CPC-361 Group Project

CPU Scheduling

Made by:

- Mahmued Ahmad Alardawi – 2135209 – CS1
- Mahmued Ahmad Alardawi – 2135209 – CS1

Submitted on 10 May 2024

- To view full project, I/O and code visit: https://github.com/MahmuedAlardawi/CPU-Scheduling-Using-Round-Robin

Table of Contents

Introduction

This project was undertaken to design and implement a program that simulates job and CPU scheduling within an operating system, as detailed in the provided specifications. The simulator manages an input stream describing arriving jobs, their requirements, and actions, ensuring jobs are processed according to priority and system capabilities.

System Design and Implementation

The program simulates an operating system's behavior in allocating resources to jobs based on their memory and device requirements and their priorities using Round Robin algorithm. Key features of the implementation include:

Main Components

- **Job Arrivals**: Jobs enter the system and are either rejected, placed in hold queues, or moved directly to the Ready Queue based on available resources.
- **Memory and Device Management**: Jobs request units of main memory and devices. If not available, they are queued or rejected.
- **Priority Handling**: Two priority levels determine the queuing strategy: higher priority jobs enter a memory-sorted queue, while lower priority jobs use a FIFO system.

Scheduling Algorithms

- **Hold Queues Management**: Jobs are managed in two hold queues based on priority and memory requirements.
- **Dynamic Round Robin Scheduling**: CPU scheduling is performed using a dynamic round robin algorithm, where the time quantum is adjusted based on the average burst times of processes in the Ready Queue.

Input Specification

The input is handled via text commands that specify system configurations, job arrivals, and system status requests.

Code Breakdown

Main Class: CPU_scheduling

Global Variables

- **static int timeQuantum**: Sets the default time quantum for process scheduling.
- **public static int time**: Tracks the current time in the simulation.
- **public static int memory**: Total system memory available.
- **public static int device**: Tracks the number of devices available in the system.
- **public static int timeStamp**: Serves as a time marker for different operations.
- **public static ArrayList<Job> jobsMatrix**: Store the jobs taken from the input file to later calculate the performance matrix of each algorithm.

Methods

- **main(String[] args):** This method initializes the simulation by setting up input and output streams and invokes the scheduling processes for both dynamic and standard Round Robin approaches. It also handles the closing of these streams after processing.

- **dynamicRoundRobin(Scanner input, PrintWriter output):** This method implements the dynamic Round Robin scheduling algorithm. It dynamically adjusts the time quantum based on the burst times of the processes in the ready queue. The method handles reading input data, scheduling jobs, and managing queues.

- **roundRobin(Scanner input, PrintWriter output):** Implements a standard Round Robin scheduling with a fixed time quantum. It processes jobs according to their arrival and simulates their execution in a fixed-time quantum cycle.

- **systemConfiguration(Scanner input, Job job):** Reads and processes system configuration commands from the input. This method adjusts global settings like system memory and devices based on the commands parsed. It also configures job-specific attributes when a job is passed as an argument.

- **processHoldingQueues(Queues queues):** Manages jobs waiting in hold queues. This method checks the availability of system resources (memory and devices) and moves jobs to the ready queue when resources become sufficient to process them.

- **dynamicProcessNewArrivals(ArrayList<Job> jobs, Queues queues, int memory0, int device0):** Processes new job arrivals for the dynamic Round Robin algorithm. This method adds new jobs to the system, allocates resources, and updates the ready and hold queues based on the current system state and job requirements.

- **printSystemStatus(Queues queues, Job j, int remainingTime):** Outputs the current state of the system, including job details and queue contents. This method provides a snapshot of the system every second.

- **timeStampPrint(ArrayList<Job> jobs, Queues queues, int jobTracker):** Outputs the current state of the system, including job details and queue contents. This method provides a snapshot of the system at any given point, which the input file provides.

- **performanceMetrics(PrintWriter output):** provides the ending performance matrix to the round robin algorithm.

Class: Queues

Global Variables

- **static Queue<Job> readyQueue**: This is a queue that holds jobs that are ready to be scheduled for execution. Implemented using LinkedList, which allows for efficient FIFO operations.

- **static Queue<Job> holdQueue1**: Priority queue that holds jobs with priority 1, sorted by memory requirements in ascending order. Implemented using PriorityQueue with a custom comparator based on memory required.

- **static Queue<Job> holdQueue2**: FIFO queue holding jobs with priority 2. Implemented using LinkedList to maintain FIFO ordering.

