This full-text paper was peer-reviewed and accepted to be presented at the IEEE WiSPNET 2016 conference.

Efficient Implementation of Multilevel Feedback Queue Scheduling

Malhar Thombare, Rajiv Sukhwani, Priyam Shah,Sheetal Chaudhari, Pooja Raundale
malharthombare@gmail.com, rajiv.sukhwani96@gmail.com, priyamshah0810@gmail.com,
sheetal_chaudhari@spit.ac.in, pooja@spit.ac.in
Sardar Patel Institute of Technology Andheri, Mumbai

***Abstract*—In CPU scheduling various algorithms exist like FCFS (First come first serve), SJF (Shortest job first), SRTF (Shortest remaining time first), Priority Scheduling, Round Robin (RR), MLQ (Multilevel queue), MLFQ (Multilevel feedback queue) scheduling. Multilevel Feedback Queue (MLFQ) algorithm allows the switching of processes between queues depending on their burst time. The processes switch to the next queue when burst time is greater than time quantum. Each queue can define its own scheduling policy. In this paper we have implemented MLFQ technique using small burst time for the first queue thus making it analogous to RR scheduling and using SJF prior to RR from second queue onwards gives better CPU utilization. Dynamic time quantum is also used which further improves the efficiency of the scheduling. Here the dynamic time quantum of the queues is calculated based on the burst time of the processes. Time quantum of the second queue is the burst time of the $(2n/3)^{th}$ process (where n is the number of processes remaining after the execution in the first queue) and time quantum of the third queue is burst time of the largest remaining process. Thus 66% of the processes get executed in the second queue and remaining processes in the last queue thus preventing the problem of starvation of huge burst time processes.**

Index Terms— CPU scheduling, Multilevel Feedback Queue, Average Waiting Time, Dynamic Time Quantum, Average Turnaround Time.

I. INTRODUCTION

The efficiency of multitasking operating systems primarily depend on the scheduling algorithm used by the CPU. The selection of scheduling policy affects the process response time (amount of time taken by process to get CPU), average waiting time (amount of time a process waits for its complete execution) and may also have impact on starvation (it occurred if a process is not getting CPU for a long time).Currently, many CPU scheduling algorithms are existing like FCFS, SJF, SRTF, Round Robin Scheduling, Priority Scheduling, MLQ and MLFQ. MLFQ may be one of most powerful strategy for CPU scheduling. It is an extended version of MLQ scheduling algorithm also Multilevel Queue Scheduling is result of combination of scheduling algorithms like FCFS and RR scheduling algorithms.

Allocating CPU to a process requires careful attention to prevent process CPU starvation. Different CPU scheduling algorithms have different objectives and may favor one type of process over another. Many performance measures have been suggested for comparing CPU scheduling algorithms. Performance measures used include CPU utilization, throughput, waiting time, response time, turnaround time, scheduler efficiency and context switching.

A good scheduling algorithm is the one that optimizes the performance measures. The optimization performance measures are maximizing CPU usage, maximizing throughput, minimizing turnaround time, minimizing waiting time, minimizing response time, minimizing the context switching.

In Multilevel Queue, processes are not allowed to switch among queues whereas, in Multilevel Feedback Queue (MFQ) allows the processes in ready queue to execute in several queues according to the scheduling policy applied to each queue and the process burst time. If a process does not complete its execution in one queue then it is passed to subsequent queue for execution. The process burst time and selection of time quantum for different queues affects the execution sequence and time of each process. In one of the work the time quantum for the second queue was computed as a complex function of burst time of processes and still did not fairly solve the problem of starvation for huge burst time processes also the complex approach reduced the program efficiency. The proposed approach uses an easier method of automatic quantum generation to minimize the process starvation, prioritizing all types of processes equally at the same time looks if each process is able to efficiently get a chance to get CPU time and resources and also takes care of short processes so that they do not have to wait for higher burst time processes to get executed. The implementation was improved by providing a short quantum value for first queue for the execution of short processes, SJF from further queue onwards prior to RR and an easier dynamic quantum generation thus providing a fair chance to huge burst processes by giving them 33% execution chance after process execution in the second queue.

