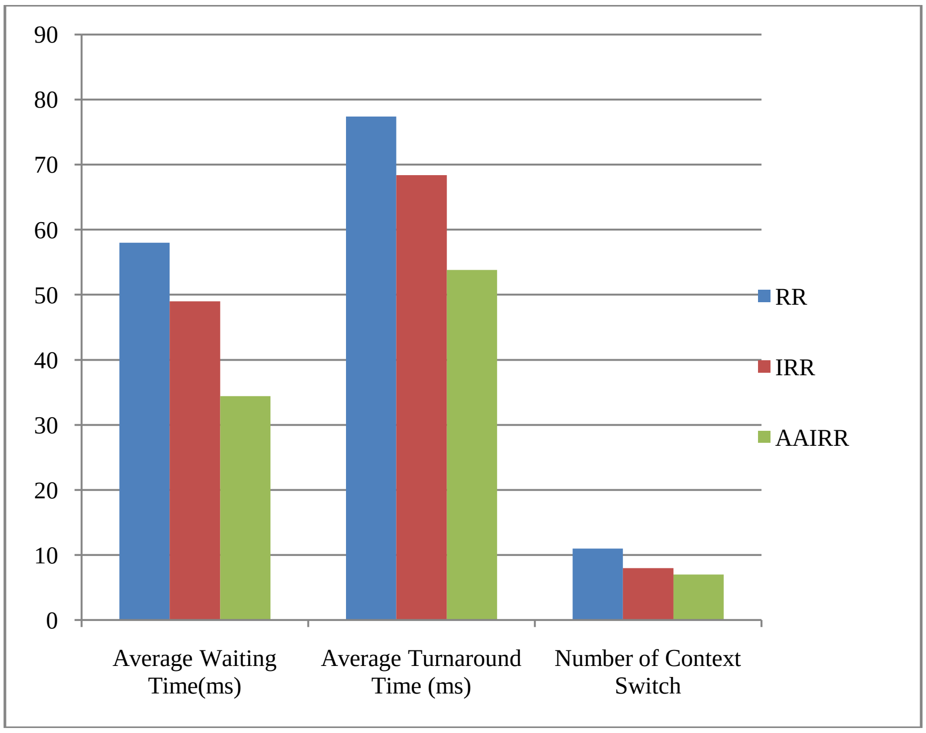

Figure 9: Bar chart of Case 1 (Decreasing order)

3) ***CPU Burst Time in Random Order:*** The ready queue with five processes P1, P2, P3, P4 and P5 arriving at time 0 with burst time 20, 34, 5, 12 and 26 respectively was considered. The comparative results of RR, IRR and proposed AAIRR are shown in Table 3. Figures 10-12 show the Gantt chart representation of RR, IRR and AAIRR respectively. Figure 13 shows the bar chart of the comparison.

Table 3: Comparative results of RR, IRR and AAIRR

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	48	67.4	11
IRR	40.4	59.8	8
AAIRR	31	50.4	6

