
# School of Computer Science and Engineering

**A Project On**

## CPU Scheduling using Improved Round Robin Algorithms

**Under the Guidance of,**

**Prof. Shaik Naseera**

**BY**

Group Members details

| Name | Reg. no: |
|---|---|
| **HARSHIT DADHICH** | **17BCI0011** |
| **ACHAL GOYAL** | **17BCI0024** |
| **SHUBHAM GUPTA** | **17BCI0044** |
| **KUMAR ABHISHEK** | **17BCI0145** |
| **VISHAL SAINI** | **17BCI0188** |

## ABSTRACT:

CPU scheduling is one of the most important operation carried out by Operating System. Different algorithms are available for CPU scheduling. Round Robin is one of the algorithms which is based on time sharing concept. The effectiveness of the Round Robin algorithm depends on the choice of time quantum. At present, Round Robin is based on static time quantum which has high average waiting time, turnaround time and context switching which is not very much helpful in real time uni-processor-multi programming Operating System. This project is focused on finding different algorithms to choose time quantum in order to reduce the average waiting time, turnaround time and context switching for uni-processor-multi programming OS.


## INTRODUCTION:

Operating System performs variety of task in which scheduling is on of the basic task. CPU Scheduler, also called **Short-term Scheduler**, is responsible for selecting one process at a time from pool of process present in the ready queue and allocates the CPU to it. The primary objective of the CPU scheduling is to maximize CPU utilization, throughput and minimize average waiting, turnaround, response time and context switching for a set of requests. In multi programmed computing system, efficiency of CPU highly depends on allocating the different process in correct fashion. Thus, CPU scheduling is one of the important entities in increasing the efficiency of the computing system.

There are many CPU scheduling algorithms with distinct benefits and shortcomings. Some of the popular algorithms are First Come First Served (FCFS), Shortest Job First (SJF), Priority scheduling, Round Robin (RR). In FCFS, the process that arrived first in ready queue served first. In SJF, process from the ready queue that has shortest burst time (it refers to the time taken by process to complete) will execute first. Priority Scheduling algorithm allocates CPU to the higher priority process present in the ready queue. In Round Robin, a small unit of time quantum is given to each process present in the ready queue which maintain the fairness factor.

In this project, five different algorithms are proposed which helps in increasing the efficiency of the Round Robin. All the algorithms are focused on selecting an appropriate time quantum based on the burst time of the process present in ready queue which helps in increasing the performance of the exiting Round Robin algorithm.

### Performance Criteria:

There are various CPU scheduling algorithms for selecting a process from a pool of processes. The criteria to measure the efficiency of any CPU scheduling algorithm are:

a. **Context Switch**: It is the process of storing and restoring context of a pre-empted process, so the execution can be resumed from same point at later time.
b. **Throughput:** It is defined as number of process completed per unit time. Context switch and throughput are inversely proportional to each other.
c. **Turnaround Time:** It refers to the total time taken by process to complete.

d. **Waiting Time:** It refers to the total time spent by process in ready queue.
e. **Response Time:** It refers to the time from the submission of a request until the first response is produced.

A good CPU scheduling algorithm has following criteria:

a. Minimum context switches.
b. Maximum throughput.
c. Minimum turnaround time.
d. Minimum waiting time.
e. Minimum response time.

## Literature Survey:

In recent time different approaches are used to increase the performance of the CPU scheduling algorithm.

I.S. Rajat[1].. merges the concept of priority scheduling and round robin in such a way that after each cycle new priority are assigned to all the process according to the remaining burst time of the process and processes are executed according to the new priority.

R.K Mishra[2] proposed a system where after one cycle of round robin, shortest job is assigned to CPU first, then the next. It is based on SJF and Round robin. It will continue until all the process are executed.

S. Verma[3] proposed new algorithm where process are arranged in ascending order on the basis of new value call factor which is calculated by adding burst time and arrival time.

S. Babu[4] proposed a method in which process are sorted in ascending order. If time quantum of any process expires before its complete execution, it is put at the end of the ready queue and control of CPU is assigned to next shortest incoming process.

Abbas Noon[5] increases the efficiency of existing round robin by using dynamic time quantum concept in which time quantum is based on mean average of the burst time.

H.S. Behera [6] proposed method in which time quantum depends on the median of the burst time. Processes are arranged in ascending order of the burst time. Time quantum is calculated after each cycle on basis of burst time of remaining process.

Samih M. Mostafa[7] uses the concept of Integer Programming that decides a value which is neither too large nor too small such that every process has got reasonable time and throughput of the system is not decreased due to unnecessarily context switches.

Tong li[8]proposed a method called Distributed Weighted Round Robin algorithm.it enables user to specify weights to thread and achieve accurate proportional CPU sharing with constant error bounds.

Rami J[9] uses the concept of median to find dynamic time quantum in round robin algorithm. If median of burst time is less than 25, then time quantum will be 25. He set minimum time quantum value as 25.

Saroj Hiranwal[10] proposed algorithm based on smart time slicing concept which depends on three aspects i.e. priority, average CPU burst time and context switch avoidance time.

