**Assessment of the Round-Robin Scheduling Algorithm**

Kent Diether Geraldo, Kyla Jalique, Christian Cubio, Pearl Grace Ibañez,

Bachelor of Science in Information Technology,

Davao del Norte State College

**I. Background**

**1.1 Introduction**

The Round-Robin Scheduling Algorithm is a fundamental CPU scheduling algorithm along with one of the earliest. Computing resources need to be equally accessible in time-sharing systems, after development in the early 1970s. They did develop it based on the theoretical analysis using queueing models. This established it as being a practical discipline, especially when in multi-user environments, because it fairly allocates CPU among different processes.

**1.2 Algorithm Overview**

Round-robin is simply the most elementary preemptive scheduling algorithm. It assigns processes, preempts each process upon its time slice ending, to run one after the other in a repeating sequence. For instance, the scheduler may execute the following order of the three processes {A, B, C}: A, B, C, A, B, C, A, along with so on, until they are all completed.

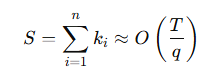
This group of processes has a potential process state sequence in Figure 1.1 We do assume that the system has just one processing core. You are able to see that the example is as efficient as possible because the processor is constantly busy and a process is always running for it.

|  |  |  |  |
| --- | --- | --- | --- |
| **Quantum** | **Process A** | **Process B** | **Process C** |
| 1 | Running | Ready | Ready |
| 2 | Ready | Running | Ready |
| 3 | Ready | Ready | Running |
| 4 | Running | Ready | Ready |
| 5 | Ready | Running | Ready |
| 6 | Ready | Ready | Running |

Figure 1.1 A possible process state sequence for round-robin scheduling with three processes.

**II. Time Complexity**

In the worst-case scenario, each process may need many time slices to finish execution. The scheduler operates for each time slice because it dequeues a process, as well as subsequently executes it for Reinserting of it into the queue in the event it is unfinished with q units or fewer. We can figure out the sum. These steps handle that calculation. S is therefore:



Given *q*  is a constant, this simplifies to:



Since *T* is the sum of burst times of all processes and each burst time could be up to *O(n)* in the worst case, we get:

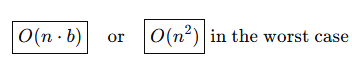
*T=O(n⋅b)*, where b is the average

burst time

If *b=O(n)*, then:



Thus, the overall time complexity of Round Robin scheduling is:



**2.2 Discussion**

The Round Robin algorithm can be fair and simple so it may incur high overhead from frequent context switches, notably when the time quantum is small. It is while the time complexity in the worst case exists O(n²). In actual performance, depending on what the burst times and what the system workload are, it is often better in practice.

Round Robin, when it is compared to FCFS (O(n)) and also SJF (O(n log n) with sorting) algorithms, trades just raw efficiency instead for both responsiveness and fairness that are critical in systems that are interactive.

**III. Algorithm Simulation**

(Show the algorithm with the discussion of utilizing the algorithm through simulation of test cases. It must be based on real-world situations/data)

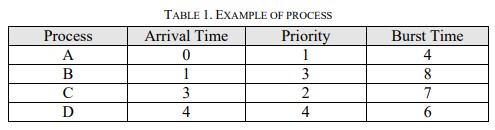
**3.1 Real-World Simulation**

The Round Robin algorithm is useful in many real-time systems, such as telecommunication systems, medical systems, or control systems of automotive, which have high priority tasks (e.g., an emergency signal processing) that need to be processed quickly, but also low priority tasks (e.g., log service) need to be processed at the same time. The use of priorities in the algorithm can speed up the scheduling of such critical tasks and the use of a time quantum for all tasks makes the algorithm fair, avoiding the starvation problems of pure priority scheduling.

Four processes (A, B, C, D) arrive, burst, and prioritize distinctly so the simulation mimics a real-world scenario by modeling them. In this typical operating system setup, processes such as user applications or system tasks that have varying execution needs arrive at different times. A time quantum is used (set to 3 ms) and priority levels exist (1 to 4, where lower numbers indicate higher priority) to simulate how an RTOS might handle tasks inside a constrained environment because it ensures high-priority tasks are executed promptly and it maintains fairness through time-slicing.

