Experiment No – 4b **Date -**

(b) PREEMPTIVE PRIORITY SCHEDULING

Aim - To implement Preemptive priority scheduling algorithm

THEORY –

Preemptive Priority Scheduling is a **CPU scheduling algorithm** in which each process is assigned a **priority value**. The **CPU is allocated** to the process with the **highest priority** (usually, smaller numerical value = higher priority).
If a new process arrives with a **higher priority** than the currently running one, the CPU **preempts (stops)** the current process and assigns the CPU to the new, higher-priority process.

How It Works

1. Each process has a **priority number**.
2. The scheduler always checks if a new process with **higher priority** arrives.
3. If so, the **running process is preempted** (paused and moved to the ready queue).
4. The higher-priority process executes first.
5. Once it completes or gets blocked, the CPU returns to the next highest-priority process.

Characteristics

- **Preemptive:** CPU can be taken away if a higher-priority process arrives.
- **Efficient for real-time systems**, where urgent tasks must be executed immediately.
- **Starvation (Indefinite Blocking):** Low-priority processes might **never get CPU time** if high-priority processes keep arriving.
- **Aging Technique:** Used to reduce starvation — gradually **increasing the priority** of waiting processes over time.

PRIORITY MECHANISM

Each process has a priority number.
Lower number = Higher priority (in most systems).
Example: Priority 1 is higher than Priority 3.
CPU is allocated to the waiting process with the highest priority.
If two processes have the same priority, then First-Come-First-Serve (FCFS) is used as a tie-breaker.

IMPORTANT TERMS

1. Arrival Time (AT)
The time at which a process arrives in the ready queue.
Example: If Process P1 has AT = 2, it means it comes to the system at time unit 2.

2. Burst Time (BT)
The total execution time a process needs to complete on the CPU.
Example: If BT = 5, the process requires 5 units of CPU time.

3. Completion Time (CT)
The time at which a process finishes execution.
Example: If a process starts at time 4 and runs for 5 units, its CT = 9.

4. Turnaround Time (TAT)
The total time a process spends in the system (from arrival to completion).
Formula:
TAT = CT - AT

5. Waiting Time (WT)
The time a process spends waiting in the ready queue before getting CPU.
Formula:
WT = TAT - BT

6. Average Turnaround Time (Avg TAT)
Average of turnaround times of all processes.

7. Average Waiting Time (Avg WT)
Average of waiting times of all processes.

Example -

Process Name	Arrival Time	Burst Time	Priority
A	2	2	3
B	0	8	4
C	3	16	1
D	7	1	2

Gantt Chart –

B	A	C	D	A	B

0 2 3 19 20 21 27

Table -

Process Name	Arrival Time	Burst Time	Priority	Completion Time	Turn Around Time (TAT)	Wait Time
A	2	2	3	21	19	17
B	0	8	4	27	27	19
C	3	16	1	19	16	0
D	7	1	2	20	13	12

AVERAGE TURN AROUND TIME IS 18.75 AND AVERAGE WAIT TIME IS 12

PROGRAM

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
using namespace std;

struct Process {
  string name;
  int at, bt, ct, tat, wt, pr;
  int remainingTime;
  bool done;
};

int main() {
  int n;
  cout << "Enter number
of processes: ";
  cin >> n;

  vector<Process> p(n);
  for (int i = 0; i < n; i++) {
    cout << "Enter
Process Name, Arrival
Time, Burst Time,
Priority for P" << i + 1 <<
": ";
    cin >> p[i].name >>
p[i].at >> p[i].bt >>
p[i].pr;
    p[i].remainingTime =
p[i].bt;
    p[i].done = false;
  }

  int time = 0,
completed = 0;
  vector<string>
ganttProcess;
  vector<int> ganttTime;
  ganttTime.push_back(
0);

  string currentProcess
= "";

  while (completed != n)
{
    int idx = -1;
    int highestPriority =
1e9;

    for (int i = 0; i < n;
i++) {
      if (p[i].at <= time
&& !p[i].done) {
        if (p[i].pr <
highestPriority) {
          highestPriority
= p[i].pr;
          idx = i;
        } else if (p[i].pr
== highestPriority) {
          if (p[i].at <
p[idx].at)
            idx = i;
        }
      }
    }

    if (idx != -1) {
      if (currentProcess
!= p[idx].name) {
        currentProcess =
p[idx].name;
        ganttProcess.pu
sh_back(currentProcess
);
        ganttTime.push_
back(time);
      }

      p[idx].remainingTi
me--;
      time++;

      if
(p[idx].remainingTime
== 0) {
        p[idx].ct = time;
        p[idx].tat =
p[idx].ct - p[idx].at;
        p[idx].wt =
p[idx].tat - p[idx].bt;
        p[idx].done =
true;
        completed++;
        currentProcess =
"";
      }
    } else {
      time++;
      if (currentProcess
!= "IDLE") {
        currentProcess =
"IDLE";
        ganttProcess.pu
sh_back(currentProcess
);
        ganttTime.push_
back(time - 1);
      }
    }
  }

  ganttTime.push_back(
time);

  int totalTAT = 0,
totalWT = 0;
  for (int i = 0; i < n; i++) {
    totalTAT += p[i].tat;
    totalWT += p[i].wt;
  }

  cout << "-----------------
-------------------------------
---------------\n";
  cout <<
"Process\tAT\tBT\tPR\tC
T\tTAT\tWT\n";
```

```
    cout << "-----------------------------------------------------------------\n";
    for (int i = 0; i < n; i++) {
        cout << p[i].name << "\t" << p[i].at << "\t" << p[i].bt << "\t"
             << p[i].pr << "\t" << p[i].ct << "\t" << p[i].tat << "\t" << p[i].wt << "\n";
    }
    cout << "-----------------------------------------------------------------\n";

    cout << "Total Turnaround Time = " << totalTAT << "\n";
    cout << "Average Turnaround Time = " << (float)totalTAT / n << "\n";
    cout << "Average Waiting Time = " << (float)totalWT / n << "\n\n";

    cout << "Gantt Chart:\n";
    cout << "-----------------------------------------------------------------\n|";
    for (size_t i = 0; i < ganttProcess.size(); i++) {
        cout << "   " << setw(2) << ganttProcess[i] << " |";
    }
    cout << "\n-----------------------------------------------------------------\n";
    for (size_t i = 0; i < ganttTime.size(); i++) {
        cout << setw(7) << ganttTime[i];
    }
    cout << "\n";

    return 0;
}
```

OUTPUT –

```
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter number of processes: 4
Enter Process Name, Arrival Time, Burst Time, Priority for P1: A
2
2
3
Enter Process Name, Arrival Time, Burst Time, Priority for P2: B
0
8
4
Enter Process Name, Arrival Time, Burst Time, Priority for P3: C
3
16
1
Enter Process Name, Arrival Time, Burst Time, Priority for P4: D
7
1
2
---------------------------------------------------------------
Process AT      BT      PR      CT      TAT     WT
---------------------------------------------------------------
A       2       2       3       21      19      17
B       0       8       4       27      27      19
C       3       16      1       19      16      0
D       7       1       2       20      13      12
---------------------------------------------------------------
Total Turnaround Time = 75
Average Turnaround Time = 18.75
Average Waiting Time = 12

Gantt Chart:
---------------------------------------------------------------
|     B |     A |     C |     D |     A |     B |
---------------------------------------------------------------
      0      0      2      3     19     20     21     27
```

CONCLUSION - Preemptive Priority Scheduling is highly effective for systems that require strict adherence to priority levels and fast response times for critical tasks.