Impheral Optimization of Round Robin in RTOS
Aditee Singh\textsuperscript{1}, Abhishek Ranjan\textsuperscript{2}, Pallab Banerjee\textsuperscript{3}, Biresh Kumar\textsuperscript{4}
Assistant Professor\textsuperscript{3,4}
Department of Computer Science and Engineering
Amity University Jharkhand, Ranchi, India

Abstract:
CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc., thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair. Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them. Some of the popular CPU scheduling algorithms are First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling and Round Robin (RR). FCFS is the simplest form of CPU scheduling algorithm. Round Robin being the most popular choice in time shared system, but it may not be suitable for real time systems because of larger waiting time, turnaround time and more number of context switches. This paper describes an improvement in RR.

I. INTRODUCTION

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithm –

• First-Come, First-Served (FCFS) Scheduling
• Shortest-Job-Next (SJN) Scheduling
• Priority Scheduling
• Shortest Remaining Time
• Round Robin(RR) Scheduling
• Multiple-Level Queues Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state. A typical process involves both I/O time and CPU time. In a uni programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Objectives of Process Scheduling Algorithm
• Max CPU utilization [Keep CPU as busy as possible]
• Fair allocation of CPU
• Max throughput [Number of processes that complete their execution per time unit]
• Min turnaround time [Time taken by a process to finish execution]
• Min waiting time [Time a process waits in ready queue]
• Min response time [Time when a process produces first response]

Scheduling criteria
There are lot of CPU scheduling algorithms having different properties, and the choice of a particular algorithm may favour one class of processes over another. For selection of an algorithm for a particular situation, we must consider properties of various algorithms. The criteria include the following:
• Context Switch: A context switch is computing process of storing and restoring state of a CPU so that execution can be resumed from same point at a later time. Context switch are usually computationally intensive, lead to wastage of time, memory, scheduler overhead so much of the design of operating system is to optimize these switches.
• Throughput: Throughput is defined as number of process completed per unit time. Throughput will be slow in round robin scheduling implementation. Context switch and throughput are proportional to each other.
• CPU Utilization: We want to keep the CPU as busy as possible.
• Turnaround Time: Turnaround time is sum of periods spent waiting to get into memory, waiting in ready queue, executing on CPU and doing input output. It should be less.
• Waiting Time: Waiting time is the amount of time during which a process spends waiting in ready queue.
• Response Time: Response time is the time it takes to start responding, not the time it takes to output the response. Large response time is a drawback in round robin architecture as it leads to degradation of system performance.

Parameters
• Arrival Time: Time at which the process arrives in the ready queue.
• Completion Time: Time at which process completes its execution.
• Burst Time: Time required by a process for CPU execution.
• Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
• Waiting Time(W.T): Time Difference between turnaround time and burst time.

Waiting Time = Turn Around Time – Burst Time

CPU scheduling algorithms
First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this
type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue. The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process. Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes. SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

II. SIMPLE RR SCHEDULING ALGORITHM

According to Silberchatz, Galvin, Gagne in operating system design and operating system by D M Dhamdhere, the simple

RR scheduling algorithm is given by following steps:-
1. The scheduler maintains a queue of ready processes and a list of blocked and swapped out processes.
2. The PCB of newly created process is added to end of ready queue. The PCB of terminating process is removed from the scheduling data structures.
3. The scheduler always selects the PCB at head of the ready queue.
4. When a running process finishes its slice, it is moved to end of ready queue.
5. The event handler perform the following action
a. When a process makes an input -output request or swapped out, its PCB is removed from ready queue to blocked/swapped out list
b. When input-output operation awaited by a process finishes or process is swapped in its process control block is removed from blocked/swapped list to end of ready queue.

III. OPTIMIZED ROUND ROBIN CPU SCHEDULING

The proposed algorithm will be executed in three phase which help to minimize a number of performance parameters such as context switches, waiting time and average turnaround time.

The algorithm performs following steps as:
Phase 1: Allocate every process to CPU, a single time by applying RR scheduling with a initial time quantum (say k units).

