Text

Description automatically generated

**Activity based**

**Project Report on**

**PROGECT MANAGEMENT BTECCE22511**

**Project - II**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**MARIYAM PATEL**

**SRN No : 202201178**

**Roll No : 34**

**Div : C**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2024-2025**

**1)Problem Statement-**

The objective is to design and implement a **dynamic CPU scheduling algorithm** that can handle variable workloads by dynamically adjusting process scheduling and resource allocation.

**2)Introduction-**

**a. Overview of the Topic:**

Dynamic CPU scheduling is critical in multi-programming operating systems where processes have varying computational needs. In such environments, workload characteristics are not static, making it essential to use algorithms that dynamically adjust resource allocation, such as the **Multi-Level Feedback Queue (MLFQ)** with dynamic quantum adjustment. This ensures better CPU utilization, lower waiting times, and higher system efficiency.

**b. Importance of the Topic:**

Effective scheduling enhances the overall performance of a system by minimizing waiting time, response time, and maximizing CPU utilization. In systems where the workload is variable, dynamic scheduling ensures that high-priority or short processes receive CPU attention promptly without allowing any process to monopolize CPU resources, making it highly relevant in operating systems, especially for real-time applications.

**3)Project Scope-**

The project involves designing, implementing, and testing a CPU scheduling algorithm that adjusts dynamically to process workloads. It will use a heap-based priority queue to schedule processes based on their remaining burst times. The simulation will cover different scenarios of process arrival times and burst times, demonstrating how dynamic time quantum and process priorities can be utilized for efficient scheduling.

**4)Objectives-**

 Implement a **dynamic CPU scheduling algorithm** that adapts to variable workloads.

 Ensure optimal resource utilization by dynamically adjusting time quantum.

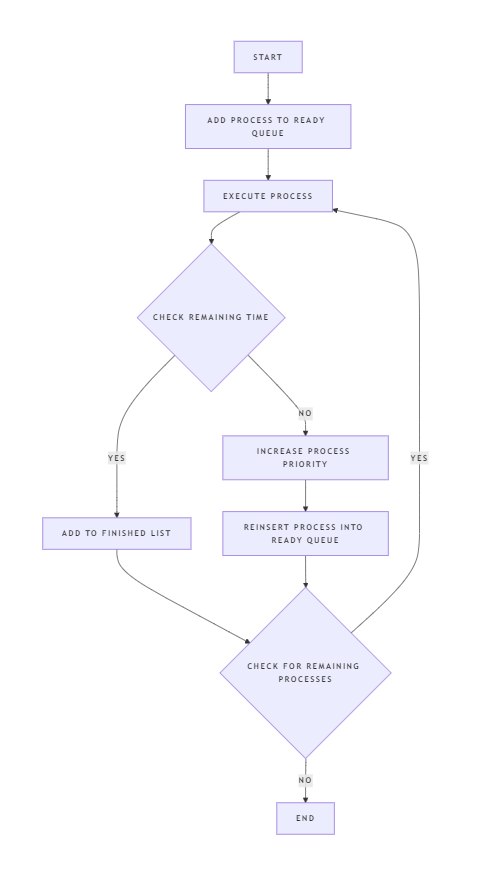
 Provide fairness among processes through dynamic priority adjustment.

 Simulate the performance of the scheduler using multiple processes with variable arrival and burst times.

 Analyze the completion times and efficiency in terms of waiting and turnaround times.

**5)System Architecture-**

**a. Diagram:**



**b.Explanation:**

 **Process Queue (Ready Queue)**: A min-heap (priority queue) is used to keep track of processes, prioritizing those with the shortest remaining burst time.

 **CPU Execution**: The CPU scheduler executes processes for a fixed quantum (e.g., 4 time units). After each quantum, it checks if the process has completed.

 **Dynamic Adjustment**: If the process is not finished, its priority is incremented, and it’s reinserted into the queue, ensuring shorter processes complete faster.

