****

FCFS

Code:

```
#include <iostream>
using namespace std;

struct ProcessNode {
int pid;
int burstTime;
int arrivalTime;
int completionTime;
int waitTime;
int turnAroundTime;
ProcessNode* next;
};

class ProcessQueue {
private:
ProcessNode* front;
ProcessNode* rear;

public:
ProcessQueue() {
front = nullptr;
rear = nullptr;
}

~ProcessQueue() {
ProcessNode* temp = front;
while (temp != nullptr) {
front = front->next;
delete temp;
temp = front;
}
}
```

```
bool isEmpty() {
return front == nullptr;
}

void enqueue(int pid, int burstTime, int arrivalTime) {
ProcessNode* newNode = new ProcessNode;
newNode->pid = pid;
newNode->burstTime = burstTime;
newNode->arrivalTime = arrivalTime;
newNode->next = nullptr;

if (rear == nullptr) {
front = newNode;
rear = newNode;

}
else {
rear->next = newNode;
rear = newNode;
}
}

void dequeue() {
if (isEmpty()) {
cout << "Queue is empty.\n";
}
else {
ProcessNode* temp = front;
front = front->next;
if (front == nullptr) {
rear = nullptr;
}
delete temp;
}
}
```

```
void computeMetrics() {
if (isEmpty()) {
cout << "Queue is empty.\n";
return;
}

int currentTime = front->arrivalTime;
ProcessNode* temp = front;

while (temp != nullptr) {
temp->waitTime = currentTime - temp->arrivalTime;
temp->completionTime = currentTime +

temp->burstTime;

temp->turnAroundTime = temp->completionTime -

temp->arrivalTime;

currentTime = temp->completionTime;
temp = temp->next;
}
}

void printMetrics() {
if (isEmpty()) {
cout << "Queue is empty.\n";
return;
}

cout << "PID\tBurst Time\tArrival Time\tCompletion
Time\tWait Time\tTurnaround Time\n";
ProcessNode* temp = front;
while (temp != nullptr) {
cout << temp->pid << "\t" << temp->burstTime << "\t\t" <<

temp->arrivalTime << "\t\t"
```

```
<< temp->completionTime << "\t\t" << temp->waitTime

<< "\t\t" << temp->turnAroundTime << endl;

temp = temp->next;
}
}
};

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

ProcessQueue pq;
int pid, burstTime, arrivalTime;
for (int i = 0; i < numProcesses; i++) {
cout << "Enter the details of process " << i + 1 << ":" << endl;
cout << "PID: ";
cin >> pid;
cout << "Burst Time: ";
cin >> burstTime;
cout << "Arrival Time: ";
cin >> arrivalTime;
pq.enqueue(pid, burstTime, arrivalTime);
}

pq.computeMetrics();
pq.printMetrics();

return 0;
}

**********
```

****

SJF

```
#include<iostream>
using namespace std;
struct process {
int id;
int bt;
int waiting_time;
int turnaround_time;
int completion_time;
};

bool compare(process p1, process p2) {
return p1.bt < p2.bt;
}

void sjf(process processes[], int n) {
for (int i = 0; i < n - 1; i++) {
int min= i;
for (int j = i + 1; j < n; j++) {
if (processes[j].bt < processes[min].bt) {
min = j;
}
}
swap(processes[min], processes[i]);
}

int total_time = 0;
cout << "Process\tBurst Time\tWaiting Time\tTurnaround Time\tCompletion
Time\n";
for(int i = 0; i < n; i++) {
cout << processes[i].id << "\t" << processes[i].bt<< "\t\t";
int waiting_time = total_time;
```

```
cout << waiting_time << "\t\t";
int turnaround_time = waiting_time + processes[i].bt;
cout << turnaround_time << "\t\t";
int completion_time = total_time + processes[i].bt;
cout << completion_time << endl;
total_time += processes[i].bt;
}
}


int main() {
int n=0;
cout << "Enter the number of processes: ";
cin >> n;
process processes[n];
for(int i = 0; i < n; i++) {
cout << "Enter the burst time for process " << i+1 << ": ";
cin >> processes[i].bt;
processes[i].id = i+1;
processes[i].waiting_time = 0;
processes[i].turnaround_time = 0;
processes[i].completion_time = 0;
}
sjf(processes, n);
return 0;
}
**********************************************************************
```

