IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-4, Special Issue-1 E-ISSN: 2347-2693

Weight-based Starvation-free Improvised Round-Robin (WSIRR) CPU Scheduling Algorithm

Dr. Anuradha Banerjee*
Assistant Professor
Kalyani Govt Engg. College
anuradha79bn@gmail.com

Nabajit Mondal, Naznin, Poulomi Basu, Prateeksha Singh
Student, Dept. of Computer Applications
Kalyani Govt Engg. College

***Abstract*—CPU is a primary computer resource, so its scheduling is central to operating system design. When multiple runnable processes exist in the ready queue, OS has the onus of responsibility to decide which one is to run fast. The part of the OS that takes this decision is called scheduler and the algorithm based on which it works, is called the scheduling algorithm. Different kinds of scheduling algorithms exist in the literature. Among them First-Come-First-Served (FCFS), Shortest-job-first (SJF), Priority Scheduling and Round-robin, are mention-worthy. This paper proposes a weight-based starvation-free improvised scheduling algorithm that allocates CPU to processes in round-robin manner while the time quantum is calculated based on the burst time of the processes waiting in the ready queue.**

***Keywords*—** *CPU, Processes, Ready Queue, Round-Robin, Scheduling, Weight*

1. INTRODUCTION

Scheduling is one of the basic functions of any operating system, because scheduling of all computer resources is done before their use. The success of a CPU scheduler depends highly on the design of a good quality scheduling algorithm that will produce high CPU utilization (i.e. no idle time and fewer context switches) with minimum waiting and turnaround time. The main focus of the proposed work is to develop a good quality scheduling algorithm suitable for all kinds of jobs. Below we discuss various scheduling algorithms in brief and compare their characteristics in terms of advantages and disadvantages [2-7].

A. First-come-first-served (FCFS)

In this technique, the process that was first inserted into the ready queue is allocated the processor first. Processes are inserted into the rear end as soon as they are submitted to the system. The next process is taken from the front end as soon as a process finishes execution [3].

In FCFS, there is no prioritization of processes. Every process completes eventually and hence, there is no starvation. The process with longest burst time can monopolize the CPU, even if the burst time of other processes is too short. Therefore, throughput is low.

B. Shortest-job-first (SJF)

The process that has the least burst time is allocated the CPU first. The processes are arranged in the ready queue in ascending order of burst time. It gives maximum throughput in most scenarios [2]. SJF reduces the average waiting time because it services smaller processes before the large ones.

The main problem with SJF is that it may affect the execution of processes with high burst time requests. They may be left unserved in the ready queue for a very long time (age problem) provided the processes with smaller burst times keep arriving at the ready queue. This leads to starvation [2].

C. Priority Scheduling (PS)

Here each process is equipped with some priority and processes are inserted into the ready queue in descending order of priority. At any point of time whenever the CPU is idle, it is allocated to the process residing at the front end of the queue i.e. the one with highest priority.

Starvation is a serious problem for this algorithm also because if high priority processes are continuously submitted one after the other, then a low priority process might have to wait for a very long time.

D. Round-robin(RR)

In this scheduling algorithm [5, 7, 8], a small unit of time called time-slice or time-quantum is assigned to the processes. The scheduler goes around the ready queue allocating the CPU to each and every process for the defined time-quantum. New processes are added at the rear end of the queue.

Deciding the time quantum is an important aspect of RR. If it is set too sort then it may cause a huge number of context switches resulting in lower CPU efficiency. On the other hand, if time quantum is set too long then it may cause poor response time and approximate FCFS.

Reference [12] proposes an improvised round-robin (IRR) scheduling algorithm that interestingly inculcates the flavor of SJF in RR, although the method suffers from the serious problem of starvation. Consider that the processes p1, p2, p3, p4 and p5 are submitted to the system having burst times 31, 1,3,4,4. Then ready queue will consist of the processes p2,p3,p4,p5 and p1 with the respective burst times 1,3,4,4 and 31 . Now another process p6 comes with burst time 26. It will be placed in ready queue before p1. Now assume that another process p7 comes with burst time 22. It will be placed before p6. So, p1 is starving, being pushed back continuously. WSIRR prevents this. Also the method of time quantum computation in proposed technique is such that it encourages the round-robin fashion of CPU sharing.

