A

Microproject report on

"Priority Scheduling Algorithm"

SUBMITTED TO M.S.B.T.E., Mumbai

Forthe Award of

DIPLOMA INCOMPUTER ENGINEERING
BY

Roll No.	Name of Student	Enrollment No.	Sign
04	Lokhande Isha Abhijit	2210740163	
20	Huke Sakshi Kishor	2210740180	
45	Asabe Ananya Dhananjay	2210740208	
59	Patil Srushti Sachin	2210740226	
61	Tapkire Mayuri Mahesh	2210740338	

UNDER THE GUIDANCE OF

Ms. V. R. Muttagi

DEPARTMENT OF COMPUTER ENGINEERING

NBA ACCREDIATED

SVERI's College of Engineering

(Polytechnic),Pandharpur Gopalpur

Pandharpur-413304 2024-25

AFFILIATED TO

M.S.B.T.E.

Annexure II

Evolution sheet for Micro Project

Academic Year: - 2024-25	**Name of Faculty: -** Ms. V.R . Muttagi
Course: - Computer Engineering	**Course code: -** CO5I
Subject: - Operating System	**Subject Code: - 22516**
Semester: -5th	**Scheme: - I**

Title of Project: -	Priority Scheduling Algorithm

Major Learning Outcomes achieved by students by doing the Project:

(a)Practical Outcomes:	
(b) Unit Outcomes in Cognitive Domain:	1. Calculate turnaround time and average waiting time for a given scheduling algorithm
(c) Outcomes in Affective Domain:	

Comments/Suggestions about teamwork/leadership/inter-personal communication (if any)

Roll No.	Name of Student	Marks out of 6 for performancein group activity	Marksout of 4 for performance in oral/Presentation	Totalmarksout of 10
04	Lokhande Isha Abhijit			
20	Huke Sakshi Kishor			
45	Asabe Ananya Dhananjay			
59	Patil Srushti Sachin			
61	Tapkire Mayuri Mahesh			

Name & Signature of faculty	Ms. V.R . Muttagi	

SVERI's COLLEGE OF ENGINEERING (POLYTECHNIC), PANDHARPUR.

CERTIFICATE

This is to certify that the Project report entitled

"Priority Scheduling Algorithm"

Submitted by

Roll No.	**Name of Student**	**Enrollment No.**
04	Lokhande Isha Abhijit	2210740163
20	Huke Sakshi Kishor	2210740180
45	Asabe Ananya Dhananjay	2210740208
59	Patil Srushti Sachin	2210740226
61	Tapkire Mayuri Mahesh	2210740338

is a bonafide work carried out by above student, under the guidance of Ms. V. R. Muttagi and it is submitted towards the fulfillment of requirement of MSBTE, Mumbai for the award of Diploma in Computer Engineering at SVERI's COE (Poly.), Pandharpur during the academic year 2024 - 2025.

(Ms. V. R. Muttagi)
Guide

(Mr. P. S. Bhandare)
HOD

(Dr. N. D. Misal)
Principal

Place: Pandharpur
Date:

Acknowledgement

Priority Scheduling Algorithm has been developed successfully with a great contribution of five students in a period of two months. We like to appreciate their guidance, encouragement and willingness since without their support the project would not have been a success. We would like to give our heartfelt gratitude to Principal Dr. N. D. Misal, HOD Mr. P. S. Bhandare & Guide Ms. V. R. Muttagi who is the supervisor of our project for helping and encouraging us in many ways to make our project a success. We would never been able to finish our work without great support and enthusiasm from friends and support from our family. We would like to thank the department of computer engineering, for giving us permission to initiate this project and successfully finish it.

INTRODUCTION

1. Rationale:

Priority scheduling in operating systems is a technique that determines the order in which processes are executed based on their priority levels. This approach ensures that critical tasks receive the necessary CPU time while allowing less urgent processes to wait.

Effective priority scheduling is vital for optimizing system performance, resource allocation, and responsiveness, especially in real-time systems where timing is crucial. Operating systems employ various algorithms to implement priority scheduling, including fixed-priority scheduling and dynamic- priority scheduling. The rationale behind adopting priority scheduling is to enhance system efficiency, ensure timely execution of important tasks, and improve overall user experience by minimizing wait times for critical processes.

2. Aim/Benefits of the Micro-project:

The aim of this micro-project is to explore the concept of priority scheduling in operating systems, focusing on its implementation, advantages, and potential challenges.

Benefits of priority scheduling include:

i. Higher-priority processes receive immediate attention, reducing latency for critical applications.
ii. Prioritizing processes ensures optimal use of CPU and other resources.
iii. Essential for applications that require strict timing, such as embedded systems and multimedia processing.