`

****

ROUND ROBIN

```
#include<iostream>
using namespace std;
void FindAVGtime(int processes[], int n, int bt[], int quantum);
void FindTAT(int processes[], int n, int bt[], int wt[], int tat[]);

int main()
{
int n,quantum;
int processes[n],burst_time[n];
cout<<"Enter number of processes :";
cin>>n;
for(int i=0;i<n;i++){
cout<<"Enter process id :";
cin>>processes[i];
cout<<"Enter burst time :";
cin>>burst_time[i];
}
cout<<"Enter Quantum :";
cin>>quantum;
FindAVGtime(processes, n, burst_time, quantum);
return 0;
}

void FindWaitingTime(int processes[], int n, int BurstTime[], int
WaitTime[], int quantum){
int Rem_BurstTime[n];
for (int i = 0 ; i < n ; i++)
Rem_BurstTime[i] = BurstTime[i];
int t = 0;
while(1)
{
bool done = true;
for (int i = 0 ; i < n; i++)
{
if (Rem_BurstTime[i] > 0)
{
done = false;
```

```
if (Rem_BurstTime[i] > quantum)
{
t += quantum;
Rem_BurstTime[i] -= quantum;
}
else
{
t = t + Rem_BurstTime[i];
WaitTime[i] = t - BurstTime[i];
Rem_BurstTime[i] = 0;
}
}
}
if(done==true)
break;
}
}
void FindTAT(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);
FindTAT(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;
}
**********************************************************************
```

****

MULTILEVEL

```
#include <iostream>
#include<stdio.h>
int main()
{
	int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;
	float wtavg, tatavg;
	printf("Enter the number of processes:");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		p[i] = i;
		printf("Enter the Burst Time of Process%d:", i);
		scanf("%d",&bt[i]);
		printf("System/User Process (0/1) ? ");
		scanf("%d", &su[i]);
	}
	for(i=0;i<n;i++)
		for(k=i+1;k<n;k++)
			if(su[i] > su[k])
			{
			temp=p[i];
			p[i]=p[k];
			p[k]=temp;
			temp=bt[i];
			bt[i]=bt[k];
			bt[k]=temp;
			temp=su[i];
			su[i]=su[k];
			su[k]=temp;
			}
```

```
        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];
                wtavg = wtavg + wt[i];
                tatavg = tatavg + tat[i];
        }
        printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST
TIME\tWAITING TIME\tTURNAROUND TIME");
        for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d
",p[i],su[i],bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        return 0;
}


}
```