2. TIME QUANTUM COMPUTATION IN WSIRR

Time quantum in WSIRR is calculated in such a manner that at least $\lfloor n/2 \rfloor$ processes have burst time higher than the time quantum where n is the total number of processes in the ready queue. This keeps the round robin characteristic intact. Let the ready queue consist of n processes p1, p2, … ,pn having burst times b1, b2, …, bn respectively. Then the time quantum tq is formulated in (1).

$$tq = \lceil (\sum_{1 \le i \le n} bi) / n \rceil - k \quad (1)$$

Here k is the lowest integer (greater than or equal to 0) so that at least $\lfloor n/2 \rfloor$ processes have burst time greater than tq.

Time quantum is calculated anew each time a new process enters into the ready queue or exits from it. If the ready queue contains only one process then time quantum is it's burst time. This technique introduces dynamism in the system and reduces the number of context switching improving CPU throughput.

3. INSERTION OF PROCESSES IN READY QUEUE IN WSIRR

Processes are arranged in the ready queue in descending order of weight when the scheduling starts, whereas the weight is defined in (2). Also whenever a new process is submitted to the system, it is inserted in the ready queue in it's proper position depending upon weight. As soon as a process is served for a time quantum, it is forcibly removed from the front end and inserted into the rear end of the queue without consideration of weight at all. This is termed as ready queue reconstruction. If a new process arrives at this time it is inserted into the ready queue only after the queue reconstruction after a time quantum is over and the process that has just been inserted at the rear end does not compete with the newcomer.

$$W(Pi) = (current\_time - age(i)+1) / remaining\_burst\_time(i) \quad (2)$$

W(Pi) denotes weight of the process pi. age(i) is age of the process in ready queue after scheduling algorithm started execution. The variable current_time is self-explanatory while remaining_burst_time(i) indicates the amount of more CPU time that is required for completion of pi. If, by any means, age of a process in ready queue crosses an upper limit U then it is put at the front of the ready queue with a special time quantum equal to the remaining burst time of the process. After the process finishes, time quantum for the remaining processes is calculated newly. If the remaining burst time of currently running process

is less than one time quantum then CPU is again allocated to the currently running process for remaining burst time. This reduces context switching and waiting time.

4. THE ALGORITHM OF WSIRR

The proposed algorithm of WSIRR is as follows:

```
WSIRR_execorder( p, n)
{
 /* this function determines the order of execution, i.e. creates the gantt chart;
     P is the array of processes and n is the number of elements*/
  While ready queue Q is not empty or a new process has come or a process has finished
   {
tq=WSIRR_timequan(p,n)
    /* WSIRR_timequan function computes the time quantum and stores in tq*/
WSIRR_calweight(p,n)
   /* WSIRR_calweight function calculates the weight of individual processes*/
Sort( Q)
   /*Q is sorted in order of weight of processes according to bubblesort technique*/
fr_process = retrieve(Q)
allocate CPU to fr_process for tq time
iffr_process.rbt<tq then
    /* the attribute rbt stands for remaining burst time */
allocate CPU to fr_process for tq time again
else
insert(fr_process,Q)
   }

WSIRR_timequan(p,n)
{
tquan =0
 For i = 1 to n
tquan= tquan+p[i].rbt
tquan = tquan/n;
cntr = 0
while(cntr is not equal to -9)
      {
       For i = 1 to n
         If (tquan< p[i].rbt)
cntr = cntr +1
if (cntr is greater than or equal to ⌈n/2⌉) then
cntr = -9
else
tquan = tquan -1
                  }
               }

WSIRR_calweight(p,n)
 {
For i = 1 to n
W(p[i])= (curtime-arrvaltime+1)/p[i].rbt
}
```

5. ILLUSTRATION OF EXECUTION IN WSIRR

Example1: Please consider the arrangement of processes in table 1.

Process id	Burst time	Arrival time in ready queue
P1	15	0
P2	3	10
P3	8	10
P4	4	10

Phase - 1

Scheduling starts at time 10.
W(p1)=11/15, W(p2)=1/3, W(p3)=1/8, W(p4)=1/4
Ready queue: p1 p2 p4 p3
Time quantum = 7
P1 executes from 0 to 7 and is inserted at the rear end of ready queue.

Phase - 2

W(p2)=2/3, W(p3)=2/8, W(p4)=2/4: Please note that W(p1) is immaterial here unless all three of p2, p3 and p4 have a turn.
Ready queue: p2 p4 p3 p1
Time quantum = 6
P2 executes from 7 to 10 and is finished.

Phase - 3

W(p3)=3/8, W(p4)=3/4Ready queue:p4 p3 p1
Time quantum = 7
P4 executes from 10 to 14 and is finished.

Phase - 4

W(p3)=1/8:
Ready queue: p3 p1
Time quantum = 8
P3 executes from 14 to 22 and is finished.

Phase-5

P1 executes from 21 to 30 and is finished.

