Experiment No. 2

Aim: Write a program to implement the "shortest Job first" (SJF) algorithm / CPU scheduling algorithm in C++ program code.

Experiment No. 2

Aim: Write a program to implement the "Shortest Job first" (SJF) algorithm / CPU scheduling algorithm in C/C++ program code.

Theory: SJF (Shortest Job First) CPU Scheduling Algorithm:
Shortest Job first (SJF) is a CPU scheduling algo. that selects the process with the smallest CPU burst time to execute next.

i) Basic concept : SJF is a non-preemptive scheduling algo. that selects the waiting process with the smallest execution time. The algorithm works on the algorithmic principle that shorter jobs should be executed before longer once to minimize average waiting time.

ii) Types of SJF :
Non-preemptive SJF : Once a process begins execution, it continues until completion.

Preemptive SJF : Also called as Shortest Remaining time first (SRTF) : If a new process arrives with a smaller burst time than the currently executing process, the CPU switches to the new processes.

iii) Advantages : Provides minimum avg. waiting time among all scheduling algorithm.

Reduces overhead as fewer context switches are needed.
Maximizes throughput by completing smaller job quickly.

ΦΦ> Disadvantages : Can lead to starvation of longer processes if shorter processes keep arriving
Requires prediction of CPU burst time, which is difficult to estimate accurately
Not practical for interactive systems.

ΦΦ> Implementation : Non-preemtive SJF :
Identify all processes in the ready state
Determine the CPU burst time for each process
Select the process with the smallest burst time.
Executed the selected process until completion.
Repeat above steps until all processes complete.

ΦΦ> Gantt chart construction :
X-axis represents time (cycles msec).
Each process execution is shown as the block on the timeline.
Process ID (PID) is typically written inside the block

C++ code:

```
#include<iostream>
using namespace std;
int main () {
        int A [100][4];
        int i, j, n, total = 0, index, temp;
        float avg_wt, avg_tat;

        cout <<"Enter number of process:";
        cin >> n;
        cout << "Enter burst time:" << endl;

        for (i=0; i<n; i++) {
            cout <<"p" << i+1 <<":";
            cin >> A[i][1];
            A[i][0] = i+1;
        }

        for (i=0; i<n; i++){
            index = i;
                for (j=1; j<n; j++){
                    if (A[j][1] < A[index][1])
                        index = j;
                temp = A[i][1];
                A[i][1] = A[index][1];
                A[index][1] = temp;
```

```
        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
    }
A[0][2] = 0;
for(i=1; i<n; i++){
        A[i][2] = 0;
        for(j=0; j<i; j++)
            A[i][2] +=A[j][1];
        total += A[i][2];
}

avg_wt = (float) total /n;
total = 0;
cout << " P    BT    WT    TAT" << endl;

for(i=0; i<n; i++){
    A[i][3] = A[i][1] + A[i][2];
    total += A[i][3];
    cout << "P" << A[i][0] << "    " << A[i][1] <<
        "    " << A[i][2] << "    " << A[i][3] <<
        << endl;
avg_tat =(float)total /n;
cout << "Average waiting time =" << avg_wt << endl;
cout <<"Average Turnaround time =" << avg_tat << endl;
}
```

Output:

Process	Arrival time	Burst Time
P1	0	6
P2	1	4
P3	2	2
P4	3	3

Gantt Chart:

P1	P3	P4	P2

0 6 8 11 15

Process	CT	WT	TAT
P1	6	0	6
P2	15	9	14
P3	8	[illegible]	6
P4	11	5	8
Avg.		4	8.5

Conclusion: Thus the understanding & execution of SJF algorithm has studied successfully.

Conclusion: Thus the understanding & execution of SJF algorithm has studied successfully.