Smart time slice=mid process burst time(if number of processes are odd)

Or

Smart time slice = Average Burst Time(if number of processes are even).

Helmy[11] introduced a proportional-share scheduling algorithm as an attempt to combine the low scheduling overhead of the round robin algorithms and favour shortest jobs.

Sanjay Kumar[12] tries to enhance the efficiency of round robin using min max dispersion algorithm. The idea is to make time quantum repeatedly adjusted using Min-Max dispersion measure in accordance with remaining CPU burst time.

D. Nayak[13] proposed algorithm is based on median and largest burst time of the processes present in ready queue. Time quantum is derived as the sum of highest burst time and median divided by 2.

Abdul razag[14] proposal is also based on selecting time quantum on the bases of the median of the burst time of the processes present in the reday queue.

S. kiran[15] proposed method is called Mean-Difference round Robin Algorith. The proposed algorithm calculates the mean burst time of all the processes in the ready queue. Next, it finds out the difference between a process burst time and calculated mean burst time. This step is repeated for all the processes in the ready queue. Then, the proposed algorithm find out the process having the largest difference value and assigns it to CPU, and execute it for one time slice.

Neetu goel[16] performs a comparative study of CPU scheduling algorithms in different environments to find out the best possible scheduling algorithms.

Rakesh Mohanty[17] proposed a new variant of Round Robin (RR) algorithm is proposed which is suitable for soft real time systems. The proposed algorithm, known as Priority Based Dynamic Round Robin Algorithm(PBDRR),which calculates intelligent time slice for individual processes and changes after every round of execution. The proposed scheduling algorithm is developed by taking dynamic time quantum concept into account.

Neetu Goel[18]In this proposed method, factor value is computed for every process as F = BT*0.2 + At*0.3.Arranging the process in ascending order and time quantum equal sum of 1st process burst time and 2nd process burst time divided by 2.

Rishi Verma[19]In this approach, dynamic time quantum is taken into account where time quantum is computed as difference between the largest and smallest burst time present in ready queue.

Shailendra Shukla[20]In his approach, process are arranged according to ascending value of arrival time + burst time. Time Quantum is calculated as: if jobs are odd then time quantum is middle job burst time. If jobs are even the time quantum=(middle burst time + (middle burst time+1)/2).

## Proposed Algorithm:

In this project, we have proposed five different algorithms to improve existing round robin algorithm.

**Assumptions:**

All the operations are performed in single process environment and all the processes are independent. All the processes have equal priority. The context switching time is equal to zero. All the processes are CPU bound. CPU will not sit idle until all the processes in the ready queue has been completed.

**ALGORITHM-0:**

This is the existing algorithm. Here, process are arriving at different time interval.

**STEP 1:** CPU scheduler picks the process from ready queue.

**STEP 2:** CPU is allocated to 1 process for 1 time quantum.

**STEP 3:** if burst time is less than 1 TQ then process will leave the CPU after completion.

**STEP 4:** If a new process arrives then put it at the tail of the process.

**STEP 5:** Remove the currently running process from the ready queue and put it at the tail of the ready queue.

**STEP 6:** END

## ALGORITHM -1:

This algorithm is based on static time quantum where process arrived at different time interval and arrival time of 1st process is zero.

TQ=time quantum

**STEP 1:** Make a ready queue of the Processes.

**STEP 2:** Select a TQ

**STEP 3:** Do steps 4,5 and 6 while ready queue becomes empty.

**STEP 4:** Pick the first process from the ready queue and allocate the CPU to it for a time interval up to 1 TQ

**STEP 5:** If the remaining CPU burst time of the currently process is less than 1 TQ then allocate CPU again to the currently running process for the remaining CPU burst time. After completion of the execution, remove it from the ready queue and go to Step 3.

**STEP 6:** If a new process arrives then put it at the tail of the process.

**STEP 7:** Remove the currently running process from the ready queue and put it at the tail of the ready queue.

**STEP 7:** END

## ALGORITHM – 2:

This algorithm is based on dynamic time quantum where time quantum will change after every cycle. Processes arrive at different time interval with arrival time of 1st process is considered zero.

TQ=time quantum

F=factor

**STEP 1:** Compute the factor analysis F=Burst time*0.3+Arrival time *0.2 for all the process.

**STEP 2:** Shuffle the processes in ascending order according to the factor of each process in the ready queue (RQ) such that the head of the ready queue contains the lowest factor process based on the burst time, arrival time of the process.

**STEP 3:** low=burst value of the first process. high= burst value of the last process.

**STEP 4:** Compute TQ=(low+high)/2

**STEP 5:** Pick first process from the ready queue and allocate the CPU to it for 1 TQ.

**STEP 6:** If remaining time of the process is less than TQ/2, allocate the CPU again to the process for remaining burst time.

**STEP 7:** calculate factor value of the running process as factor=remaining burst time*0.3 and allocate it to the ready queue. Go to Step 2.

