Dynamic Round Robin with Controlled Preemption (DRRCP)

Ashiru Simon[1], Saleh Abdullahi[2] and Sahalu Junaidu[3]

[1,2,3] Mathematics Department, Ahmadu Bello University (ABU), Zaria, Nigeria

Abstract

Round Robin (RR) CPU scheduling algorithm has been designed chiefly for time sharing systems. The RR algorithm has proven to be more useful in multiprogramming environment in which time slice or quantum is given to processes in the ready queue. An ideal classical RR uses a static quantum time which is gotten from the average of processes in the ready queue. One of the major challenges in classical RR is poor timing in performing context switching. This will eventually lead to unnecessary context switching. Using Dynamic Round Robin with Controlled Preemption (DRRCP), variable quantum time is used to eliminate this shortcoming. In an attempt to eliminate unnecessary context switching, the average waiting time, average turnaround time and number of context switching were as well improved. All dataset used for this analysis are generated using normal distribution function.

Keywords: *DRRCP, Quantum time (TQ), Waiting time, Turnaround time, Round Robin, Context switching.*

1. Introduction

In computing system many processes are created. These processes are in need of one or more system resource(s) that are highly limited. It implies that processes will have to compete for these available resources. Since that is the case, how are these resources allocated to these processes? Which process should wait for a resource and for how long? Which process should be assign a resource (CPU)? These questions are answered by a technique used by the operating system called ***scheduling***. The basic CPU scheduling algorithms are: First come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling and Round Robin (RR).

The idea of Round Robin (RR), one of the basic CPU scheduling algorithms is to allocate equal time slice (quantum time) in a circular manner to processes in the ready queue. Peradventure the quantum time is greater or equal to the burst time of the process it will run to completion without being preempted. Otherwise, the process must be preempted after it must have exhausted its quantum time and then return to the tail of the ready queue to take turn. The beauty of RR is fairness in assigning CPU to all the processes in the ready queue because equal time slice is given to each. Its greatest challenge is what should be the quantum time. Having a small quantum time will increase the number of context switching thereby reducing the general performance of the system. A larger one will practically degrade the system to First Come First Serve (FCFS) scheduling.

Just like any other CPU scheduling algorithm, RR has its peculiar features that make it unique. Each time RR algorithm is evaluated against SJF and FCFS through any of the evaluation techniques; it average waiting time and average turnaround time is always higher. But it has gained more popularity and application in time sharing systems, and as such the most widely use CPU scheduling algorithm. This is to say that apart from average waiting time and average turnaround time, there are other factors that were considered for it acceptance. Among these factors are: multiprogramming, response time and so on.

1. 1. Preliminaries

A process is an instance of a program in execution [5]. You might think of it as the collection of data structures that fully describes how far the execution of the program has progressed [5]. Processes are like human beings: they are generated, they have a more or less significant life, they optionally generate one or more child processes, and eventually they die. A program by itself is not a process; a program is a passive entity, such as a file containing a list of instructions stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction and set of associated resources. A program becomes a process when an executable file is loaded into memory [10]. It is these processes that are schedule for resources (CPU), and these processes can be in the following states:

- ***New:*** when a new process is just created.
- ***Running:*** the process is being executed.
- ***Waiting:*** the process is waiting for some event to occur such as I/O event and so on.
- ***Ready:*** the process is ready to be assign a processor.
- ***Terminated:*** the process has finished execution.

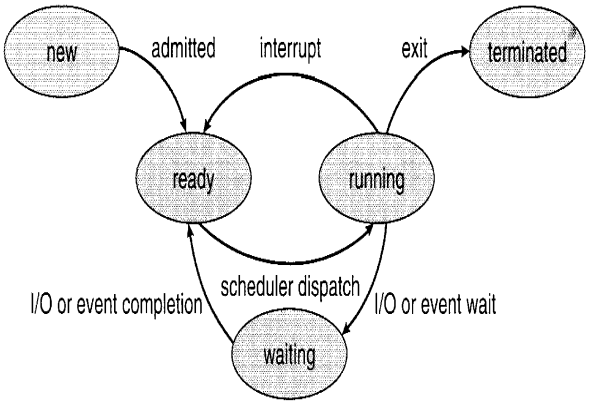


Fig 1: Process state diagram

Fig 1 shows that when a new process is created it is being admitted in to the ready queue. The scheduler will dispatch the process in the ready queue for the processor. At this point, the process is said to be in a running state. The running process upon completion will exit thereby changing its state to terminated. Sometimes, running process may be preempted caused by an interrupt. This will force the running process to change its state to ready state which will be schedule later. Also, a running process may change its state to waiting state because it is waiting for an I/O event to occur. Similarly, the same process in its waiting state may return to ready state upon completion of I/O event. It is important to note that only one process can be assign a CPU at a time. However, many processes may be waiting and ready at the same time.