MULTILEVEL CPP CODE

```
#include <iostream>
using namespace std;
struct Process {
int pid;
int type;
int burst_time;
int priority;
int arrival_time;
int completion_time;
```

```
int turnaround_time;
int waiting_time;
int remaining_time;

};
void printGanttChart(Process *processes, int n) {
int total_time = 0;
for (int i = 0; i < n; i++) {
total_time += processes[i].burst_time;
}
cout << " ";
for (int i = 0; i < total_time; i++) {
cout << "-";
}
cout << endl;
int current_time = 0;
while (current_time < total_time) {
int selected_process = -1;
for (int i = 0; i < n; i++) {
if (processes[i].arrival_time <= current_time &&

processes[i].remaining_time > 0) {

if (selected_process == -1) {
selected_process = i;
} else if (processes[i].type <

processes[selected_process].type) {
selected_process = i;
} else if (processes[i].type ==

processes[selected_process].type) {

if (processes[i].type == 5 && processes[i].priority <

processes[selected_process].priority) {
selected_process = i;
```

```
} else if (processes[i].type != 5 &&

processes[i].remaining_time < processes[selected_process].remaining_time) {

selected_process = i;
}
}
}
}
if (selected_process == -1) {
cout << "|";
current_time++;
continue;
}
cout << "|";
if (processes[selected_process].remaining_time > 2 &&

processes[selected_process].type == 2) {

cout << "RR";
current_time += 2;
processes[selected_process].remaining_time -= 2;
} else if (processes[selected_process].remaining_time > 4 &&
processes[selected_process].type == 3) {

cout << "RR";
current_time += 4;
processes[selected_process].remaining_time -= 4;
} else {
cout << "P" << processes[selected_process].pid;
current_time += processes[selected_process].remaining_time;
processes[selected_process].remaining_time = 0;
processes[selected_process].completion_time = current_time;
processes[selected_process].turnaround_time =

processes[selected_process].completion_time -
processes[selected_process].arrival_time;
```

```
processes[selected_process].waiting_time =

processes[selected_process].turnaround_time -
processes[selected_process].burst_time;
}
}
cout << "|" << endl;
cout << " ";
for (int i = 0; i < total_time; i++) {
cout << "-";
}
cout << endl;
cout << " ";
for (int i = 0; i < n; i++) {
printf("P%d ", processes[i].pid);
}
cout << endl;
}
int main() {
int n;
cout << "Enter the number of processes: ";
cin >> n;
Process *processes = new Process[n];
for (int i = 0; i < n; i++) {
processes[i].pid = i+1;

cout << "Enter the type of process for P" << i+1 << " (1 for system, 2 for
interactive, 3 for interactive editing, 4 for batch, 5 for student): ";
cin >> processes[i].type;
cout << "Enter the burst time for P" << i+1 << ": ";
cin >> processes[i].burst_time;
if (processes[i].type == 5) {
cout << "Enter the priority for P" << i+1 << ": ";
cin >> processes[i].priority;
} else {
processes[i].priority = 0;
```

```
}
processes[i].arrival_time = 0;
processes[i].remaining_time = processes[i].burst_time;
}
for (int i = 0; i < n-1; i++) {
for (int j = 0; j < n-i-1; j++) {
if (processes[j].arrival_time > processes[j+1].arrival_time) {
Process temp = processes[j];
processes[j] = processes[j+1];
processes[j+1] = temp;
}
}
}
printGanttChart(processes, n);
float avg_turnaround_time = 0;
float avg_waiting_time = 0;
for (int i = 0; i < n; i++) {
avg_turnaround_time += processes[i].turnaround_time;
avg_waiting_time += processes[i].waiting_time;
}
avg_turnaround_time /= n;
avg_waiting_time /= n;
cout << "PID\tType\tBurst Time\tPriority\tArrival Time\tCompletion
Time\tTurnaround Time\tWaiting Time" << endl;
for (int i = 0; i < n; i++) {
printf("%d\t%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t\t%d\n",
processes[i].pid, processes[i].type, processes[i].burst_time,
processes[i].priority, processes[i].arrival_time,
processes[i].completion_time, processes[i].turnaround_time,

processes[i].waiting_time);
}
cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
cout << "Average Waiting Time = " << avg_waiting_time << endl;
delete[] processes;
return 0;
}
```

****

****

SRTF

```
using namespace std;
int main()
{
    int a[10],b[10],x[10];
    int waiting[10],turnaround[10],completion[10];
    int i,j,smallest,count=0,time,n;
    double avg=0,tt=0,end;

    cout<<"\nEnter the number of Processes: ";  //input
    cin>>n;
    for(i=0; i<n; i++)
    {
        cout<<"\nEnter arrival time of process: ";  //input
        cin>>a[i];
    }
    for(i=0; i<n; i++)
    {
```

```
        cout<<"\nEnter burst time of process: ";  //input
        cin>>b[i];
    }
    for(i=0; i<n; i++)
        x[i]=b[i];

    b[9]=9999;
    for(time=0; count!=n; time++)
    {
        smallest=9;
        for(i=0; i<n; i++)
        {
            if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
                smallest=i;
        }
        b[smallest]--;

        if(b[smallest]==0)
        {
            count++;
            end=time+1;
            completion[smallest] = end;
            waiting[smallest] = end - a[smallest] - x[smallest];
            turnaround[smallest] = end - a[smallest];
        }
    }
    cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time"
<<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<<
"\t"<<"completion-time"<<endl;
    for(i=0; i<n; i++)
    {

cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<waiting[i]<<"\t\t"<<turn
around[i]<<"\t\t"<<completion[i]<<endl;
        avg = avg + waiting[i];
        tt = tt + turnaround[i];
    }
    cout<<"\n\nAverage waiting time ="<<avg/n;
    cout<<"  Average Turnaround time ="<<tt/n<<endl;
}
```

