# LABORATORY EXERCISE 8

# PRIORITY SCHEDULING

**Learning Objectives**

* To understand and implement priority scheduling in process management.
* To differentiate between preemptive and non-preemptive scheduling approaches.
* To analyze the efficiency of scheduling algorithms based on priority levels and observe their impact on process completion times.

**Prerequisite student experiences and knowledge**

A basic understanding of CPU scheduling algorithms and process management in operating systems is essential for optimizing how processes are executed and managed. Familiarity with key concepts such as burst time, waiting time, and turnaround time, along with their calculation in scheduling algorithms, helps analyze and improve system performance. Additionally, experience in C or C++ programming, particularly with loops, arrays, and basic input/output handling, is crucial for implementing these scheduling algorithms effectively and understanding their impact on real-time process execution.

**Background**

Priority scheduling is an approach used by operating systems to determine the order in which processes run based on their assigned priorities. In preemptive priority scheduling, a higher-priority process can interrupt a currently running process if it arrives later but has a higher priority. In non-preemptive scheduling, however, once a process starts executing, it completes its burst time without interruption. Understanding these two approaches allows students to gain insights into efficient process management and the challenges of balancing fairness and responsiveness in an operating system.

This exercise will involve implementing both preemptive and non-preemptive priority scheduling in C++ and observing how each approach impacts the turnaround and waiting times of processes..

**Materials/Resources**

* PC/Internet
* Pen
* C++ (such as DevC++, TurboC++, or any standard IDE).
* Web Browser (Internet Explorer, Mozilla, Google Chrome, Etc.)
* Word-processing program

**Laboratory Activity**

**Instructions:** *Perform the following steps.*

**Step 1: Define the Process Structure**

* + Start your C++ IDE and create a new file called priority\_scheduling.cpp.
  + Define a structure called Process with attributes such as pid, arrival\_time, burst\_time, priority, start\_time, finish\_time, and waiting\_time.

**Step 2: Input Process Information**

* + Write a function that prompts the user to enter the number of processes and, for each process, input the arrival\_time, burst\_time, and priority values.
  + Store these processes in an array or vector.

**Step 3: Implement Non-Preemptive Priority Scheduling**

* + Write a function named nonPreemptiveScheduling() that sorts the processes based on their priority (higher priority processes run first).
  + Calculate the start\_time and finish\_time for each process. Ensure that the program waits until a process arrives before scheduling it.
  + Calculate the waiting time and turnaround time for each process and display the results.

**Step 4: Implement Preemptive Priority Scheduling**

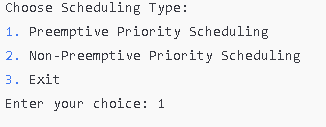
* + Write a function called preemptiveScheduling() that continuously checks for any arriving higher-priority processes while a process is running.
  + If a higher-priority process arrives during execution, preempt the current process, push it back into the queue, and start the higher-priority process.
  + Use time-slicing to simulate real-time process arrival and track the remaining\_burst\_time of each process.