3. Course Outcomes Achieved:

Apply scheduling algorithm to calculate turnaround time and average waiting time

4. Literature Review:

A literature review on priority scheduling in operating systems reveals a rich landscape of methodologies and advancements aimed at optimizing process management. Traditional fixed-priority scheduling assigns static priorities to tasks, demonstrating effectiveness in periodic task management but often encountering issues such as priority inversion, where lower-priority tasks block higher- priority ones.

• Introduction to CPU Scheduling:-

CPU scheduling is the process of determining which process in the ready queue should be executed by the CPU at any given time. In a multitasking environment, multiple processes may be ready for execution, but the CPU can handle only one process at a time. The scheduler's role is to maximize CPU utilization, improve system performance, and ensure fairness among all processes by deciding the order in which processes should run.

➢ Types of CPU Scheduling:-

1. First-Come, First-Served (FCFS)
2. Shortest Job Next (SJN) / Shortest Job First (SJF)
3. Priority Scheduling
4. Round Robin (RR)
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling
7. Shortest Remaining Time First (SRTF)
8. Longest Job First (LJF)
9. Longest Remaining Time First (LRTF)

- **Priority Scheduling:-**

Priority Scheduling is a CPU scheduling method where each process is assigned a priority, and the CPU is allocated to the process with the highest priority. This allows more critical or important processes to be executed first, improving the responsiveness of systems that need to prioritize certain tasks over others. The priority can be determined based on various factors such as the importance of the task, user-defined criteria, or the process's resource requirements. Once a high-priority process arrives, it is either allowed to preempt the currently running process (preemptive priority scheduling) or waits for the running process to finish before being executed (non-preemptive priority scheduling).

One of the main advantages of priority scheduling is its flexibility. It allows the system to handle tasks of varying importance by dynamically adjusting the order in which they are executed. For example, time-sensitive tasks such as real-time processing can be given higher priority, ensuring they are handled quickly without unnecessary delay. This can improve overall system efficiency, especially in environments where certain tasks are more urgent than others.

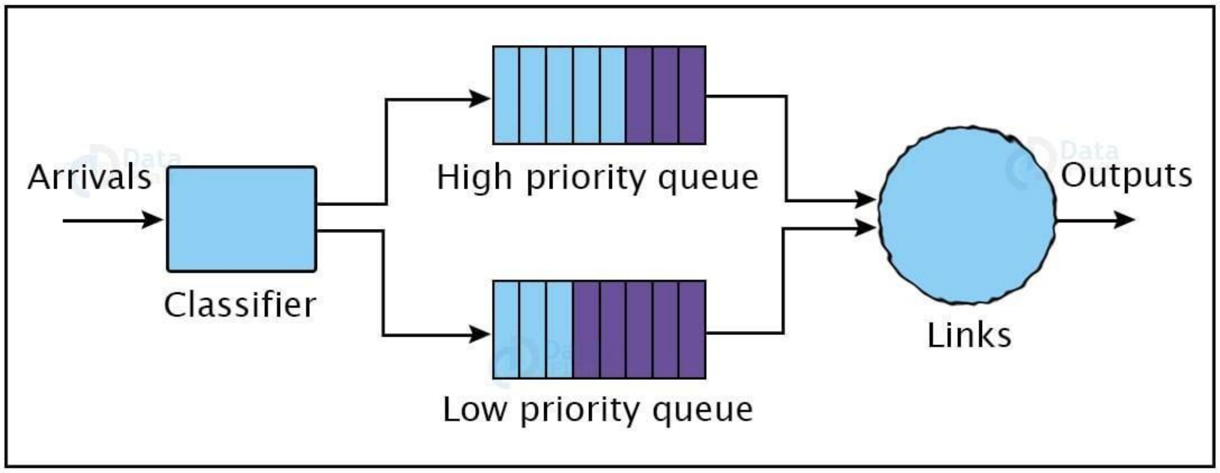


Fig. Priority Scheduling

This diagram represents a **priority queue system**. Incoming processes (arrivals) are classified into **high priority** and **low priority** by the classifier. The classified processes are placed into their respective queues (high or low). The high-priority queue is serviced before the low-priority queue. Both queues feed into a shared set of output links, and the processes are dispatched based on their priority.