3. Scheduling criteria

- CPU utilization: The idea is to keep the CPU as busy as possible. This criterion should be maximized. The CPU should be busy 100%.
- Throughput: This is the number of tasks that can be completed per unit amount of time. If the CPU is highly utilized, then the number of tasks that can be completed within a time unit will be high. Just as CPU utilization, throughput should be maximized.
- Turnaround Time: This is concern on how long it takes to finish executing a process. The time a process will take from when it is submitted for execution to when it finishes execution. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O [10]. The turnaround time should be minimized.
- Waiting Time: This is the sum of the time spend waiting in the ready queue. As for waiting time, the goal is to minimize it.
- Response Time: In an interactive system, turnaround time may not be the best criterion. Often, a process may produce some output fairly early and continue computing new results while previous results are being output to the user. Response time is the time from the submission of a request to when the first response is produced. It is the time taking to starts responding. This also should be minimized.
- Number of context switching: Context switching (CS) is the act of switching the CPU to another process while performing a state save of the current process and a state restore of a different process. Even though switching is pure overhead because the system does no useful work while switching, it is needed in time sharing systems [10]. So, an optimal switching is required for high system performance.

2. Motivation

The major challenge of RR algorithm is what should be the quantum time (QT). Having a small quantum time will increase the number of context switching thereby reducing the general performance of the system. A larger one will practically degrade the system to First Come First Serve (FCFS) scheduling. The classical RR uses the average of processes in the ready queue as its QT, and it is static. But this will always create even greater problems of unnecessary context switching. This will lead to poor average waiting time, poor average turnaround time and poor number of context switch. If a process using the CPU is preempted with a little left over time, preempting the process should be considered unnecessary. The unnecessary switching that was done shall lead to poor average waiting time and poor average turnaround time. In this kind of scenario, if some of the preempted processes with little left over time are allowed to run to completion without preemption, it will produce a better result.

4. Related Works

In recent past years, various researches have been conducted to improve on the setbacks on the classical Round Robin CPU scheduling algorithm. Among these are: Variable Quantum Time (VQT) algorithm which is based on averaging technique to allocated a variable quantum time (QT) to each process in the ready queue[13]. In Even Odd Round Robin (EORR), there are two QT (QT1 and QT2). QT1 is the average of processes that are in odd position in the ready queue, while QT2 is the average of processes in even position in the ready queue. QT1 is compared to QT2 and the greatest is use as the QT in that round [6]. Dynamic Quantum Time using Mean Average

uses the average of processes in the ready queue in each cycle as its QT [1]. It implies that each cycle will have a different QT. In Average Mid Max Round Robin (AMMRR), quantum time is the mean of the summation of the average and the maximum burst time of the processes in the ready queue in each cycle. As for Ascending Quantum Minimum and Maximum Round Robin (AQMMRR), QT is calculated by multiplying the summation of the minimum and maximum CPU burst by 80 percent [2]. Multi Dynamic Quantum time Round Robin (MDQTRR) uses two different QT in single cycle. Up to the median process, the quantum time used is gotten using the median quartile formula MQT (Median Quantum Time) while for the succeeding processes, the Upper Quartile formula is used to calculate the quantum time, UQT (Upper Quartile Quantum time) [4].