The Gantt chart is as follows:

P1	P2	P4	P3	P1
0	7	10	14	22	30

Average waiting time = (25+7+10+14)/4 = 14
Average turnaround time = (10+14+22+40)/4 = 21.5

Comparison with IRR

In IRR, the ready queue stores the processes in ascending order of burst time; there is nothing to do with age. The time quantum is the average of burst times of all the processes.

Scheduling starts at time 10. Time quantum = 8

Phase-1

Ready queue: p2 p4 p3 p1
P2 executes from 0 to 3 and is finished.

Phase-2

Ready queue: p4 p3 p1
P4 executes from 3 to 7 and is finished.

Phase-3

Ready queue: p3 p1
P3 executes from 7 to 15 and is finished.

Phase-4

Ready queue: p1
P1 executes from 15 to 30 and is finished.

The Gantt Chart is as follows:

P2	P4	P3	P1

0 3 7 15 30

Average waiting time = (0+3+7+25)/4 = 8.75
Average turnaround time = (3+7+15+40)/4=16.25

Explanation for the difference in performance:-

In order to avoid starvation, which is a must-avoid in all scheduling algorithms, WSIRR gives more weight to a aged process even if it has higher burst time. For this, average waiting time and average turnaround time of processes have to be sacrificed; because if a process with higher burst time gains access to CPU before the other with smaller burst times, waiting time for those smaller burst time processes increase and as a result, turnaround time also increases in WSIRR compared to IRR. But this is acceptable in the context of fair sharing i.e. no starvation. Also, it is noticeable that a process with high weight is not allowed to monopolize the CPU in WSIRR. Because as soon as a its time quantum finishes, it is inserted at the tail of ready queue and cannot compete unless all other waiting processes have a turn.

Example2: Please consider the arrangement of processes in table 2.

Process id	Burst time	Arrival time in ready queue
P1	6	0
P2	1	0
P3	8	0
P4	5	5
P5	9	6
P6	21	7
P7	28	8
P8	35	9

Phase - 1

Scheduling starts at time 0.
Time = 0

W(p1)=1/6, W(p2)=1, W(p3) = 1/8
Ready queue: p2 p1 p3
Time quantum = 7
P2 executes from 0 to 1 and is finished.

Phase - 2

Time = 1
W(p1)=2/6, W(p3) = 2/8
Ready queue: p1 p3
Time quantum = 7
P1 executes from 1 to 7 and is finished.

Phase - 3

Time = 7
W(p3) = 1, W(p4)=3/5, W(p5)=2/9, W(p6)=1/21
Ready queue: p3 p4 p5 p6
Time quantum = 8
P3 executes from 7 to 15 and is finished.

Phase - 4

Time = 15
W(p4)=11/5, W(p5)=10/9, W(p6)=9/21, W(p7)=8/28, W(p8)=7/35
Ready queue: p4 p5 p6p7p8
Time quantum = 20
P4 executes from 15 to 20 and is finished.
Phase - 5

Time = 20
W(p5)=16/9, W(p6)=15/21, W(p7)=14/28, W(p8)=13/35
Ready queue: p5 p6p7p8
Time quantum = 27
P5 executes from 20 to 29 and is finished.

Phase - 6

Time = 29
W(p6)=23/21, W(p7)=22/28, W(p8)=21/35
Ready queue: p6p7p8
Time quantum = 28
P6 executes from 29 to 50 and is finished.

Phase - 7

Time = 50
W(p7)=43/28, W(p8)=42/35
Ready queue: p7p8
Time quantum = 32
P7 executes from 50 to 78 and is finished.

Phase - 8

Time = 78

W(p8)=70/35
Ready queue: p8
Time quantum = 35
P8 executes from 78 to 113 and is finished.

The Gantt chart is as follows:

P2	P1	P3	P4	P5	P6	P7	P8

0 1 7 15 20 29 50 78 113

Average waiting time = (0+1+7+10+14+22+42+69)/8 = 19.375
Average turnaround time = (1+7+15+15+24+43+70+104)/8 = 34.875

Comparison with IRR

In IRR, the ready queue stores the processes in ascending order of burst time; there is nothing to do with age. The time quantum is the average of burst times of all the processes.

Scheduling starts at time 0. Time quantum = 8

Phase-1

Ready queue: p2 p1p3
P2 executes from 0 to 1 and is finished.

Phase-2

Ready queue: p1p3
P1 executes from 1 to 7 and is finished.

Phase-3

Ready queue: p4 p3 p5 p6
P4 executes from 7 to 12 and is finished.

