Lab 5: Preemptive and Non-Preemptive Priority Scheduling Algorithm

Param Makwana	F004	F1	02/02/2024

1. Introduction

Scheduling algorithms play a crucial role in the management of processes in an operating system. Priority scheduling is one such algorithm that assigns priorities to each process and determines the order in which they are executed. In this lab, you will explore two variations of priority scheduling: preemptive and non-preemptive. Preemptive priority scheduling allows a higher-priority process to interrupt the execution of a lower-priority process, while non-preemptive priority scheduling completes the execution of the current process before switching to the highest-priority one.

2. Objective

The objective of this lab is to understand and implement preemptive and non-preemptive priority scheduling algorithms, including their procedures and differences.

3. Equipment and Software Requirements

- A computer with a suitable operating system (e.g., Windows, Linux, macOS)

- Programming environment (e.g., C/C++ compiler)

- Code editor (e.g., Visual Studio Code, Dev C++, or any preferred IDE)

4. **Preemptive Priority Scheduling Algorithm**

4.1. Theory

Preemptive Priority Scheduling is an algorithm that selects the process with the highest priority for execution. If a higher-priority process arrives during the execution of a lower-priority process, it preempts the lower-priority one and starts executing.

4.2. Procedure

1. Assign priorities to each process. Lower numbers typically indicate higher priorities.

2. Initialize a queue to hold the processes.

3. While there are processes in the queue:

   a. Select the process with the highest priority from the queue.

   b. Execute the selected process.

c. If a higher-priority process arrives while executing, preempt the current process.

d. Remove the executed or preempted process from the queue.

e. Continue this process until all processes have been executed.

Process	Burst Time	Priority	AT
P_1	10	3	3
P_2	1	1	0
P_3	2	4	2
P_4	1	5	1
P_5	5	2	4

Create a Gantt chart to illustrate the execution order of these processes using preemptive priority scheduling.

5. Non-Preemptive Priority Scheduling Algorithm

5.1. Theory

Non-preemptive Priority Scheduling is an algorithm that selects the highest-priority process for execution. However, once a process starts executing, it will continue until completion, even if a higher-priority process arrives later.

5.2. Procedure

1. Assign priorities to each process. Lower numbers typically indicate higher priorities.

2. Initialize a queue to hold the processes.

3. Sort the processes in the queue based on their priorities, with the highest-priority process at the front.

4. Execute the process at the front of the queue.

5. Remove the executed process from the queue.

6. Repeat steps 3 to 5 until all processes have been executed.

Process	Burst Time	Priority	AT
P_1	10	3	3
P_2	1	1	0
P_3	2	4	2
P_4	1	5	1
P_5	5	2	4

Create a Gantt chart to illustrate the execution order of these processes using non-preemptive priority scheduling.

6. Conclusion

In this lab, you have learned about preemptive and non-preemptive priority scheduling algorithms, their procedures, and differences. You have also applied these algorithms to a practical example to understand their execution order. Scheduling algorithms are essential for managing processes efficiently in operating systems.

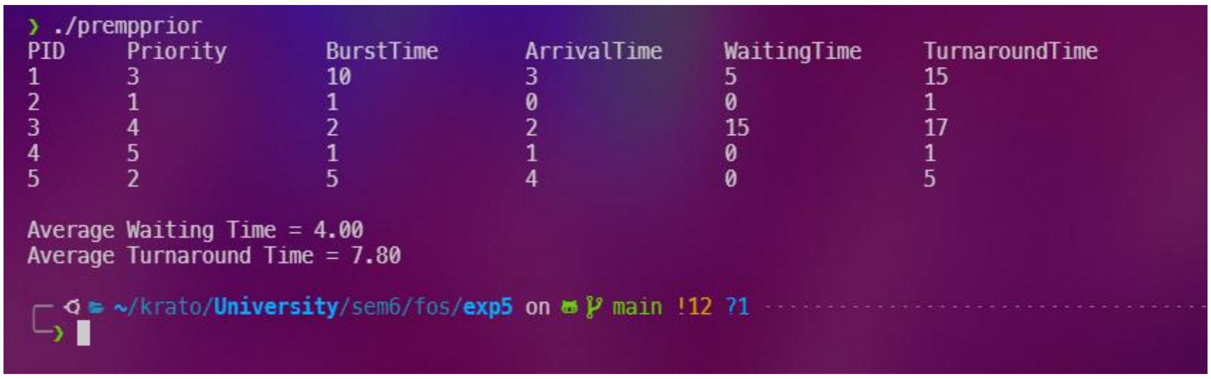
7. References

[1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts, 10th Edition. Wiley.

