Practical No. 9

NAME :_____ STD.:_____ DIV.:_____

Page :
Date :

Name :- Pranav Sahasrabuddhe

Roll NO :- 2171944

Branch :- IT

Batch :- 2nd

Date :-

Sign :- 13/11/19

Aim :: Implementation of shortest job first algorithm

shortest job first Algorithm:

- shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next SJN is a non-preemptive algorithm

- It is a Greedy algorithm.

- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.

- It is practically infeasible as operating system may not know burst time & therefore may not sort them while it is not possible to predict execution time.

- sort all the process according to the arrival time.

- Then select the that process which has minimum arrival time & minimum burst time.

- After completion of process make a pool of process which after till the completion of previous process and select that process among the.

Algorithm:

Process	Duration:	order	Arrival time
P_1	6	1	0
P_2	8	2	0
P_3	7	3	0
P_4	3	4	0

Process	waiting time:
P_1	0
P_2	3
P_3	9
P_4	10

Total time = 24

Average waiting time:
(0+3+9+16)/4
= 7

Condusion :-

Thus we study & implementation of shortest job first algorithm.

Program:

```
#include<iostream>
using namespace std;
int mat[10][6];

void swap(int *a, int *b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
	for(int i=0; i<num; i++)
	{
		for(int j=0; j<num-i-1; j++)
		{
			if(mat[j][1] > mat[j+1][1])
			{
				for(int k=0; k<5; k++)
				{
					swap(mat[j][k], mat[j+1][k]);
				}
			}
		}
	}
}

void completionTime(int num, int mat[][6])
{
	int temp, val;
	mat[0][3] = mat[0][1] + mat[0][2];
	mat[0][5] = mat[0][3] - mat[0][1];
	mat[0][4] = mat[0][5] - mat[0][2];

	for(int i=1; i<num; i++)
	{
		temp = mat[i-1][3];
		int low = mat[i][2];
		for(int j=i; j<num; j++)
		{
			if(temp >= mat[j][1] && low >= mat[j][2])
			{
				low = mat[j][2];
				val = j;
```

```
                }
            }
            mat[val][3] = temp + mat[val][2];
            mat[val][5] = mat[val][3] - mat[val][1];
            mat[val][4] = mat[val][5] - mat[val][2];
            for(int k=0; k<6; k++)
            {
                swap(mat[val][k], mat[i][k]);
            } }}
int main()
{
    int num, temp;

    cout<<"Enter number of Process: ";
    cin>>num;

    cout<<"...Enter the process ID...\n";
    for(int i=0; i<num; i++)
    {
        cout<<"...Process "<<i+1<<"...\n";
        cout<<"Enter Process Id: ";
        cin>>mat[i][0];
        cout<<"Enter Arrival Time: ";
        cin>>mat[i][1];
        cout<<"Enter Burst Time: ";
        cin>>mat[i][2];
    }
    cout<<"Before Arrange...\n";
    cout<<"Process ID\tArrival Time\tBurst Time\n";
    for(int i=0; i<num; i++)
    {
    cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";
    }
    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout<<"Final Result...\n";
    cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for(int i=0; i<num; i++)
    {

    cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[i][4]<<"\t\t"<<mat[i][5]<<"
\n";
    }
}
```

Output:

/*Enter number of Process: 2
...Enter the process ID...
...Process 1...
Enter Process Id: 15
Enter Arrival Time: 14
Enter Burst Time: 12
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 14
Enter Burst Time: 16
Before Arrange...

Process ID	Arrival Time	Burst Time
15	14	12
2	14	16

Final Result...

Process ID	Arrival Time	Burst Time	Waiting Time	Turnaround Time
15	14	12	0	12
2	14	16	12	28

Process exited after 14.68 seconds with return value 0
Press any key to continue . . .*/