To Schedule or Not To Schedule?

Ahmet E. Hacıoğlu,2020400132

(Tested on MacOS)

1 Introduction

The multi-level queue scheduler project aims to implement a priority-based scheduling algorithm for a set of processes. The scheduler categorizes processes into different priority levels and executes them based on their priority. This report provides an overview of the implementation approach, challenges faced, and a focus on the round-robin process within the scheduler.

2 Implementation

The 'Process' structure defines a process with attributes including its name, priority, arrival time, type, an array to store instructions, execution-related parameters, and flags indicating its status (defined, finished, arrived), serving as a comprehensive representation for process management in a system.

The enqueue function inserts a given process into a priority queue based on its arrival time, with additional consideration for priority and lexicographical ordering of process names when arrival times are identical.

A series of operations are carried out to initialize, read, and process instructions and process definitions. Initially, an array of instructions is filled by reading from a file, and subsequently, a priority queue is constructed based on process definitions read from another file. Further, the instructions for each process are read from individual files, updating relevant data structures.

Additionally, there's an implementation for managing unfinished processes, including a loop that iterates through the priority queue to determine the next process for execution based on priority and arrival time considerations. The code snippet concludes with the initialization of variables for tracking unfinished processes and the scheduling of the initial process. The presented code represents a basic workflow for process and instruction handling in a simulated system.

The loop systematically evaluates processes in the priority queue, marking them as "arrived" and skipping those already marked as "finished." For each uncompleted process, it compares priorities, considering arrival times and lexicographical ordering of names in case of ties. Additionally, it identifies the Platinum process with the highest priority.

Based on these comparisons, it determines the next process to execute and updates the current time accordingly. If the selected process differs from the previous one, it increments the current time by 10 units. The loop also maintains an array and count of unfinished processes. This iterative process efficiently manages the execution order of processes, considering their priority, arrival time, and type.

The loop iterates through the instructions of a selected process in the priority queue, updating the current time based on the duration of each instruction. It keeps track of the last executed instruction and counts the quantum time spent executing instructions.

Additionally, it checks for newly arrived processes and updates their status and priority count. The process dynamically adjusts its type and quantum count based on execution time, considering conditions related to priorities and process types. If certain criteria are met, the process transitions to a higher-priority type (e.g., SILVER to GOLD) and resets the quantum count.

The last part adjusts arrival times for round robin and tracks completed processes. It checks for completed instructions in incomplete processes, updating their status and metrics. If all processes are done, it updates their status and metrics, considering potential new arrivals. The loop index is reset based on process completion or if multiple processes share the highest priority. Finally, it calculates and prints the average turnaround and waiting times for the executed processes.

3 Difficulties Encountered

Throughout the implementation of the scheduling algorithm, I encountered various challenges, mainly revolving around managing numerous edge cases and identifying subtle mistakes in the sequencing of process executions. The complexity of the project demanded careful consideration of diverse scenarios to guarantee the correct and efficient execution of processes.

One notable challenge involved the dynamic adjustments to process priorities and types during execution. Balancing priorities based on arrival time, quantum time, and process types required intricate logic to prevent unintended behaviors and maintain fairness in the scheduling algorithm.

The initial strategy, involving the maintenance of Round Robin as a numeric value incremented with each quantum, faced challenges in prioritization based on arrival time and process names. Inconsistencies arose, prompting a reevaluation of the approach. The decision to dynamically update arrival times and move completed processes to the end of the queue emerged as a robust solution, effectively addressing the challenges in accurately prioritizing processes.

In summary, the project underscored the significance of a meticulous approach in handling intricacies within process scheduling. It emphasized the need to ensure that the algorithm adeptly manages priorities, arrivals, and execution order in a simulated system. .