**Assignment 3**

**Name : Yadnesh Patil**

**Div : C Roll No : 23**

**Problems to be solved in the lab:** **CPU Scheduling Algorithms**

1. **First Come First Served (FCFS)**

**Code:**

*#include* <bits/stdc++.h>

using namespace std;

class Process

{

private:

    int at;

    int bt;

    int ct;

    int tat;

    int wt;

    int pid;

public:

    int &operator[](string var)

    {

*if* (var == "at")

*return* at;

*if* (var == "bt")

*return* bt;

*if* (var == "ct")

*return* ct;

*if* (var == "tat")

*return* tat;

*if* (var == "wt")

*return* wt;

*return* pid;

    }

    void update\_after\_ct()

    {

        tat = ct - at;

        wt = tat - bt;

    }

    void display()

    {

        printf("%d\t%d\t%d\t%d\t%d\t%d\n", pid, at, bt, ct,

               tat, wt);

    }

};

float average(vector<Process> P, string var)

{

    int total = 0;

*for* (auto temp : P)

    {

        total += temp[var];

    }

*return* (float)total / P.size();

}

int main()

{

    int n;

    cout<<"enter no of process: ";

    cin >> n;

    cout<<"enter arrival and burst time for each process: ";

    int counter = 0;

    vector<Process> P(n);

*for* (Process &temp : P)

    {

        temp["id"] = counter++;

        cin >> temp["at"] >> temp["bt"];

    }

    sort(P.begin(), P.end(),

         [](Process first, Process second)

         {

*return* first["at"] < second["at"];

         });

    printf("pid\tat\tbt\tct\ttat\twt\n");

    P[0]["ct"] = P[0]["at"] + P[0]["bt"];

    P[0].update\_after\_ct();

    P[0].display();

*for* (int i = 1; i < P.size(); i++)

    {

*if* (P[i]["at"] < P[i - 1]["ct"])

        {

            P[i]["ct"] = P[i - 1]["ct"] + P[i]["bt"];

        }

*else*

        {

            printf("curr['at'] : %d, prev['ct'] : %d\n",

                   P[i]["at"], P[i - 1]["ct"]);

            P[i]["ct"] = P[i]["at"] + P[i]["bt"];

        }

        P[i].update\_after\_ct();

        P[i].display();

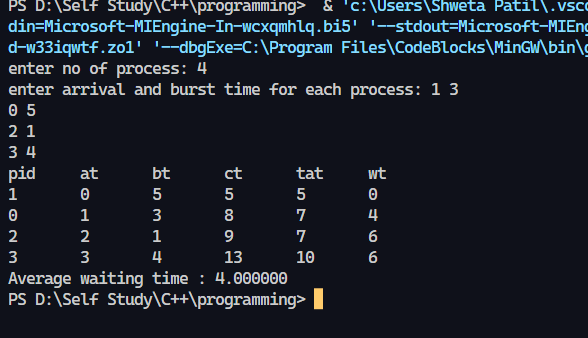
    }

    printf("Average waiting time : %f\n", average(P, "wt"));

*return* 0;

}

**Output:**

****

1. **Shortest Job First (SJF)**

**Code:**

*#include*<iostream>

using namespace std;

int main()

{

      int n,temp,tt=0,min,d,i,j;

      float atat=0,awt=0,stat=0,swt=0;

      cout<<"enter no of process"<<endl;

      cin>>n;

      int a[n],b[n],e[n],tat[n],wt[n];

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

      {

            cout<<"enter arival time "; *//input*

            cin>>a[i];

      }

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

      {

            cout<<"enter brust time "; *//input*

            cin>>b[i];

      }

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

      {

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

          {

*if*(b[i]>b[j])

                {

                      temp=a[i];

                      a[i]=a[j];

                      a[j]=temp;

                      temp=b[i];

                      b[i]=b[j];

                      b[j]=temp;

                }

          }

      }

      min=a[0];

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

      {

*if*(min>a[i])

            {

                  min=a[i];

                  d=i;

            }

      }

      tt=min;

      e[d]=tt+b[d];

      tt=e[d];

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

      {

*if*(a[i]!=min)

            {

                  e[i]=b[i]+tt;

                  tt=e[i];

            }

      }

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

      {

            tat[i]=e[i]-a[i];

            stat=stat+tat[i];

            wt[i]=tat[i]-b[i];

            swt=swt+wt[i];

      }

      atat=stat/n;

      awt=swt/n;

      cout<<"Process  Arrival-time(s)  Burst-time(s)  Waiting-time(s)  Turnaround-time(s)\n";

