Simulation Based Assignment

Title: Dynamic Priority Scheduling

As a Field work for Course

Operating System CSE(316)

By

Registration No	Name	Roll No	Section
12201534	**S.Mustafeezulla Khan**	**RK22UNB48**	**K22UN**

Lovely Professional University

Jalandhar, Punjab, India.

Abstract

The simulation program described in this report addresses the challenges of efficiently scheduling processes in a computing environment. It implements a non-preemptive priority scheduling algorithm that considers both the estimated run time and the waiting time of each process. The program allows for dynamic user input, real-time display of process priorities, and the generation of a Gantt chart. Additionally, it calculates individual and average waiting times, providing a comprehensive overview of process scheduling performance.

Introduction

In the context of operating systems and process scheduling, the efficient allocation of computing resources is essential to maximize system productivity. Traditional scheduling algorithms often focus solely on factors such as process priority or estimated run time. However, these algorithms may not fully address the dynamic nature of computing environments where processes arrive and compete for resources in real-time.

The purpose of this simulation program is to implement a non-preemptive priority scheduling algorithm that considers not only the estimated run time of processes but also the time they have spent waiting. By assigning higher priority to processes that have waited longer, the algorithm aims to prevent indefinite postponement and improve overall system performance. The program allows users to input processes dynamically, visualize process priorities in real-time, generate a Gantt chart to represent the execution sequence, and calculate individual and average waiting times.

Objectives :- The primary objectives of this simulation program are as follows:

Priority-Based Process Scheduling: The core objective of the program is to implement a non-preemptive priority scheduling algorithm that assigns priorities to processes based on their estimated run time and the time they have spent waiting. This approach aims to improve scheduling efficiency by ensuring that processes with longer waiting times are given higher priority for execution.

Dynamic user input: The program allows users to provide input dynamically at runtime. Users can specify the number of processes, their arrival times, and burst times. This feature makes the simulation adaptable to various scenarios and provides users with real-time control over process scheduling.

Real time Priority: The program provides real-time visualization of process priorities. Users can observe how process priorities change as they wait for execution. This objective allows users to gain insights into the scheduling decisions made by the algorithm.

Gantt Chart Visualization: The program generates a Gantt chart that visually represents the sequence in which processes are scheduled for execution. This visualization helps users understand the order in which processes are executed and the overall scheduling performance.

Waiting Time Calculation: The program calculates and displays the waiting time for each individual process. This objective allows users to assess the impact of waiting times on process scheduling and performance.

Average Waiting Time Calculation: In addition to individual waiting times, the program computes and displays the average waiting time for all processes. This metric provides an overall measure of scheduling efficiency and can be used to evaluate the algorithm's performance under different scenarios.

Methodology:-

The methodology of the provided code, which simulates a basic Dynamic Priority Scheduling, can be broken down into several key steps:

1. Data Structures and Initialization:

- Initialize necessary variables, including n (the number of processes), arrival time and burst time arrays to store process details, current time to track the current time, completed to count the number of completed processes, and completed time to record the completion time of each process.

-Introduce the execution order array to record the order of process execution for Gantt chart visualization.

2. User Input:

- Prompt the user for the number of processes (n) and details for each process, such as arrival time and burst time. This dynamic input enables users to simulate different scenarios.

3. Priority Calculation:

- Calculate the priority for each process based on the provided formula: Priority = 1 + (Waiting time / Estimated run time). The priority reflects the process's estimated run time and waiting time.

4. Main Simulation Loop:

- Implement a loop that continues until all processes are completed (completed == n).

In each iteration, calculate priorities and determine the process with the highest priority among the waiting processes.

5. Gantt Chart Generation:

- Record the execution order of processes in the execution_order array as they are selected for execution based on their priority.

6. Individual Waiting time Calculation:

- Calculate the waiting time for each process, which is the time a process spends waiting in the queue before execution.

7. Average waiting time Calculation:

- Calculate the average waiting time by summing up the waiting times of all processes and dividing by the total number of processes.

11. Termination and Exit:

- Allow the user to exit the simulation by selecting the appropriate menu option.

The code provides a simplified simulation of a file management system with a focus on file allocation, creation, deletion, renaming, and disk status reporting. While it serves as a basic illustration, it can be extended and enhanced for more complex scenarios and real-world use cases.

PSEUDO CODE:-

1. Input:

- Prompt user for the number of processes (num_processes).
- For each process i from 1 to num_processes:
  - Prompt user for the arrival time (arrival_times[i]) and burst time (burst_times[i]).

2. Initialization:

- Set current_time to 0.
- Set completed_count to 0.
- Initialize an array completion_times of size num_processes, all elements set to 0.
- Set total_wait_time to 0.

3. Main Simulation Loop:

- While completed_count is less than num_processes:
  - For each process i from 1 to num_processes:
    - If arrival_times[i] <= current_time and completion_times[i] is 0:
      - Calculate the priority of process i using the formula:
        - priority = 1.0 + (current_time - arrival_times[i]) / burst_times[i].
      - Display the process priority.

  - Find the process with the highest priority:
    - Initialize selected_index to -1.
    - Initialize max_priority to -1.
    - For each process i from 1 to num_processes:
      - If arrival_times[i] <= current_time and completion_times[i] is 0:
        - Calculate the priority of process i.
        - If priority > max_priority, update max_priority and selected_index.