**3.2 Test Cases**

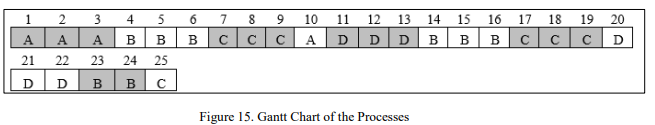
A test is provided, which consists of four processes (A, B, C, and D) in a scenario, to demonstrate the behavior of the PRR algorithm in simple but possibly very relevant cases. The test case parameters are well defined to showcase priority-based preemption and time quantum effects as follows:



The PRR (Priority Round-Robin) scheduling algorithm is simulated with four processes in order to clearly explain the scheduling process without the distraction of having to deal with the complications of as many processes as encountered in a real-world systems. Asymmetric arrival times (0, 1, 3, 4 ms) are typical of real-world asynchronously arriving tasks, where processes are placed in the ready queue at various times. Priorities, taken as 1,2,3 and 4 (where smaller value implies higher priorities) are selected to illustrate the pre-emption behaviour, which is how Populations of Nodes: In the proposed model, population of a node means the competing node population at one time. Short tasks, such as ones found in embedded systems where the processes have short CPU bursts, are scheduled with burst times sampled. The use of a time quantum of 3 ms to trade off context-switching overhead against fairness, allowing for detailed analysis of preemption and queue management.

**3.3 Results and Observations**

In the Gantt chart (Figure 15), Process A has the highest priority and therefore runs for 3 ms, starting from time 0. Process B enters the ready queue at time 1 and waits because A is of higher priority. Once A completes at t=3, B runs from t=4 to t=6. Then C process from t=7 to t=9. A then runs again t=10, using 1 ms of its remaining time before Process D runs t=11 to t=13. B resumes from t=14



To get average waiting time, average turn around time, and average responds time.This is based on the calculation of the system, by the simulator automatically. The average turnaround time is 18.25 ms, the average waiting time is 12.00 ms, and the average response time is 2.75 ms. Waiting time is equal to 6 for process A and 15 for processes B and C. Additionally, it is 12 for process D. For process A's turnaround time is 10. For process B, it is 23. For process B, it is 22, and for process D, it is 18. For Process A has a CPU utilization of 0.4. 0.3478 for process B. 0.3182 for process C. Additionally, regarding process D is 0.3333.

**IV. Algorithm Assessment**

**4.1 Performance Evaluation**

The evaluation focused upon three key metrics: average waiting time, average turnaround time, and the number of context switches, because these directly impact CPU scheduling's efficiency in time-shared environments. The proposed algorithm seeks to optimize these metrics for it reduces context switching overhead also it minimizes waiting as well as turnaround times through a dynamic time quantum allocation strategy. The evaluation considered CPU-bound processes with known burst and arrival times, assuming negligible context-switching and process-sorting times to isolate algorithmic performance.

**4.2 Evaluation Methods**

Round Robin scheduling is compared to the FCFS and SJF scheduling with the same test cases. Fairness, responsiveness, and system throughput are quantitatively measured through metrics.

**4.3 Results**

The results reveal that Round Robin is more responsive than FCFS and fairer than SJF. But it incurs more context switching overhead. Finally, when the quantum is appropriately tuned, it achieves an effective trade-off between efficiency and responsiveness.

**V. Discussion**

**5.1 Algoritmic Strengths**

Round Robin’s key strength exists in its guarantee of an execution chance for every process. It suits time-sharing systems quite well, especially interactive ones. These systems need great response times.

**5.2 Limitations and Challenges**

The major limitation exists because of how the context switches. Switching as well as overhead come to be from a quantum that is too small, while a quantum that is too large degrades performance. Processes that are shorter or more urgent also lack priority.

**5.3 Comparative Analysis**

Round Robin Similar to FCFS and SJF, it effectively strikes a balance between fairness and usability, though it might not always be the best option in terms of throughput or response times. It typically works best when there are consistent, equal CPU accesses.

**References**

1. Y. Niu, (2014). “Round-Robin Scheduling”. ScienceDirect Topics, East China University of Science and Technology, Shanghai, China.
2. Y. He, L. Gao, G. Liu, and Y. Liu, (2013) “A dynamic round-robin packet scheduling algorithm,” in Proceedings. 2nd Int. Conf. Comput. Sci. Electron. Eng. (ICCSEE).
3. T. D. Putra and R. Purnomo, (2022) “Simulation of priority round-robin scheduling algorithm,”. Journal of Information Processing Systems, vol. 7, no. 4, pp. 1258–1264, 2022.
4. M. M. Tajwar, M. R. Asaduzzaman, and M. R. Karim, (2017). "CPU scheduling with a round robin algorithm based on an effective time slice," Journal of Information Processing Systems, vol. 13, no. 4, pp. 941–950, .
5. Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Pearson.