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

    {

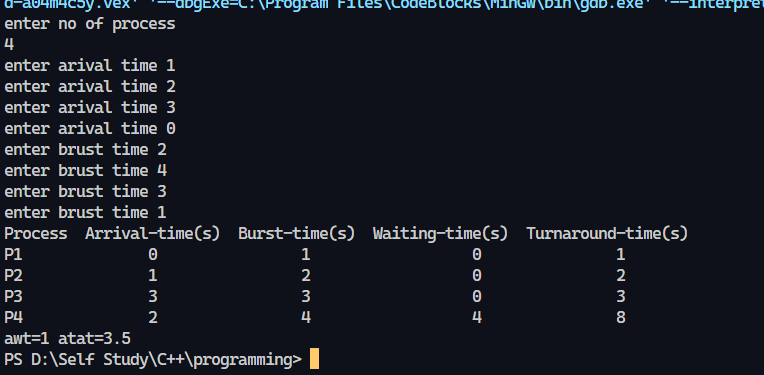
    cout<<"P"<<i+1<<"              "<<a[i]<<"                "<<b[i]<<"                  "<<wt[i]<<"               "<<tat[i]<<endl;

    }

    cout<<"awt="<<awt<<" atat="<<atat; *//average waiting time and turn around time*

}

**Output:**



1. **Shortest Remaining Time First**

**Code:**

*#include* <iostream>

*#include* <algorithm>

*#include* <iomanip>

*#include* <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

int main() {

    int x;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilization;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    float throughput;

    int burst\_remaining[100];

    int is\_completed[100];

    memset(is\_completed,0,sizeof(is\_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>x;

*for*(int i = 0; i < x; i++) {

        cout<<"Enter arrival time ofthe process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of the process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        p[i].pid = i+1;

        burst\_remaining[i] = p[i].burst\_time;

        cout<<endl;

    }

    int current\_time = 0;

    int completed = 0;

    int prev = 0;

*while*(completed != x) {

        int idx = -1;

        int mn = 10000000;

*for*(int i = 0; i < x; i++) {

*if*(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

*if*(burst\_remaining[i] < mn) {

                    mn = burst\_remaining[i];

                    idx = i;

                }

*if*(burst\_remaining[i] == mn) {

*if*(p[i].arrival\_time < p[idx].arrival\_time) {

                        mn = burst\_remaining[i];

                        idx = i;

                    }

                }

            }

        }

*if*(idx != -1) {

*if*(burst\_remaining[idx] == p[idx].burst\_time) {

                p[idx].start\_time = current\_time;

                total\_idle\_time += p[idx].start\_time - prev;

            }

            burst\_remaining[idx] -= 1;

            current\_time++;

            prev = current\_time;

*if*(burst\_remaining[idx] == 0) {

                p[idx].completion\_time = current\_time;

                p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

                p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

                p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

                total\_turnaround\_time += p[idx].turnaround\_time;

                total\_waiting\_time += p[idx].waiting\_time;

                total\_response\_time += p[idx].response\_time;

                is\_completed[idx] = 1;

                completed++;

            }

        }

*else* {

             current\_time++;

        }

    }

    int min\_arrival\_time = 10000000;

    int max\_completion\_time = -1;

*for*(int i = 0; i < x; i++) {

        min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

        max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / x;

    avg\_waiting\_time = (float) total\_waiting\_time / x;

    avg\_response\_time = (float) total\_response\_time / x;

    cpu\_utilization = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

    throughput = float(x) / (max\_completion\_time - min\_arrival\_time);

    cout<<endl<<endl;

    cout<<"Process\t"<<"Arrival Time\t"<<"Burst Time\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

*for*(int i = 0; i < x; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilization<<"%"<<endl;

    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;

}

**Output:**

Text

Description automatically generated

1. **Round Robin**

**Code :**

*// C++ program for implementation of RR scheduling*

*#include*<iostream>

using namespace std;

*// Function to find the waiting time for all*

*// processes*

void findWaitingTime(int processes[], int n,

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

{

*// Make a copy of burst times bt[] to store remaining*

*// burst times.*

    int rem\_bt[n];

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

        rem\_bt[i] = bt[i];

    int t = 0; *// Current time*

*// Keep traversing processes in round robin manner*

*// until all of them are not done.*

*while* (1)

    {

        bool done = true;

*// Traverse all processes one by one repeatedly*

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

        {

*// If burst time of a process is greater than 0*

*// then only need to process further*

*if* (rem\_bt[i] > 0)

            {

                done = false; *// There is a pending process*

*if* (rem\_bt[i] > quantum)

                {

*// Increase the value of t i.e. shows*

*// how much time a process has been processed*

                    t += quantum;

*// Decrease the burst\_time of current process*

*// by quantum*

                    rem\_bt[i] -= quantum;

                }

*// If burst time is smaller than or equal to*

*// quantum. Last cycle for this process*

*else*

                {

*// Increase the value of t i.e. shows*

*// how much time a process has been processed*

                    t = t + rem\_bt[i];

*// Waiting time is current time minus time*

*// used by this process*

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

*// As the process gets fully executed*

*// make its remaining burst time = 0*

                    rem\_bt[i] = 0;