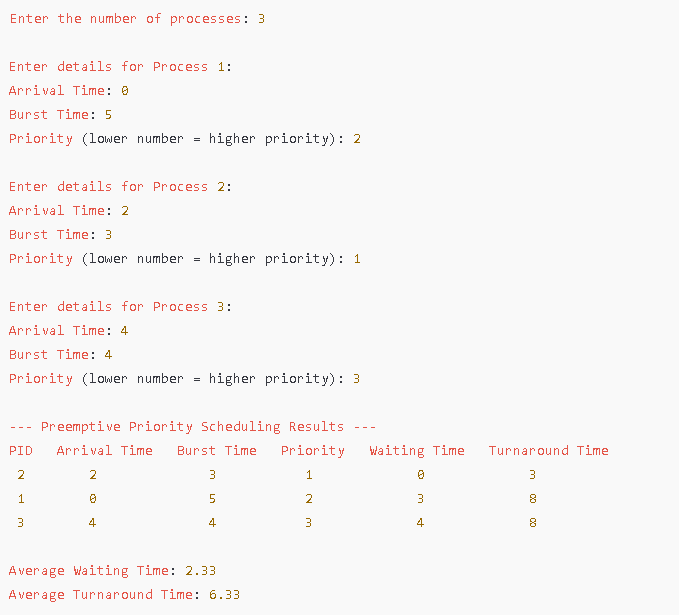
**Step 5: Display Results**

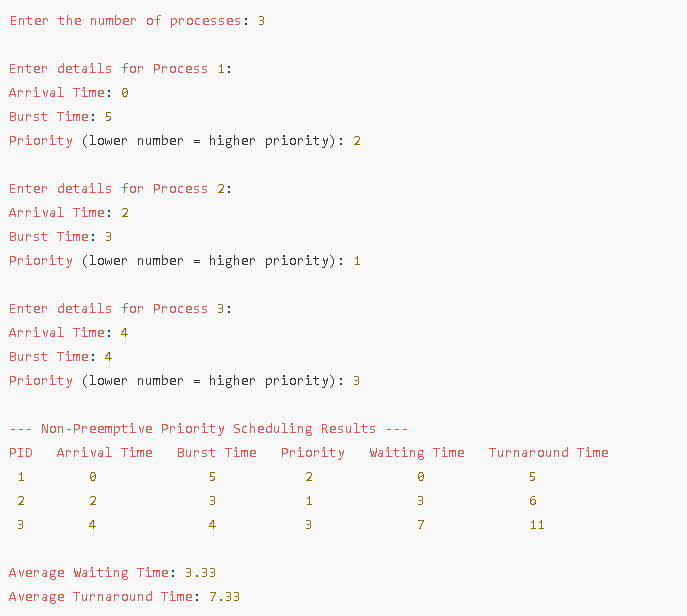
* + After executing each scheduling function, display the following details for each process:
    - Process ID (PID)
    - Arrival Time
    - Burst Time
    - Priority
    - Waiting Time
    - Turnaround Time

**Step 6: Testing and Output**

* + Run both preemptive and non-preemptive scheduling algorithms with the same process data.
  + Capture and print the results of both algorithms to compare waiting and turnaround times.
  + Take screenshots of the console output for documentation.

**Sample:**

**Preemptive**

** Non-Preemptive**

**Test Table**

**Table 1 (PREEMPTIVE)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process Number** | **Arrival Time** | **Burst Time** | **Priority** | **Completion Time** | **Waiting Time** | **Turnaround Time** |
| **A** | **0** | **9** | **5** | **13** | **4** | **13** |
| **B** | **3** | **1** | **6** | **4** | **0** | **1** |
| **C** | **1** | **6** | **2** | **39** | **32** | **38** |
| **D** | **8** | **2** | **3** | **29** | **19** | **21** |
| **E** | **2** | **8** | **3** | **27** | **17** | **25** |
| **F** | **7** | **3** | **6** | **10** | **0** | **3** |
| **G** | **0** | **4** | **2** | **33** | **29** | **33** |
| **H** | **4** | **2** | **1** | **41** | **35** | **37** |
| **I** | **2** | **5** | **4** | **19** | **12** | **17** |
| **J** | **5** | **1** | **5** | **14** | **8** | **9** |

**Average Waiting Time**

**AWT = (4+0+32+19+17+0+29+35+12+8)/10**

**= 156/10**

**= 15.6**

**Average Turnaround Time**

**ATAT= (13+1+38+21+25+3+33+37+17+9)/10**

**= 197/10**

**= 19.7**

**Gantt Chart**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **A** | **A** | **B** | **A** | **A** | **A** | **F** | **F** | **F** | **A** | **J** | **I** | **E** | **D** | **G** | **C** | **H** |

**0 1 2 3 4 5 6 7 8 9 10 13 14 19 27 29 33 39 41**

**Table 1.2 (NON-PREEMPTIVE)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process Number** | **Arrival Time** | **Burst Time** | **Priority** | **Completion Time** | **Waiting Time** | **Turnaround Time** |
| **A** | **1** | **3** | **5** | **13** | **9** | **12** |
| **B** | **0** | **1** | **1** | **29** | **28** | **29** |
| **C** | **2** | **5** | **4** | **18** | **11** | **16** |
| **D** | **3** | **3** | **8** | **8** | **2** | **5** |
| **E** | **2** | **1** | **3** | **19** | **16** | **17** |
| **F** | **0** | **5** | **6** | **5** | **0** | **5** |
| **G** | **6** | **4** | **2** | **28** | **18** | **22** |
| **H** | **4** | **3** | **2** | **24** | **17** | **20** |
| **I** | **2** | **2** | **7** | **10** | **6** | **8** |
| **J** | **5** | **2** | **3** | **21** | **14** | **16** |