Constructors

- **Queues()**: Initializes an instance of the Queues class.

Methods

- **getReadyQueue()**: Returns the queue containing jobs that are ready for scheduling.
- **getHoldQueue1()**: Returns the priority queue of jobs with priority 1.
- **getHoldQueue2()**: Returns the FIFO queue of jobs with priority 2.
- **toString()**: Provides a string representation of the Queues object, listing the current states of the ready queue, holdQueue1, and holdQueue2.

Class: Job

Attributes

- **int arrivalTime**: Represents the arrival time of a job.
- **int jobId**: Unique identifier for each job.
- **int memoryRequired**: Amount of memory required by the job.
- **int devicesRequired**: Number of devices required by the job.
- **int burstTime**: The amount of CPU time required for the job to complete execution.
- **int priority**: The priority level assigned to this job, used in priority queues.
- **int startTime**: The time when a job starts executing in the Ready Queue.
- **int completionTime**: The time when the job is terminated from the Ready Queue.
- **int turnaroundTime**: The total time taken from a job's arrival to its completion, including both processing and waiting periods.
- **int waitingTime:** The time a job spends waiting in the ready queue before it gets CPU time to start or resume execution.

Constructors

- **Job()**: A default constructor initializing a Job object without parameters.

Methods

- **Getter and Setter Methods**: These methods allow getting and setting values for each of the job attributes (e.g., getArrivalTime, setArrivalTime, getJobId, etc.).

- **toString()**: Provides a string representation of the Job object, listing all its attributes (arrival time, job ID, memory required, devices required, burst time, and priority).

- **isJob()**: Returns true if the job attributes indicate a valid job (not empty or with default values), false otherwise.

- **canProcess(int memory, int device)**: Checks if the job can be processed given the available system memory and devices. Returns true if the job's memory and device requirements are satisfied, false otherwise.

- **public void calculateMetrics()**: This method calculates the turnaround time and waiting time for a job. The turnaround time is computed as the difference between the completion time and the arrival time, while the waiting time is the turnaround time minus the burst time.

- **public Job copy(Job source)**: This method creates a new job by duplicating the attributes of a given source job, ensuring that all relevant properties are identical between the original and copied job objects.

Challenges

During the project, several challenges were encountered, including:

- **Dynamic Time Quantum Management**: Implementing a dynamic round robin that adjusts time quantum dynamically based on jobs available in the Ready Queue.
- **Resource Allocation**: Managing the allocation and deallocation of memory and devices dynamically to maximize efficiency and minimize job wait times.
- **Queue Management**: Efficiently managing multiple queues (two hold queues and one ready queue) while ensuring that priority and resource requirements are met.

Performance Matrices Output

DRR Output

```
Final Metrics for All Jobs:
Job ID: 1, Arrival Time: 10, Start Time: 10, Burst Time: 8, Completion
Time: 18, Turnaround Time: 8, Waiting Time: 0
Job ID: 2, Arrival Time: 13, Start Time: -1, Burst Time: 13,
Completion Time: -1, Turnaround Time: 0, Waiting Time: 0
Job ID: 3, Arrival Time: 15, Start Time: 18, Burst Time: 5, Completion
Time: 23, Turnaround Time: 8, Waiting Time: 3
Job ID: 4, Arrival Time: 19, Start Time: 23, Burst Time: 7, Completion
Time: 33, Turnaround Time: 14, Waiting Time: 7
Job ID: 5, Arrival Time: 21, Start Time: 28, Burst Time: 4, Completion
Time: 38, Turnaround Time: 17, Waiting Time: 13
Job ID: 6, Arrival Time: 28, Start Time: 33, Burst Time: 8, Completion
Time: 50, Turnaround Time: 22, Waiting Time: 14
Job ID: 7, Arrival Time: 34, Start Time: 38, Burst Time: 12,
Completion Time: 54, Turnaround Time: 20, Waiting Time: 8
Job ID: 8, Arrival Time: 35, Start Time: 54, Burst Time: 8, Completion
Time: 62, Turnaround Time: 27, Waiting Time: 19

Average Turnaround Time: 116.0
Average Waiting Time: 64.0
```