- **Implementation of Priority Scheduling Algorithm–**

To implement the **Priority Scheduling Algorithm**, follow these steps:

i. Input Process Details:
Gather details for each process, including its burst time (execution time) and priority value. The lower the priority number, the higher the priority (this can be system-specific).

ii. Sort Processes by Priority:
Sort the processes in the ready queue based on their priority. If two processes have the same priority, you can break the tie using their arrival time (FCFS for equal priority).

iii. Select Highest Priority Process:
Pick the process with the highest priority (i.e., the lowest priority number). If the scheduling is preemptive, check if a new process with a higher priority arrives, and if so, preempt the currently running process.

iv. Execute the Process:
Allocate the CPU to the selected process and start its execution. For non-preemptive scheduling, let the process complete before moving to the next. For preemptive scheduling, switch to the new process if it has a higher priority.

v. Calculate Times:
Track waiting time, turnaround time, and completion time for each process. These metrics are essential to evaluate the performance of the scheduling algorithm.
Waiting Time = Start Time - Arrival Time. Turnaround Time = Completion Time - Arrival Time.
Completion Time is when the process finishes execution.

vi. Repeat for Remaining Processes:
Continue selecting the next highest-priority process from the ready queue until all processes are completed.

vii. Output Results:
Display the order of execution, waiting times, turnaround times, and average waiting/turnaround times for all processes to assess the efficiency of the scheduling.