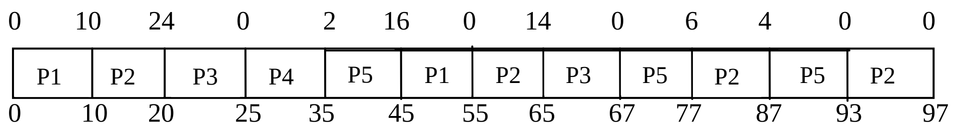

Figure 10: Gantt chart representation of RR

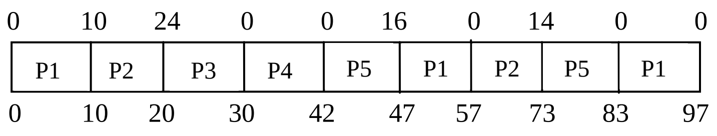

Figure 11: Gantt chart representation of IRR

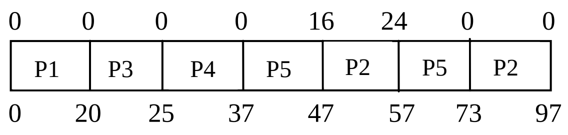

Figure 12: Gantt chart representation of AAIRR

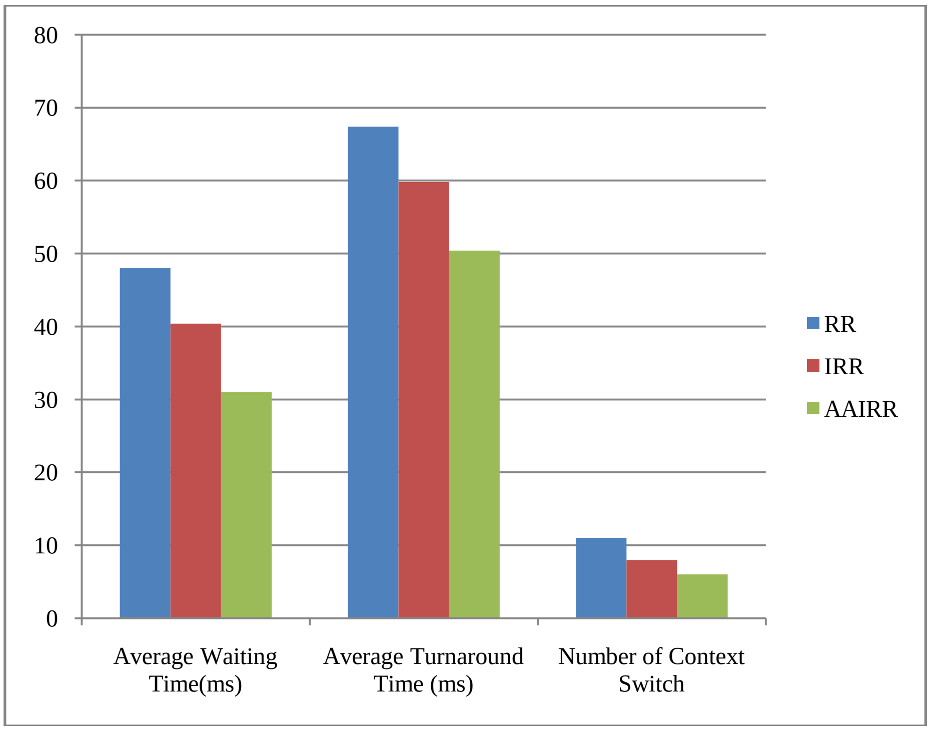

Figure 13: Bar chart of Case 1 (Random order)

B. ***CASE 2 – Non-Zero Arrival Time:***

In this case arrival time has been considered non-zero and CPU burst time has been taken in increasing, decreasing and random orders. Time quantum is 10 milliseconds.

1) ***CPU Burst Time in Increasing Order:*** The ready queue with five processes P1, P2, P3, P4 and P5 arriving at time 0, 4, 10, 15 and 17 with burst time 7, 18, 27, 30 and 36 respectively was considered. The comparative results of RR, IRR and proposed AAIRR are shown in Table 4. Figures 14-16 show the Gantt chart representation of RR, IRR and AAIRR respectively. Figure 17 shows the bar chart of the comparison.

Table 4: Comparison of RR, IRR and AAIRR

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	42	65.6	11
IRR	32	55.6	9
AAIRR	30	53.6	7

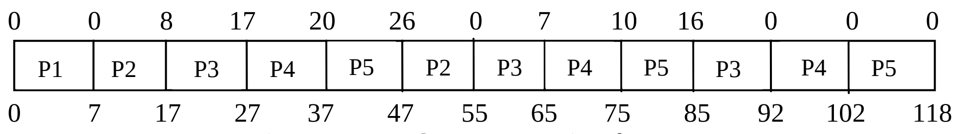

Figure 14: Gantt chart representation of RR

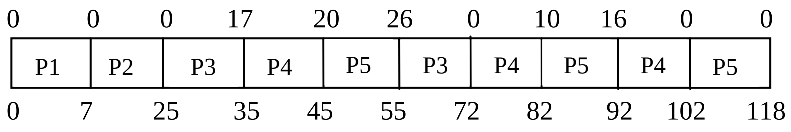

Figure 15: Gantt chart representation of IRR

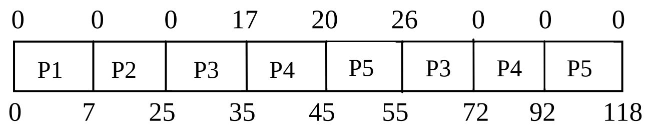

Figure 16: Gantt chart representation of AAIRR

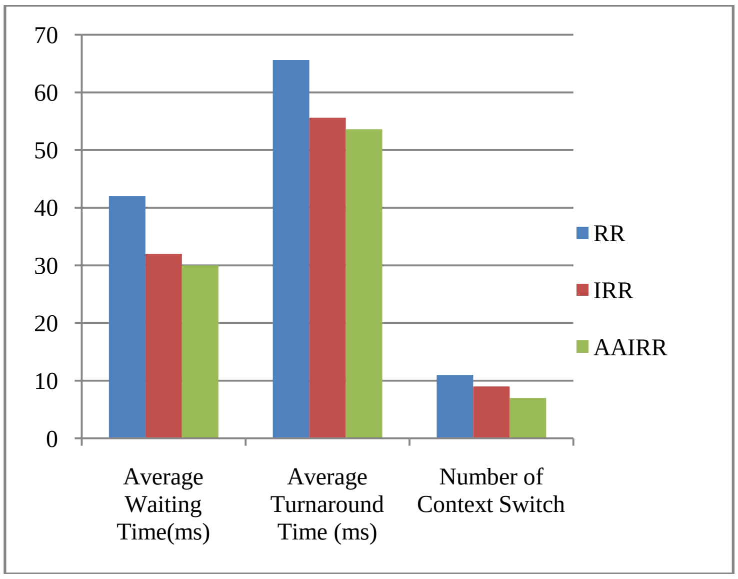

Figure 17: Bar chart of Case 2 (Increasing order)