II. EXISTING MFQ SCHEDULING APPROACH

A number of approaches have been made and researches have been carried and cited in the field of process scheduling using MFQ. To improve the efficiency authors[2] have used concept of applying SJF prior to RR from first queue onwards while some also proposed a method of implementing dynamic quantum[1] on basis of burst time of process[7] using some complex functions. The authors of different papers focused on

978-1-4673-9338-6/16/$31.00 ©2016 IEEE

both the above implementations but this also does not solve completely the basis of multilevel feedback queue as no feedback was provided from bottom level to top level queues and the lower level queue processes kept starving for CPU and resources. Through this work the problem of lower level queue starvation is solved. Also the low burst time processes waiting time and dynamic time quantum computation complexity is reduced.

III. PROPOSED ALGORITHM/ IMPLEMENTATION

The following algorithms are used to calculate the time quantum of the queues. According to the proposed algorithm the processes get executed within three queues q1, q2, q3. The time quantum of the first queue that is q1 is kept short so that the short processes get executed within the first queue itself. The time quantum of q1 is 10 so the processes within the burst time 10 get executed and the burst time are added in the variable qc. The variable qc is created so as to calculate the time quantum of the second and third queue. If the burst time is greater than 10 the processes enter the second queue. In the second queue the processes are arranged according to SJF scheduling algorithm and executed further. If the burst time exceeds the time quantum that is calculated it enters the third queue. 75% of the processes get executed in the second queue. The remaining 25% processes get executed in the third queue.

Algorithm: Quantum calculation

1. Short input array of processes according to their arrival time(processes[i][0]) and burst time(processes[i][1])
2. int qc=0(quantum value)
   int i=0(counting variable)
   int total(total no of processes)
3. While (i++<total) repeat steps 4,5
4. if(processes[i][1]<=10) burst time<10

Transfer process contents in queue1 (nq1 queue 1 variable) and add burst time value to quantum variable qc

- q1[nq1][0]=processes[i][0];
- q1[nq1][1]=processes[i][1];
- q1[nq1++][2]=i;
- qc+=processes[i][1];

5. else

Transfer process contents in queue 2,3 (j queue 2,3 variable) and add burst time value to quantum variable qc

Choose from second queue and update tt,wt,clock

- wt+=clock-q2[i][0];
- tt+=clock-q2[i][0]+q2[i][1];
- clock+=q2[i++][1];

10. if (j<nq3)

Choose from third queue and update tt,wt,clock

- q2q3[j][0]=processes[i][0];
- q2q3[j][1]=processes[i][1];
- q2q3[j++][2]=i;
- qc+=processes[i][1];

6. f=j;
   i=0;
7. While(i++<f-1) repeat step 8,9
8. j=0
9. while(j++<f-i-1) repeat step 10
10. sort processes according to burst time as in SJF
    If (q2q3 [j] [1]>q2q3 [j+1] [1])
    - q2q3[j][0]=q2q3[j][0]+q2q3[j+1][0]-(q2q3[j+1][0]=q2q3[j][0]);
    - q2q3[j][1]=q2q3[j][1]+q2q3[j+1][1]-(q2q3[j+1][1]=q2q3[j][1]);
    - q2q3[j][2]=q2q3[j][2]+q2q3[j+1][2]-(q2q3[j+1][2]=q2q3[j][2]);
11. qc=q2q3[(3*f)/4][1];(3/4 so that 75% processes are implemented under queue 2)

Algorithm: 2/3 execution chance to the current level queue 1/3 chance to lower level queue

1. i=0
2. while(i++<f) repeat steps 3,4
3. if(q2q3[i][1]<=qc)
   Transfer process in queue2 if burst time is less than calculated quantum
4. else transfer contents in queue 3
5. i=0
6. while(i++<nq2) repeat steps 7,8,9,10
7. set waiting time ,tur around time, execution clock of queue 2,3
   - wt+=clock-q2[i][0];
   - tt+=clock-q2[i][0]+q2[i][1];
   - clock+=q2[i++][1];
8. if (i<nq2)