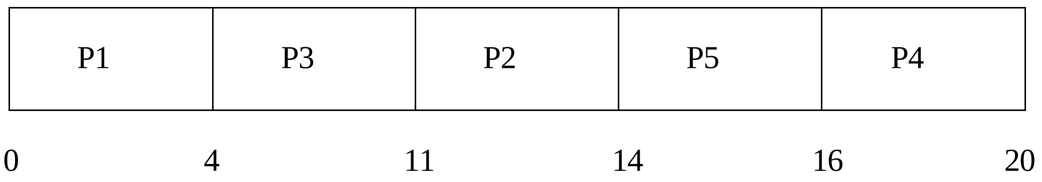
- **Example of Priority Scheduling**

The Jobs are scheduled as below:-

Process	Priority	Burst Time
P1	1	4
P2	2	3
P3	1	7
P4	3	4
P5	2	2

Solution:-

Gantt Chart:-



Waiting Time:-
P1=0
P2=11
P3=4
P4=16
P5=14

Average Waiting Time $= \frac{0+4+11+14+16}{5}$

$= \frac{45}{5}$

= 9ms

Turnaround time
P1=4
P3=11
P2=14
P4=20
P5=16

Average Turnaround Time $= \frac{4+11+14+20+16}{5}$

$= \frac{65}{5}$

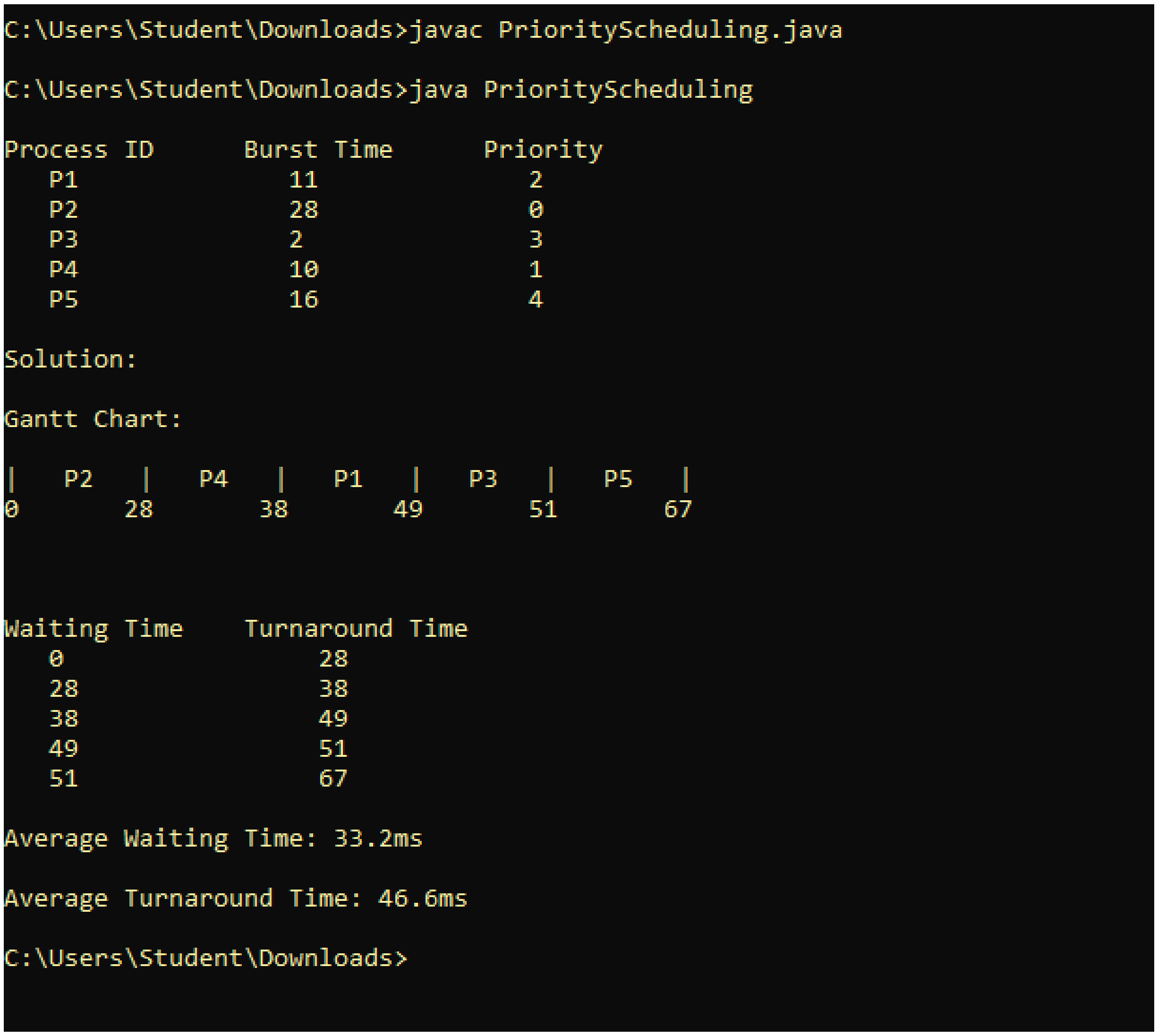
= 13ms

- **Implementation of Priority Scheduling algorithm using Java Code**

Java Code:-

```
import java.util.*;

class Process {
    String pid;
    int burst_time;
    int priority;

    Process(String i, int b, int p) {
        pid = i;
        burst_time = b;
        priority = p;
    }
}

class PriorityScheduling {
    public static void main(String arg[]) {
        List<Process> l = new ArrayList<>();
        l.add(new Process("P1", 11, 2));
        l.add(new Process("P2", 28, 0));
        l.add(new Process("P3", 2, 3));
        l.add(new Process("P4", 10, 1));
        l.add(new Process("P5", 16, 4));

        System.out.println("\nProcess ID\tBurst Time\tPriority");
        for (Process p : l) {
            System.out.println("   " + p.pid + "\t\t   " + p.burst_time + "\t\t   " + p.priority);
        }

        Collections.sort(l, new Comparator<Process>() {
            public int compare(Process p1, Process p2) {
                return Integer.compare(p1.priority, p2.priority);
             }
         });
System.out.println("\nSolution: ");

        System.out.println("\nGantt Chart:");
        System.out.print("\n|");
        for (Process p : l) {
            System.out.print("   " + p.pid + "   |");
        }
        System.out.println();
        System.out.print("0");

        int currentTime = 0;
        for (Process p : l) {
            currentTime += p.burst_time;
            System.out.print("        " + currentTime);
        }
        System.out.println();
        int[] waitingTime = new int[l.size()];
        int[] turnaroundTime = new int[l.size()];
        waitingTime[0] = 0;

        for (int i = 1; i < l.size(); i++) {
            waitingTime[i] = waitingTime[i - 1] + l.get(i - 1).burst_time;
        }

        for (int i = 0; i < l.size(); i++) {
            turnaroundTime[i] = waitingTime[i] + l.get(i).burst_time;
        }

        System.out.println("\n\n\nWaiting Time\tTurnaround Time");
        for (int i = 0; i < l.size(); i++) {
            System.out.println("   " + waitingTime[i] + "\t\t      " + turnaroundTime[i]);
        }

        double averageWaitingTime = 0.0;
        for (int i = 0; i < l.size(); i++) {
            averageWaitingTime += waitingTime[i];
        }
        averageWaitingTime = averageWaitingTime / l.size();
        System.out.println("\nAverage Waiting Time: " + averageWaitingTime + "ms");

        double averageTurnaroundTime = 0.0;
        for (int i = 0; i < l.size(); i++) {
            averageTurnaroundTime += turnaroundTime[i];
        }
        averageTurnaroundTime = averageTurnaroundTime / l.size();
        System.out.println("\nAverage Turnaround Time: " + averageTurnaroundTime + "ms");
    }
}
```