- If a process with the highest priority is found (selected_index is not -1):

  - Mark process selected_index as completed:

    - Set completion_times[selected_index] to current_time + burst_times[selected_index].

    - Update total_wait_time with the waiting time of the selected process.

    - Update current_time by adding burst_times[selected_index].

    - Increment completed_count.

    - Display the completion time of the selected process.

- If no process is selected for execution:

  - Increment current_time by 1 (time unit without process execution).

4. Calculate Metrics:

   - Calculate the average waiting time as average_wait_time = total_wait_time / num_processes.

5. Display Results:

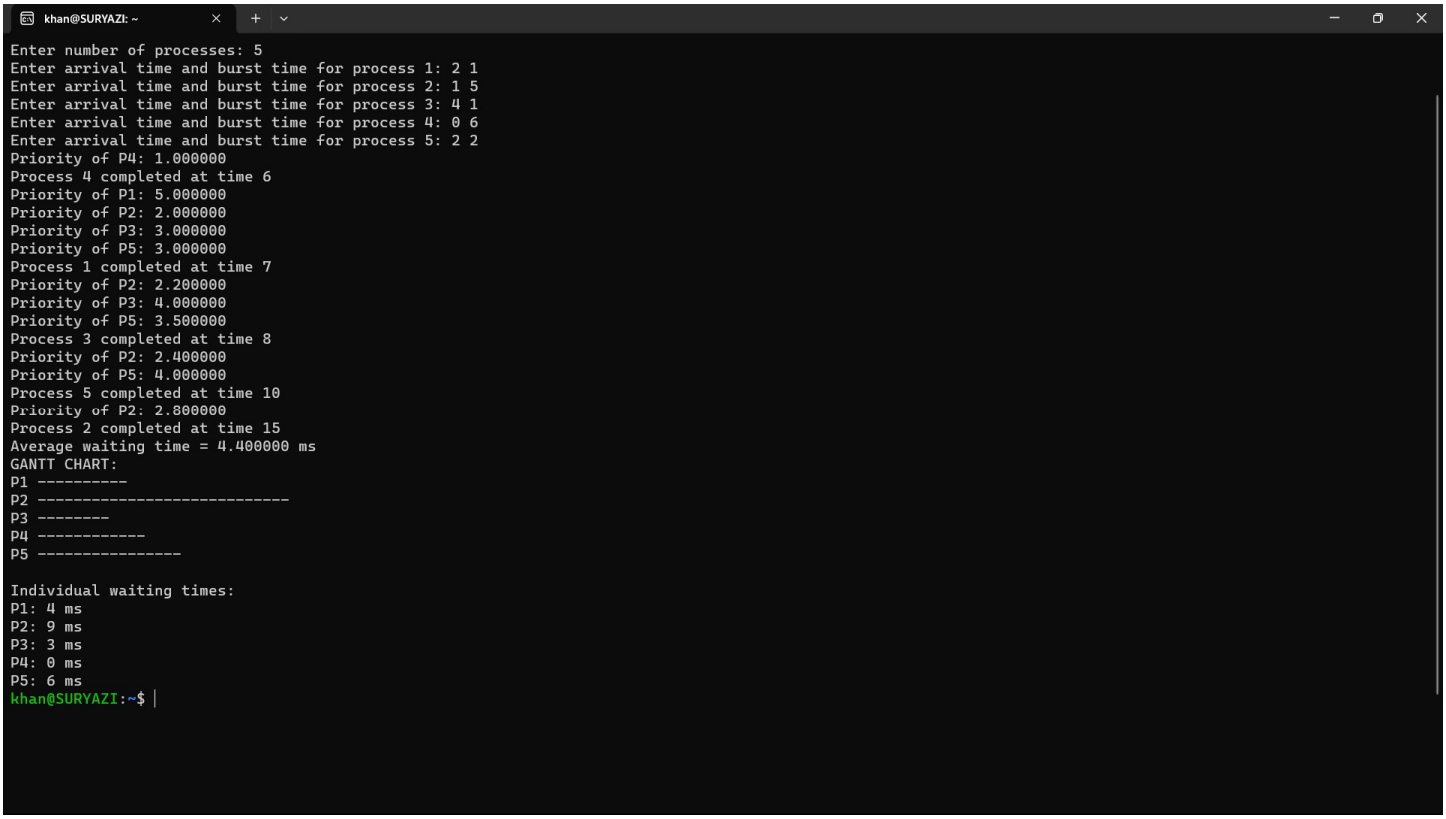
   - Display the average waiting time.

   - Generate and display the Gantt chart to visualize the execution sequence.

   - Display individual waiting times for each process.

6. End of Simulation.

OUTPUT :-



GITHUB LINK:

https://github.com/smustafeezullakhan/Priority-Scheduling.git

CONCLUSION :-

The simulation program presented here serves as an essential tool for understanding and evaluating the non-preemptive priority scheduling algorithm with dynamic user input, real-time process priority display, Gantt chart visualization, and waiting time calculations. The conclusions drawn from this simulation hold significant importance in the context of process scheduling and resource allocation in computing environments.

The program offers valuable insights into the efficiency and effectiveness of the implemented scheduling algorithm. It allows for the examination of scheduling decisions based on process priorities, estimated run times, and waiting times. The real-time priority display offers a hands-on approach to observe how priorities change as processes wait for execution. This dynamic feature enhances the understanding of process scheduling dynamics.