Choose from second queue and update tt,wt,clock

- wt+=clock-q2[i][0];
- tt+=clock-q2[i][0]+q2[i][1];
- clock+=q2[i++][1];

9. if (i<nq2)
   - wt+=clock-q3[j][0];
   - q3w+=clock-q3[j][0];
   - q3tt+=clock-q3[j][0]+q3[j][1];
   - tt+=clock-q3[j][0]+q3[j][1];
   - clock+=q3[j++][1];

IV. EXPERIMENT IMPLEMENTATION AND ANALYSIS

To analyze the performance of MFQ scheduling using the simulator, CPU burst & arrival time consider the example consisting of seven processes and while doing so the system ignores the nature (CPU or I/O Bound) of each process. The arrival time and burst time of the process has to be entered by the user.

```
Enter no of processes 7
p0 0 8
p1 3 133
p2 2 21
p3 8 39
p4 19 67
p5 33 114
p6 33 54
```

Fig.2. To enter Arrival and Burst time of Processes

```
avg Queue3 WT=228

avg Queue3 TAT=351
```

Fig.1. Results using static time quantum

```
avg Queue3 WT=119

avg Queue3 TAT=252
```

Fig.3. Results using dynamic time quantum

```
quantum: 110
Queue 1: p0
Queue 2: p2 p3 p6 p4
Queue 3: p5 p1

Execution FCFS-SJF-SJF

Queue 1: 0 [   P0   ] 8

Queue 2: 8 [   P2   ] 29 [   P3   ] 68 [   P6   ] 122 [   P4   ] 189

Queue 3: 189 [   P5   ] 303 [   P1   ] 436
```

Fig.4. Process execution using static time quantum

```
quantum: 114
Queue 1: p0
Queue 2: p2 p3 p6 p4 p5
Queue 3: p1

Execution FCFS-DYNAMIC SJF

Queue 1: 0 [   P0   ] 8

Queue 2&3: 8 [   P2   ] 29 [   P3   ] 68 [   P6   ] 122 [   P1   ] 255 [   P4   ] 322 [   P5   ] 436
```

Fig.5. Process execution using dynamic time quantum

The average waiting time and turnaround time of the processes can be found out by two methods either by using static time quantum or by dynamically generating time quantum. In Fig. 4 example is solved using SJF and RR along with static time quantum. In the second case Fig. 6 dynamic time quantum is calculated by using the ratio (2n/3) where n is the number of processes left after execution of the processes in the first queue. The burst time of the (2n/3) process is the time quantum of the second queue. And the largest burst time left is the time quantum of the third queue. Average waiting time and turnaround time is computed in both cases as shown the respective outputs in Fig. 5 and Fig. 7.

V. OBSERVATION

TABLE I. COMPARISON BETWEEN STATIC AND DYNAMIC RESULTS

Cases	Average Waiting time	Average Turnaround time
Static Implementation	228	351
Dynamic Implementation	119	252

From the above analysis it can be seen that the average waiting time and turnaround time of the second case is less as compared to the first case where static time quantum is used. The efficiency can be improved to a larger extent using dynamic time quantum. The following graph describes the efficiency of the dynamic time quantum further.

VI. CONCLUSION AND FUTURE WORK

From all the above experiments it can be clearly concluded that dynamically generating time quantum by the method explained above and using SJF before RR helps us in improving the CPU and resource utilization. By experimenting with various combinations of jobs and scheduling policies, we have preferred Round Robin algorithm for fair use of CPU in first queue and SJF prior to RR in next queues (which we called as SJFRR) to reduce the average waiting time as well as turned around time. Using our analysis and observations in this work through our own program, more efficient MLFQ can be designed to help manage the processes by also considering other aspects such as the nature of processes submitted to the system. We are presently in process of enhancing the experiment for a more realistic analysis of processes.