5. Proposed Approach

The main concern of DRRCP technique is the control applied on preemption to processes that are using the CPU in RR scheduling. As much as possible, DRRCP tries not to ruin the basic RR idea but to improve on its preemptive technique. The classical RR uses the average of processes in the ready queue as the quantum time, so is DRRCP. The only addition to classical RR is avoidance to unnecessary preemption of processes. This is the reason why this technique has a dynamic QT as opposed to the classical RR which has a static QT. Let us look at it this way, why should a process be preempted with left over job of 5 percent or less having completed 95 percent of its job? In fact, there are cases where 1 or less than 1 percent is preempted. At least, even if multiprogramming cannot be achieved in a particular set of processes in the ready queue, you should be able to achieve minimal average waiting time and average turnaround time. But this is not the case with classical RR, it always incur unnecessary preemption cost which can be avoided. The proposal is if the quantum time allows a process to execute up to 95 percent of its job, it should be allowed to execute its left over job without being preempted. In this scheduling, a process using the CPU may or may not be preempted even if its quantum time is exhausted. It can only be preempted if its quantum time finishes and within the time slice it only processed less than 95 percent of its job. On the other hand, if the quantum time finishes and greater than or equal to 95 percent of the job is processed, that process should be allowed to run to completion, otherwise, it should be preempted. This technique can be applied to all RR algorithms, classical or dynamic. It gives priority to processes using the CPU while having a smaller left over time of 5 percent or less to run to completion. The dynamism has to do with allowing a process not to be preempted unnecessarily. As long as 95 percent of a process job is executed, quantum time increases automatically from the average value to its actual CPU burst.

Algorithm 1: DRRCP ALGORITHM

1. //N= Number of processes
//P_i= i^{th} Process
//i=1 Loop variable
QT = quantum time
//BT= Burst time of the processes

2. While(RQ !=NULL)
// RQ= Ready Queue
Set Sum=0, Count=0
// Count= Counts number of processes in the ready queue.
//Calculation of Quantum time (QT)
for i=1 to N Loop
{ Sum = Sum + BT_i
Count++ }
QT =Sum/Count // take the floor value

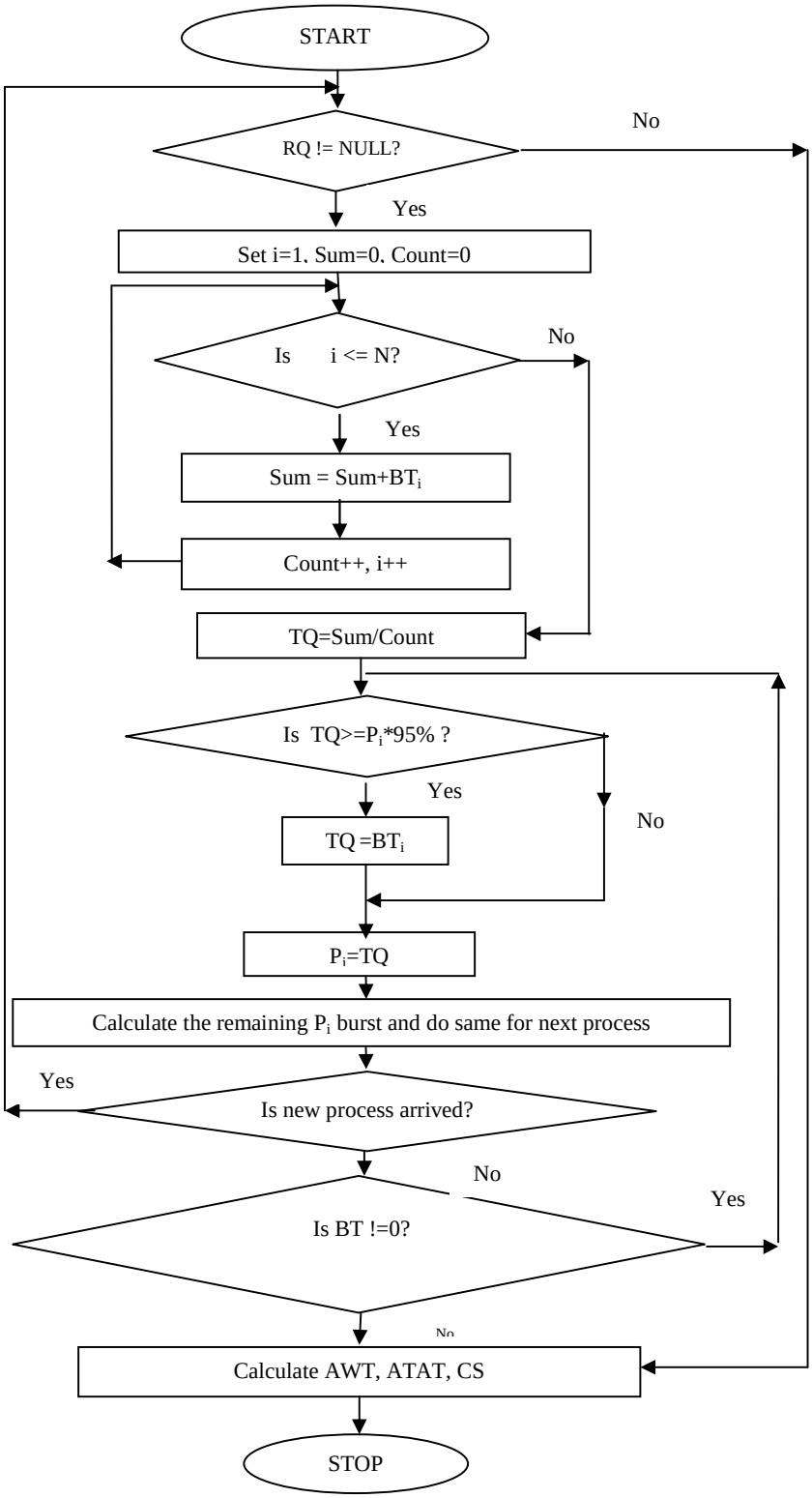
3. // Assign QT to (1 to N) processes.
for i=1 to N loop
{ If P_i*95% >= QT
QT = BT_i
Else
QT: remain unchanged
End if }
P_i=QT
Calculate the remaining P_i Burst time of the process.
End of for

