**FAR WESTERN UNIVERSITY**

**MAHENDRANAGAR , KANCHANPUR**



**Lab Report on**

**Operating system**

**Submitted to:**

**Er. Rohit Bist**

**Submittted By:**

**Menuka Paneru**

**Roll.no 25**

**B.E COMPUTER 5TH SEM**

**2024, March 02**

1. **BANKERS ALGORITHM**

**CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**int max[100][100];**

**int alloc[100][100];**

**int need[100][100];**

**int avail[100];**

**int n,r;**

**void input();**

**void show();**

**void cal();**

**int main()**

**{**

**int i,j;**

**printf("\*\*\*\*\*\*\*\*\*\* Baner's Algo \*\*\*\*\*\*\*\*\*\*\*\*\n");**

**input();**

**show();**

**cal();**

**getch();**

**return 0;**

**}**

**void input()**

**{**

**int i,j;**

**printf("Enter the no of Processes\t");**

**scanf("%d",&n);**

**printf("Enter the no of resources instances\t");**

**scanf("%d",&r);**

**printf("Enter the Max Matrix\n");**

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

**for(j=0;j<r;j++) {**

**scanf("%d",&max[i][j]);**

**}}**

**printf("Enter the Allocation Matrix\n");**

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

**for(j=0;j<r;j++) {**

**scanf("%d",&alloc[i][j]);**

**}}**

**printf("Enter the available Resources\n");**

**for(j=0;j<r;j++) {**

**scanf("%d",&avail[j]);**

**}}**

**void show() {**

**int i,j;**

**printf("Process\t Allocation\t Max\t Available\t");**

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

**printf("\nP%d\t ",i+1);**

**for(j=0;j<r;j++) {**

**printf("%d ",alloc[i][j]); }**

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

**for(j=0;j<r;j++) {**

**printf("%d ",max[i][j]); }**

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

**if(i==0) {**

**for(j=0;j<r;j++)**

**printf("%d ",avail[j]);**

**}}}**

**void cal()**

**{**

**int finish[100],temp,need[100][100],flag=1,k,c1=0;**

**int safe[100];**

**int i,j;**

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

**finish[i]=0; }**

**//find need matrix**

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

**for(j=0;j<r;j++) {**

**need[i][j]=max[i][j]**

**-alloc[i][j];**

**}}**

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

**while(flag) {**

**flag=0;**

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

**int c=0;**

**for(j=0;j<r;j++) {**

**if((finish[i]==0)&&(need[i][j]<=avail[j])) {**

**c++;**

**if(c==r) {**

**for(k=0;k<r;k++) {**

**avail[k]+=alloc[i][j];**

**finish[i]=1;**

**flag=1; }**

**printf("P%d->",i);**

**if(finish[i]==1) {**

**i=n;**

**}}}}}}**

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

**if(finish[i]==1) {**

**c1++;**

**}**

**else**

**{printf("P%d->",i);**

**}}**

**if(c1==n)**

**{printf("\n The system is in safe state");**

**}**

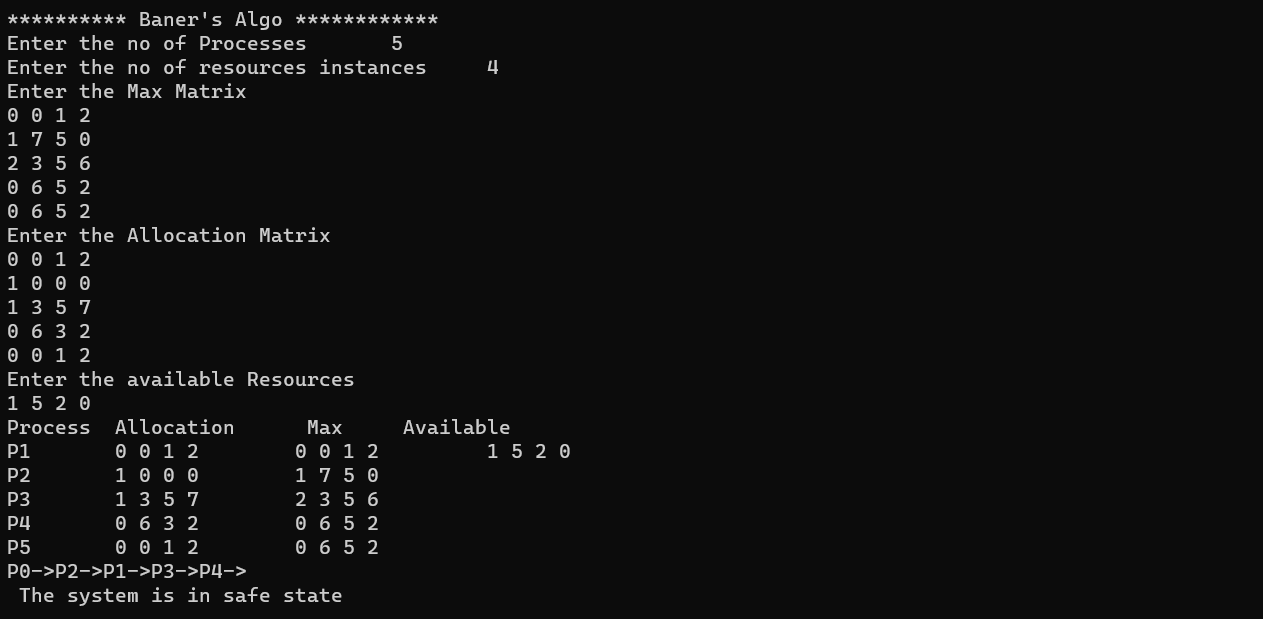
**else**

**{**

**printf("\n Process are in dead lock");**

**printf("\n System is in unsafe state");**

**}}**



1. **BEST FIT**

**CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**#define max 25**

**void main()**

**{**

**int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;**

**static int bf[max],ff[max];**

**printf("\nEnter the number of blocks:");**

**scanf("%d",&nb);**

**printf("Enter the number of files:");**

**scanf("%d",&nf);**

**printf("\nEnter the size of the blocks:-\n");**

**for(i=1;i<=nb;i++)**

**{**

**printf("Block %d:",i);**

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

**}**

**printf("Enter the size of the files :-\n");**

**for(i=1;i<=nf;i++)**

**{**

**printf("File %d:",i);**

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

**}**

**for(i=1;i<=nf;i++)**

**{**

**for(j=1;j<=nb;j++)**

**{**

**if(bf[j]!=1)**

**{**

**temp=b[j]-f[i];**

**if(temp>=0)**

**if(lowest>temp)**

**{**

**ff[i]=j;**

**lowest=temp;**

**}**

**}**

**}**

**frag[i]=lowest;**

**bf[ff[i]]=1;**

**lowest=10000;**

**}**

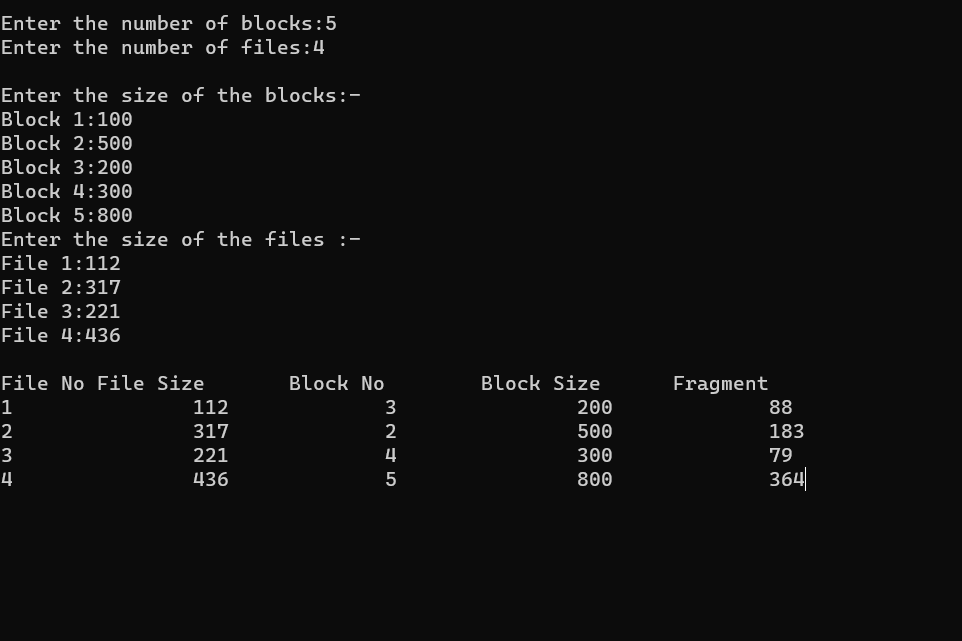
**printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");**