Program for preemptive priority scheduling:

```
#include <stdio.h>
#include <stdbool.h>

#define N 5  // Number of processes

// Process structure
typedef struct {
   int pid;
   int priority;
   int burstTime;
   int arrivalTime;
   int remainingTime;
   int completionTime;
   int waitingTime;
   int turnaroundTime;
} Process;
```

```
// Function declarations
void calculateCompletionTime(Process proc[], int n);
void calculateTurnaroundTime(Process proc[], int n);
void calculateWaitingTime(Process proc[], int n);
void preemptivePriorityScheduling(Process proc[], int n);
void printProcesses(Process proc[], int n);
void calculateAverageTimes(Process proc[], int n);

int main() {
    Process proc[N] = {
        {1, 3, 10, 3, 10, 0, 0, 0},
        {2, 1, 1, 0, 1, 0, 0, 0},
        {3, 4, 2, 2, 2, 0, 0, 0},
        {4, 5, 1, 1, 1, 0, 0, 0},
        {5, 2, 5, 4, 5, 0, 0, 0}
    };

    preemptivePriorityScheduling(proc, N);
    calculateTurnaroundTime(proc, N);
    calculateWaitingTime(proc, N);
    printProcesses(proc, N);
    calculateAverageTimes(proc, N);

    return 0;
}

void preemptivePriorityScheduling(Process proc[], int n) {
    int currentTime = 0, completed = 0, prev = 0;
    bool isCompleted[N] = {false};

    while(completed != n) {
        int idx = -1;
        int min = 10000;  // Assuming a very high priority number
        for(int i = 0; i < n; i++) {
            if(proc[i].arrivalTime <= currentTime && !isCompleted[i]) {
                if(proc[i].priority < min) {
                    min = proc[i].priority;
                    idx = i;
                }
                if(proc[i].priority == min) {
                    if(proc[i].arrivalTime < proc[idx].arrivalTime) {
                        min = proc[i].priority;
                        idx = i;
                    }
                }
            }
        }
        if(idx != -1) {
            proc[idx].remainingTime--;
            currentTime++;
            if(proc[idx].remainingTime == 0) {
                proc[idx].completionTime = currentTime;
                completed++;
                isCompleted[idx] = true;
            }
        } else {
            currentTime++;
        }
    }
}

void calculateTurnaroundTime(Process proc[], int n) {
    for(int i = 0; i < n; i++)
        proc[i].turnaroundTime = proc[i].completionTime - proc[i].arrivalTime;
}

void calculateWaitingTime(Process proc[], int n) {
    for(int i = 0; i < n; i++)
        proc[i].waitingTime = proc[i].turnaroundTime - proc[i].burstTime;
}

void printProcesses(Process proc[], int n) {
    printf("PID\tPriority\tBurstTime\tArrivalTime\tWaitingTime\tTurnaroundTime\n");
    for(int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].pid, proc[i].priority, proc[i].burstTime,
               proc[i].arrivalTime, proc[i].waitingTime, proc[i].turnaroundTime);
    }
}

void calculateAverageTimes(Process proc[], int n) {
    float totalWaitingTime = 0, totalTurnaroundTime = 0;
    for(int i = 0; i < n; i++) {
        totalWaitingTime += proc[i].waitingTime;
        totalTurnaroundTime += proc[i].turnaroundTime;
    }
    printf("\nAverage Waiting Time = %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time = %.2f\n", totalTurnaroundTime / n);
}
-------------------Output----------------------------------
```