4. If (new process arrived)
then go to step1
else if (new process is not arrived and BT!=0)
go to step 3
else
go to step 5
end of if
end of while

5. Calculate AWT, ATAT, CS
//ATAT=Average Turnaround time.
//AWT=Average waiting time.
//CS=Number of context switch.

6. End

Figure 1: DRRCP FLOW CHART



6. Illustrations/Analysis

Case1:
Using mean =80 and deviation=60, the following processes and their associated CPU burst are generated.
{P1=110, P2=89, P3=113, P4=137, P5=86, P6=131, P7=95}

6.1. Classical RR

In a classical RR the quantum time is the average of processes CPU burst time in the ready queue.
Quantum time (QT) = (110+89+113+137+86+131+95)/7 =761/7 = 109.
The following left over time are obtained after applying Round Robin:
Processes left over time are:: P1=1, P3=4, P4=28, P6=22. These lefts over will be use in round two (2) with same QT.

6.1.1. Analysis

Context switching: According to case1, in classical RR, P1, P3, P4 and P6 having associated left over time of 1, 4, 28 and 22 respectively, went for second round while the rest ran to completion in the first round. They displayed some level of multiprogramming in which some were unnecessary. What is the use of preempting P1 and P3 having just a little left over time of 1 and 4 respectively? If P1 with burst time of 110 and P3 with burst time 113 will be allowed to run up to QT of 109 time unit in their first round, it does not make much sense to preempt these processes for just 1 and 4 left over time for P1 and P3 respectively. It may not be a good practice to preempt a process which is very close to finishing its task. P1 and P3 do not deserve to go for the next round. In other word, their switching time was not good at all. This problem is common in an ideal classical RR. Their average results are:
AWT=509, **ATAT**=617.71, **CS**=11

6.2. DRRCP

In DRRCP, QT is the same as the classical RR. That is:
Quantum time (QT) = (110+89+113+137+86+131+95)/7 =761/7 = 109. It only addition is that a process may run to completion even if it should have a left over time. Once its QT will allow up to 95 percent a process' job to be executed, it will run to completion, otherwise it will be preempted.