**for(i=1;i<=nf && ff[i]!=0;i++)**

**printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);**

**getch();**

**}**



1. **DINING PHILOSOPHER**

**CODE:**

**#include <pthread.h>**

**#include <semaphore.h>**

**#include <stdio.h>**

**#define N 5**

**#define THINKING 2**

**#define HUNGRY 1**

**#define EATING 0**

**#define LEFT (phnum + 4) % N**

**#define RIGHT (phnum + 1) % N**

**int state[N];**

**int phil[N] = { 0, 1, 2, 3, 4 };**

**sem\_t mutex;**

**sem\_t S[N];**

**void test(int phnum)**

**{**

**if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)**

**{**

**// state that eating**

**state[phnum] = EATING;**

**sleep(2);**

**printf("Philosopher %d takes fork %d and %d\n",phnum + 1, LEFT + 1, phnum + 1);**

**printf("Philosopher %d is Eating\n", phnum + 1);**

**// sem\_post(&S[phnum]) has no effect**

**// during takefork**

**// used to wake up hungry philosophers**

**// during putfork**

**sem\_post(&S[phnum]);**

**}**

**}**

**// take up chopsticks**

**void take\_fork(int phnum)**

**{**

**sem\_wait(&mutex);**

**// state that hungry**

**state[phnum] = HUNGRY;**

**printf("Philosopher %d is Hungry\n", phnum + 1);**

**// eat if neighbours are not eating**

**test(phnum);**

**sem\_post(&mutex);**

**// if unable to eat wait to be signalled**

**sem\_wait(&S[phnum]);**

**sleep(1);**

**}**

**// put down chopsticks**

**void put\_fork(int phnum)**

**{**

**sem\_wait(&mutex);**

**// state that thinking**

**state[phnum] = THINKING;**

**printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);**

**printf("Philosopher %d is thinking\n", phnum + 1);**

**test(LEFT);**

**test(RIGHT);**

**sem\_post(&mutex);**

**}**

**void\* philosopher(void\* num)**

**{**

**while (1) {**

**int\* i = num;**

**sleep(1);**

**take\_fork(\*i);**

**sleep(0);**

**put\_fork(\*i);**

**}**

**}**

**int main()**

**{**

**int i;**

**pthread\_t thread\_id[N];**

**// initialize the semaphores**

**sem\_init(&mutex, 0, 1);**

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

**sem\_init(&S[i], 0, 0);**

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

**{**

**// create philosopher processes**

**pthread\_create(&thread\_id[i], NULL, philosopher, &phil[i]);**

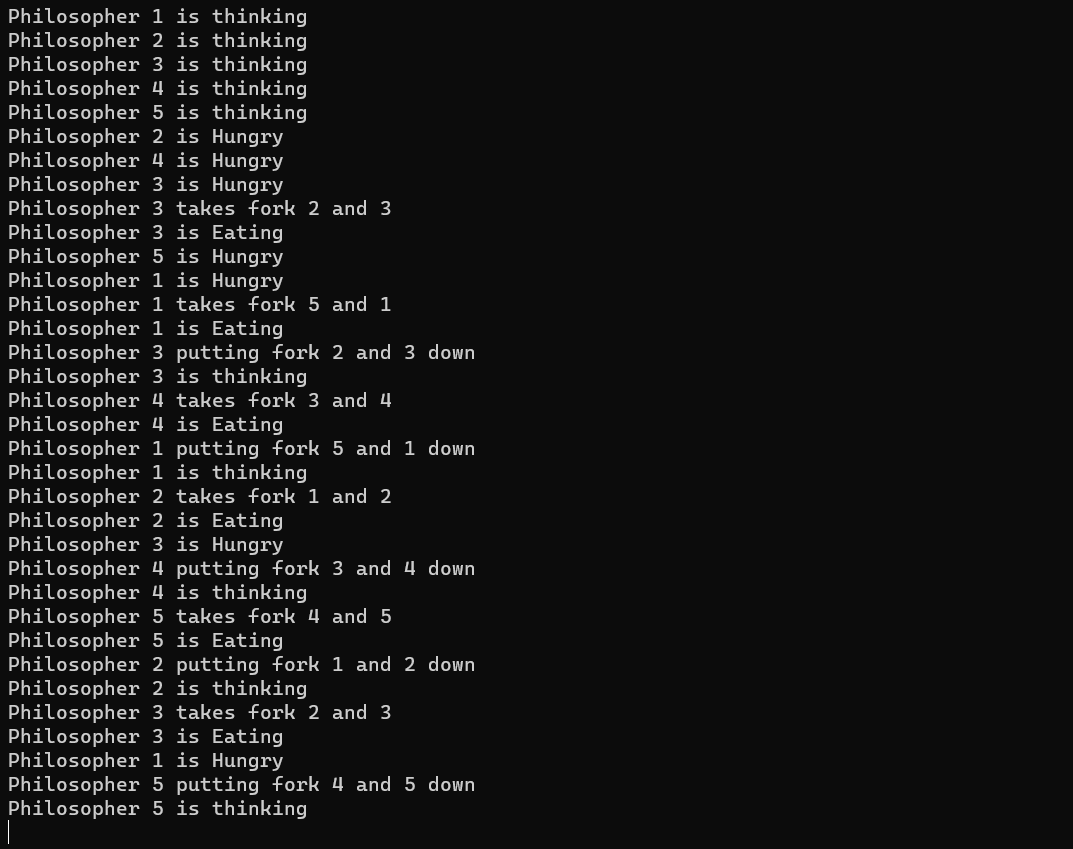
**printf("Philosopher %d is thinking\n", i + 1);**

**}**

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

**pthread\_join(thread\_id[i], NULL);**

**}**



1. **FCFS SCHEDULING**

**CODE:**

**#include<stdio.h>**

**int main()**

**{**

**int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;**

**printf("Enter total number of processes(maximum 20):");**

**scanf("%d",&n);**

**printf("\nEnter Process Burst Time:\t");**

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

**{**

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

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

**}**

**wt[0]=0;**

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

**{**

**wt[i]=0;**

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

**wt[i]+=bt[j];**

**}**

**printf("Process\t\tBurst Time\tWaiting Time\tTurnaround Time");**

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

**{**

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

**avwt+=wt[i];**

**avtat+=tat[i];**

**printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);**

**}**

**avwt/=i;**

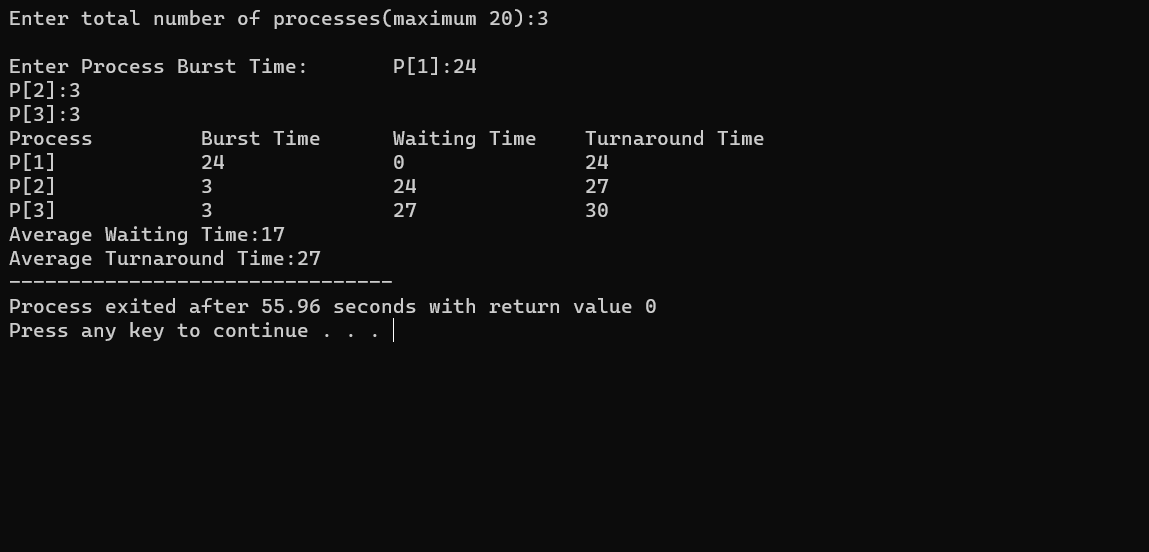
**avtat/=i;**

**printf("\nAverage Waiting Time:%d",avwt);**

**printf("\nAverage Turnaround Time:%d",avtat);**

**return 0;**

**}**



1. **FIRST FIT**

**CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**#define max 25**

**void main()**

**{**

**int frag[max],b[max],f[max],i,j,nb,nf,temp;**

**static int bf[max],ff[max];**

**printf("\n\tMemory Management Scheme - First Fit");**

**printf("\nEnter the number of blocks:");**

**scanf("%d",&nb);**

**printf("Enter the number of files:");**

**scanf("%d",&nf);**

**printf("\nEnter the size of the blocks:-\n");**

**for(i=1;i<=nb;i++)**

**{**

**printf("Block %d:",i);**

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

**}**

**printf("Enter the size of the files :-\n");**

**for(i=1;i<=nf;i++)**

**{**

**printf("File %d:",i);**

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

**}**

**for(i=1;i<=nf;i++)**

**{**

**for(j=1;j<=nb;j++)**

**{**

**if(bf[j]!=1)**

**{**

**temp=b[j]-f[i];**

**if(temp>=0)**

**{**

**ff[i]=j;**

**break;**

**}**

**}**

**}**

**frag[i]=temp;**

**bf[ff[i]]=1;**

**}**

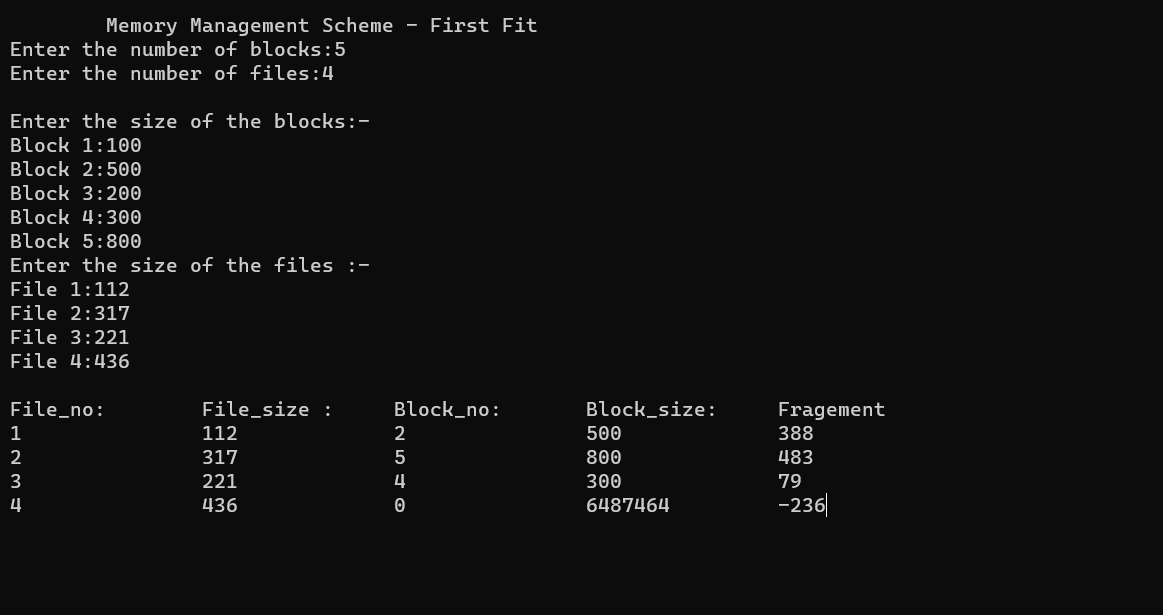
**printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");**