Output:-



```
C:\Users\Student\Downloads>javac PriorityScheduling.java

C:\Users\Student\Downloads>java PriorityScheduling

Process ID      Burst Time      Priority
   P1              11              2
   P2              28              0
   P3              2               3
   P4              10              1
   P5              16              4

Solution:

Gantt Chart:

|   P2   |   P4   |   P1   |   P3   |   P5   |
0       28       38       49       51       67



Waiting Time    Turnaround Time
   0                28
   28               38
   38               49
   49               51
   51               67

Average Waiting Time: 33.2ms

Average Turnaround Time: 46.6ms

C:\Users\Student\Downloads>
```

- **Advantages of Priority Scheduling:-**

Efficient Task Management: Priority scheduling ensures that critical tasks are completed first, making it suitable for real-time systems where high-priority processes must meet deadlines.

Flexibility**:** It allows for dynamic task prioritization. Priorities can be adjusted based on the importance of tasks, allowing the system to respond effectively to changing requirements.

Reduced Waiting Time for Critical Processes**:** High-priority tasks experience minimal waiting time, improving the responsiveness of important applications or services.

Better Resource Utilization: By allocating resources based on priority, important processes can be executed efficiently, preventing less important tasks from monopolizing resources.

Adaptability: It can be used in both preemptive and non-preemptive modes, depending on whether processes can be interrupted.

Improved Performance in Certain Workloads: In environments where some tasks are significantly more important than others (e.g., operating systems or network routers), priority scheduling enhances overall system performance by focusing on the most critical tasks.

.

- **Disadvantages of Priority Scheduling:-**

Starvation (Indefinite Blocking):- Low-priority processes may never get executed if higher-priority processes continue to arrive, leading to starvation. This is particularly problematic in systems with a large number of high-priority tasks.

Complexity in Priority Assignment:- Determining and assigning appropriate priorities can be difficult and may require constant adjustment based on task urgency or system conditions.

Lack of Fairness:- Priority scheduling can be unfair to lower-priority processes, as they may experience excessive delays, even if they have been waiting for a long time.

Potential for Priority Inversion:- A lower-priority task may hold a resource that a higher-priority task needs, causing the higher-priority task to wait. This can lead to performance issues, known as priority inversion, unless mechanisms like priority inheritance are implemented.

5. Actual Resources Used:

Sr.No	Name of Recourses	Specification	Qty	Remark
1	Laptop	RAM: 8GB, Processor: i5, HDD: 1TB, Graphic: 2GB	1	For coding and testing the scheduling algorithm
2	JDK (Java Development Kit)	JDK Version 8 or above	1	Used for compiling and running Java code
3	Operating System	Windows/Linux	1	Required for running the scheduling simulations
4	Reference Books/Online Resources	Algorithms, Scheduling, OS Concepts	Multiple	For studying and implementing the priority scheduling algorithm

6. Skill Developed/Learning outcome of Micro-Project:

The implementation of the priority scheduling algorithm enhances understanding of task prioritization in real-world systems, such as operating systems and network management. It offers insights into resource allocation, time management, and process scheduling, while improving analytical skills to resolve issues like starvation and priority inversion. The project also fosters problem-solving skills in designing efficient scheduling mechanisms and optimizing system performance through better task management.

7. Applications of this Micro-Project:

1. Operating Systems: Used to prioritize critical system tasks (e.g., kernel processes) over user applications to ensure smooth system operation.
2. Real-Time Systems: Essential for time-sensitive applications like air traffic control, where high-priority tasks must meet strict deadlines.
3. Network Routers: Helps in prioritizing high-priority packets (e.g., voice or video traffic) over regular data traffic to ensure quality of service.
4. Embedded Systems: Useful in devices like pacemakers, where certain tasks must execute with minimal delay to ensure functionality.
5. Multimedia Applications: Ensures real-time media processing tasks (e.g., video rendering) get priority over less urgent tasks, improving user experience.

8. Conclusion:

The priority scheduling algorithm effectively demonstrates how processes can be prioritized based on their importance or urgency in a multitasking environment. By assigning different priority levels to tasks, the system ensures that critical processes are executed first, improving overall efficiency in handling time-sensitive operations. The implementation of this algorithm provides valuable insights into real-world scheduling techniques used in operating systems, network management, and embedded systems. Although it has several advantages, such as reduced waiting time for high-priority tasks, there are challenges like potential starvation and priority inversion.

9. References:

https://data-flair.training/blogs/priority-scheduling-algorithm-in-operating-system/

https://www.geeksforgeeks.org/priority-cpu-scheduling-with-different-arrival-time-set-2/

https://unstop.com/blog/scheduling-algorithms-in-operating-system/