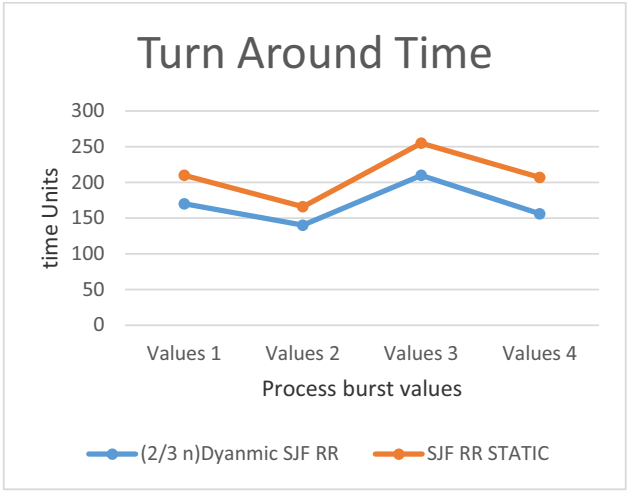

Fig.6. Comparison graph for turnaround time

As seen in the graph the turnaround time obtained is less in dynamic implementation. This method helps us in achieving at least 25% more efficiency than the static implementation method system.

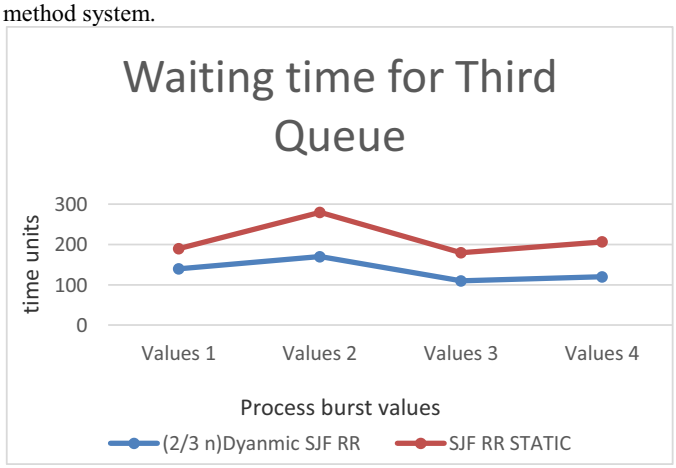

Fig.7. Comparision graph for waiting time

REFERENCES

[1] Dwivedi, Sanjay. K., and Ritesh Gupta. "A simulator based performance analysis of multilevel feedback queue scheduling", *2014 International Conference on Computer and Communication Technology (ICCCT)*, 2014.

[2] Rakesh Kumar Yadav, Anurag Upadhayay, "A fresh loom For Multilevel Feedback Queue Scheduling Algorithm", *International Journal of Advances in Engineering Sciences* Vol.2, Issue 3, July 2012 Tavel, P. 2007 Modelling and Simulation Design AK

[3] Deepali Maste, Leena Ragha and Nilesh Marathe "Intelligent Dynamic Time Quantum Allocation in MLFQ Scheduling" *International Journal of Information and Computation Technology* ISSN 0974-2239 Volume 3, Number 4 (2013), pp. 311-322 © International Research Publications House.

[4] M. V. Panduranga Rao and K. C. Shet, "Analysis of New Multilevel Feedback Queue Scheduler for Real Time Kernel" *International Journal Of computational cognition" (*http://www.ijcc.us),

vol. 8, no. 3, September 2010.

[5] Rakesh Kumar Yadav, Abhishek K Mishra,Naveen Prakash, Himanshu Sharma, "An Improved Round Robin Scheduling Algorithm", *International Journal on Computer Science and Engineering,* 2010, Volume 2, Number 04, pp: 1064-1066.

[6] H.S. Behera, Reena Kumari Naik, Suchilagna Parida "Improved Multilevel Feedback Queue Scheduling Using Dynamic Time Quantum and Its Performance Analysis". *(IJCSIT) International Journal of Computer Science and Information Technologies,* Vol. 3 (2), 2012, 3801-3807