**Average Waiting Time**

**AWT = (9+28+11+2+16+0+18+17+6+14 ) / 10**

**= 121 / 10**

**= 12.1**

**Average Turnaround Time**

**ATAT = (12+29+16+5+17+5+22+20+8+16) / 10**

**= 150 / 10**

**= 15**

**Gantt Chart**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **F** | **D** | **I** | **H** | **C** | **E** | **J** | **H** | **G** | **B** |

**0 5 8 10 13 18 19 21 24 28 29**

**QUESTIONS (10 pts each)**

1. Describe a scenario where preemptive priority scheduling is more beneficial than non-preemptive priority scheduling.

-Little time to waste when a medical emergency comes along. In non-preemptive scheduling, if a high-priority task arrived while the execution of a lower priority was being carried out then the former had to wait until the latter completed its execution. In a preemptive scheduling, the machine can right away switch over to the higher-priority task, and makes sure that tasks are attended to immediately that are life-critical.

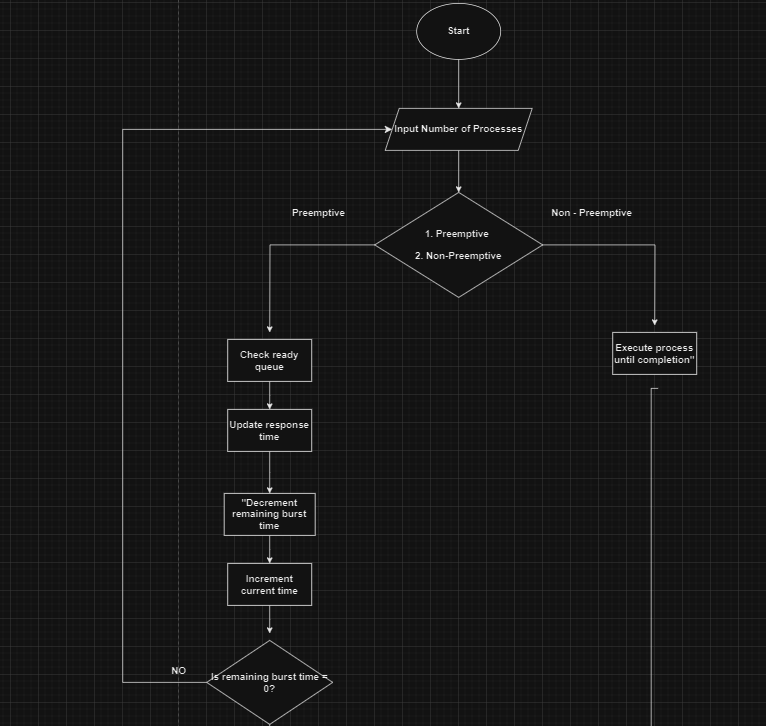
1. Discuss the impact of frequent process switching in preemptive scheduling on system performance.