**STEP 8:** END

**ALGORITHM – 3:**

This algorithm is based on dynamic time quantum where process arrived at different time interval with arrival time of 1st process as zero.

TQ=Time Quantum

RQ=Ready Queue

n=number of process

**STEP 1:** Arrange the process in ascending order to the burst time.

**STEP 2:** n=number of processes in RQ

**STEP 3:** compute TQ=(sum of the burst time of process in RQ)/n

**STEP 4:** allocate each process to CPU for 1 TQ.

**STEP 5:** calculate new burst time of the process.

**STEP 6:** If new burst time > 0 then add the process in ready queue for next round of execution.

**STEP 7:** If new process arrives, go to STEP 1.

**STEP 8:** If RQ is not empty, go to STEP 2.

**STEP 9:** END

**ALGORITHM – 4:**

This algorithm is based on dynamic time quantum where arrival time of the process is zero.

TQ=Time Quantum

RQ=Ready Queue

**STEP 1:** Arrange the process in ascending order to the burst time in ready queue.

**STEP 2:** Initialize TQ=burst time of the first process.

**STEP 3:** Allocate CPU to each process for 1 TQ.

**STEP 4:** Calculate new burst time as burst-TQ.

**STEP 5:** If new burst time > 0 then allocate the process the RQ else eliminate it from RQ.

**STEP 6:** If new arrives or RQ is not empty, go to STEP 1.

**STEP 7:** END

**ALGORITHM – 5:**

This algorithm is based on dynamic time quantum where arrival time of all the process is zero.

TQ=Time Quantum

**STEP 1:** Arrange all the process in ascending order to the burst time in ready queue.

**STEP 2:** Compute TQ

If (maximum burst time – minimum burst time) < minimum burst time then TQ=minimum burst time.

If (maximum burst time – minimum burst time) equals zero then TQ=minimum burst time.

TQ = minimum burst time – maximum burst time

**STEP 3:** Allocate each process to 1 TQ.

**STEP 4:** compute new burst time of all the process.

**STEP 5:** If new burst time > 0 then allocate the process in RQ else eliminate the process.

**STEP 6:** If new process arrives or RQ is not empty, then go to STEP 1.

**STEP 7:** END

## EXPERIMENTAL ANALYSIS

We will analyse the efficiency of all the processes in three different cases:

1. Burst Time in Increasing Order.
2. Burst Time in Decreasing Order.
3. Burst Time in Random Order.

**SET 1:** When process arrives with different arrival time.

### CASE 1: Burst Time in Increasing Order

**EXAMPLE 1**

| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| 1 | 0 | 8 |
| 2 | 2 | 15 |
| 3 | 3 | 20 |
| 4 | 5 | 22 |
| 5 | 8 | 24 |
| 6 | 10 | 28 |
| 7 | 12 | 32 |
| 8 | 14 | 40 |
| 9 | 17 | 42 |
| 10 | 21 | 50 |
| 11 | 25 | 55 |
| 12 | 29 | 65 |
| 13 | 32 | 68 |
| 14 | 38 | 70 |
| 15 | 45 | 75 |

**EXAMPLE 2**

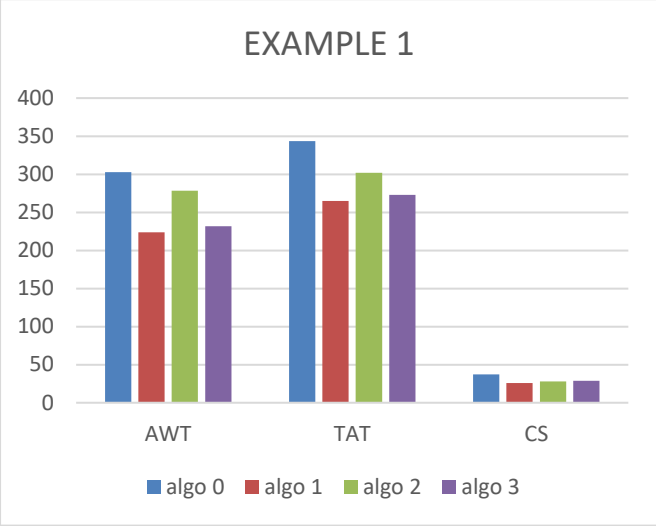
| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| 1 | 0 | 2 |
| 2 | 1 | 4 |
| 3 | 5 | 8 |
| 4 | 8 | 17 |
| 5 | 12 | 20 |
| 6 | 17 | 22 |
| 7 | 28 | 28 |
| 8 | 32 | 32 |
| 9 | 33 | 36 |
| 10 | 39 | 42 |
| 11 | 40 | 52 |
| 12 | 42 | 66 |
| 13 | 50 | 70 |
| 14 | 56 | 75 |
| 15 | 60 | 80 |