**for(i=1;i<=nf;i++)**

**printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);**

**getch();**

**}**



1. **HRNN SCHEDULING**

**CODE:**

**#include <stdio.h>**

**// Defining process details**

**struct process {**

**char name;**

**int at, bt, ct, wt, tt;**

**int completed;**

**float ntt;**

**} p[10];**

**int n;**

**// Sorting Processes by Arrival Time**

**void sortByArrival()**

**{**

**struct process temp;**

**int i, j;**

**// Selection Sort applied**

**for (i = 0; i < n - 1; i++) {**

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

**// Check for lesser arrival time**

**if (p[i].at > p[j].at) {**

**// Swap earlier process to front**

**temp = p[i];**

**p[i] = p[j];**

**p[j] = temp;**

**}**

**}**

**}**

**}**

**int main()**

**{**

**int i, j, t, sum\_bt = 0;**

**char c;**

**float avgwt = 0, avgtt = 0;**

**n = 5;**

**// predefined arrival times**

**int arriv[] = { 0, 2, 4, 6, 8 };**

**// predefined burst times**

**int burst[] = { 3, 6, 4, 5, 2 };**

**// Initializing the structure variables**

**for (i = 0, c = 'A'; i < n; i++, c++) {**

**p[i].name = c;**

**p[i].at = arriv[i];**

**p[i].bt = burst[i];**

**// Variable for Completion status**

**// Pending = 0**

**// Completed = 1**

**p[i].completed = 0;**

**// Variable for sum of all Burst Times**

**sum\_bt += p[i].bt;**

**}**

**// Sorting the structure by arrival times**

**sortByArrival();**

**printf("\nName\tArrival Time\tBurst Time\tWaiting Time");**

**printf("\tTurnAround Time\t Normalized TT");**

**for (t = p[0].at; t < sum\_bt;) {**

**// Set lower limit to response ratio**

**float hrr = -9999;**

**// Response Ratio Variable**

**float temp;**

**// Variable to store next process selected**

**int loc;**

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

**// Checking if process has arrived and is Incomplete**

**if (p[i].at <= t && p[i].completed != 1) {**

**// Calculating Response Ratio**

**temp = (p[i].bt + (t - p[i].at)) / p[i].bt;**

**// Checking for Highest Response Ratio**

**if (hrr < temp) {**

**// Storing Response Ratio**

**hrr = temp;**

**// Storing Location**

**loc = i;**

**}**

**}**

**}**

**// Updating time value**

**t += p[loc].bt;**

**// Calculation of waiting time**

**p[loc].wt = t - p[loc].at - p[loc].bt;**

**// Calculation of Turn Around Time**

**p[loc].tt = t - p[loc].at;**

**// Sum Turn Around Time for average**

**avgtt += p[loc].tt;**

**// Calculation of Normalized Turn Around Time**

**p[loc].ntt = ((float)p[loc].tt / p[loc].bt);**

**// Updating Completion Status**

**p[loc].completed = 1;**

**// Sum Waiting Time for average**

**avgwt += p[loc].wt;**

**printf("\n%c\t\t%d\t\t", p[loc].name, p[loc].at);**

**printf("%d\t\t%d\t\t", p[loc].bt, p[loc].wt);**

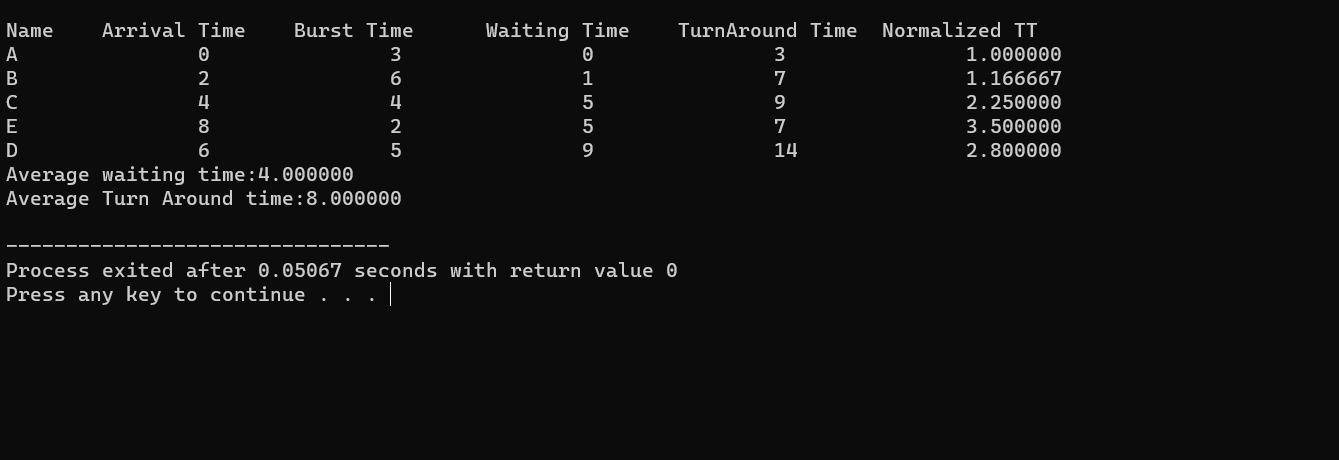
**printf("%d\t\t%f", p[loc].tt, p[loc].ntt);**