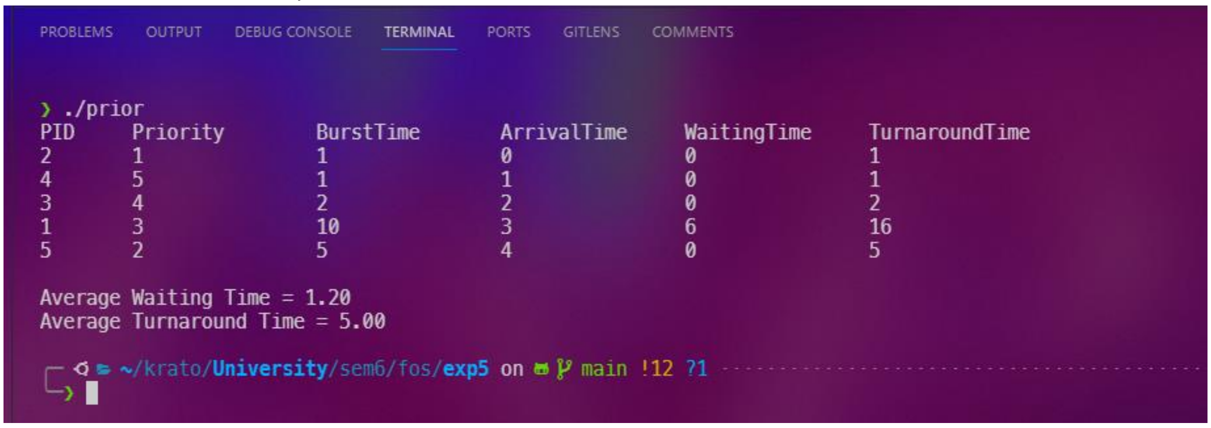
```
❯ ./prempprior
PID     Priority        BurstTime       ArrivalTime     WaitingTime     TurnaroundTime
1       3               10              3               5               15
2       1               1               0               0               1
3       4               2               2               15              17
4       5               1               1               0               1
5       2               5               4               0               5

Average Waiting Time = 4.00
Average Turnaround Time = 7.80

╭─ ~/krato/University/sem6/fos/exp5 on  main !12 ?1
╰─❯
```

Program for Non-preemptive priority scheduling

```c
#include <stdio.h>
#include <stdlib.h>

#define N 5  // Number of processes

// Process structure
typedef struct {
   int pid;
   int priority;
   int burstTime;
   int arrivalTime;
   int waitingTime;
   int turnaroundTime;
   int completionTime;
} Process;

// Function declarations
void sortByArrival(Process proc[], int n);
void nonPreemptivePriorityScheduling(Process proc[], int n);
void calculateWaitingTime(Process proc[], int n);
void calculateTurnaroundTime(Process proc[], int n);
void calculateAverageTimes(Process proc[], int n);
void printProcesses(Process proc[], int n);

int main() {
   Process proc[N] = {
      {1, 3, 10, 3, 0, 0, 0},
      {2, 1, 1, 0, 0, 0, 0},
      {3, 4, 2, 2, 0, 0, 0},
      {4, 5, 1, 1, 0, 0, 0},
      {5, 2, 5, 4, 0, 0, 0}
   };

   sortByArrival(proc, N);
```

