**Round Robin:**

#include <stdio.h>

int main()

{

// Initialize the variables

int i, NOP, sum = 0, y, quant, wt = 0, tat = 0;

int at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf("Total number of processes in the system: ");

scanf("%d", &NOP);

y = NOP; // Assign the number of processes to variable y

// Use for loop to enter the details of the process like Arrival time and Burst Time

for (i = 0; i < NOP; i++) {

printf("\nEnter the Arrival and Burst time of the Process[%d]\n", i + 1);

printf("Arrival time: ");

scanf("%d", &at[i]); // Accept arrival time

printf("Burst time: ");

scanf("%d", &bt[i]); // Accept the Burst time

temp[i] = bt[i]; // Store the burst time in temp array

}

// Accept the Time Quantum

printf("Enter the Time Quantum for the process: ");

scanf("%d", &quant);

// Display the process No, Burst Time, Turn Around Time and Waiting Time

printf("\nProcess No\tBurst Time\tTAT\tWaiting Time");

for (sum = 0, i = 0; y != 0; ) {

// Check if the process has arrived before processing

if (temp[i] > 0 && at[i] <= sum) {

if (temp[i] <= quant) { // Define the conditions

sum += temp[i];

temp[i] = 0;

y--; // Decrement the process count

printf("\nProcess No[%d]\t\t%d\t\t%d\t\t%d",

i + 1, bt[i], sum - at[i], sum - at[i] - bt[i]);

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

tat += sum - at[i];

} else {

temp[i] -= quant;

sum += quant;

}

}

// Move to the next process

i = (i + 1) % NOP; // Wrap around to the first process

}

// Represents the average waiting time and Turn Around time

avg\_wt = (float)wt / NOP;

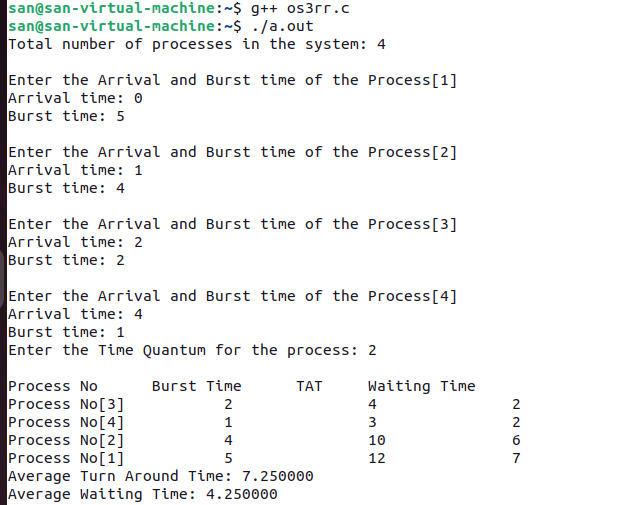
avg\_tat = (float)tat / NOP;

printf("\nAverage Turn Around Time: %f", avg\_tat);

printf("\nAverage Waiting Time: %f\n", avg\_wt);

return 0;

}



**Shortest Job First(non-preemptive):**

#include <stdio.h>

int main()

{

// Matrix for storing Process Id, Burst Time, Average Waiting Time & Average Around Time.

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of process: ");

scanf("%d", &n);

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

// User Input Burst Time and alloting Process Id.

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

printf("P%d: ", i + 1);

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

// Sorting process according to their Burst Time.

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

index = i;

for (j = i + 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;

// Calculation of Waiting Times

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;

printf("P BT WT TAT\n");

// Calculation of Turn Around Time and printing the data.

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],

A[i][1], A[i][2], A[i][3]);

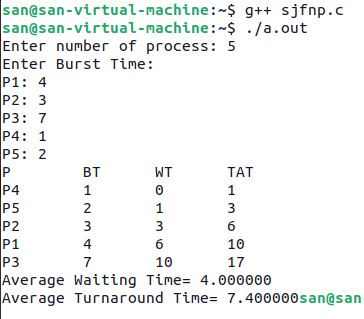
}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt);

printf("\nAverage Turnaround Time= %f", avg\_tat);

}



**Shortest Remaining Time First(preemptive):**

#include <stdio.h>

#include <limits.h>

struct Process

{

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

// Function to find the waiting time for all processes

void findWaitingTime(struct Process proc[], int n, int wt[])

{

int rt[n];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

int check = 0; // changed boolean to integer

// Process until all processes gets completed

while (complete != n)

{

// Find process with minimum remaining time among the processes that arrives till the current time

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

{

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0)

{

minm = rt[j];

shortest = j;

check = 1; // changed boolean to integer

}

}

if (check == 0)

{

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely executed

if (rt[shortest] == 0)

{

// Increment complete

complete++;

check = 0; // changed boolean to integer

// Find finish time of current process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Function to calculate turn around time

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {

// calculating turnaround time by adding

// bt[i] + wt[i]

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

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

}

// Function to calculate average time

void findavgTime(struct Process proc[], int n) {

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

total\_tat = 0;

// Function to find waiting time of all processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all details

printf(" P\t\t" "BT\t\t" "WT\t\t" "TAT\t\t\n");

// Calculate total waiting time and total turnaround time

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

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

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

printf(" %d\t\t"

"%d\t\t %d"

"\t\t %d\n", proc[i].pid,

proc[i].bt, wt[i], tat[i]);

}

printf("\nAverage waiting time = "

"%f", (float)total\_wt / (float)n);

printf("\nAverage turn around time = "

"%f", (float)total\_tat / (float)n);

}

// Driver code

int main() {

struct Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 }, { 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