                }

            }

        }

*// If all processes are done*

*if* (done == true)

*break*;

    }

}

*// Function to calculate turn around time*

void findTurnAroundTime(int processes[], int n,

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

{

*// calculating turnaround time by adding*

*// bt[i] + wt[i]*

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

        tat[i] = bt[i] + wt[i];

}

*// Function to calculate average time*

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

                                    int quantum)

{

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

*// Function to find waiting time of all processes*

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

*// Function to find turn around time for all processes*

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

*// Display processes along with all details*

    cout << "PN\t "<< " \tBT "

        << " WT " << " \tTAT\n";

*// Calculate total waiting time and total turn*

*// around time*

*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;

}

*// Driver code*

int main()

{

*// process id's*

    int processes[] = { 1, 2, 3};

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

*// Burst time of all processes*

    int burst\_time[] = {10, 5, 8};

*// Time quantum*

    int quantum = 2;

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

*return* 0;

}

**Output :**

Text

Description automatically generated

1. **Priority Scheduling**

**Code:**

*#include*<iostream>

 using namespace std;

int main()

{

    int a[10],b[10],x[10];

    int waiting[10],turnaround[10],completion[10],p[10];

    int i,j,smallest,count=0,time,n;

    double avg=0,tt=0,end;

   cout<<"\nEnter the number of Processes: ";

    cin>>n;

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

    {

      cout<<"\nEnter arrival time of process: ";

      cin>>a[i];

    }

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

    {

      cout<<"\nEnter burst time of process: ";

      cin>>b[i];

    }

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

    {

      cout<<"\nEnter priority of process: ";

      cin>>p[i];

    }

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

        x[i]=b[i];

    p[9]=-1;

*for*(time=0; count!=n; time++)

    {

        smallest=9;

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

        {

*if*(a[i]<=time && p[i]>p[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"<<"\t"<<"Priority"<<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"<<turnaround[i]<<"\t\t"<<completion[i]<<"\t\t"<<p[i]<<endl;

        avg = avg + waiting[i];

        tt = tt + turnaround[i];

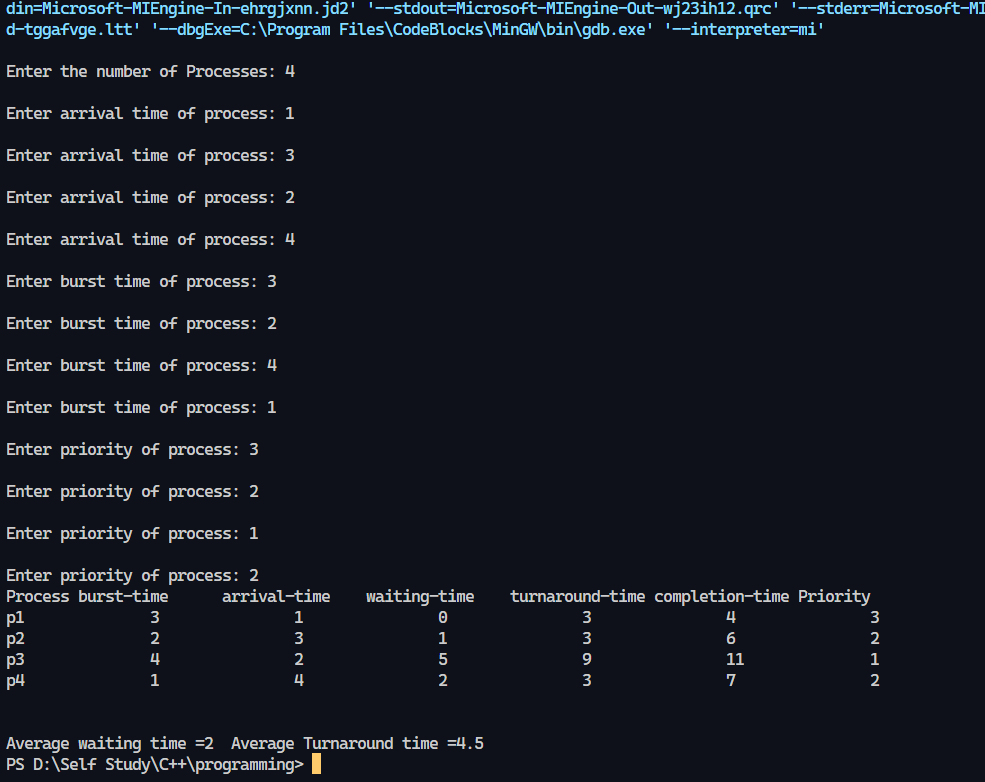
    }

   cout<<"\n\nAverage waiting time ="<<avg/n;

    cout<<"  Average Turnaround time ="<<tt/n<<endl;

}

**Output:**

****