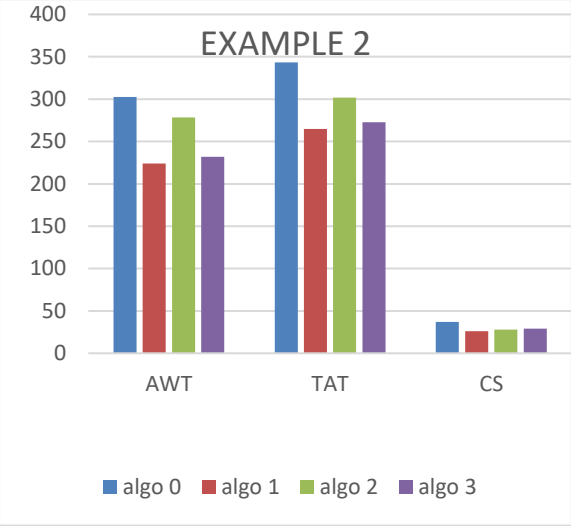
AWT = Average Waiting Time

TAT = Average Turnaround Time

CS = Context Switches

| Algorithm | AWT | TAT | CS | | | AWT | TAT | CS |
|---|---|---|---|---|---|---|---|---|
| 0 | 302.6 | 343.53 | 37 | | | 210.40 | 247.33 | 33 |
| 1 | 223.93 | 264.87 | 26 | | | 150.20 | 187.13 | 25 |
| 2 | 278.33 | 301.87 | 28 | | | 190.40 | 196.60 | 28 |
| 3 | 231.93 | 272.87 | 29 | | | 144.20 | 181.13 | 23 |



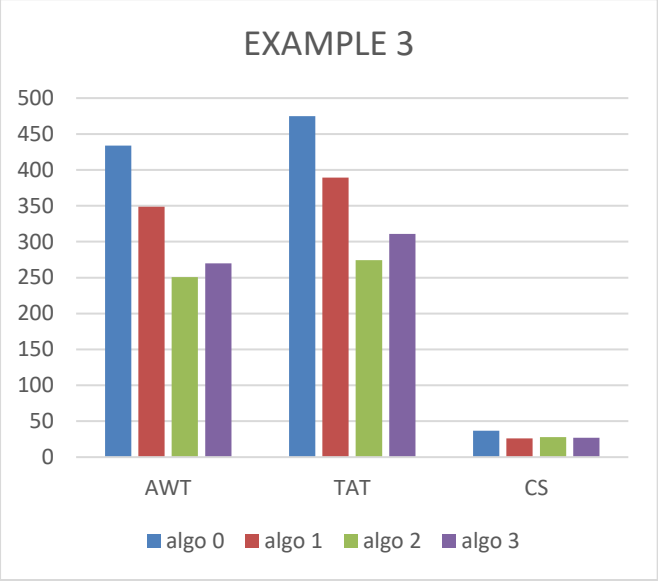



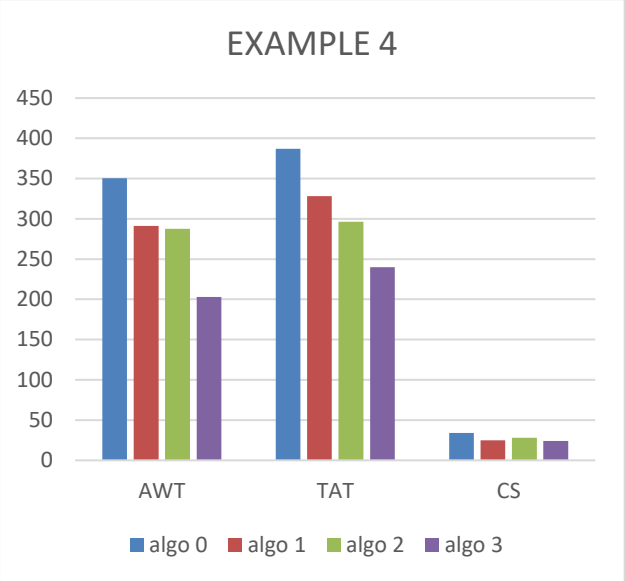

**CASE 2: BURST TIME IN DECREASING ORDER**

**EXAMPLE 3** **EXAMPLE 4**

| PROCESS | ARRIVAL TIME | BURST TIME | | PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|---|---|---|---|
| 1 | 0 | 75 | | 1 | 0 | 80 |
| 2 | 2 | 70 | | 2 | 1 | 75 |
| 3 | 3 | 68 | | 3 | 5 | 70 |
| 4 | 5 | 65 | | 4 | 8 | 66 |
| 5 | 8 | 55 | | 5 | 12 | 52 |
| 6 | 10 | 50 | | 6 | 17 | 42 |
| 7 | 12 | 42 | | 7 | 28 | 36 |
| 8 | 14 | 40 | | 8 | 32 | 32 |
| 9 | 17 | 32 | | 9 | 33 | 28 |
| 10 | 21 | 28 | | 10 | 39 | 22 |
| 11 | 25 | 24 | | 11 | 40 | 20 |
| 12 | 29 | 22 | | 12 | 42 | 17 |
| 13 | 32 | 20 | | 13 | 50 | 8 |
| 14 | 38 | 15 | | 14 | 56 | 4 |
| 15 | 45 | 8 | | 15 | 60 | 2 |

| Algorithm | AWT | TAT | CS | | | AWT | TAT | CS |
|---|---|---|---|---|---|---|---|---|
| 0 | 433.93 | 474.87 | 37 | | | 350.27 | 387.20 | 34 |
| 1 | 348.53 | 389.47 | 26 | | | 291.40 | 328.33 | 25 |
| 2 | 250.80 | 274.33 | 28 | | | 287.53 | 296.27 | 28 |
| 3 | 270.07 | 311.00 | 27 | | | 202.87 | 239.80 | 24 |