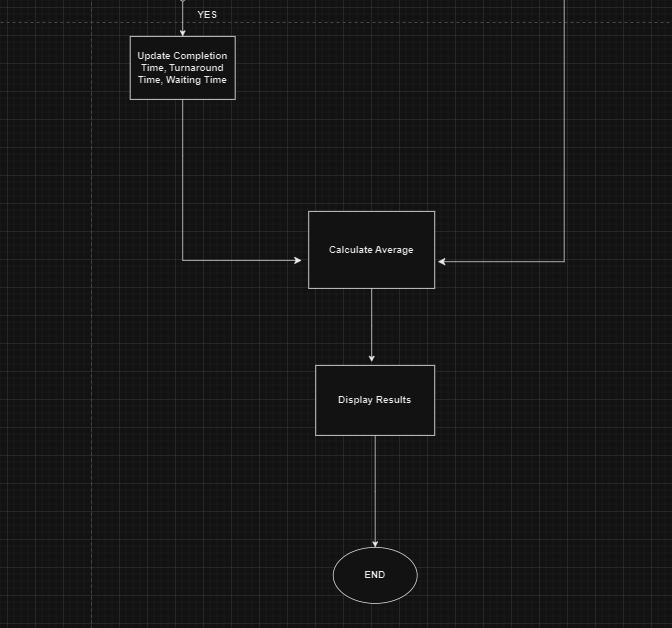
- The major source of overhead in preemptive scheduling is frequent context switching due to the loss of time incurred in switching contexts, which reduces the available CPU time for actual computations. It might also create problems with caches and synchronization by introducing too many race conditions and deadlocks because of interruptions to too many processes. However, it ensures timely handling of high-priority tasks to improve responsiveness of time-critical systems like those in real-time applications.

1. How would the priority scheduling algorithm change if lower numbers indicate higher priority?

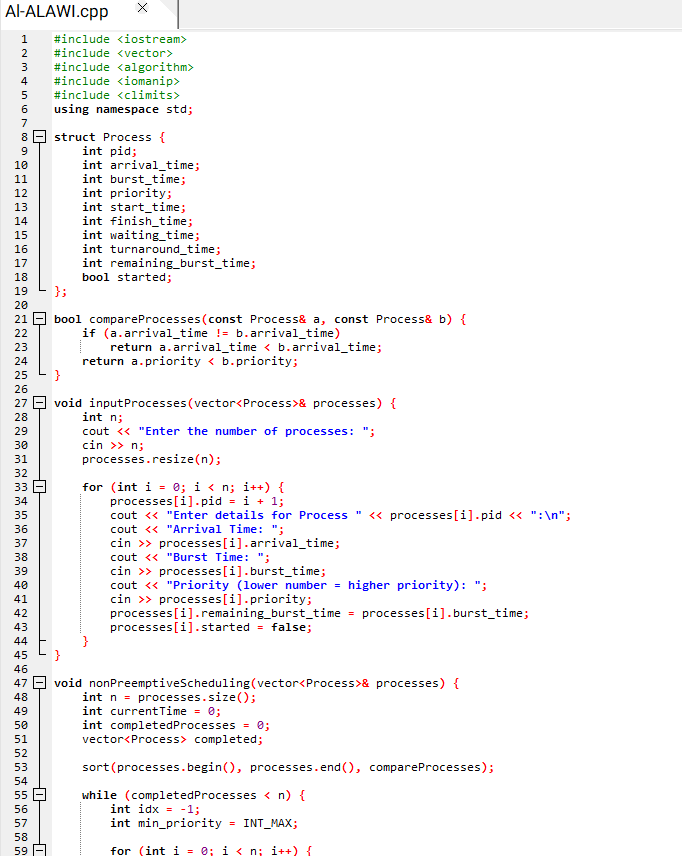
-It would schedule the process with the lowest numerical value first in a priority scheduling algorithm that uses lower numbers to represent higher priority. Processes with lower-priority numbers-for example, priority 1-will therefore be executed before those with higher numbers-for example, priority 5. The scheduling will ensure that higher-priority tasks (lower numbers) are handled sooner, which consequently enables more urgent processes to preempt others.