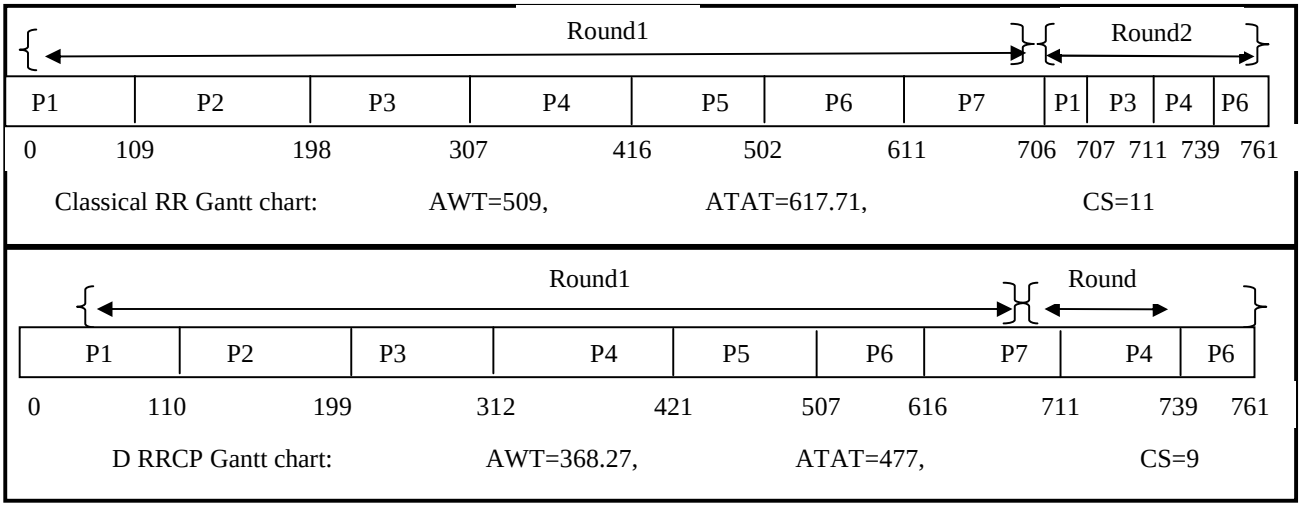
Method:
Step1: Calculate 95% of each processes burst.
P1=110, its 95% = 110*0.95=104.5, P2=89, its 95%= 89*0.95=84.6, P3=113, its 95%=113*0.95=107.4, P4=137, its 95%=137*0.95=130.1, P5=86, its 95%=86*0.95=81.7, P6=131, its 95%=131*0.95=124.5, P7=95, its 95%=95*0.95=90.3.
Step2: Compare QT (109) with 95% of each processes burst.
Once the QT is greater than or equal to 95 percent of the process' burst, that process will be allowed to run to completion. In this case, it implies that P1, P2, P3, P5, and P7 will run to completion. But P4 and P6 will be preempted for the next round because QT is less than 95 percent of each of their processes burst. In the case of P4 and P6, the QT which is 109 will be use, leading to left over time of 28 and 22 for P4 and P6 round respectively. This will then be use in the second round.

6.2.1. Analysis

Context switching: According to case1, In DRRCP, P4 and P6 are the only processes that went for second round. All the rest processes ran to completion in the first round. Surely, this takes care of P1 and P3 that went for second round in the classical RR which were considered unnecessary. It provides a better switching time for P1 and P3 by allowing them to run to completion. This solution is common with DRRCP. Their average results are:
AWT=368.27, **ATAT**=312, **CS**=9

Figure 1 below shows the Gantt between Classical RR and DRRCP of case 1.

Fig 2: Classical RR and DRRCP Gantt chart



Case2:
Using mean=54 and deviation=32, the following processes and their associated CPU burst are generated.{P1=74, P2=62, P3=51, P4=61, P5=64, P6=58, P7=46}
QT=(74+62+51+61+64+58+46)/7=59

Classical RR:
Context switching: P1, P2, P4 and P5 went for second round while the rest ran to completion in the first round. After the first round, P1 has left over time of 15, P2 has 3, P4 has 2 and P5 has 5. At least, looking at their left over time, P2 and P4 were not supposed to go for the second round. Their average results are:
AWT=304.43, **ATAT**=363.86, **CS**=11

DRRCP:
Context switching: Only P1 and P5 went for second round while the rest ran to completion in the first round. The problem encountered for allowing P1 and P5 to go for next round in the classical RR is solve here. P2 and P4 need not to go for next round with just left over time of 2 and 3 time unit respectively.
AWT=240.43, **ATAT**=299.85, **CS**=9

Case3:
Using mean=163 and deviation=76, the following processes and their associated CPU burst are generated.
{P1=191, P2=187, P3=191, P4=219, P5=202, P6=178, P7=165}
QT= (191+187+191+219+202+178+165)/7=190

Classical RR:
Context switching: Only P1, P3, P4 and P5 went for the next round having associated burst of 1, 1, 29 and 12 respectively, while the rest ran to completion in the first round. Clearly, P1 and P3 were not supposed to go the second round. Their left over burst is too little which should have been completed in the first round.
AWT=956.57, **ATAT**=1147, **CS**=11

