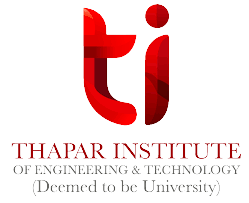
UCT-401

OPERATING SYSTEMS

Submitted To: Dr.Rupali Bhardwaj



By: Divija 102018056 CSBS-3

**1.Basic Codes**

**1.1 FACTORIAL**

echo " enter a number:"

read num

fact=1

while [ $num -gt 1 ]

do

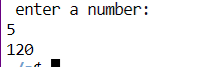
fact=`expr $num \\* $fact`

num=`expr $num - 1`

done

echo $fact

Output:



**1.2 PRIME NUMBER**

echo "enter a number:"

read num

for((i=2; i<=num/2; i++))

do

if [ `expr $num \% $i` -eq 0 ]

then

echo "$num is not a prime number."

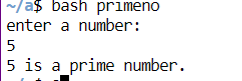
exit

fi

done

echo "$num is a prime number."

Output:



**1.3 EVEN ODD**

echo "enter a number"

read a

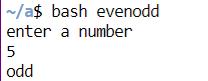
if [ `expr $a \% 2` -eq 0 ]

then echo "even"

else echo "odd"

fi

output:



**1.4 SOME OF SERIES**

echo " enter a number:"

read num

sum=0

while [ $num -gt 0 ]

do

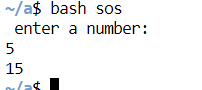
sum=`expr $num + $sum`

num=`expr $num - 1`

done

echo $sum

Output:



**1.5 FIBONACCI**

echo "enter a number:"

read N

a=0

b=1

echo "The Fibonacci series is : "

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

do

echo -n "$a "

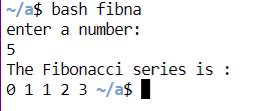
ans=`expr $a + $b`

a=$b

b=$ans

done

Output:



**2.FORK WITHOUT GETPID()**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

//pid\_t is just a data type like int char etc

pid\_t p;

printf("before fork\n");

//fork is used to create a new process

p=fork();

fork();

if(p<0)

{

printf("Error");

}

else if(p==0)//child process

{

printf("child");

}

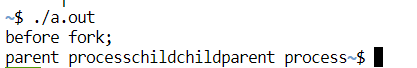
else //p>0 parent process

{ wait(NULL);

printf("parent process");

}

Output:



**3.FORK WITH GETPID()**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

//pid\_t is just a data type like int char etc

pid\_t p;

printf("before fork\n");

//fork is used to create a new process

p=fork();

if(p<0)

{

printf("Error");

}

else if(p==0)//child process

{

//getpid() prints the process id of any process

//getppid prints the process id of parent

printf("I am child having id %d\n",getpid());

printf("My parent's id is %d\n",getppid());

}

else //p>0 parent process

{

//if parent wants to print the process id of child toh

//usko print using p(jaha fork use kra)

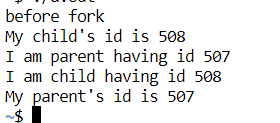
printf("My child's id is %d\n",p);

printf("I am parent having id %d\n",getpid());

}

}

Output:



**4.RACE CONDITION**

#include<pthread.h>

#include<stdio.h>

#include<unistd.h>

void \*fun1();

void \*fun2();

//shared variable

int shared = 5;

int main(){

pthread\_t thread1 , thread2 ;

pthread\_create(&thread1 , NULL , fun1 , NULL);

pthread\_create(&thread2 , NULL , fun2 , NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

printf("final value of shared is %d\n" , shared);

}

void \*fun1()

{

int x ;

x = shared;

printf("thread1 reads the value of shared variable as %d\n" , x);

x++;

printf("Local updation by thread1 : %d\n", x);

sleep(1);

shared = x;

printf("value of shared variable updated by thread1 is : %d\n" , shared);

}

void \*fun2()

{

int y ;

y = shared;

printf("thread2 reads the value of shared variable as %d\n" , y);

y--;

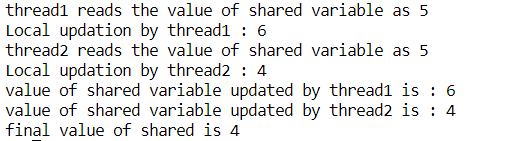
printf("Local updation by thread2 : %d\n", y);

sleep(1);

shared = y;

printf("value of shared variable updated by thread2 is : %d\n" , shared);