**CASE 3: Burst Time in Random Order**

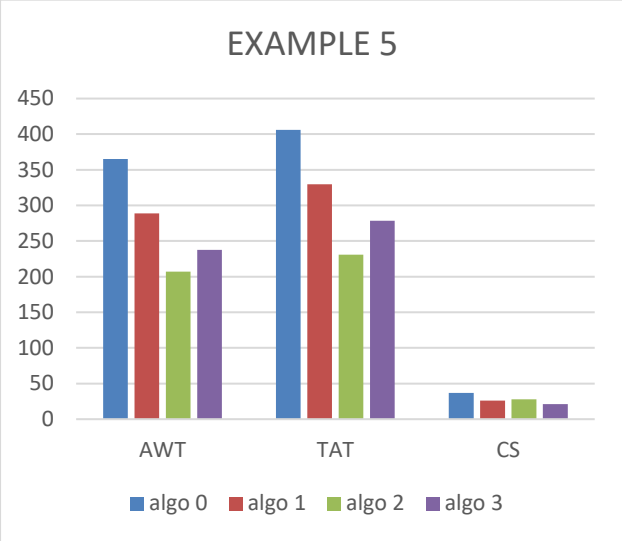
**EXAMPLE 5**

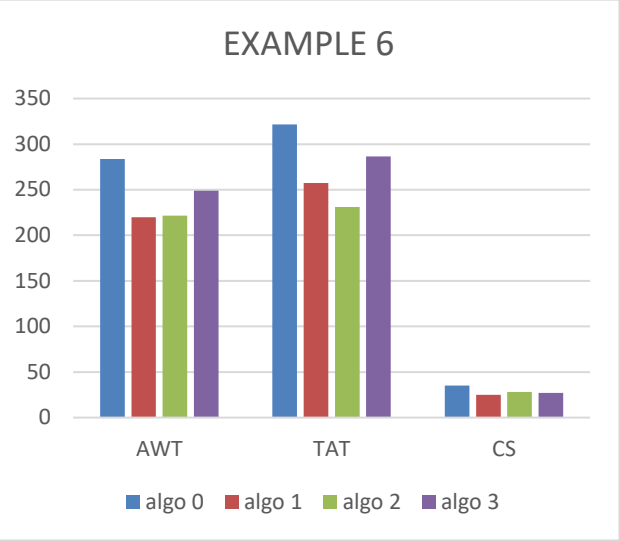
| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| 1 | 0 | 20 |
| 2 | 2 | 32 |
| 3 | 3 | 40 |
| 4 | 5 | 70 |
| 5 | 8 | 68 |
| 6 | 10 | 75 |
| 7 | 12 | 42 |
| 8 | 14 | 8 |
| 9 | 17 | 15 |
| 10 | 21 | 65 |
| 11 | 25 | 55 |
| 12 | 29 | 24 |
| 13 | 32 | 28 |
| 14 | 38 | 50 |
| 15 | 45 | 22 |

**EXAMPLE 6**

| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| 1 | 0 | 32 |
| 2 | 1 | 8 |
| 3 | 5 | 52 |
| 4 | 8 | 4 |
| 5 | 12 | 75 |
| 6 | 17 | 70 |
| 7 | 28 | 80 |
| 8 | 32 | 28 |
| 9 | 33 | 2 |
| 10 | 39 | 36 |
| 11 | 40 | 42 |
| 12 | 42 | 32 |
| 13 | 50 | 66 |
| 14 | 56 | 17 |
| 15 | 60 | 22 |

| Algorithm | AWT | TAT | CS | | | AWT | TAT | CS |
|---|---|---|---|---|---|---|---|---|
| 0 | 365.13 | 406.07 | 37 | | | 283.73 | 321.47 | 35 |
| 1 | 288.80 | 329.73 | 26 | | | 219.67 | 257.40 | 25 |
| 2 | 207.13 | 230.67 | 28 | | | 221.60 | 231.13 | 28 |
| 3 | 237.67 | 278.60 | 21 | | | 248.87 | 286.60 | 27 |