DRRCP:
Context switching: Only P4 and P5 went for the next round while the rest ran to completion in the first round. The problem encountered for allowing P1 and P3 to go for next round in the classical RR is solve here. P2 and P4 need not to go for next round with just left over time of 1 for each. They ran to completion.
AWT=696.86, **ATAT**=887.29, **CS**=9

7. Description of the Simulation

The simulator is designed using Visual Basic 6.0 as the programming language and was executed on windows 7operating system. The size of the ram used is 2.0GB, the speed is 1.56GHz and the size of the hard disk is 300GB. The simulator considers all processes to be in the ready queue and their arrival time set to be zero. Also, processes are considered to be of same priority. The simulator uses normal distribution function which requires three input parameters to generate processes

and their associated CPU burst. The input parameters are: mean (μ), standard deviation (σ), and number of process. These parameters are variables, meaning they can assume any value. The maximum number of process is set to be 100 while the maximum number of the mean and standard deviation is set to be 1000 each. When the program is executing, it will request the user to supplier number of process, the mean and the standard deviation. The simulator computes the average waiting time, average turnaround time, and number of context switching for each algorithm (Classical RR and DRRCP). The results of the experiments are given in the table and figure below when the simulator is run for 10 times with number of processes starting with 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. The simulator is able to find the average waiting time, average turnaround time and the number of context switch. Fig 3 below shows the interface of the simulator.

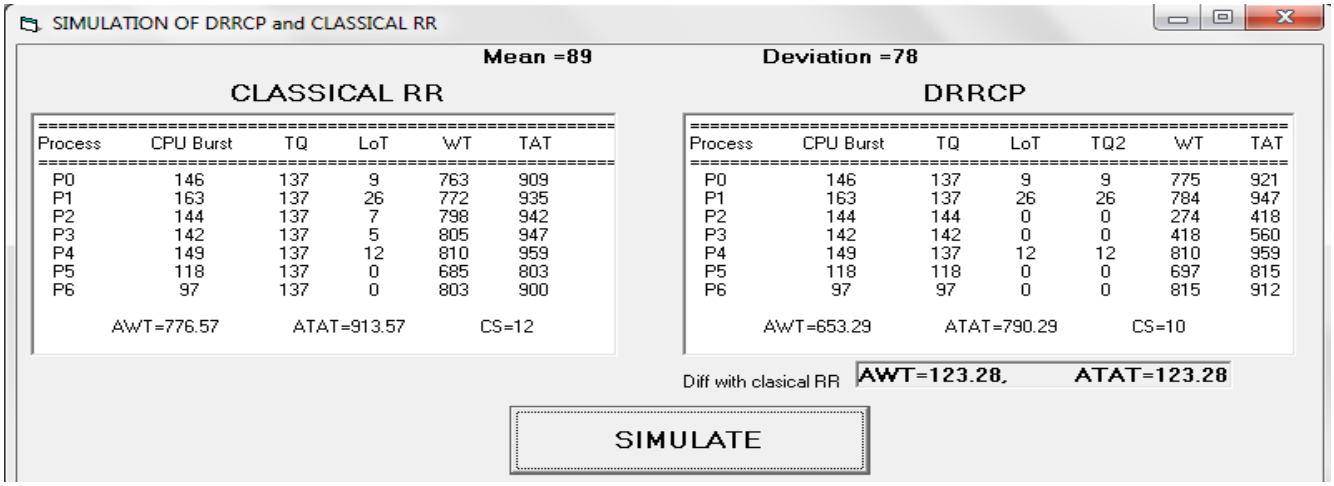


Fig 3: Interface of the simulator

Simulation result between Classical RR and DRRCP								
			Classical RR			DRRCP		
No of Process	Deviation	Mean	AWT	ATAT	CS	AWT	ATAT	CS
10	67	98	854.7000	995.0000	14	799.000	939.3000	13
20	321	107	2792.700	3041.700	30	2597.700	2846.700	29
30	131	307	7404.330	7777.270	44	6863.700	7236.630	40
40	356	407	16752.18	17351.60	59	16189.08	16788.50	57
50	543	589	30987.04	31841.02	76	29153.40	30007.38	72
60	354	453	26860.93	27492.00	92	24069.20	24700.27	84
70	754	687	50562.21	51606.37	105	46628.97	47673.13	98
80	567	832	62758.11	63850.98	123	53881.90	54974.76	105
90	765	876	78786.34	80052.39	135	74473.78	75739.82	128
100	895	986	101629.4	103074.75	152	95753.86	97199.21	145