2) ***CPU Burst Time in Decreasing Order:*** The ready queue with five processes P1, P2, P3, P4 and P5 arriving at time 0, 4, 10, 15 and 17 with burst time 36, 30, 27, 18 and 7 respectively was considered. The comparative results of RR, IRR and proposed AAIRR are shown in Table 5. Figure 18-20 show the Gantt chart representation of RR, IRR and AAIRR respectively. Figure 21 shows the bar chart of the comparison.

Table 5: Comparison of RR, IRR and AAIRR

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	60.6	84.2	12
IRR	51.4	75	9
AAIRR	38	61.6	7

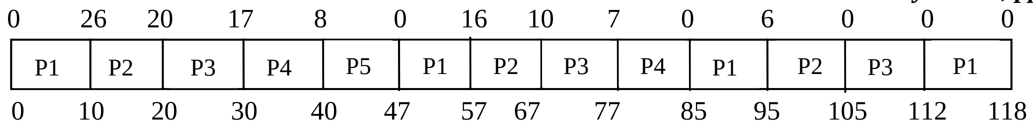

Figure 18: Gantt chart representation of RR

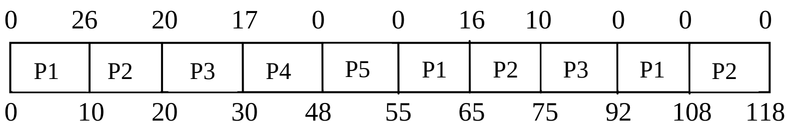

Figure 19: Gantt chart representation of IRR

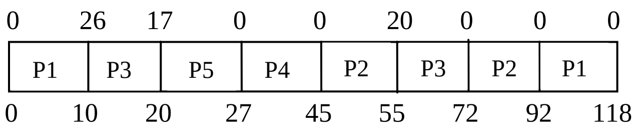

Figure 20: Gantt chart representation of AAIRR

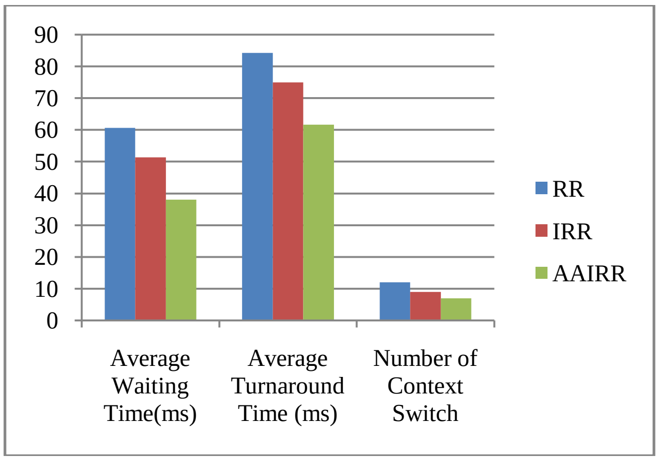

Figure 21: Bar chart of Case 2 (Decreasing order)

3) ***CPU Burst Time in Random Order:*** The ready queue with five processes P1, P2, P3, P4 and P5 arriving at time 0, 4, 10, 15 and 17 with burst time 27, 7, 30, 36 and 18 respectively was considered. The comparative results of RR, IRR and proposed AAIRR are shown in Table 6. Figure 22-24 show the Gantt chart representation of RR, IRR and AAIRR respectively. Figure 25 shows the bar chart of the comparison.

Table 6: Comparison of RR, IRR and AAIRR

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	52	75.6	11
IRR	40	63.6	9
AAIRR	34	57.6	7