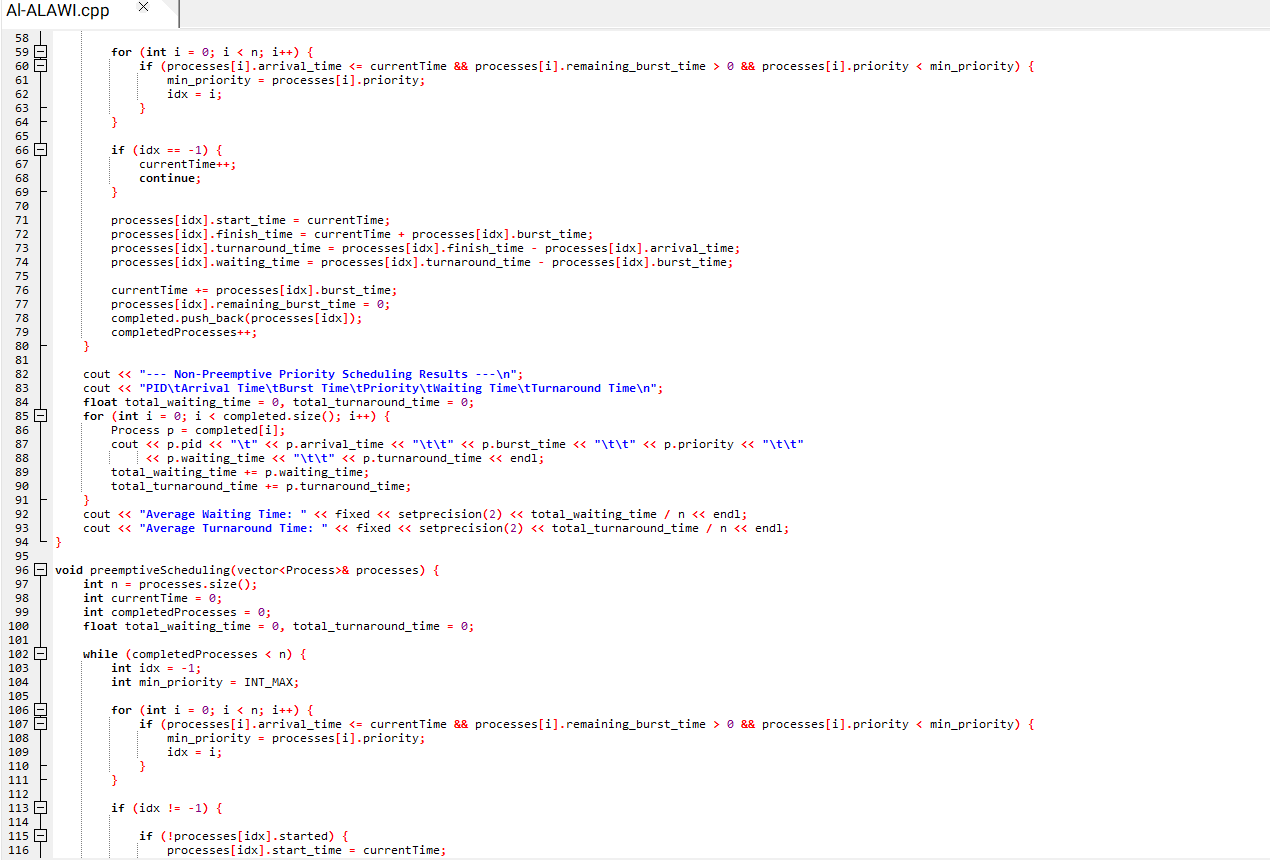
1. Provide a Flowchart of the given task.