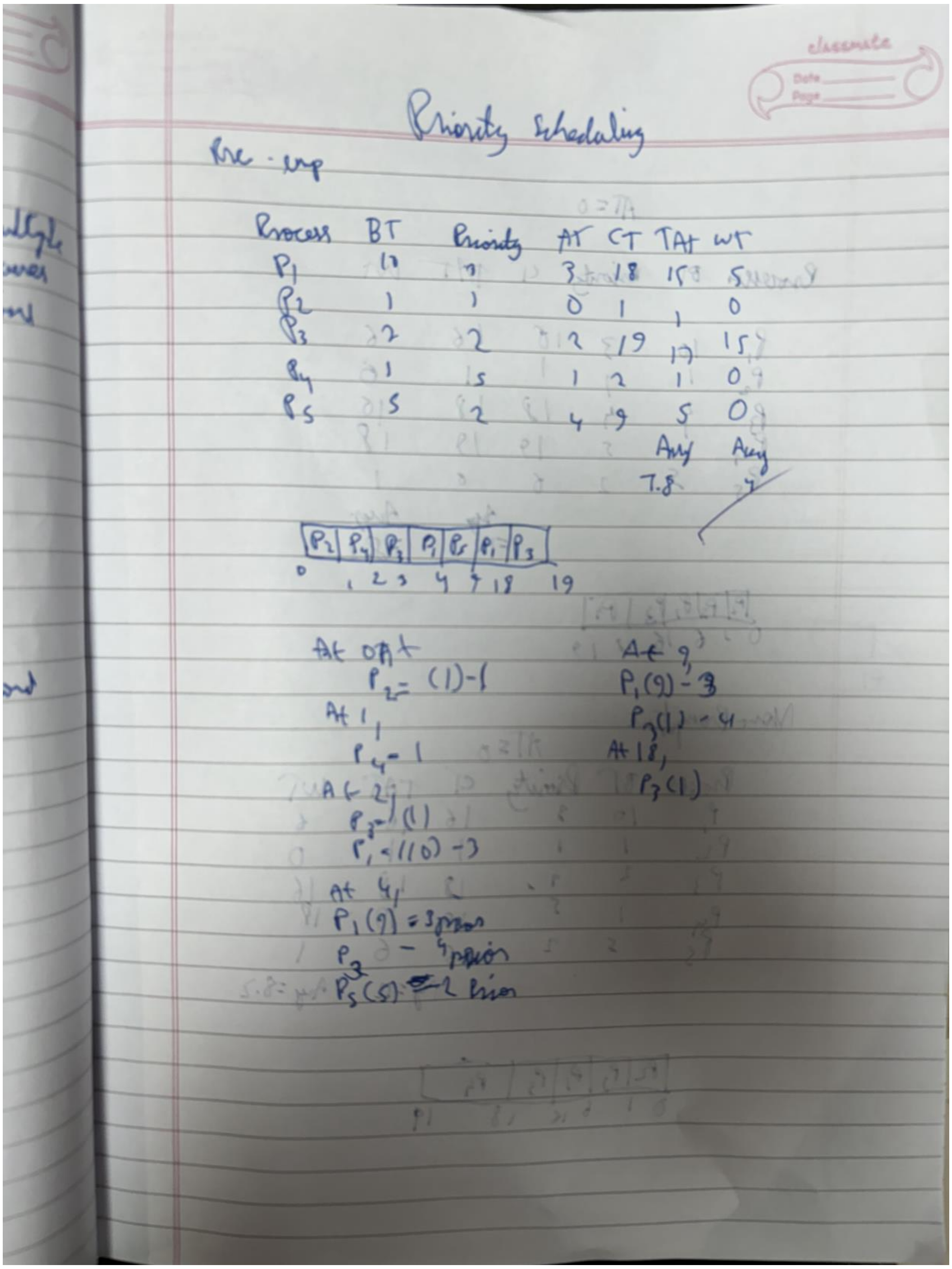
```
    nonPreemptivePriorityScheduling(proc, N);
    calculateWaitingTime(proc, N);
    calculateTurnaroundTime(proc, N);
    printProcesses(proc, N);
    calculateAverageTimes(proc, N);

    return 0;
}

void sortByArrival(Process proc[], int n) {
    Process temp;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].arrivalTime > proc[j].arrivalTime) {
                temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }
}

void nonPreemptivePriorityScheduling(Process proc[], int n) {
    int currentTime = 0, completed = 0;
    int isCompleted[N] = {0};
    while (completed != N) {
        int idx = -1;
        int min = 10000;  // Assuming a very high priority number
        for (int i = 0; i < N; i++) {
            if (proc[i].arrivalTime <= currentTime && !isCompleted[i]) {
                if (proc[i].priority < min) {
                    min = proc[i].priority;
                    idx = i;
                }
                if (proc[i].priority == min) {
                    if (proc[i].arrivalTime < proc[idx].arrivalTime) {
                        idx = i;
                    }
                }
            }
        }
        if (idx != -1) {
            currentTime += proc[idx].burstTime;
            proc[idx].completionTime = currentTime;
            completed++;
            isCompleted[idx] = 1;
        } else {
            currentTime++;
        }
    }
}

void calculateWaitingTime(Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].waitingTime = proc[i].completionTime - proc[i].burstTime - proc[i].arrivalTime;
    }
}

void calculateTurnaroundTime(Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnaroundTime = proc[i].completionTime - proc[i].arrivalTime;
    }
}

void calculateAverageTimes(Process proc[], int n) {
    float totalWaitingTime = 0, totalTurnaroundTime = 0;
    for (int i = 0; i < n; i++) {
        totalWaitingTime += proc[i].waitingTime;
        totalTurnaroundTime += proc[i].turnaroundTime;
    }
    printf("\nAverage Waiting Time = %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time = %.2f\n", totalTurnaroundTime / n);
}

void printProcesses(Process proc[], int n) {
    printf("PID\tPriority\tBurstTime\tArrivalTime\tWaitingTime\tTurnaroundTime\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].pid, proc[i].priority, proc[i].burstTime,
               proc[i].arrivalTime, proc[i].waitingTime, proc[i].turnaroundTime);
    }
}
```