**}**

**printf("\nAverage waiting time:%f\n", avgwt / n);**

**printf("Average Turn Around time:%f\n", avgtt / n);**

**}**



1. **PRIORITY SCHEDULING**

**CODE:**

**#include <stdio.h>**

**#define max 5**

**int main()**

**{**

**int i,j,n,t,p[max],bt[max],pr[max],wt[max],tat[max],Total\_wt=0,Total\_tat=0;**

**float awt=0,atat=0;**

**printf("Enter the number of processes\n");**

**scanf("%d",&n);**

**//Enter the processes according to their arrival times**

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

**{**

**printf("Enter the process number\n");**

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

**printf("Enter the burst time of the process\n");**

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

**printf("Enter the priority of the process\n");**

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

**}**

**//Apply the bubble sort technique to sort the processes according to their priorities times**

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

**{**

**for(j=0;j<n-i-1;j++)**

**{**

**if(pr[j]>pr[j+1])**

**{**

**// Sort according to priorities**

**t=pr[j];**

**pr[j]=pr[j+1];**

**pr[j+1]=t;**

**// Sorting burst times**

**t=bt[j];**

**bt[j]=bt[j+1];**

**bt[j+1]=t;**

**// Sorting Process numbers**

**t=p[j];**

**p[j]=p[j+1];**

**p[j+1]=t;**

**} //if**

**} //for**

**} //for**

**printf("Processid \t Burst Time\t Priority\tWaiting Time\t Turn Around Time\n");**

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

**{**

**wt[i]=0;**

**tat[i]=0;**

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

**wt[i]=wt[i]+bt[j];**

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

**Total\_wt=Total\_wt+wt[i];**

**Total\_tat=Total\_tat+tat[i];**

**printf("%d\t\t %d\t\t%d\t\t %d\t\t %d\n",p[i],bt[i],pr[i],wt[i],tat[i]);**

**}**

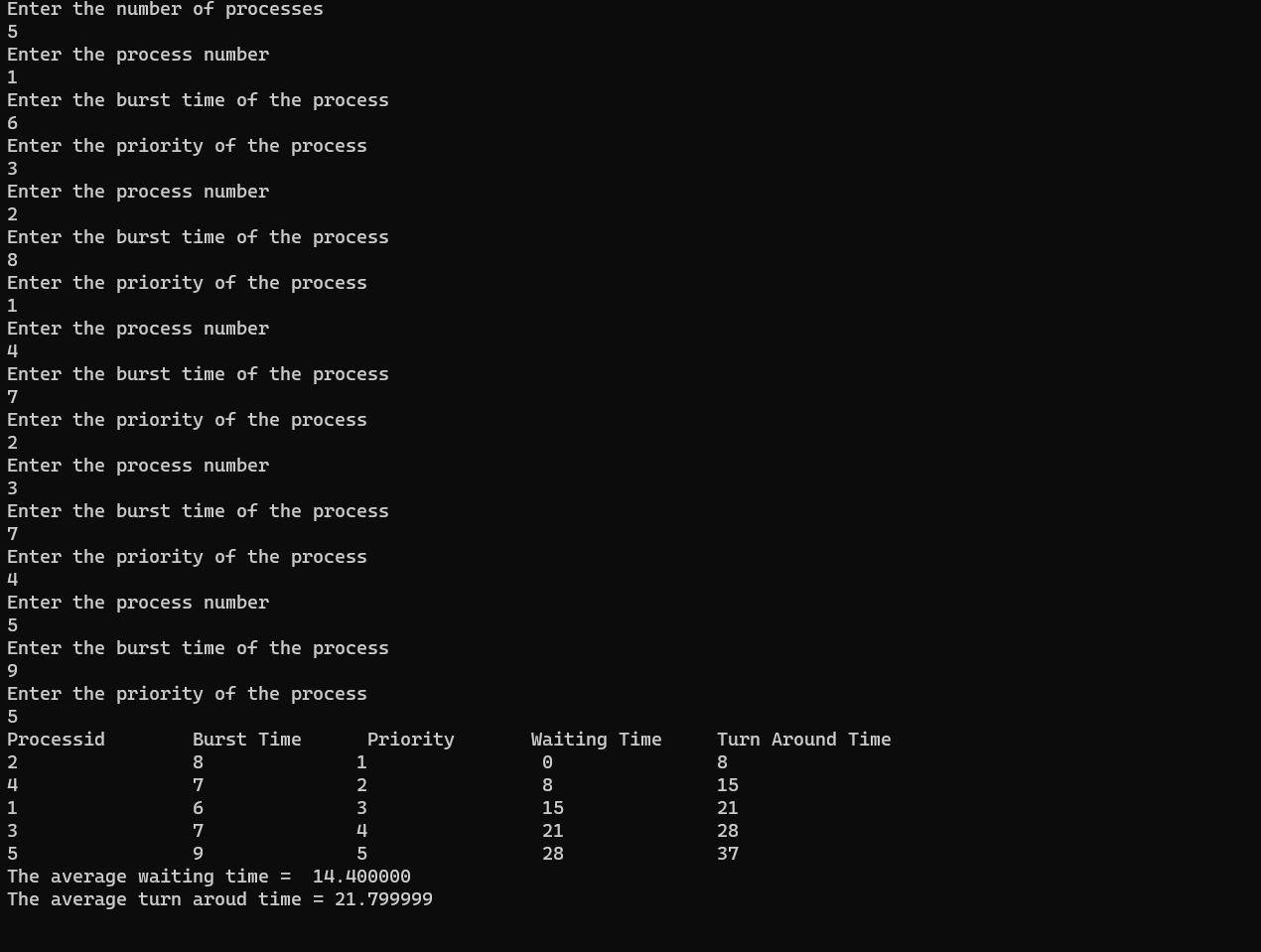
**awt=(float)Total\_wt/n;**

**atat=(float)Total\_tat/n;**

**printf("The average waiting time = %f\n",awt);**

**printf("The average turn aroud time = %f\n",atat);**

**}**



1. **PRODUCER-CONSUMER PROBLEM**

**CODE:**

**#include<stdio.h>**

**int mutex=1,full=0,empty=3,x=0;**

**main()**

**{**

**int n;**

**void producer();**

**void consumer();**

**int wait(int);**

**int signal(int);**

**printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");**

**while(1) {**

**printf("\nENTER YOUR CHOICE\n");**

**scanf("%d",&n);**

**switch(n)**

**{ case 1:**

**if((mutex==1)&&(empty!