**OS-PT-1**

**HARSH MITTAL**

**2K19CSUN01082**

**CSE4B**

The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking.

In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

#include<stdio.h>

using namespace std;

void findWaitingTime(int processes[], int n,

int bt[], int wt[], int quantum)

{

int rem\_bt[n];

for (int i = 0 ; i < n ; i++)

rem\_bt[i] = bt[i];

int t = 0;

while (1)

{

bool done = true;

for (int i = 0 ; i < n; i++)

{

if (rem\_bt[i] > 0)

{

done = false;

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

else

{

t = t + rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done == true)

break;

}

}

void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[])

{

for (int i = 0; i < n ; i++)

tat[i] = bt[i] + wt[i];

}

void findavgTime(int processes[], int n, int bt[],

int quantum)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

for (int i=0; i<n; i++)

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

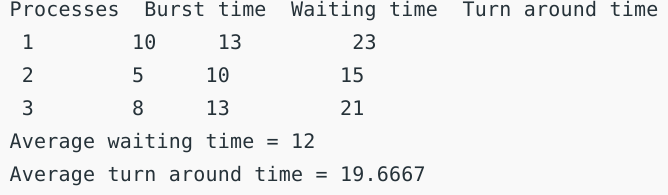
int burst\_time[] = {10, 5, 8};

int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

return 0;

}



Write and Implement the program for the same and include the snapshot of your implementation. 2. Write and implement the program for Pre-emptive and non pre-emptive SJF scheduling algorithm same and include the snapshot of your implementation.

**NON-PREEMPTIVE**

#include <stdio.h>

int main()

{

int i, n, p[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, min, k = 1, btime = 0;

int bt[10], temp, j, at[10], wt[10], tt[10], ta = 0, sum = 0;

float wavg = 0, tavg = 0, tsum = 0, wsum = 0;

printf(" -------Shortest Job First Scheduling ( NP )-------\n");

printf("\nEnter the No. of processes :");

scanf("%d", &n);

for (i = 0; i < n; i++)

{

printf("\tEnter the burst time of %d process :", i + 1);

scanf(" %d", &bt[i]);

printf("\tEnter the arrival time of %d process :", i + 1);

scanf(" %d", &at[i]);

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if (at[i] < at[j])

{

temp = p[j];

p[j] = p[i];

p[i] = temp;

temp = at[j];

at[j] = at[i];

at[i] = temp;

temp = bt[j];

bt[j] = bt[i];

bt[i] = temp;

}

}

}

for (j = 0; j < n; j++)

{

btime = btime + bt[j];

min = bt[k];

for (i = k; i < n; i++)

{

if (btime >= at[i] && bt[i] < min)

{

temp = p[k];

p[k] = p[i];

p[i] = temp;

temp = at[k];

at[k] = at[i];

at[i] = temp;

temp = bt[k];

bt[k] = bt[i];

bt[i] = temp;

}

}

k++;

}

wt[0] = 0;

for (i = 1; i < n; i++)

{

sum = sum + bt[i - 1];

wt[i] = sum - at[i];

wsum = wsum + wt[i];

}

wavg = (wsum / n);

for (i = 0; i < n; i++)

{

ta = ta + bt[i];

tt[i] = ta - at[i];

tsum = tsum + tt[i];

}

tavg = (tsum / n);

printf("\*\*\*\*\*\*\*\*");

printf("\n RESULT:-");

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around\n");

for (i = 0; i < n; i++)

{

printf("p%d\t\t %d\t\t %d\t\t %d\t\t %d\n", p[i], bt[i], at[i], wt[i], tt[i]);

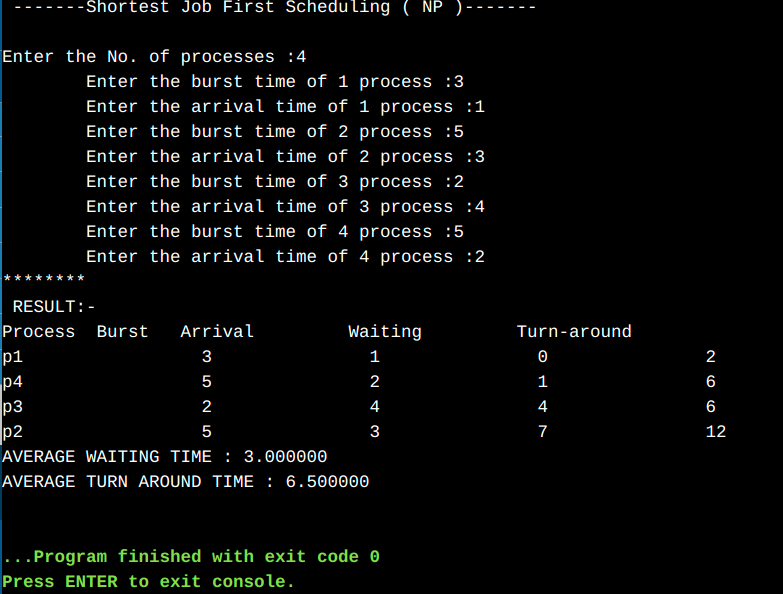
}

printf("AVERAGE WAITING TIME : %f\n", wavg);

printf("AVERAGE TURN AROUND TIME : %f\n", tavg);

return 0;

}



**Pre-emptive**

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processesn", limit);

for(i = 0; i < limit; i++)

{

printf("\nEnter Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Enter Burst Time:\t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

for(i = 0; i < limit; i++)

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

printf("\n\nAverage Waiting Time:\t%lf\n", average\_waiting\_time);

printf("Average Turnaround Time:\t%lf\n", average\_turnaround\_time);

return 0;

}