}



**5.SEMAPHORE FOR RACE CONDITION**

#include<stdio.h>

#include<semaphore.h>

#include<unistd.h>

#include<pthread.h>

void \*p1(void \*argv);

void \*p2(void \*argv);

int shared = 5;

sem\_t s;

int main(){

pthread\_t thread1;

pthread\_t thread2;

//The value of the initialised semaphore here is 1.

sem\_init(&s,0,1);

pthread\_create(&thread1,NULL,p1,NULL);

pthread\_create(&thread2,NULL,p2,NULL);

pthread\_join(thread1,NULL);

pthread\_join(thread2,NULL);

printf("shared = %d\n",shared);

}

void \*p1(void \*argv)

{

sem\_wait(&s);

int x=shared;

x++;

printf("P1 in critical section\n");

sleep(1);

shared =x;

printf("shread variable updation in function 1 = %d\n",shared);

//sem\_post() function unlocks the semaphore

printf("P1 is out of the critical section\n");

sem\_post(&s);

}

void \*p2(void \*argv)

{

sem\_wait(&s);

int y=shared;

y--;

printf("P2 in critical section\n");

sleep(1);

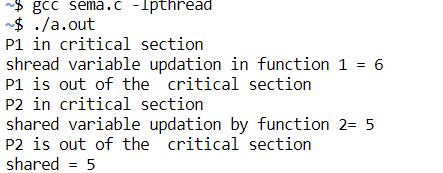
shared=y;

printf("shared variable updation by function 2= %d\n",shared);

printf("P2 is out of the critical section\n");

sem\_post(&s);

}



**6.READER WRITER**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt\*2;

printf("Writer %d modified cnt to %d\n",(\*((int \*)wno)),cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1) {

sem\_wait(&wrt); // If this id the first reader, then it will block the writer

}

pthread\_mutex\_unlock(&mutex);

// Reading Section

printf("Reader %d: read cnt as %d\n",\*((int \*)rno),cnt);

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader--;

if(numreader == 0) {

sem\_post(&wrt); // If this is the last reader, it will wake up the writer.

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10],write[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&wrt,0,1);

int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer

for(int i = 0; i < 10; i++) {

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

for(int i = 0; i < 10; i++) {

pthread\_join(read[i], NULL);

}

for(int i = 0; i < 5; i++) {

pthread\_join(write[i], NULL);

}

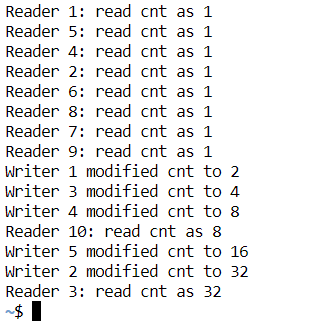
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

Output:



**7.DINNING PHILOSPHER**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t chopstick[5];

void \* philos(void \*);

void eat(int);

int main()

{

int i,n[5];

pthread\_t T[5];

for(i=0;i<5;i++)

sem\_init(&chopstick[i],0,1);

for(i=0;i<5;i++){

n[i]=i;

pthread\_create(&T[i],NULL,philos,(void \*)&n[i]);

}

for(i=0;i<5;i++)

pthread\_join(T[i],NULL);

}

void \* philos(void \* n)

{

int ph=\*(int \*)n;

printf("Philosopher %d wants to eat\n",ph);

printf("Philosopher %d tries to pick left chopstick\n",ph);

sem\_wait(&chopstick[ph]);

printf("Philosopher %d picks the left chopstick\n",ph);

printf("Philosopher %d tries to pick the right chopstick\n",ph);

sem\_wait(&chopstick[(ph+1)%5]);

printf("Philosopher %d picks the right chopstick\n",ph);

eat(ph);

sleep(2);

printf("Philosopher %d has finished eating\n",ph);

sem\_post(&chopstick[(ph+1)%5]);

printf("Philosopher %d leaves the right chopstick\n",ph);

sem\_post(&chopstick[ph]);

printf("Philosopher %d leaves the left chopstick\n",ph);