=0))**

**producer();**

**else**

**printf("BUFFER IS FULL");**

**break;**

**case 2:**

**if((mutex==1)&&(full!=0))**

**consumer();**

**else**

**printf("BUFFER IS EMPTY");**

**break;**

**case 3:**

**exit(0);**

**break; }}}**

**int wait(int s)**

**{**

**return(--s);**

**}**

**int signal(int s) {**

**return(++s); }**

**void producer()**

**{**

**mutex=wait(mutex);**

**full=signal(full);**

**empty=wait(empty);**

**x++;**

**printf("\nproducer producesthe item%d",x);**

**mutex=signal(mutex);**

**}**

**void consumer() {**

**mutex=wait(mutex);**

**full=wait(full);**

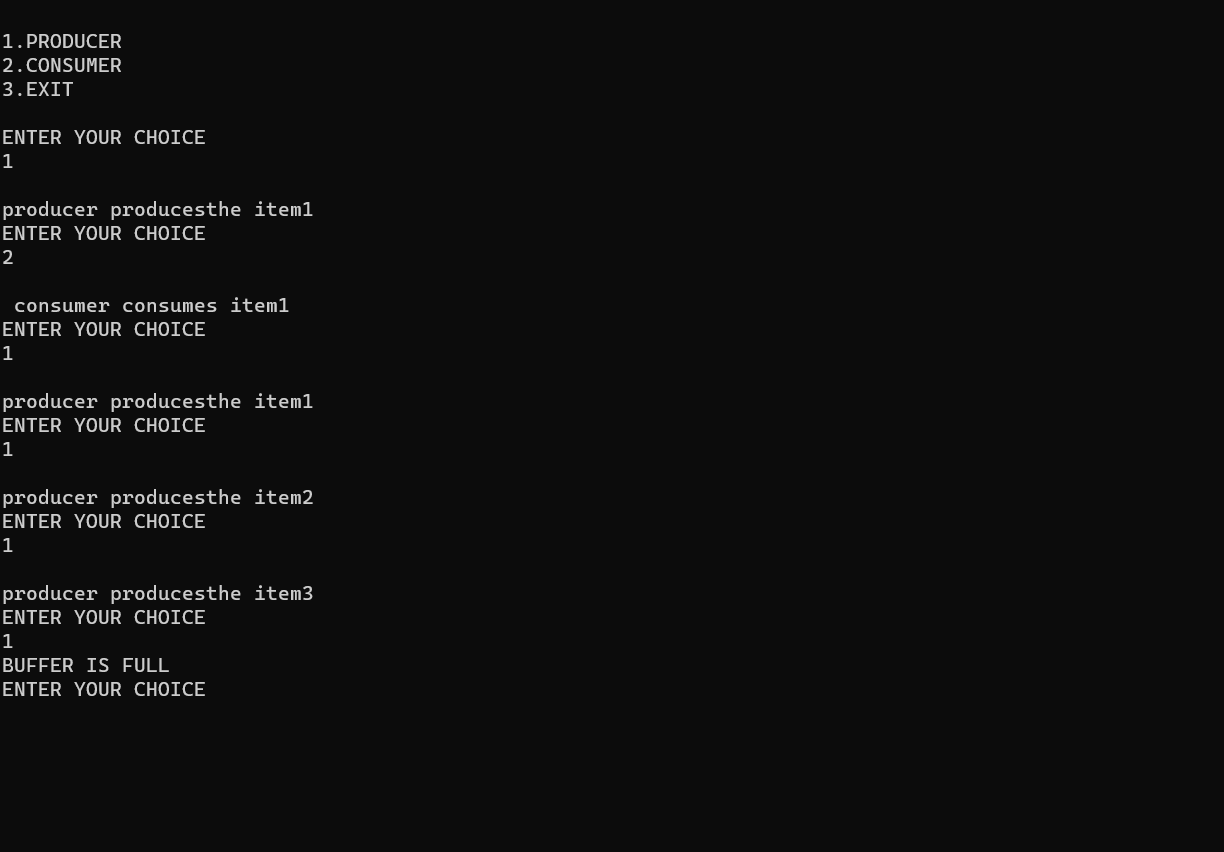
**empty=signal(empty);**

**printf("\n consumer consumes item%d",x);**

**x--;**

**mutex=signal(mutex);**

**}**



1. **RR SCHEDULING**

**CODE:**

**#include<stdio.h>**

**int main()**

**{**

**int i, limit, total = 0, x, counter = 0, time\_quantum;**

**int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];**

**float average\_wait\_time, average\_turnaround\_time;**

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

**scanf("%d", &limit);**

**x = limit;**

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

**{**

**printf("\nEnter Details of Process[%d]\n", i + 1);**

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

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

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

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

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

**}**

**printf("\nEnter Time Quantum:\t");**

**scanf("%d", &time\_quantum);**

**printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");**

**for(total = 0, i = 0; x != 0;)**

**{**

**if(temp[i] <= time\_quantum && temp[i] > 0)**

**{**

**total = total + temp[i];**

**temp[i] = 0;**

**counter = 1;**

**}**

**else if(temp[i] > 0)**

**{**

**temp[i] = temp[i] - time\_quantum;**

**total = total + time\_quantum;**

**}**

**if(temp[i] == 0 && counter == 1)**

**{**

**x--;**

**printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);**

**wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];**

**turnaround\_time = turnaround\_time + total - arrival\_time[i];**

**counter = 0;**

**}**

**if(i == limit - 1)**

**{**

**i = 0;**

**}**

**else if(arrival\_time[i + 1] <= total)**

**{**

**i++;**

**}**

**else**

**{**

**i = 0;**

**}**

**}**

**average\_wait\_time = wait\_time \* 1.0 / limit;**

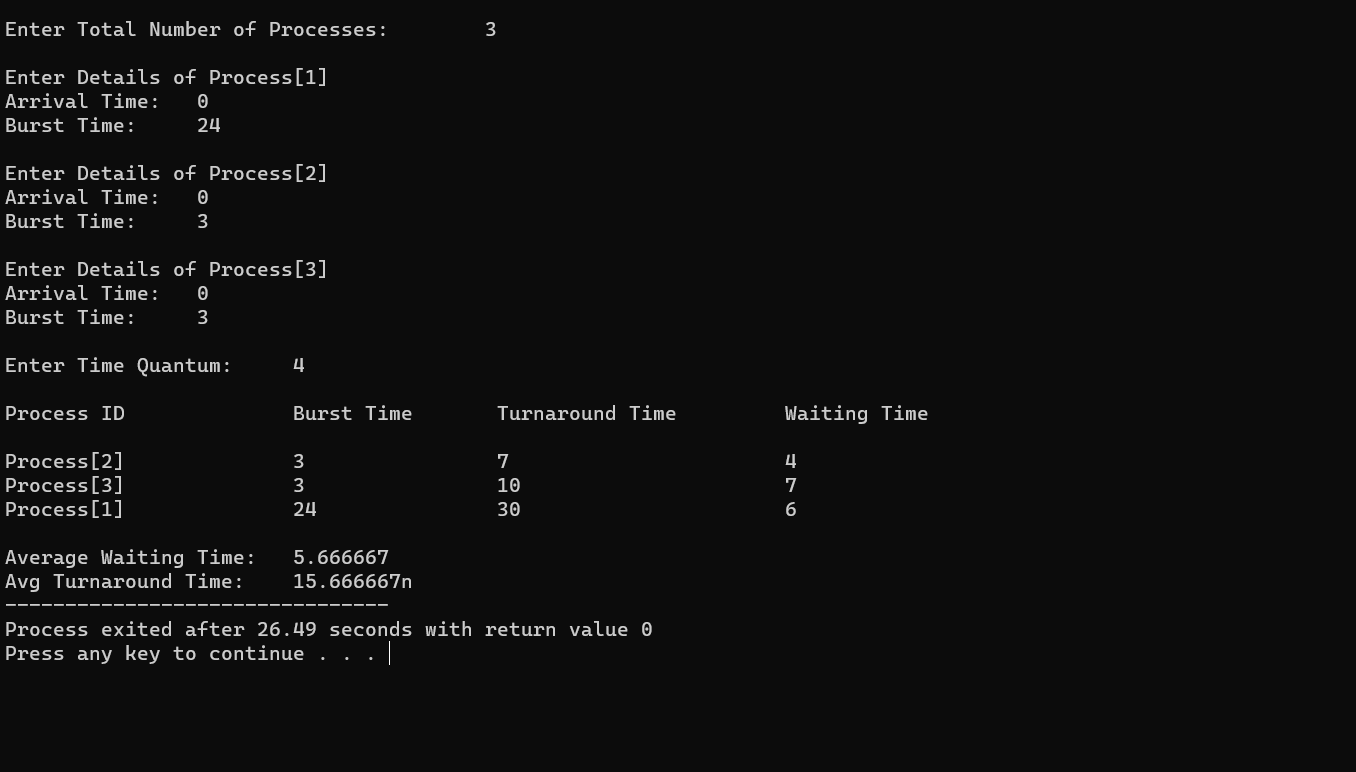
**average\_turnaround\_time = turnaround\_time \* 1.0 / limit;**