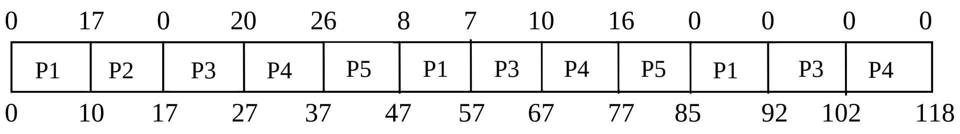

Figure 22: Gantt chart representation of RR

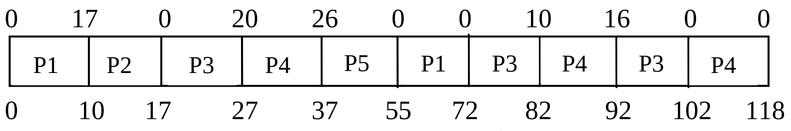

Figure 23: Gantt chart representation of IRR

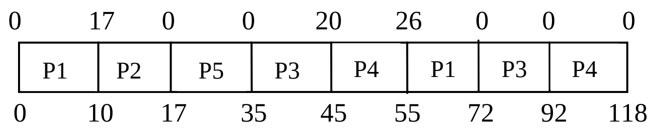

Figure 24: Gantt chart representation of AAIRR

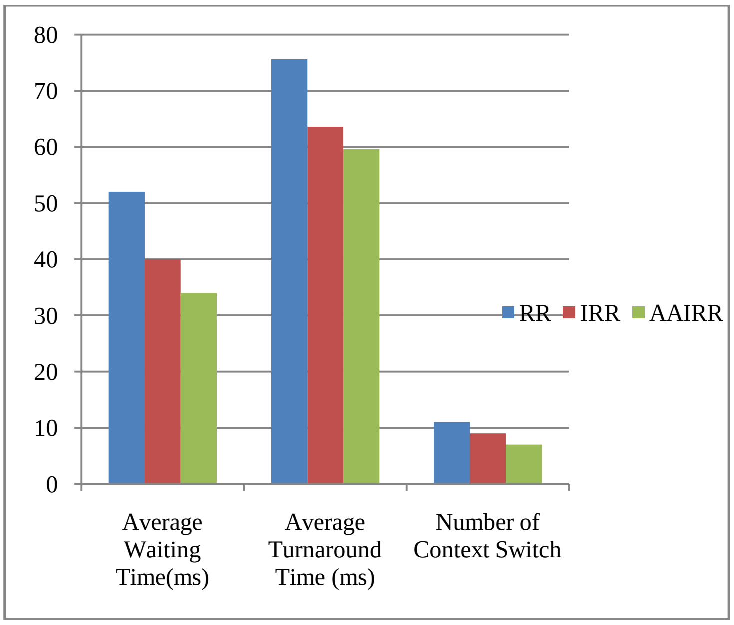

Figure 25: Bar chart of Case 2 (Random order)

VIII. CONCLUSION AND FUTURE SCOPE

In this paper, we have presented a more improved Round Robin CPU scheduling algorithm which improved on the Improved Round Robin CPU scheduling algorithm [3]. Results have shown that the proposed algorithm gives better results in terms of average waiting time, average turnaround time and number of context switches in all cases of process categories than the simple Round Robin CPU scheduling algorithm and the Improved Round Robin CPU scheduling algorithm The general problem in any form of Round Robin CPU scheduling algorithm is the choice of time quantum which when it is too small will result in Processor Sharing algorithm and in this case the number of context switches will be very high, and if the time quantum is very high, then the algorithm will be the same as First Come First Serve (FCFS) CPU scheduling algorithm [6]. So, a future scope is to determine time quantum dynamically, so as to overcome this problem.

References

[1] Ajit, S, Priyanka, G and Sahil, B (2010): An Optimized Round Robin Scheduling Algorithm for CPU Scheduling, International Journal on Computer Science and Engineering (IJCSE), Vol. 02, No. 07, 2383-2385, pp 2382-2385.

[2] Ishwari, S. R and Deepa, G (2012): A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems, International Journal of Innovations in Engineering and Technology (IJIET), ISSN: 2319 – 1058, Vol. 1 Issue 3, pp 1-11.

[3] Manish K. M. and Abdul Kadir K. (2012): An Improved Round Robin CPU Scheduling Algorithm, Journal of Global Research in Computer Science, ISSN: 2229-371X, Volume 3, No. 6, pp 64-69.

[4] Oyetunji, E.O and Oluleye, A. E (2009): Performance Assessment of Some CPU Scheduling Algorithms, Journal of Information Technology 1(1): 22-26, ISSN: 2041-3114, pp 22-26.

[5] Silberschatz, P. B. Galvin, and G. Gagne, “Operating System Concepts”, 7th Edn., John Wiley and Sons Inc, 2005, ISBN 0-471-69466-5.

[6] Soraj, H and Roy, K.C: Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice", International Journal of Data Engineering (IJDE), Volume 2, Issue 3, www.cscjournals.org/csc/manuscript/Journals/IJDE/.../IJDE-57.pdf ,accessed 10th December 2012.

[7] Suri, P.K and Sumit, M (2012): Design of Stochastic Simulator for Analyzing the Impact of Scalability on CPU Scheduling Algorithms, International Journal of Computer Applications (0975 – 8887) Volume 49– No.17, pp 4-9.