A. Pick the first process from the ready queue and allocate the CPU to it for a time interval of up to k time quantum.
B. If the remaining CPU burst time of the currently running process is less than k time quantum then allocate CPU again to the currently running process for remaining CPU burst time.
C. After completion of execution, removed it from the ready queue.

Phase 2: After completing first cycle perform the following steps:
a. Double the initial time quantum (2k units).
b. And repeat the Phase1 with 2k time quantum.

Phase3: For the complete execution of all the processes we have to repeat phase 1 and 2 cycle.

Illustrations

EXAMPLE-1: Considering a ready queue with four processes P1, P2, P3 and P4 arriving at time 0, 1, 2, 3 with burst time 15, 7, 9 and 6 respectively. Time quantum (TQ) has been assumed 3 milliseconds (ms). Our proposed ORR CPU scheduling picks the first process P1 from the ready queue and allocate the CPU to it for a time interval of 3ms. After executing P1 for 3ms, the remaining CPU burst of P1 is 12ms. Now, process P2 is selected and allocated the CPU for a time interval of 3ms. After executing P2 for 3ms, the remaining CPU burst of P2 is 4ms. After process P2 process P3 is picked from the ready queue and allocate the CPU to it for a time interval of 3ms. After executing P3 for 3ms, the remaining CPU burst of P3 is 6ms. And finally process P4 is executed and its remaining CPU burst is 3ms. Now after first cycle, the time quantum will be doubled according to ORR i.e. 6ms (before 3ms) and execution will start as according to Simple RR. For the second cycle, our proposed ORR CPU scheduling picks the first process P1 from the ready queue and allocate the CPU to it for a time interval of 6ms. After executing P1 for 6ms, the remaining CPU burst of P1 is 6ms. Now, process P2 is selected and allocated the CPU for a time interval of 6ms. After executing P2 for 6ms, the remaining CPU burst of P2 is 0ms. After process P2 process P3 is picked from the ready queue and allocate the CPU to it for a time interval of 6ms. After executing P3 for 6ms, the remaining CPU burst of P3 is 0ms. And finally process P4 is executed and its remaining CPU burst is 0ms. Now for third cycle, process P2, P3, P4 have remaining CPU burst 0ms. For process P1, after executing P1 for 6ms, the remaining CPU burst of P1 is 0ms.

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>ARRIVAL TIME</th>
<th>BURST TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

TQ=3MS

EXAMPLE-2: Considering a ready queue with four processes P1, P2, P3 and P4 arriving at time 0, 1, 2, 3 with burst time 15, 7, 9 and 6 respectively. Time quantum (TQ) has been assumed 3 milliseconds (ms). Our proposed ORR CPU scheduling picks the first process P1 from the ready queue and allocate the CPU to it for a time interval of 3ms. After executing P1 for 3ms, the remaining CPU burst of P1 is 12ms. Now, process P2 is selected and allocated the CPU for a time interval of 3ms. After executing P2 for 3ms, the remaining CPU burst of P2 is 4ms. After process P2 process P3 is picked from the ready queue and allocate the CPU to it for a time interval of 3ms. After executing P3 for 3ms, the remaining CPU burst of P3 is 6ms. And finally process P4 is executed and its remaining CPU burst is 3ms. Now after first cycle, the time quantum will be doubled according to ORR i.e. 6ms (before 3ms) and execution will start as according to Simple RR. For the second cycle, our proposed ORR CPU scheduling picks the first process P1 from the ready queue and allocate the CPU to it for a time interval of 3ms. After executing P1 for 6ms, the remaining CPU burst of P1 is 0ms. Now, process P2 is selected and allocated the CPU for a time interval of 3ms. After executing P2 for 3ms, the remaining CPU burst of P2 is 4ms. After process P2 process P3 is picked from the ready queue and allocate the CPU to it for a time interval of 3ms. After executing P3 for 3ms, the remaining CPU burst of P3 is 0ms. And finally process P4 is executed and its remaining CPU burst is 0ms. Now for third cycle, process P2, P3, P4 have remaining CPU burst 0ms. For process P1, after executing P1 for 6ms, the remaining CPU burst of P1 is 0ms.
allocated the CPU for a time interval of 6ms. After executing P2 for 6ms, the remaining CPU burst of P2 is 0ms. After process P2 process P3 is picked from the ready queue and allocate the CPU to it for a time interval of 6ms. After executing P3 for 6ms, the remaining CPU burst of P3 is 0ms. Process P4 already has remaining CPU burst time as 0ms. Now for third cycle, process P2, P3, P4 have remaining CPU burst 0ms. For process P1, after executing P1 for 6ms, the remaining CPU burst of P1 is 0ms.