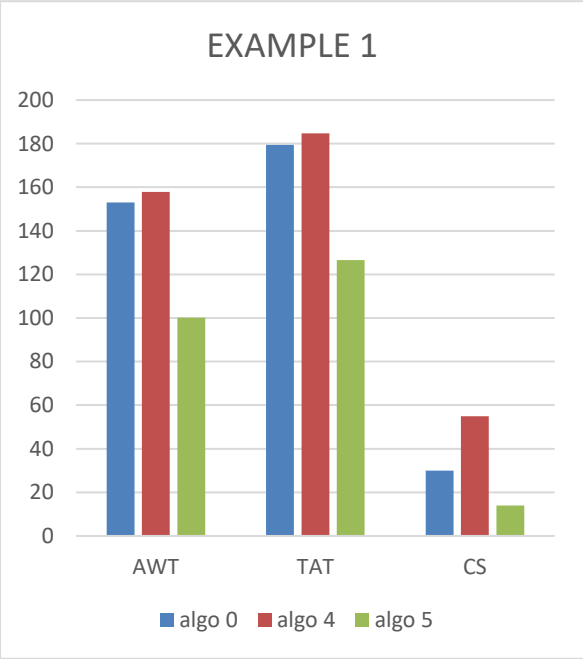



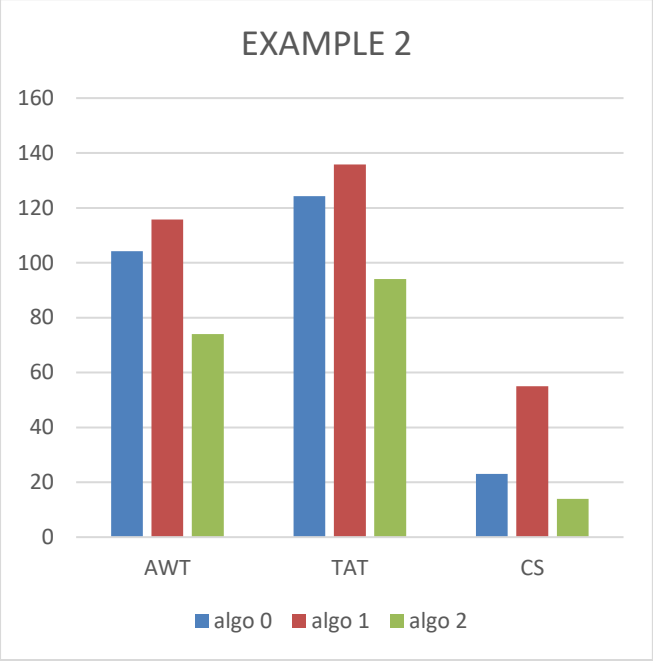

**SET 2: When process arrives with arrival time equals zero.**

**EXAMPLE 1** **EXAMPLE 2**

| PROCESS | BURST TIME | | PROCESS | BURST TIME |
|---|---|---|---|---|
| 1 | 8 | | 1 | 8 |
| 2 | 15 | | 2 | 10 |
| 3 | 20 | | 3 | 12 |
| 4 | 22 | | 4 | 15 |
| 5 | 24 | | 5 | 18 |
| 6 | 28 | | 6 | 22 |
| 7 | 32 | | 7 | 24 |
| 8 | 35 | | 8 | 28 |
| 9 | 38 | | 9 | 30 |
| 10 | 42 | | 10 | 34 |

| Algorithm | AWT | TAT | CS | | AWT | TAT | CS |
|---|---|---|---|---|---|---|---|
| 0(TQ=10) | 153.00 | 179.40 | 30 | | 104.20 | 124.30 | 23 |
| 4 | 157.80 | 184.80 | 55 | | 115.70 | 135.80 | 55 |
| 5 | 100.20 | 126.60 | 14 | | 74.00 | 94.10 | 14 |



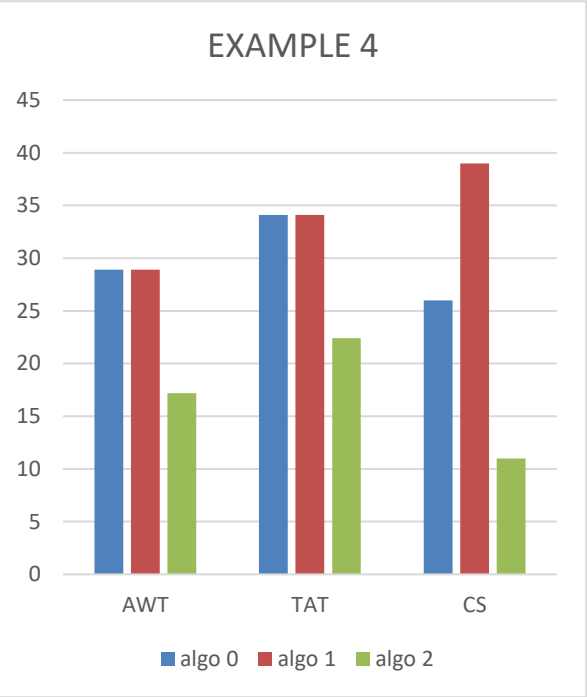

**EXAMPLE 3**

| PROCESS | BURST TIME |
|---|---|
| 1 | 75 |
| 2 | 70 |
| 3 | 68 |
| 4 | 65 |
| 5 | 55 |
| 6 | 50 |
| 7 | 51 |
| 8 | 52 |
| 9 | 66 |
| 10 | 67 |