Table 1

Table 1 above shows that when the simulator is run for 10 processes, using a deviation of 67 and mean of 98, the classical RR will have AWT to be 854.7 time unit, ATAT to be 995 time unit, and number of context switching to be 14, while the DRRCP will have AWT to be 799, ATAT to be 939.3 and contexts switching to be 13. The procedure is repeated when number of process increases from 10 to 100 with variable mean and standard deviation. The results gotten are recorded in table 1. The below charts of Average Waiting Time (AWT), Average Turnaround Time (ATAT) and number of context switching (CS) were generated from table 1.

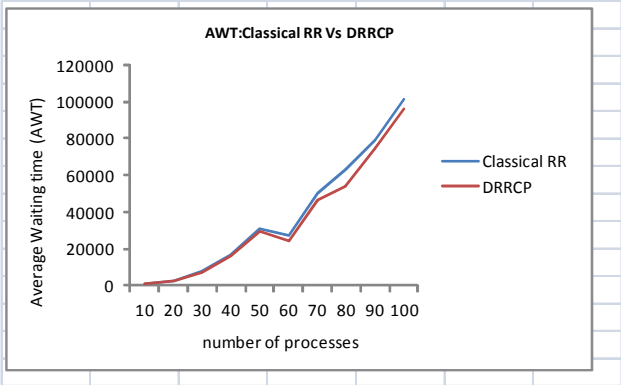


Chart 1: average waiting time (AWT) graph

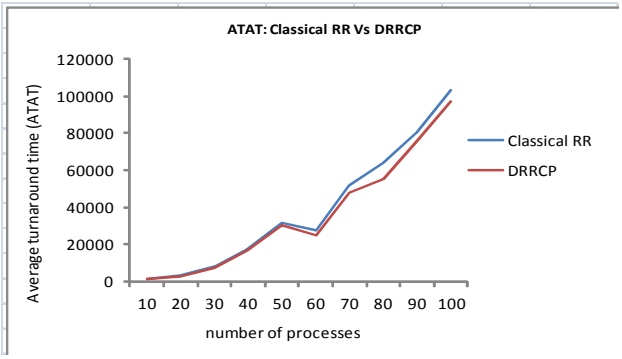


Chart 2: average turnaround time (ATAT) graph

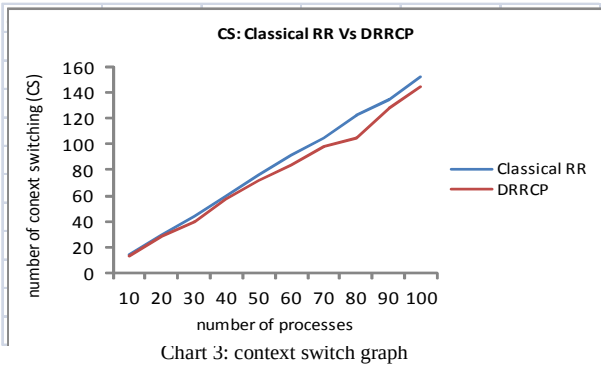


Chart 3: context switch graph

From chart 1, 2, and 3 shown above, it is clear that DRRCP is a direct improvement of the classical RR. DRRCP minimizes average waiting time, average turnaround time and number of context switching than its classical RR counterpart.

8. Why DRRCP is better than Classical RR

DRRCP algorithm tries as much as possible to preserve the characteristics of the classical RR. It changes nothing but support the classical RR. The quantum time calculation and the operation are the same. The only addition to this algorithm is the ability to controlled unnecessary preemption so as to achieve optimal performance. On one hand, it may work exactly as classical RR, preempting processes whose CPU burst are above average. But on the other hand, processes that are due for preemption are executed to completion without being preempted. As long as 95 percent of a process job is executed, that process will not be preempted even if its QT has been exhausted. It may be considered as process blocking because process that is due for preemption may or may not be preempted for certain reason. The simple reason is that: a particular process that has executed 95 percent of its job, it may not be wise to be preempted. For example, consider a process with CPU burst of 89ms having a quantum time of 88ms. This process will be preempted after running for 88ms leaving left over time of 1ms. This does not make much sense. This is because if it must be allowed for that long time of 88ms, it should be allowed for just 1ms to run to completion so as to minimize average waiting time, average turnaround time and as well as reduce the cost of performing context switching.