 **Termination**: The system finishes when all processes have completed, and results are printed.

**6)Implementation of code and Output-**

import heapq

class Process:

    def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

        self.pid = pid

        self.arrival\_time = arrival\_time

        self.burst\_time = burst\_time

        self.remaining\_time = burst\_time

        self.priority = 0

    def \_\_lt\_\_(self, other):

        return self.remaining\_time < other.remaining\_time

class CPU\_Scheduler:

    def \_\_init\_\_(self):

        self.time = 0

        self.ready\_queue = []

        self.finished\_processes = []

    def add\_process(self, process):

        heapq.heappush(self.ready\_queue, process)

    def execute(self):

        while self.ready\_queue:

            process = heapq.heappop(self.ready\_queue)

            print(f"Executing process {process.pid} at time {self.time}")

            # Simulate process execution with dynamic scheduling

            self.time += min(4, process.remaining\_time)  # Time quantum is dynamic

            process.remaining\_time -= 4

            # Check if the process is finished

            if process.remaining\_time <= 0:

                process.remaining\_time = 0

                print(f"Process {process.pid} finished at time {self.time}")

                self.finished\_processes.append(process)

            else:

                # Reinsert the process with increased priority if not finished

                process.priority += 1

                self.add\_process(process)

    def print\_results(self):

        print("\nFinished Processes:")

        for process in self.finished\_processes:

            print(f"Process {process.pid} completed with burst time {process.burst\_time}.")

# Simulate dynamic CPU scheduling

scheduler = CPU\_Scheduler()

# Add processes with variable workloads

processes = [

    Process(1, 0, 10),

    Process(2, 1, 15),

    Process(3, 2, 7),

    Process(4, 3, 20),

]

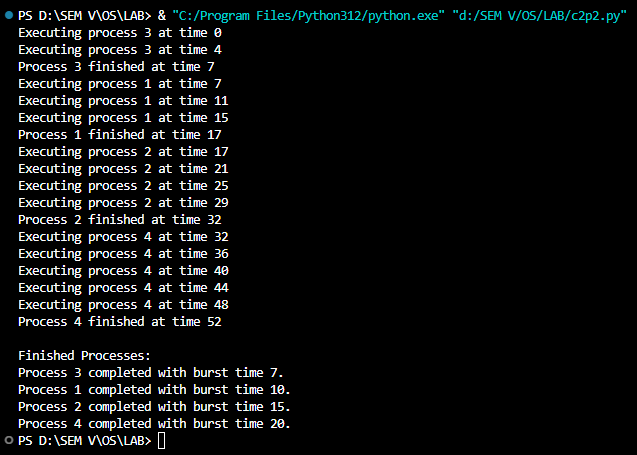
for p in processes:

    scheduler.add\_process(p)

scheduler.execute()

scheduler.print\_results()

**OUTPUT-**



**7)Key Algorithms and Data Structures to be Used-**

**a. Explanation of Algorithms:-**

 **Multi-Level Feedback Queue (MLFQ)**: Processes are prioritized based on their burst time and are preempted if not completed within a time quantum. A process can be dynamically reinserted into the queue with increased priority if it’s not finished.

 **Shortest Remaining Time First (SRTF)**: The scheduler picks the process with the shortest remaining time for execution, ensuring that shorter processes finish first.

**b. Data Structures:-**

 **Priority Queue (Min-Heap)**: A heap data structure (heapq in Python) is used to ensure that processes with the shortest remaining time are prioritized.

 **Process Object**: The process object stores attributes like process ID, arrival time, burst time, remaining time, and priority.

**8)Conclusion-**

The project successfully implements a dynamic CPU scheduling algorithm capable of handling variable workloads through the use of dynamic time quantum and process preemption. This approach ensures optimal CPU utilization and provides fair scheduling. By prioritizing processes with shorter remaining times and dynamically adjusting priorities, the algorithm reduces waiting time and maximizes system efficiency, demonstrating its applicability in real-world operating systems with varying workloads.

**9)References-**

 Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. John Wiley & Sons.

 Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems*. Pearson.

 Python Documentation on heapq: [**https://docs.python.org/3/library/heapq.html**](https://docs.python.org/3/library/heapq.html)

 CPU Scheduling Algorithms Overview:[**https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/**](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)