**printf("\n\nAverage Waiting Time:\t%f", average\_wait\_time);**

**printf("\nAvg Turnaround Time:\t%fn", average\_turnaround\_time);**

**return 0;**

**}**



1. **SJF SCHEDULING**

**CODE:**

**#include <stdio.h>**

**int main()**

**{**

**// Matrix for storing Process Id, Burst**

**// Time, Average Waiting Time & Average**

**// Turn 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]);**

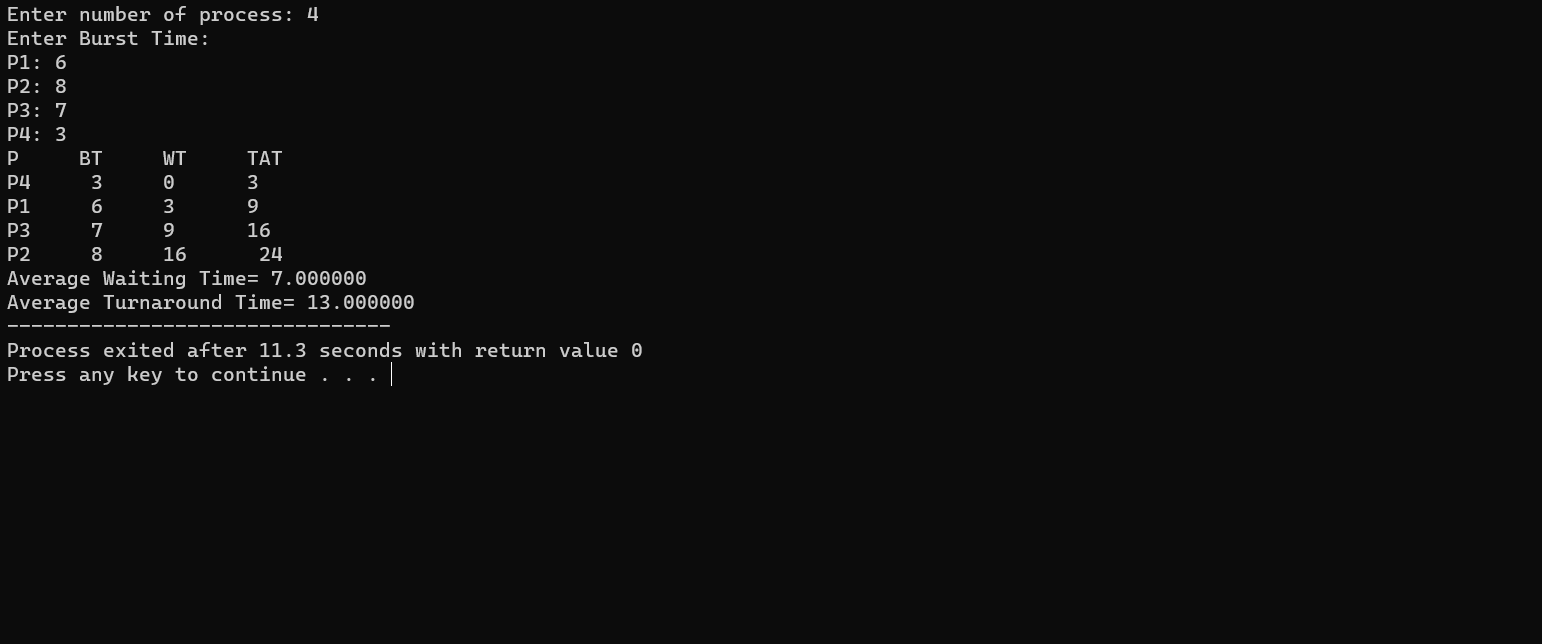
**}**

**avg\_tat = (float)total / n;**

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

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

**}**



1. **SLEEPING-BARBER PROBLEM**

**CODE:**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <semaphore.h>**

**#include <time.h>**

**#include <sys/types.h>**

**#include <sys/time.h>**

**void \*barber\_function(void \*idp);**

**void \*customer\_function(void \*idp);**

**void serve\_customer();**

**void \*make\_customer\_function();**

**/\* Mutex \*/**

**pthread\_mutex\_t srvCust;**

**/\* Semaphores \*/**

**sem\_t barber\_ready;**

**sem\_t customer\_ready;**

**sem\_t modifySeats;**

**/\* Inputs \*/**

**int chair\_cnt;**

**int total\_custs;**

**int available\_seats;**

**int no\_served\_custs = 0;**

**time\_t waiting\_time\_sum;**

**void \*barber\_function(void \*idp)**

**{**

**int counter = 0;**

**while (1)**

**{**

**/\* Lock semaphore "customer\_ready" - try to get a customer or sleep if there is none \*/**

**sem\_wait(&customer\_ready);**

**/\* Lock semaphore "modifySeats" - try to get access to seats \*/**

**sem\_wait(&modifySeats);**

**/\* Increment by 1 the available seats \*/**

**available\_seats++;**

**/\* Unlock semaphore "modifySeats" \*/**

**sem\_post(&modifySeats);**

**/\* Unlock semaphore "barber\_ready" - set barber ready to serve \*/**

**sem\_post(&barber\_ready);**

**/\* Lock mutex "srvCust" - protect service by the same barber from other threads \*/**

**pthread\_mutex\_lock(&srvCust);**

**/\* Serve customer \*/**

**serve\_customer();**

**/\* Unlock mutex "srvCust" - finished service \*/**

**pthread\_mutex\_unlock(&srvCust);**

**printf("Customer was served.\n");**

**counter++;**

**if (counter == (total\_custs - no\_served\_custs))**

**break;**

**}**

**pthread\_exit(NULL);**

**}**

**void \*customer\_function(void \*idp)**

**{**

**struct timeval start, stop;**

**/\* Lock semaphore "modifySeats" \*/**

**sem\_wait(&modifySeats);**

**/\* If there is available seat \*/**

**if (available\_seats >= 1)**

**{**

**/\* Occupy a seat \*/**

**available\_seats--;**

**printf("Customer[pid = %lu] is waiting.\n", pthread\_self());**

**printf("Available seats: %d\n", available\_seats);**

**/\* Start waiting-time counter \*/**

**gettimeofday(&start, NULL);**

**/\* Unlock semaphore "customer\_ready" - set the customer ready to be served \*/**

**sem\_post(&customer\_ready);**

**/\* Unlock semaphore "modifySeats" \*/**

**sem\_post(&modifySeats);**

**/\* Lock semaphore "barber\_ready" - wait for barber to get ready \*/**

**sem\_wait(&barber\_ready);**

**/\* Stop waiting-time counter \*/**

**gettimeofday(&stop, NULL);**

**double sec = (double)(stop.tv\_usec - start.tv\_usec) / 1000000 + (double)(stop.tv\_sec - start.tv\_sec);**

**/\* Assign the time spent to global variable (ms) \*/**

**waiting\_time\_sum += 1000 \* sec;**

**printf("Customer[pid = %lu] is being served. \n", pthread\_self());**

**}**

**else**

**{**

**/\* Unlock semaphore "modifySeats" \*/**

**sem\_post(&modifySeats);**

**no\_served\_custs++;**

**printf("A Customer left.\n");**

**}**

**pthread\_exit(NULL);**

**}**

**void serve\_customer() {**

**/\* Random number between 0 and 400 (miliseconds) \*/**

**int s = rand() % 401;**

**/\* Convert miliseconds to microseconds \*/**

**s = s \* 1000;**

**usleep(s);**

**}**

**void \*make\_customer\_function() {**

**int tmp;**

**int counter = 0;**

**while (counter < total\_custs)**

**{**

**/\* Declare and create a customer thread \*/**

**pthread\_t customer\_thread;**

**tmp = pthread\_create(&customer\_thread, NULL, (void \*)customer\_function, NULL);**

**if (tmp)**

**printf("Failed to create thread.");**

**/\* Increment the counter \*/**

**counter++;**

**/\* Sleep for 100ms before creating another customer \*/**

**usleep(100000);**

**}**

**}**

**int main() {**

**/\* Initialization, should only be called once \*/**

**srand(time(NULL));**

**/\* Barber 1 thread \*/**

**pthread\_t barber\_1;**

**/\* Thread that creates customers \*/**

**pthread\_t customer\_maker;**

**int tmp;**

**/\* Initialize mutex \*/**

**pthread\_mutex\_init(&srvCust, NULL);**

**/\* Initialize semaphores \*/**

**sem\_init(&customer\_ready, 0, 0);**

**sem\_init(&barber\_ready, 0, 0);**

**sem\_init(&modifySeats, 0, 1);**

**printf("Please enter the number of seats: \n");**

**scanf("%d", &chair\_cnt);**

**printf("Please enter the total customers: \n");**

**scanf("%d", &total\_custs);**

**available\_seats = chair\_cnt;**

**/\* Create barber thread \*/**

**tmp = pthread\_create(&barber\_1, NULL, (void \*)barber\_function, NULL);**

**if (tmp)**

**printf("Failed to create thread.");**

**/\* Create customer\_maker thread \*/**

**tmp = pthread\_create(&customer\_maker, NULL, (void \*)make\_customer\_function, NULL);**

**if (tmp)**

**printf("Failed to create thread.");**

**/\* Wait for threads to finish \*/**

**pthread\_join(barber\_1, NULL);**

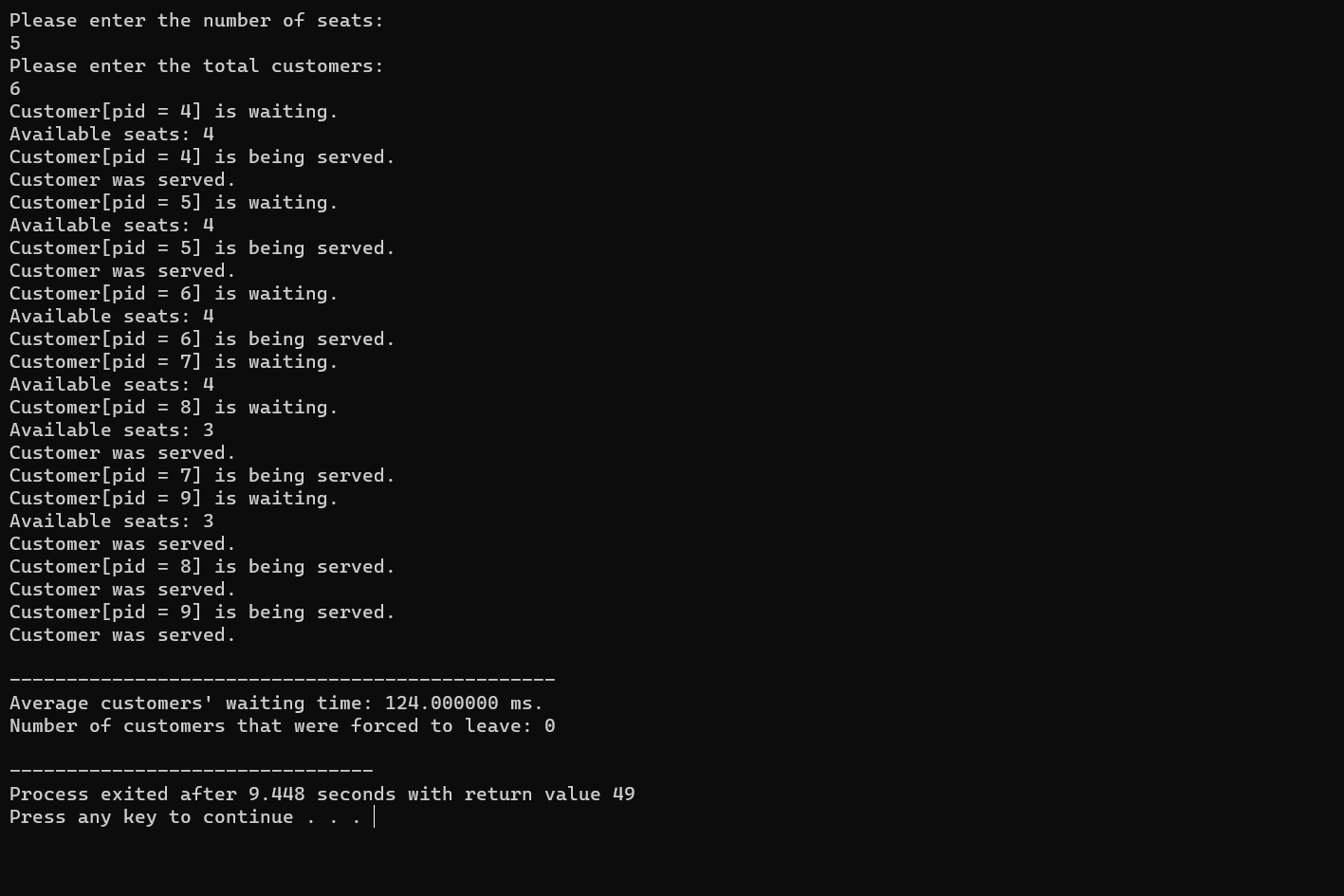
**pthread\_join(customer\_maker, NULL);**

**printf("\n------------------------------------------------\n");**

**printf("Average customers' waiting time: %f ms.\n", (waiting\_time\_sum / (double) (total\_custs - no\_served\_custs)));**