---------------------------output-----------------



```
❯ ./prior
PID     Priority        BurstTime       ArrivalTime     WaitingTime     TurnaroundTime
2       1               1               0               0               1
4       5               1               1               0               1
3       4               2               2               0               2
1       3               10              3               6               16
5       2               5               4               0               5

Average Waiting Time = 1.20
Average Turnaround Time = 5.00

~/krato/University/sem6/fos/exp5 on main !12 ?1
❯
```

Practical Notes



Priority Scheduling

Pre-emp

Process	BT	Priority	AT	CT	TAT	WT
P_1	10	3	3	18	15	5
P_2	1	1	0	1	1	0
P_3	2	2	2	19	17	15
P_4	1	5	1	2	1	0
P_5	5	2	4	9	5	0
					Avg	Avg
					7.8	4

P_2	P_4	P_3	P_1	P_5	P_1	P_3

0 1 2 3 4 9 18 19

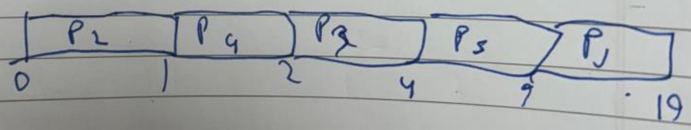
At 0 AT
$P_2 = (1) - 1$
At 1,
$P_4 - 1$
At 2,
$P_3 - (1)$
$P_1 < (10) - 3$
At 4,
$P_1 (9) = 3$ prior
P_3 - 4 prior
$P_5 (5)$ ~~=~~ 2 prior

At 9,
$P_1 (9) - 3$
$P_3 (1) - 4$
At 18,
$P_3 (1)$

Non-Preemptive

Process	AT	BT	Priority	CT	TAT	WT
P_1	3	10	3	19	16	6
P_2	0	1	1	1	1	0
P_3	2	2	4	4	2	0
P_4	1	1	5	2	1	0
P_5	4	5	2	9	5	0
			Avg		5	1.2

Gantt Chart.



P004

[illegible] 2/12/2024

AT = 0

Process	BT	Priority	CT	TAT	WT
P_1	10	3	16	16	6
P_2	1	1	1	1	0
P_3	2	4	18	18	16
P_4		5	19	19	18
P_5	5	2	6	6	1
				Avg =12	Avg =8.2

```
| P2 | P5 | P1 | P3 | P4 |
0    1    6    16   18   19
```

Non Preemptive

AT = 0

Process	BT	Priority	CT	TAT	WT
P_1	10	3	16	16	6
P_2	1	1	1	1	0
P_3	2	4	18	18	16
P_4	1	5	19	19	18
P_5	5	2	6	6	1
				Avg =12	Avg =8.2

```
| P2 | P5 | P1 | P3 | P4 |
0    1    6    16   18   19
```