****

PREEMPTIVE PRIORITY

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
void preemptive_priority_scheduling(int num_processes, int *arrival_time,
int *burst_time){
int waiting_time[num_processes], turnaround_time[num_processes],
remaining_time[num_processes];
for(int i=0; i<num_processes; i++){
waiting_time[i] = 0;
turnaround_time[i] = 0;
remaining_time[i] = burst_time[i];
}
int priority[num_processes];
srand(time(0));
for(int i=0; i<num_processes; i++){
priority[i] = rand() % 10 + 1;
}
int current_time = 0, num_completed = 0;
while(num_completed < num_processes){
int highest_priority = 11, selected_process = -1;
for(int i=0; i<num_processes; i++){

if(arrival_time[i] <= current_time && remaining_time[i] > 0 && priority[i] <
highest_priority){

highest_priority = priority[i];
selected_process = i;
}
}
if(selected_process != -1){
waiting_time[selected_process] += current_time -
arrival_time[selected_process];
remaining_time[selected_process]--;
if(remaining_time[selected_process] == 0){
```

```
num_completed++;
turnaround_time[selected_process] = current_time + 1 -

arrival_time[selected_process];
}
}
current_time++;
}
cout << "Process\t\tPriority\tBurst Time\tArrival Time\tWaiting
Time\tTurnaround Time\n";
int total_waiting_time = 0;
for(int i=0; i<num_processes; i++){
cout << "P" << i+1 << "\t\t" << priority[i] << "\t\t" << burst_time[i] << "\t\t" <<
arrival_time[i]
<< "\t\t" << waiting_time[i] << "\t\t" << turnaround_time[i] << endl;
total_waiting_time += waiting_time[i];
}
cout << "Average waiting time = " << (float)total_waiting_time /
num_processes << endl;
}
int main(){
int num_processes;
cout << "Enter the number of processes: ";
cin >> num_processes;
int arrival_time[num_processes], burst_time[num_processes];
for(int i=0; i<num_processes; i++){
cout << "Enter the arrival time and burst time for process " << i+1 << ": ";
cin >> arrival_time[i] >> burst_time[i];
}
preemptive_priority_scheduling(num_processes, arrival_time, burst_time);

return 0;
}
**********
```

PRIORITY

```
#include <iostream>
using namespace std;
struct process
{
int id, burst_time, wait_time, comp_time, tat_time, priority;
process *next;

};
bool compare(process p1, process p2)
{
return p1.priority < p2.priority;
}
void display(process p[], int n)
{
cout << "PID\tBT\tCT\tTAT\tWT" << endl;
for (int i = 0; i < n; i++)
{
cout << p[i].id << "\t" << p[i].burst_time << "\t"
<< p[i].comp_time << "\t" << p[i].tat_time << "\t" << p[i].wait_time << endl;
}
}
void priority(process pro[], int n)
{
for (int i = 0; i < n; i++)
{
int min = i;
for (int j = i + 1; j < n; j++)
{
if (compare(pro[j], pro[min]))
{
min = j;
}
}
```

```
swap(pro[min], pro[i]);
}
int total = 0;
for (int i = 0; i < n; i++)
{
pro[i].wait_time = total;
pro[i].tat_time = pro[i].wait_time + pro[i].burst_time;
pro[i].comp_time = total+pro[i].burst_time;
total = total + pro[i].burst_time;
}
display(pro, n);
}
int main()

{
int n;
cout << "Enter number of process = " << endl;
cin >> n;
process pro[n];
for (int i = 0; i < n; i++)
{
pro[i].id = i + 1;
cout << "Enter burst time of process " << i + 1 << endl;
cin >> pro[i].burst_time;
cout << "Enter the priority of the process " << endl;
cin >> pro[i].priority;
pro[i].comp_time = 0;
pro[i].tat_time = 0;
pro[i].wait_time = 0;
}
priority(pro, n);
return 0;
}
```