Phase-4

Ready queue: p3 p5 p6 p7 p8
P3 executes from 12 to 20 and is finished.

Phase-5

Ready queue: p5 p6 p7 p8
P5 executes from 20 to 28, then from 28 to 29 and is finished.

Phase-6

Ready queue: p6 p7 p8
P6 executes from 29 to 37.

Phase-7

Ready queue: p7 p8 p6

P7 executes from 37to 45.

Phase-8

Ready queue: p8 p6p7
P8 executes from 45to 53.

Phase-9

Ready queue: p6p7p8
p6 executes from 53to 61.

Phase-10

Ready queue: p7p8p6
P7 executes from 61 to 69.

Phase-11

Ready queue: p8p6p7
P8 executes from 69to 77.
Phase-12

Ready queue: p6p7p8
P6 executes from 77to 82 and is finished.

Phase-13

Ready queue: p7p8
P7 executes from 82to 90.

Phase-14

Ready queue: p8p7
P8 executes from 90to 98.

Phase-15

Ready queue: p7p8
P7 executes from 98to 102 and is finished.

Phase-16

Ready queue: p8
P8 executes from 102 to 113.

The Gantt chart is as follows:

P2	P1	P4	P3	P5	P6	P7	P8	P6	P7	P8	P6	P7	P8	P7	P8	P8

0 1 7 12 20 29 37 45 53 61 69 77 82 90 98 102 110 113

Average waiting time = (0+1+2+12+14+54+67+69)/8 = 27.375
Average turnaround time = (1+7+7+20+23+75+94+104)/8 = 41.375

Explanation for the difference in performance:-

From the above example it is seen that WSIRR produces much lesser average waiting time and turnaround time when processes with larger burst time start arriving at the ready queue after the scheduling has started. This is due to the dynamic time quantum computation facility of WSIRR which is not possible in IRR. As large burst time processes arrive in ready queue in WSIRR, the time quantum increases significantly decreasing the number of context switches and thereby decreasing the average waiting time. On the other hand, when processes with smaller burst times arrive in the ready queue, WSIRR considers their ages and assigns weights accordingly.

6. CONCLUSION

The proposed algorithm WSIRR produces better performance than IRR (IRR produces better results than FCFS and RR [12]) when processes are submitted with larger burst times after scheduling has started. Otherwise, whenever the need arises to relieve a process from starvation, WSIRR sacrifices average waiting and turnaround times.

REFERENCES

[1]. http://en.wikipedia.org/wiki/Scheduling_(computing)

[2]. Sindhu M, Rajkamal R, Vigneshwaran P. An Optimum Multilevel CPU Scheduling Algorithm.2010 International Conference on Advances in Computer Engineering

[3]. Wei Zhao, John A. Stankovic. Performance Analysis of FCFSand Improved FCFS Scheduling Algorithms for DynamicReal-Time Computer Systems.IEEE 1989.

[4]. DavenderBabbar, Phillip Krueger. A Performance Comparison of Processor Allocation and Job Scheduling Algorithms for Mesh-Connected Multiprocessors.IEEE 1994.

[5]. Umar Saleem and Muhammad YounusJaved. Simulation Of CPU Scheduling Algorithms. IEEE 2000.

[6]. SnehalKamalapur, Neeta Deshpande. Efficient CPU Scheduling: A Genetic Algorithm based Approach. IEEE 2006.

[7]. Nikolaos D. Doulamis, Anastasios D. Doulamis, Emmanouel A. Varvarigos, and Theodora A. Varvarigou. Fair Scheduling Algorithms in Grids. IEEE Transactions On Parallel And Distributed Systems, Vol. 18, No. 11, November 2007

[8]. Xiao-jing Zhu, Hong-boZeng, Kun Huang, Ge Zhang. Round-robin based scheduling algorithms for FIFO IQ switch. IEEE 2008.

[9]. Apurva Shah, KetanKotecha. Efficient Scheduling Algorithms for Real-Time Distributed Systems. 2010 1st International Conference on Parallel, Distributed and Grid Computing

[10]. DevendraThakor, Apurva Shah. D_EDF: An efficient Scheduling Algorithm for Real-Time Multiprocessor System.IEEE 2011.

[11]. Tong Li, Dan Baumberger, Scott Hahn. Efficient and Scalable Multiprocessor Fair Scheduling Using Distributed Weighted Round-Robin. ACM 2009

[12]. A. Sirohi, A. Pratap, M. Aggarwal, Improvised Round-Robin CPU Scheduling Algorithm, International Journal of Computer Applications, vol. 99, no. 18, August 2014

.