RR Output

```
Final Metrics for All Jobs:
Job ID: 1, Arrival Time: 10, Start Time: 10, Burst Time: 8, Completion Time: 18, Turnaround Time: 8, Waiting Time: 0
Job ID: 2, Arrival Time: 13, Start Time: -1, Burst Time: 13, Completion Time: -1, Turnaround Time: 0, Waiting Time: 0
Job ID: 3, Arrival Time: 15, Start Time: 18, Burst Time: 5, Completion Time: 23, Turnaround Time: 8, Waiting Time: 3
Job ID: 4, Arrival Time: 19, Start Time: 23, Burst Time: 7, Completion Time: 30, Turnaround Time: 11, Waiting Time: 4
Job ID: 5, Arrival Time: 21, Start Time: 30, Burst Time: 4, Completion Time: 34, Turnaround Time: 13, Waiting Time: 9
Job ID: 6, Arrival Time: 28, Start Time: 34, Burst Time: 8, Completion Time: 42, Turnaround Time: 14, Waiting Time: 6
Job ID: 7, Arrival Time: 34, Start Time: 42, Burst Time: 12, Completion Time: 54, Turnaround Time: 20, Waiting Time: 8
Job ID: 8, Arrival Time: 35, Start Time: 54, Burst Time: 8, Completion Time: 62, Turnaround Time: 27, Waiting Time: 19

Average Turnaround Time: 101.0
Average Waiting Time: 49.0
```

Comparative Analysis

The comparative analysis of the Dynamic Round Robin and Round Robin scheduling algorithms is based on the metrics of turnaround time and waiting time for each job, as provided in the output data. The analysis focuses on determining which scheduling approach performs better in terms of responsiveness (turnaround time) and efficiency (waiting time).

Average Turnaround Time

- **Dynamic Round Robin (DRR):** The average turnaround time is 116.0.
- **Round Robin (RR):** The average turnaround time is 101.0.

Analysis: The average turnaround time is lower in the Round Robin (RR) scheduling algorithm compared to the Dynamic Round Robin (DRR). This suggests that RR provides a quicker completion of jobs on average, which may be attributed to a more consistent allocation of CPU time quantum)that minimizes the impact of context switching and scheduling overhead.

Average Waiting Time

- **Dynamic Round Robin (DRR):** The average waiting time is 64.0.
- **Round Robin (RR):** The average waiting time is 49.0.

Analysis: Like the turnaround time, the average waiting time is also lower in the Round Robin approach. This indicates that RR tends to keep jobs waiting for a shorter duration in the ready queue compared to DRRF. The consistent quantum in RR might be reducing the waiting periods between consecutive allocations of CPU time to the same job.

Individual Job Analysis

- **Job 2** did not run in either scheduling, showing -1 for times, suggesting it might have prerequisites or conditions not met during the scheduling.
- **Jobs with Short Burst Times:** For jobs like Job 3, both DRR and RR achieve similar turnaround and waiting times, indicating that job length might influence the effectiveness of the scheduling strategy less significantly.
- **Jobs with Long Burst Times:** For longer jobs (e.g., Job 6), DRR tends to have a higher waiting time compared to RR. This might indicate that DRRF with its dynamic quantum adjustments, may sometimes penalize longer jobs if shorter jobs are frequently ready to run.

Performance and Responsiveness

- **Response Time:** Round Robin demonstrates better overall responsiveness, as evidenced by generally lower turnaround times across all jobs.
- **Fairness:** RR tends to treat jobs more uniformly, not allowing any single job to monopolize CPU time, which can be critical in environments where fairness is a priority.

Optimal Use-Cases

- **Dynamic Round Robin (DRR):** Might be more suitable in environments where job priorities frequently change or where the job characteristics are highly varied and dynamic quantum adjustment could potentially optimize CPU utilization.
- **Round Robin (RR):** Better suited for environments requiring consistent and predictable scheduling times, and where job priorities are static, and fairness is a key requirement.

Conclusion

This project successfully demonstrates the complexities and requirements of simulating an operating system's job and CPU scheduling. Through the implementation of various scheduling algorithms and management of system resources, the simulator provides valuable insights into the operational dynamics of job processing.

References

- **Jenny's Lectures CS IT**, "Round Robin(RR) CPU Scheduling Algorithm in OS with example," YouTube, [Published:24-April-2019]. [Video]. Available: https://youtu.be/-jFGYDfWkXI. [Accessed: 10-May-2024].
- **Neso Academy**, "Round Robin Scheduling - Solved Problem (Part 1)," YouTube, [Published:16-October-2019]. [Video]. Available: https://youtu.be/QlCmgBOMjlI. [Accessed: 10-May-2024].
- "Java Tutorial," *TutorialsPoint*. [Online]. Available: https://www.tutorialspoint.com/java/index.htm. [Accessed: 10-May-2024].