**printf("Number of customers that were forced to leave: %d\n", no\_served\_custs);**

**}**



1. **SRTF SCHEDULING**

**Code:**

**#include<stdio.h>**

**int main()**

**{**

**int at[10],bt[10],rt[10],endTime,i,smallest;**

**int remain=0,n,time,sum\_wait=0,sum\_turnaround=0;**

**printf("Enter no of Processes : ");**

**scanf("%d",&n);**

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

**{**

**printf("Enter arrival time for Process P%d : ",i+1);**

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

**printf("Enter burst time for Process P%d : ",i+1);**

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

**rt[i]=bt[i];**

**}**

**printf("\n\nProcess\t|Turnaround Time| Waiting Time\n\n");**

**rt[9]=9999;**

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

**{**

**smallest=9;**

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

**{**

**if(at[i]<=time && rt[i]<rt[smallest] && rt[i]>0)**

**{**

**smallest=i;**

**}**

**}**

**rt[smallest]--;**

**if(rt[smallest]==0)**

**{**

**remain++;**

**endTime=time+1;**

**printf("\nP[%d]\t|\t%d\t|\t%d",smallest+1,endTime-at[smallest],endTime-bt[smallest]-at[smallest]);**

**sum\_wait+=endTime-bt[smallest]-at[smallest];**

**sum\_turnaround+=endTime-at[smallest];**

**}**

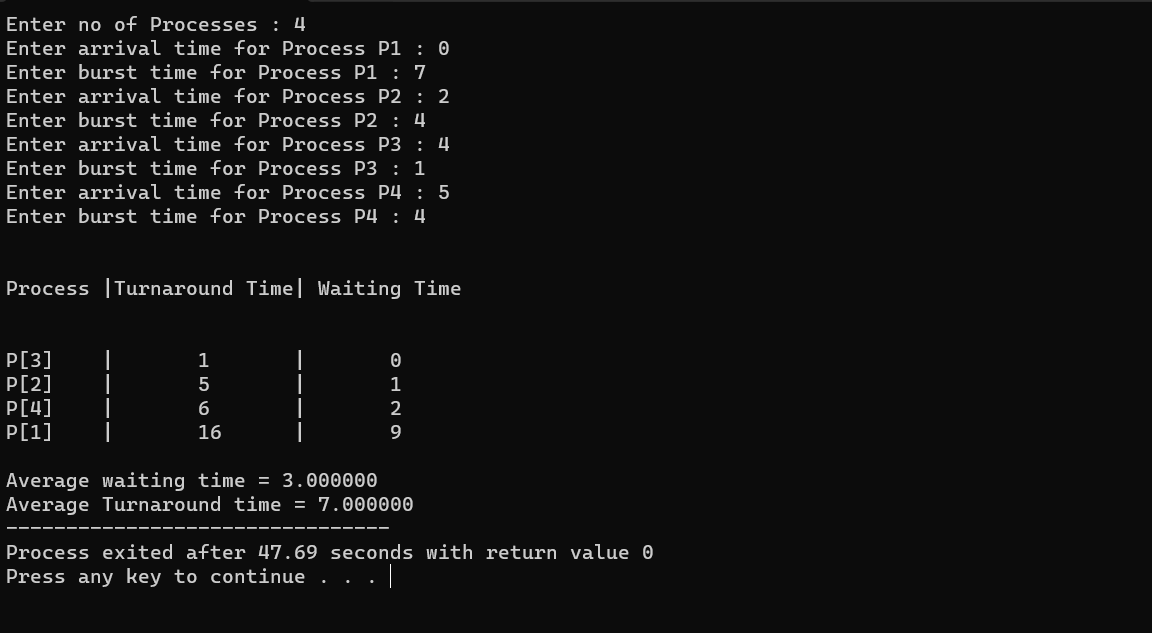
**}**

**printf("\n\nAverage waiting time = %f\n",sum\_wait\*1.0/n);**

**printf("Average Turnaround time = %f",sum\_turnaround\*1.0/n);**

**return 0;**

**}**



**13.WORST FIT**

**CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**#define max 25**

**int main()**

**{**

**int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;**

**static int bf[max],ff[max];**

**printf("\n\tMemory Management Scheme - Worst Fit");**

**printf("\nEnter the number of blocks:");**

**scanf("%d",&nb);**

**printf("Enter the number of files:");**

**scanf("%d",&nf);**

**printf("\nEnter the size of the blocks:-\n");**

**for(i=1;i<=nb;i++)**

**{**

**printf("Block %d:",i);**

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

**}**

**printf("Enter the size of the files :-\n");**

**for(i=1;i<=nf;i++)**

**{**

**printf("File %d:",i);**

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

**}**

**for(i=1;i<=nf;i++)**

**{**

**for(j=1;j<=nb;j++)**

**{**

**if(bf[j]!=1) //if bf[j] is not allocated**

**{**

**temp=b[j]-f[i];**

**if(temp>=0)**

**if(highest<temp)**

**{**

**ff[i]=j;**

**highest=temp;**

**}**

**}**

**}**

**frag[i]=highest;**

**bf[ff[i]]=1;**

**highest=0;**

**}**

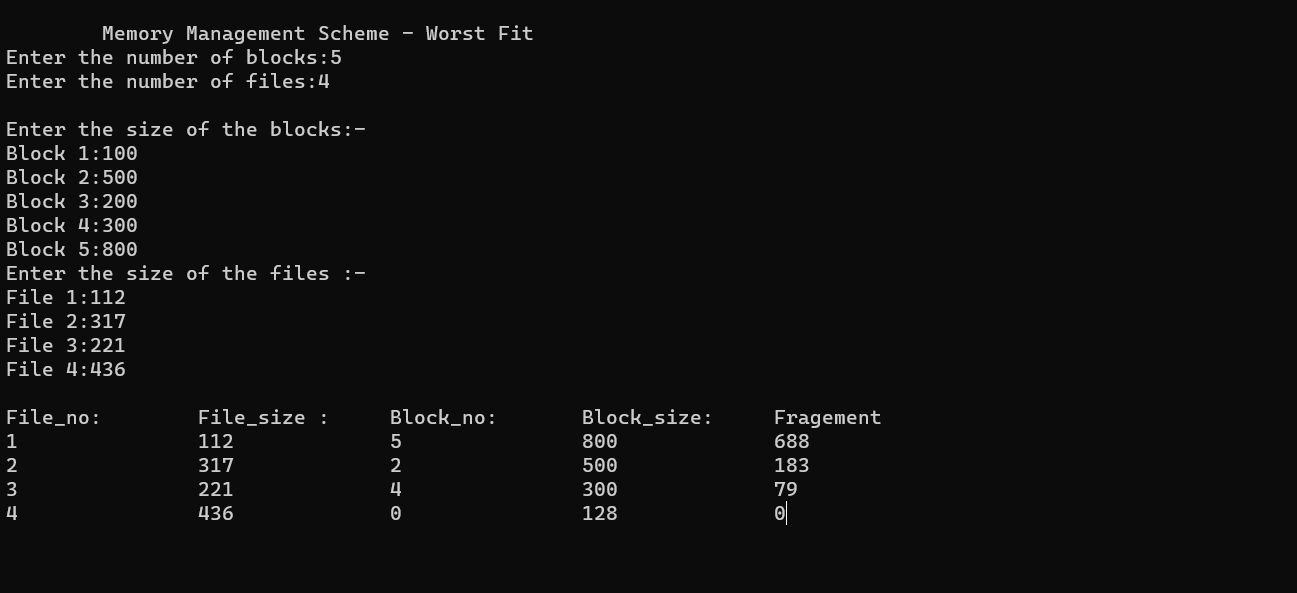
**printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");**