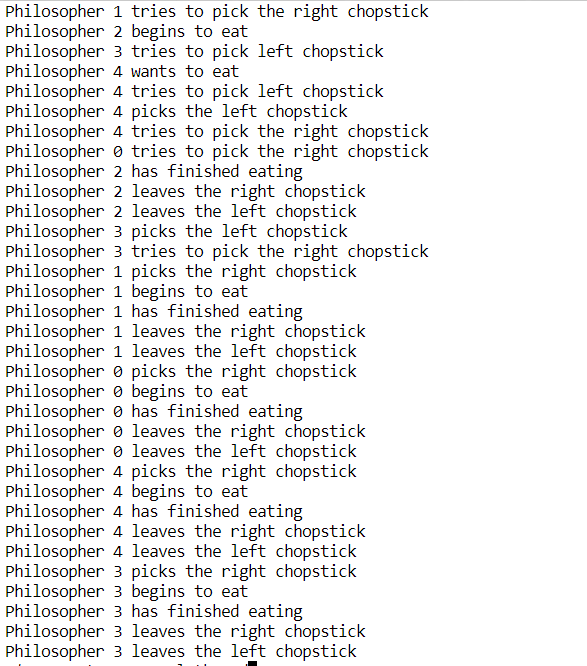
}

void eat(int ph)

{

printf("Philosopher %d begins to eat\n",ph);

}



**8.PRODUCER CONSUMER**

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume

#define BufferSize 5 // Size of the buffer

sem\_t empty;

sem\_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*pno)

{

int item;

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

item = rand(); // Produce an random item

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno),buffer[in],in);

in = (in+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*cno)

{

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

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

printf("Consumer %d: Remove Item %d from %d\n",\*((int \*)cno),item, out);

out = (out+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main()

{

pthread\_t pro[5],con[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&empty,0,BufferSize);

sem\_init(&full,0,0);

int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

for(int i = 0; i < 5; i++) {

pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_join(pro[i], NULL);

}

for(int i = 0; i < 5; i++) {

pthread\_join(con[i], NULL);

}

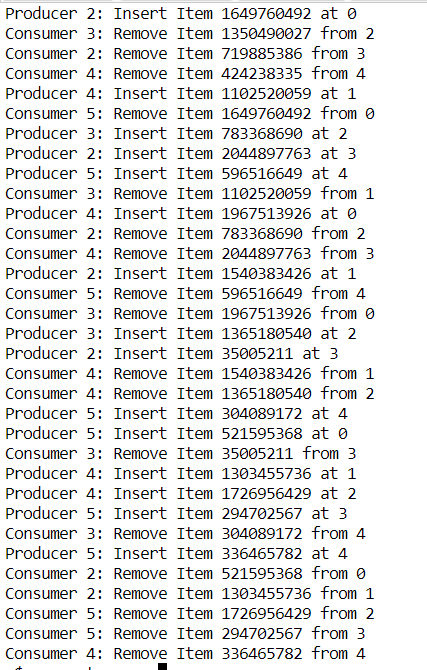
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}



**9.PETERSON**

#include<stdio.h>

#include<pthread.h>

void \*fun1();

void \*fun2();

#define TRUE 1

#define FALSE 0

int flag[2] ={FALSE , FALSE};

int turn=0;

int i = 0 ;

int j = 1;

int main()

{

pthread\_t t1,t2;

pthread\_create(&t1,NULL,fun1,NULL);

pthread\_create(&t2,NULL,fun2,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

// printf("final shared= %d\n",shared);`

}

void \*fun1()

{

flag[i] = TRUE;

turn = j;

printf("process 1 tries to enter\n");

while(flag[j] == TRUE && turn == j);

printf("process 1 is in critical sectio\n");

sleep(2);

flag[i] = FALSE;

printf("process 1 is out\n");

}

void \*fun2()

{

flag[j] = TRUE;

turn = i;

printf("process 2 tries to enter\n");

while(flag[i] == TRUE && turn == i);

printf("process 2 is in critical section\n");

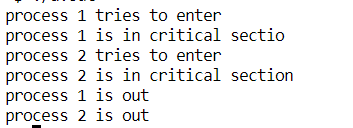
sleep(1);

flag[j] = FALSE;

printf("process 2 is out\n");