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>ARRIVAL TIME</th>
<th>BURST TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

TQ = 3MS

IV. EXPERIMENTAL ANALYSIS

Assumptions:
To evaluate the performance, we assumed that the environment where all the experiments are performed is a single processor environment and all the processes are independent. All the processes have equal priority. All the attributes like burst time, number of processes and the time slice of all the processes are known before submitting the processes to the processor. The context switching time is equal to zero i.e., there is no context switch overhead incurred in switching from one process to another. All processes are CPU bound. No processes are I/O bound. The time quantum is taken in milliseconds.

Experiments Performed: For performance evaluation of our proposed ORR algorithm, we have taken two different cases. In first case burst time of process P4 has been considered six and CPU arrival time has been taken in increasing order. In second case burst time of process P4 has been considered five (for showing special characteristics of our ORR algorithm) and CPU arrival time has been taken in increasing order.

CASE-1: Process P4 having burst time as 6ms.
We consider the ready queue with five processes P1, P2, P3, P4 arriving at time 0, 1, 2, 3 with burst time 15, 7, 9 and 6 respectively. The comparison result of RR and proposed IRR are shown in Table 1, which shows the Gantt chart representation of RR and IRR respectively.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>AVERAGE WAITING TIME (MS)</th>
<th>AVERAGE TURNAROUND TIME (MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>18.75</td>
<td>19.5</td>
</tr>
<tr>
<td>ORR</td>
<td>28</td>
<td>29</td>
</tr>
</tbody>
</table>

CASE-2: Process P4 having burst time as 5ms.
We consider the ready queue with five processes P1, P2, P3, P4 arriving at time 0, 1, 2, 3 with burst time 15, 7, 9 and 5 respectively. The comparison result of RR and proposed IRR are shown in Table 2, which shows the Gantt chart representation of RR and IRR respectively.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>AVERAGE WAITING TIME (MS)</th>
<th>AVERAGE TURNAROUND TIME (MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>18.5</td>
<td>28</td>
</tr>
<tr>
<td>ORR</td>
<td>15.5</td>
<td>24.5</td>
</tr>
</tbody>
</table>

Gantt Chart for Case-1:

ROUND ROBIN

Gantt Chart for Case-2

ROUND ROBIN
V. CONCLUSION

1. Optimised round robin is more effective than the simple round robin.
2. It involves less number of context switches.
3. The average waiting time for optimised round robin is less as compared to the simple round robin.
4. The optimised round robin has lower average turnaround time than simple round robin.
5. The overall all comparative study helped us to analyse the context switches, waiting time and turnaround time.

VI. REFERENCES:


[2]. https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html

[3]. https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm


[7]. AN IMPROVED ROUND ROBIN CPU SCHEDULING ALGORITHM Manish Kumar Mishra, Abdul Kadir Khan. Department of Information Technology, Haramaya University, Dire Dawa, Ethiopia mishrasoft1@gmail.com. Department of Computer Science, Haramaya University, Dire Dawa, Ethiopia qadirforu@gmail.com

[8]. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/

[9]. An Optimized Round Robin Scheduling Algorithm for CPU Scheduling Ajit Singh, PriyankaGoyal, SahilBatra, Department of Computer Science and Engineering, The Technological Institute of Textile and Science, Bhiwani, Haryana

[10]. https://www.studytonight.com/operating-system/cpu-scheduling