**EXAMPLE 4**

| PROCESS | BURST TIME |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 4 |
| 5 | 3 |
| 6 | 8 |
| 7 | 5 |
| 8 | 10 |
| 9 | 6 |
| 10 | 4 |

| Algorithm | AWT | TAT | CS | | | AWT | TAT | CS |
|---|---|---|---|---|---|---|---|---|
| 0(**TQ=25**) | 496.60 | 558.50 | 28 | | TQ=2 | 28.90 | 34.10 | 26 |
| 4 | 455.20 | 517.10 | 55 | | | 28.90 | 34.10 | 39 |
| 5 | 434.80 | 496.70 | 20 | | | 17.20 | 22.40 | 11 |

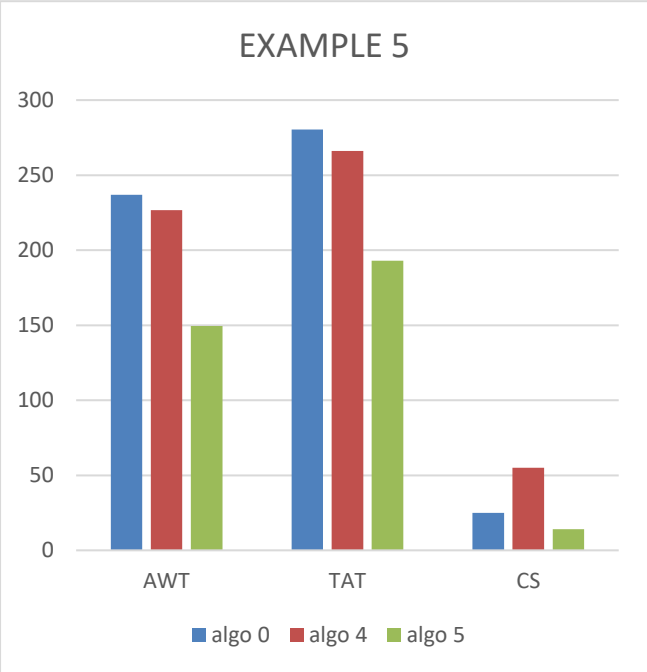

**EXAMPLE 5**

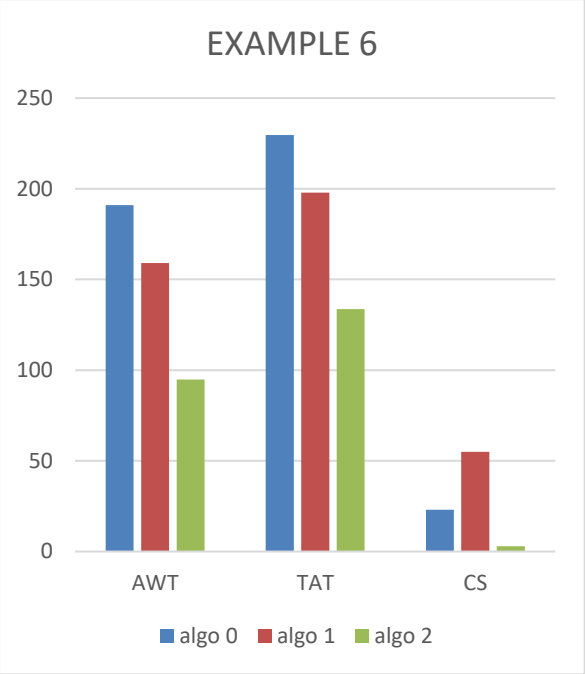
| PROCESS | BURST TIME | |
|---|---|---|
| 1 | 20 | |
| 2 | 32 | |
| 3 | 40 | |
| 4 | 70 | |
| 5 | 68 | |
| 6 | 75 | |
| 7 | 42 | |
| 8 | 8 | |
| 9 | 15 | |
| 10 | 65 | |

**EXAMPLE 6**

| PROCESS | BURST TIME |
|---|---|
| 1 | 32 |
| 2 | 8 |
| 3 | 52 |
| 4 | 4 |
| 5 | 75 |
| 6 | 70 |
| 7 | 80 |
| 8 | 28 |
| 9 | 2 |
| 10 | 36 |

| Algorithm | AWT | TAT | CS | | | AWT | TAT | CS |
|---|---|---|---|---|---|---|---|---|
| 0(TQ=20) | 237.00 | 280.50 | 25 | | | 191.00 | 229.70 | 23 |
| 4 | 222.60 | 266.10 | 55 | | | 159.10 | 197.80 | 55 |
| 5 | 149.40 | 192.90 | 14 | | | 94.90 | 133.60 | 11 |




## Result Analysis:

**SET 1:** When arrival time of the processes are different.

In case of **static time quantum,** performance of **Algorithm 1** is very good in terms of AWT, TAT and Context Switching.