Another issue is a situation whereby a process will context switched itself. For example, let say P1=88ms and P2=89ms while QT=88ms. P1 will run to completion, P2 will run for just 88ms with left over time of 1ms. This same P2 will have to context switched itself. That is, P2 being the last and the only process will have to go for second round as a result of timer interrupt that will be generated. Even though its waiting time will not increase but the switching has now become an overhead.

The reason for chosen DRRCP to be better than the classical RR is: given any dataset, DRRCP will perform the same or better than the classical RR. Mathematically, it can be stated as performance for DRRCP >= Performance of classical RR algorithm. The justification for this equation is that depending on the given dataset, DRRCP may work purely as classical RR. If the data set may not favor classical RR, it will adjust itself and perform better than classical RR. On a final note, the concept of DRRCP algorithm can also fit into any of the proposed dynamic RR CPU scheduling algorithms.

7. Conclusion

This has demonstrated clearly why DRRCP is better than the classical RR. The QT is made dynamic for maximum improvement. By simply performing a checking, unnecessary context switching encountered in the classical RR are avoided. Through this simple technique, average waiting time, average turnaround time and context switching are greatly improved.

References

[1] Abbas N, Ali K and Seifedine K., 2011. "A New Round Robin Based Scheduling Algorithm for operating systems: Dynamic Quantum using the mean average". IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No.1. pp. 224-229.

[2] Ali D.J. 2012. "Improving efficiency of Round Robin scheduling using Ascending Quantum and Minimum-Maximum burst time", J. of university of anbar for pure science. Vol.6:NO.2. pp.23-27.

[3] Bashir A., Doja1 M.N., Biswas R., and Alam .M. 2011. "Fuzzy Priority CPU Scheduling Algorithm". IJCSI International Journal of Computer Science Issues, Vol. 8. pp. 386-390

[4] Behera H.S., Rakesh M., Sabyasachi S. and Sourav B.K. 2011. "Comparative performance analysis of Multi-dynamic Quantum time Round Robin (MDQTRR) Algorithm with Arrival Time". Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2. pp. 262-271.

[5] Bovet D. P, Cesati M. 2006. 3rd edition, Understanding the Linux Kernel, O'Reilly Media, Inc., USA. pp. 1- 923.

[6] Pallab B., Proba B. and Shweta D.S. 2012a. "Comparative Performance Analysis of Even Odd Round Robin Scheduling Algorithm (EORR) using Dynamic Quantum time with Round Robin Scheduling Algorithm using static Quantum time". International Journal of Advanced Research in Computer Science and Software Engineering. Volume 2. pp. 62-70.

[7] Pallab B. Proba B. and Shweta D.S. 2012b. "Comparative Performance Analysis of Average Max Round Robin Scheduling Algorithm (AMRR) using Dynamic Quantum time with Round Robin Scheduling Algorithm using static Quantum time", International Journal of Innovative Technology and Exploring Engineering (IJITEE). Volume-1. pp. 56-62.

[8] Pallab B., Proba B. and Shweta D.S. 2012c. "Performance Evaluation of a New Proposed Average Mid Max Round Robin (AMMRR) Scheduling Algorithm with Round Robin Scheduling Algorithm". International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2. pp.143-151.

[9] Puneet V.K., Nadeem A. and Faridul S.H. 2012. "Efficient CPU Scheduling Algorithm Using Fuzzy Logic". International Conference on Computer Technology and Science, IPCSIT vol. 47. pp. 13-18.

[10] Silberschatz A, Galvin P.B. and Gagne .G, 2005, Operating Systems Concepts, (7th ed), John Wiley and Sons, USA. pp. 1-885.

[11] Sunita B., Bhavik K. and Chittaranjan H. 2011. "Dynamic Task-Scheduling in Grid Computing using Prioritized Round Robin Algorithm". IJCSI International Journal of Computer Science Issues, Vol. 8. pp. 472-477.

[12] Tanenbaun A.S. 2008. "Modern Operating Systems". (3rd ed), Prentice Hall, pp. 1104.

[13] Yashasvini S. 2013. "Determining the Variable Quantum Time (VQT) In Round Robin and importance over Average Quantum Time Method", International Journal of Science, Engineering and Technology Research (IJSETR) Volume 2. pp. 613-617.