}

Output:



**10.FIRST COME FIRST SERVER**

#include <conio.h>

#include <stdio.h>

// computation of waiting time

void waitingTime(int p[], int n, int burst\_time[], int arrival\_time[])

{

int waiting[10], cpu;

float avt = 0;

waiting[0] = 0;

cpu = arrival\_time[0];

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

{

cpu += burst\_time[i - 1];

if (arrival\_time[i] > cpu)

{

waiting[i] = arrival\_time[i] - cpu;

}

else

{

waiting[i] = 0;

}

}

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

{

avt = avt + waiting[i];

printf("Waiting:%d", waiting[i]);

printf(" ");

}

avt = avt / n;

printf("\nAverage waiting time:%f", avt);

}

void completionTime(int p[], int n, int burst\_time[], int arrival\_time[])

{

int completion[10];

float avc = 0;

completion[0] = burst\_time[0] + arrival\_time[0];

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

{

if (completion[i - 1] < arrival\_time[i])

{

completion[i] = burst\_time[i] + arrival\_time[i];

}

else

{

completion[i] = completion[i - 1] + burst\_time[i];

}

}

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

{

avc = avc + completion[i];

printf("Completion:%d", completion[i]);

printf(" ");

}

avc = avc / n;

printf("\nAverage completion time:%f", avc);

}

void fcfs(int p[], int n, int burst\_time[], int arrival\_time[])

{

waitingTime(p, n, burst\_time, arrival\_time);

completionTime(p, n, burst\_time, arrival\_time);

}

int main()

{

int p[] = {1, 2, 3, 4};

int burst\_time[] = {4,

2,

3,

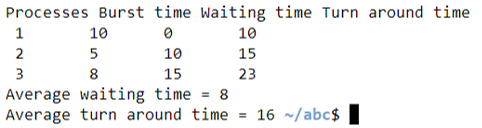
1};

int arrival\_time[] = {2, 7, 8, 9};

int n = 4;

fcfs(p, n, burst\_time, arrival\_time);

}

Output: 

**11.SHORTEST JOB FIRST**

#include <bits/stdc++.h>

using namespace std;

void swap(int x, int y){

int temp=x;

x=y;

y=x;

}

void waitingTime(int wt[], int bt[], int at[], int n){

wt[0]=0;

int sum=0;

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

sum+=bt[i-1];

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

}

}

void turnaroundTime(int tat[], int bt[], int wt[], int n){

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

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

}

int main(){

int i,j,k=1,n=3;

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

int bt[]={5,1,2};

int at[]={0,1,2};

int wt[n],tat[n];

double wtsum=0,tatsum=0;

//sort by arrival time:

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

{

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

{

if(bt[i]>bt[j])

{

swap(at[i],at[j]);

swap(bt[i],bt[j]);

swap(p[i], p[j]);

}

}

}

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

int b=0;

b+=bt[i];

int min=bt[k];

for(j=k; j<n; j++){

if(b>at[i] && bt[i]<min){

swap(p[j],p[k]);

swap(at[j],at[k]);

swap(bt[j],bt[k]);

}

}

k++;

}

waitingTime(wt,bt,at,n);

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

wtsum+=wt[i];

turnaroundTime(tat,bt,wt,n);

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

tatsum+=tat[i];

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

cout<<endl<<i+1<<". Process p"<<p[i]<<endl;

cout<<"waiting time "<<wt[i];

cout<<"\nturn around time "<<tat[i]<<endl;

}

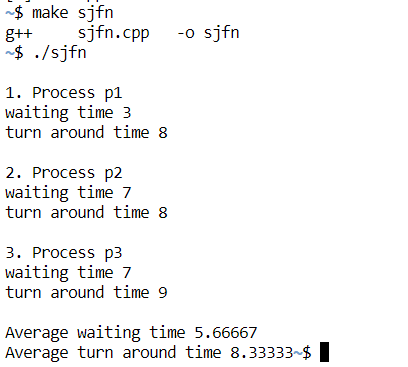
cout<<"\nAverage waiting time "<<wtsum/n;

cout<<"\nAverage turn around time "<<tatsum/n;

return 0;

}

Output:



Preemptive

#include <bits/stdc++.h>

using namespace std;

void waiting\_time(int wt[], int bt[], int at[], int n){

int rt[n];

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

rt[i]=bt[i];

int p=0, t=0, flag=0, x=0, m=INT\_MAX, finish\_time;

// Process until all processes gets completed

while(p!=n){

// Find process with minimum remaining time among the

//processes that arrives till the current time

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

if(at[i]<=t && rt[i]<m && rt[i]>0){

m=rt[i];

x=i;

flag=1;

}

}

if(flag==0){

t++; continue;

}

//reduce remaining time, update minumum

rt[x]--;

m=rt[x];

if(m==0) m=INT\_MAX;

if(rt[x]==0){

p++; flag=0;

finish\_time=t+1;

//waiting time:

wt[x]=finish\_time-bt[x]-at[x];

if(wt[x]<0) wt[x]=0;

}

t++;

}

}

void turn\_around\_time(int wt[], int bt[], int tat[], int n){

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

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

}

int main(){

int i, n=3;

int bt[n] = {5, 1, 2};

int at[n] = {0, 1, 2};

int wt[n], tat[n];

float total\_wt=0, total\_tat=0;

waiting\_time(wt, bt, at, n);

turn\_around\_time(wt, bt, tat, n);

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

cout<<"\nProcess "<<i+1<<endl;

cout<<"Waiting time: "<<wt[i]<<endl;

cout<<"Turn around time: "<<tat[i]<<endl;

total\_wt+=wt[i];

total\_tat+=tat[i];

}

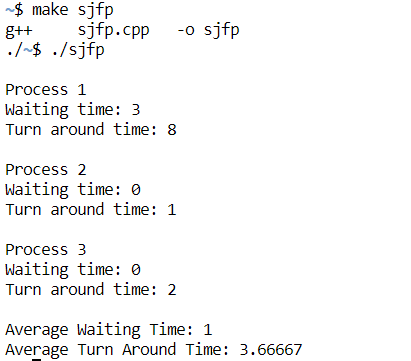
cout<<"\nAverage Waiting Time: "<<total\_wt/n<<endl;

cout<<"Average Turn Around Time: "<<total\_tat/n<<endl;

return 0;

}

Output:



**12.PRIORITY**

#include<stdio.h>

void priorityOrdering(int processes[],int bt[], int priority[],int new\_bt[], int n){

int order = 0;

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

order = priority[i];

processes[i] = order;

new\_bt[i] = bt[order];

}

}

void waitingTime(int processes[],int bt[],int wt[],int priority[],int new\_bt[],int n){

priorityOrdering(processes,bt,priority,new\_bt,n);

wt[0] = 0;

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

wt[i] = wt[i-1] + new\_bt[i-1];

}

}

void turnAroundTime(int processes[], int bt[], int wt[], int tat[],int priority[],int new\_bt[], int n){

priorityOrdering(processes,bt,priority,new\_bt,n);

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

tat[i] = new\_bt[i] + wt[i];

}

}

void findAverage(int processes[],int bt[], int wt[], int tat[],int priority[],int new\_bt[], int n){

int total\_wt = 0;

int total\_tat = 0;

waitingTime(processes, bt, wt, priority, new\_bt, n);

turnAroundTime(processes, bt, wt, tat, priority, new\_bt,n);

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

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

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

}

printf("\nProcesses\tBurst Time\tWaiting Time\tTurn Around Time\n");

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

printf("\nP%d\t\t%d\t\t%d\t\t%d",processes[i],new\_bt[i],wt[i],tat[i]);

}

printf("\nAverage Waiting Time = %f",(float)total\_wt/n);

printf("\nAverage Turn Around Time = %f",(float)total\_tat/n);

}

void main(){

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

int bt[] = {10,5,8};

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

int new\_bt[3];

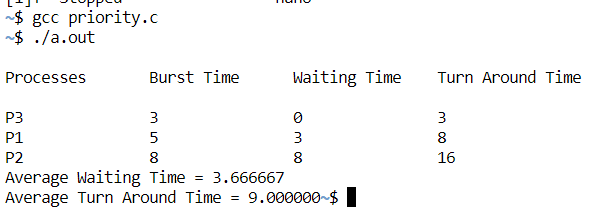
int n = sizeof(processes)/sizeof(processes[0]);

int wt[n], tat[n];

findAverage(processes, bt, wt, tat, priority, new\_bt, n);