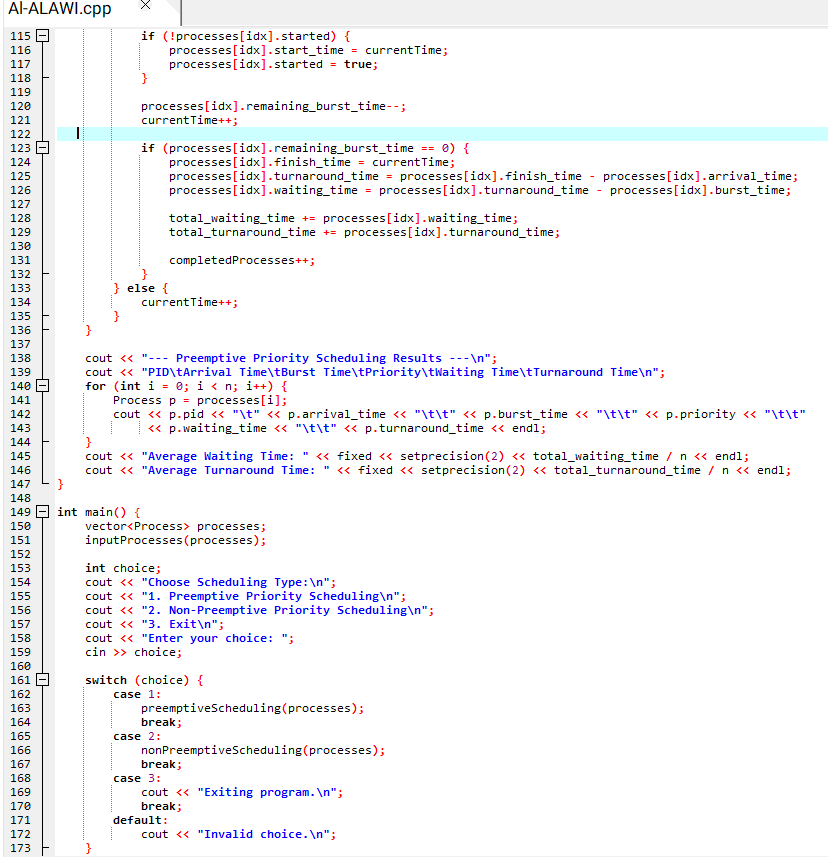


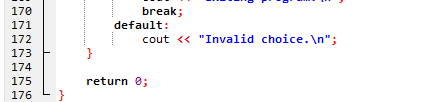


**Source Code**

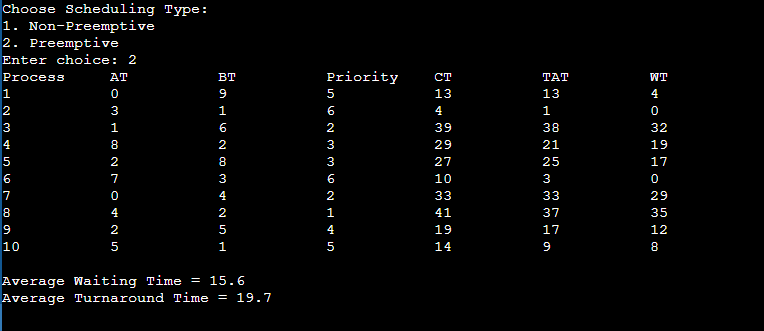


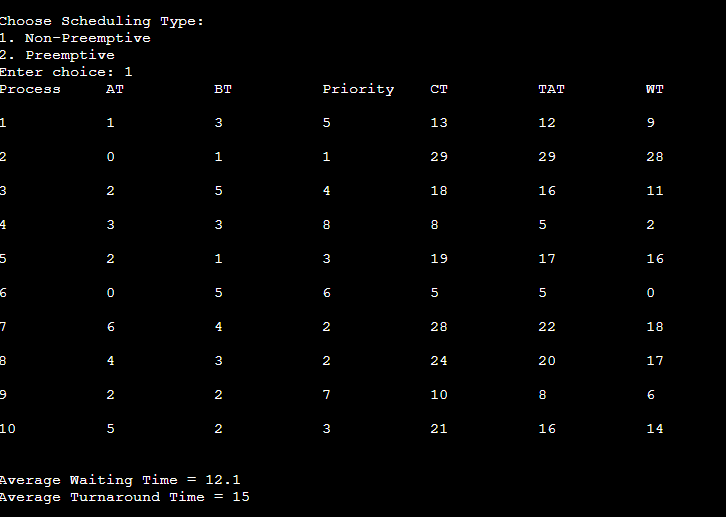






**Output / Results**





**Conclusion**

**Priority Scheduling is a CPU methodology in which processes are assigned priority; that is, the process holding the highest priority, which, by convention, is the lowest number, runs first. It can run in two modes-preemptive or nonpreemptive. In preemptive mode, higher priority processes may interrupt lower priority ones, while under a nonpreemptive mode, a process runs to completion after having been started. Although this algorithm is useful for prioritizing critical tasks, it has significant drawbacks in starvation, where low-priority processes may not get executed unless mechanisms like aging are implemented. Generally, Priority Scheduling is very effective for the management of differently ranked importance levels, but a lot of careful management needs to be done to avoid fairness issues while allowing the smooth execution of all processes.**