**for(i=1;i<=nf;i++)**

**printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);**

**getch();**

**}**



**14.FIRST IN FIRST OUT PAGE REPLACEMENT**

**CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**int main()**

**{**

**int incomingStream[] = {4 , 1 , 2 , 4 , 5};**

**int pageFaults = 0;**

**int frames = 3;**

**int m, n, s, pages;**

**pages = sizeof(incomingStream)/sizeof(incomingStream[0]);**

**printf(" Incoming \t\t Frame 1 \t\t Frame 2 \t\t Frame 3 ");**

**int temp[ frames ];**

**for(m = 0; m < frames; m++)**

**{**

**temp[m] = -1;**

**}**

**for(m = 0; m < pages; m++)**

**{**

**s = 0;**

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

**{**

**if(incomingStream[m] == temp[n])**

**{**

**s++;**

**pageFaults--;**

**}**

**}**

**pageFaults++;**

**if((pageFaults <= frames) && (s == 0))**

**{**

**temp[m] = incomingStream[m];**

**}**

**else if(s == 0)**

**{**

**temp[(pageFaults - 1) % frames] = incomingStream[m];**

**}**

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

**printf("%d\t\t\t",incomingStream[m]);**

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

**{**

**if(temp[n] != -1)**

**printf(" %d\t\t\t", temp[n]);**

**else**

**printf(" - \t\t\t");**

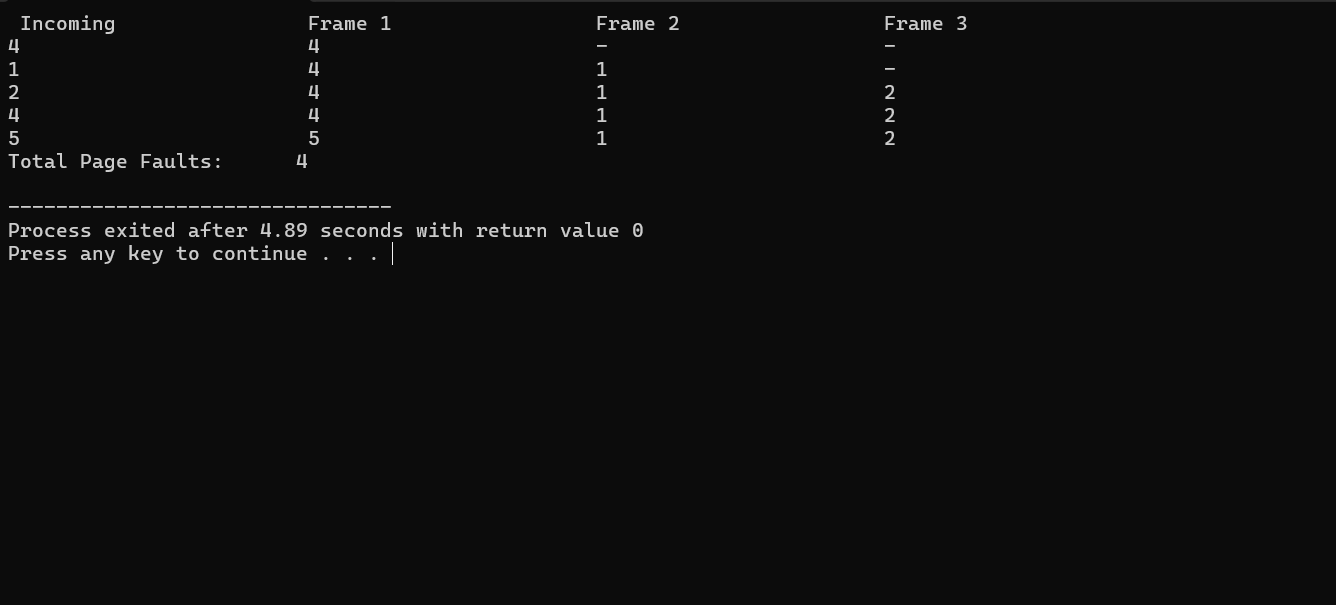
**}**

**}**

**printf("\nTotal Page Faults:\t%d\n", pageFaults);**

**return 0;**

**}**



**15.LEAST RECENTLY USED PAGE REPLACEMENT ALGORITHM**

**CODE:**

**#include<stdio.h>**

**main()**

**{**

**int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];**

**printf("Enter no of pages:");**

**scanf("%d",&n);**

**printf("Enter the reference string:");**

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

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

**printf("Enter no of frames:");**

**scanf("%d",&f);**

**q[k]=p[k];**

**printf("\n\t%d\n",q[k]);**

**c++;**

**k++;**

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

**{**

**c1=0;**

**for(j=0;j<f;j++)**

**{**

**if(p[i]!=q[j])**

**c1++;**

**}**

**if(c1==f)**

**{**

**c++;**

**if(k<f)**

**{**

**q[k]=p[i];**

**k++;**

**for(j=0;j<k;j++)**

**printf("\t%d",q[j]);**

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

**}**

**else**

**{**

**for(r=0;r<f;r++)**

**{**

**c2[r]=0;**

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

**{**

**if(q[r]!=p[j])**

**c2[r]++;**

**else**

**break;**

**}**

**}**

**for(r=0;r<f;r++)**

**b[r]=c2[r];**

**for(r=0;r<f;r++)**

**{**

**for(j=r;j<f;j++)**

**{**

**if(b[r]<b[j])**

**{**

**t=b[r];**

**b[r]=b[j];**

**b[j]=t;**

**}**

**}**

**}**

**for(r=0;r<f;r++)**

**{**

**if(c2[r]==b[0])**

**q[r]=p[i];**

**printf("\t%d",q[r]);**

**}**

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

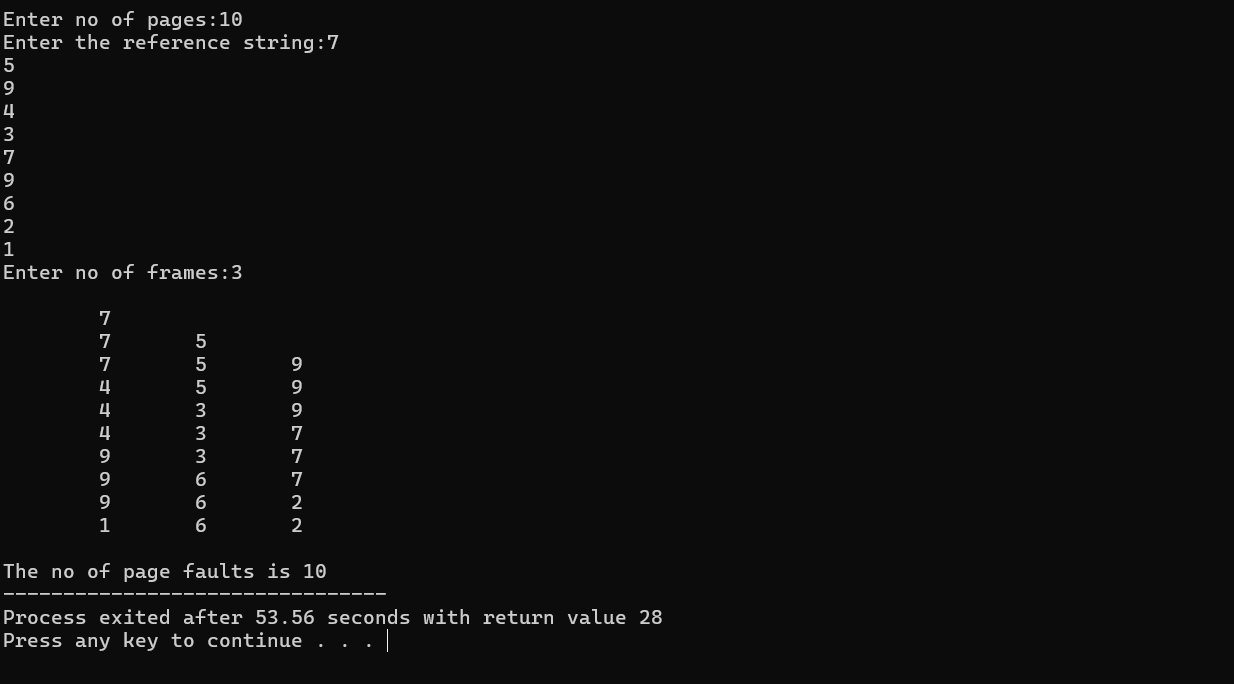
**}**

**}**

**}**

**printf("\nThe no of page faults is %d",c);**

**}**



**16.OPTIMAL PAGE REPLACEMENT ALGORITHM**

**CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**main()**

**{**

**int i,j,k,l,m,n,p,c=0,s;**

**int a[20],b[20],q,max;**

**printf("enter no. of reference string: ");**

**scanf("%d",&n);**

**printf("enter size of frame: ");**

**scanf("%d",&m);**

**printf("enter the elements of ref. string: \n");**

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

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

**for(j=0; j<m; j++)**

**b[j]=-1; //initialize all frame elements with -1**

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

**{**

**for(k=0; k<m; k++)**

**if(b[k]==a[i])**

**goto here;**

**for(j=0; j<m; j++)**

**{**

**if(b[j]==-1)//check if element already present in frame,if true then no page fault.**

**{**

**b[j]=a[i];**

**c++;**

**goto here;**

**}**

**}**

**if(j==m)**

**{**

**l=i+1,max=0;**

**for(j=0; j<m; j++)**

**{**

**for(s=l; s<n; s++)**

**{**

**if(a[s]==b[j])**

**{**

**if(s>max)**

**{**

**max=s;**

**p=j;**

**}**

**break;**

**}**

**}**

**if(s==n)**

**{**

**max=s;**

**p=j;**

**}**

**}**

**}**

**b[p]=a[i];**

**c++;**

**here:**

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

**for(k=0; k<m; k++)**

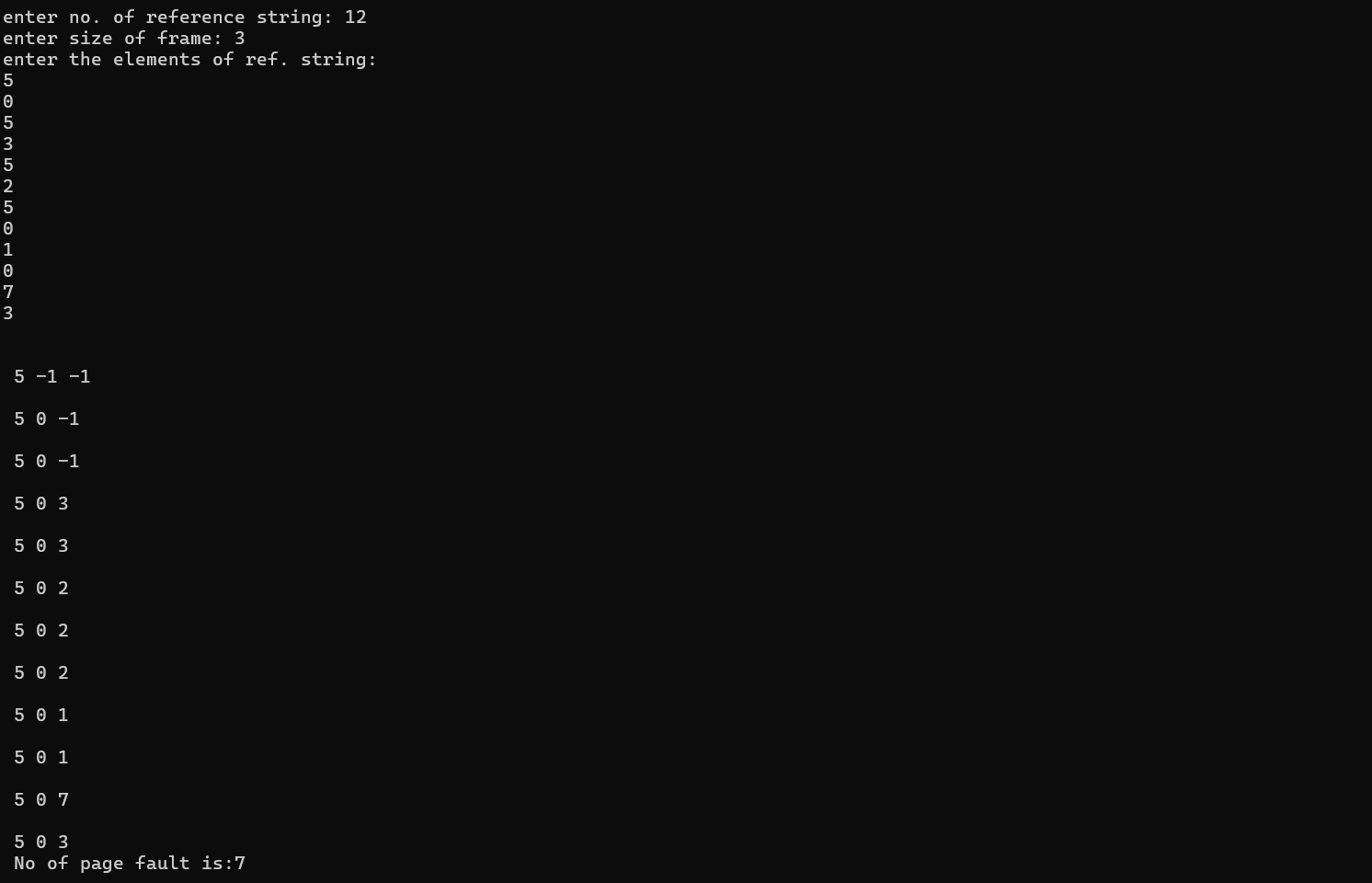
**printf(" %d",b[k]);**

**}**

**printf("\n No of page fault is:%d",c);**

**getch();**

**}**



**17. Write a Shell program to swap the two integers**

echo "Enter Two Numbers"

read a b temp=$a a=$b b=$temp

echo "after swapping" echo $a $b

**18. Write a Shell program to find the factorial of a number**

echo "Enter a Number"

read n

i=`expr $n - 1`

p=1

while [ $i -ge 1 ]

do

n=`expr $n \\* $i`

i=`expr $i - 1`

done

echo "The Factorial of the given Number is $n"

**19. Write a Shell program to check the given year is leap year or not.**

echo "Enter the year"

read y

b=`expr $y % 4`

if [ $b -eq 0 ]

then

echo "$y is a leap year"

else

echo "$y is not a leap year"

fi

**20. Write a Shell program to check the given number is even or odD.**

echo "Enter the Number"

read n

r=`expr $n % 2`

if [ $r -eq 0 ]

then

echo "$n is Even number"

else

echo "$n is Odd number"

fi