}

Output:



**13.ROUND ROBIN**

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

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

{

printf("Arrival Time and Burst Time for Process %d :",count+1);

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

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

rt[count]=bt[count];

}

printf("Enter the time Quantum:\t");

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

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

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

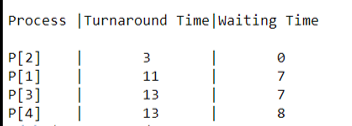
printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

Output:



**14.DEADLOCK**

**14.1BANKER’S SAFETY ALGORITHM**

#include<iostream>

using namespace std;

int main(){

//n = number of processes

int n=5;

//m = number of resources

int m=3;

//total resources

int total[3]={10,5,7};

int available[3]={3,3,2};

// copy of available to make changes when the resources are released

int work[3]={3,3,2};

//chaiye kitne hai ek particular resource ko

int max[5][3] = { { 7, 5, 3 }, { 3, 2, 2 }, { 9, 0, 2 }, { 2, 2, 2 }, { 4, 3, 3 } }; // max matrix for P0,P1,P2,P3

//already kitne given hai

int alloc[5][3] = { { 0, 1, 0 }, { 2, 0, 0 },{ 3, 0, 2 },{ 2, 1, 1 },{ 0, 0, 2 } }; //allocation matrix for P0,P1,P2,P3

//need = max-alloc

int need[n][m];

//step 1:

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

{

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

{

need[i][j]=max[i][j]-alloc[i][j];

}

}

bool finish[n]={0};

int safeseq[n];

int count =0;

while(count<n){

bool found =false;

//step 2:

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

{

if(finish[i]==false)

{

int a=0;

while(a<m)

{

//agr need jyada hai process ki toh ageh chlo

if(need[i][a]>work[a])

{

break;

}

a++;

}

//step : 3

if(a==m)

{

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

{

work[k]+=alloc[i][k];

}

//step 4:

finish[i]=1;

found=true;

safeseq[count]=i;

count++;

}

}

}

if (found == false)

{

cout<< "System is not in safe state";

return 0;

}

}

cout << "System is in safe state.\nSafe sequence is: ";

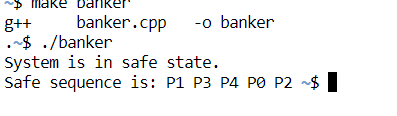
for (int i = 0; i < n ; i++)

cout <<"P"<<safeseq[i]<< " ";

return 0;

}

**Output:**

****

**14.2RESOURCE ALLOCATION**

#include<stdio.h>  
#include<conio.h>  
  
int proc,res,i,j,row=0,flag=0;  
static int pro[3][3],req[3][3],st\_req[3][3],st\_pro[3][3];  
  
void main()  
{  
 clrscr();  
  
 printf("\nEnter the number of Processes:");  
 scanf("%d",&proc);  
 printf("\nEnter the number of Resources:");  
 scanf("%d",&res);  
  
 printf("\nEnter the Process Matrix:");  
 for(i=0;i<proc;i++)  
  for(j=0;j<res;j++)  
 scanf("%d",&pro[i][j]);  
  
 printf("\nEnter the Request Matrix:");  
 for(i=0;i<res;i++)  
  for(j=0;j<proc;j++)  
 scanf("%d",&req[i][j]);  
  
 row=0;  
 while(!kbhit())  
 {  
  for(i=0;i<res;i++)  
  {  
   if(pro[row][i]==1)  
 {  
 if(st\_pro[row][i]>1 && flag==1)  
 {  
   printf("\nDeadlock Occured");  
   getch();  
   exit(0);  
 }  
 st\_pro[row][i]++;  
 row=i;  
 break;  
 }  
  }  
  
  for(i=0;i<proc;i++)  
  {  
   if(req[row][i]==1)  
 {  
 if(st\_req[row][i]>1)  
 {  
   printf("\nDeadlock Occured");  
   getch();  
   exit(0);  
 }  
 st\_req[row][i]++;  
 row=i;  
 flag=1;  
 break;  
 }  
  }  
 }  
 cprintf("\nNo Deadlock Detected");  
 getch();  
 getch();  
}