In case of **dynamic time quantum,** performance of **Algorithm 3** is good in terms of AWT, TAT and Context Switching. Algorithm 2 also shows a good result but its working is more similar to the FCFS algorithm.

**SET 2:** When arrival time of all the process is zero.

In this case, **Algorithm 5** shows a very good performance in comparison to the other algorithms. Algorithm 4 shows good result in terms of AWT and TAT but its performance is bad in terms of context switching. It happens because TQ in algorithm 4 depends on minimum burst time.

## CONCLUSION:

This project proposes some new algorithms which is based on existing Round robin Algorithm. The experimental analysis shows that proposed algorithms works better than the existing one in different cases by reducing the waiting time, turnaround time and context switches. In static time quantum environment, algorithm 1 can replace that existing one. This paper has successfully solved the problem of selecting time quantum as proposed algorithms decides time quantum based on the arrival time and burst time of the process. For overall performance, Algorithm 3 is good for dynamic time quantum environment.

For future research, expression to find factor analysis can be optimized in algorithm 2 to enhance its performance.

## REFERENCES:


**[1]** Rajput, I. S., & Gupta, D. (2012). A priority based round robin CPU scheduling algorithm for real time systems. *International Journal of Innovations in Engineering and Technology*, *1*(3), 1-11.

**[2]** Yadav, R. K., Mishra, A. K., Prakash, N., & Sharma, H. (2010). An improved round robin scheduling algorithm for CPU scheduling. *International Journal on Computer Science and Engineering*, *2*(04), 1064-1066.

**[3]** Varma, P. S. Optimized Round Robin CPU Scheduling Algorithm.

**[4]** Sukumar Babu, B., Neelima Priyanka, N., & Suresh Varma, P. (2012). Optimized round robin CPU scheduling algorithm. *Global J Comput Sci Technol*, *12*(11), 21-25.

**[5]** Noon, A., Kalakech, A., & Kadry, S. (2011). A new round robin based scheduling algorithm for operating systems: dynamic quantum using the mean average. *arXiv preprint arXiv:1111.5348*

**[6]** Behera, H. S., Mohanty, R., & Nayak, D. (2011). A new proposed dynamic quantum with re-adjusted round robin scheduling algorithm and its performance analysis. *arXiv preprint arXiv:1103.3831*..

**[7]** Mostafa, S. M., Rida, S. Z., & Hamad, S. H. (2010). Finding time quantum of round robin CPU scheduling algorithm in general computing systems using integer programming. *International Journal of Research and Reviews in Applied Sciences (IJRRAS)*, *5*(1), 64-71.

**[8]** Li, T., Baumberger, D., & Hahn, S. (2009, February). Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. In *ACM Sigplan Notices* (Vol. 44, No. 4, pp. 65-74). ACM.

**[9]** Matarneh, R. J. (2009). Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes. *American Journal of Applied Sciences*, *6*(10), 1831.

**[10]** Hiranwal, S., & Roy, K. C. (2011). Adaptive round robin scheduling using shortest burst approach based on smart time slice. *International Journal of Computer Science and Communication*, *2*(2), 319-323.

**[11]** Helmy, T., & Dekdouk, A. (2007). Burst round robin as a proportional-share scheduling algorithm. *IEEEGCC 2007*.

**[12]** Panda, S. K., & Bhoi, S. K. (2014). An effective round robin algorithm using min-max dispersion measure. *arXiv preprint arXiv:1404.5869*.

**[13]** Nayak, D., Malla, S. K., & Debadarshini, D. (2012). Improved round robin scheduling using dynamic time quantum. *International Journal of Computer Applications (0975–8887) Volume*.

**[14]** Abdulrahim, A., Abdullahi, S. E., & Sahalu, J. B. (2014). A new improved round robin (NIRR) CPU scheduling algorithm. *International Journal of Computer Applications*, *90*(4).

**[15]** Kiran, R. S., Babu, P. V., & Krishna, B. M. (2014). Optimizing CPU scheduling for real time applications using mean-difference round robin (MDRR) algorithm. In *ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol I* (pp. 713-721). Springer, Cham.

**[16]** Goel, N., & Garg, R. B. (2013). A comparative study of cpu scheduling algorithms. *arXiv preprint arXiv:1307.4165*.

**[17]** Mohanty, R., Behera, H. S., Patwari, K., Dash, M., & Prasanna, M. L. (2011). Priority based dynamic round robin (PBDRR) algorithm with intelligent time slice for soft real time systems. *arXiv preprint arXiv:1105.1736*.

**[18]** Goel, N., & Garg, R. B. (2016). Performance analysis of cpu scheduling algorithms with novel omdrrs algorithm. *Int J Adv Comput Sci Appl*, *7*(1), 216-221.

**[19]** Verma, R., Mittal, S., & Singh, V. (2014). A Round Robin Algorithm using Mode Dispersion for Effective Measure. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 166-174.

**[20]** Shukla, S., & Kishore, L. A Dynamic Time Quantum SJRR CPU Scheduling Algorithm. *International Journal of Engineering, Management & Sciences (IJEMS), ISSN*, 2348-3733.