UCT-401

OPERATING SYSTEMS

Submitted To: Dr.Rupali Bhardwaj

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

By: Divija 102018056 CSBS-3

Index

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:

```
 enter a number:
5
120
```

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:

```
~/a$ bash primeno
enter a number:
5
5 is a prime number.
```

1.3 EVEN ODD

```
echo "enter a number"
read a
if [ `expr $a \% 2` -eq 0 ]
then echo "even"
else echo "odd"
fi
```

output:

```
~/a$ bash evenodd
enter a number
5
odd
```

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:

```
~/a$ bash sos
 enter a number:
5
15
~/a$
```

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:

```
~/a$ bash fibna
enter a number:
5
The Fibonacci series is :
0 1 1 2 3 ~/a$
```

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:

```
~$ ./a.out
before fork;
parent processchildchildparent process~$
```

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:

```
before fork
My child's id is 508
I am parent having id 507
I am child having id 508
My parent's id is 507
~$
```

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);
}
```

```
thread1 reads the value of shared variable as 5
Local updation by thread1 : 6
thread2 reads the value of shared variable as 5
Local updation by thread2 : 4
value of shared variable updated by thread1 is : 6
value of shared variable updated by thread2 is : 4
final value of shared is 4
```

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);
}
```

```
~$ gcc sema.c -lpthread
~$ ./a.out
P1 in critical section
shread variable updation in function 1 = 6
P1 is out of the  critical section
P2 in critical section
shared variable updation by function 2= 5
P2 is out of the  critical section
shared = 5
```

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:

```
Reader 1: read cnt as 1
Reader 5: read cnt as 1
Reader 4: read cnt as 1
Reader 2: read cnt as 1
Reader 6: read cnt as 1
Reader 8: read cnt as 1
Reader 7: read cnt as 1
Reader 9: read cnt as 1
Writer 1 modified cnt to 2
Writer 3 modified cnt to 4
Writer 4 modified cnt to 8
Reader 10: read cnt as 8
Writer 5 modified cnt to 16
Writer 2 modified cnt to 32
Reader 3: read cnt as 32
~$
```

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);

}
```

```
Philosopher 1 tries to pick the right chopstick
Philosopher 2 begins to eat
Philosopher 3 tries to pick left chopstick
Philosopher 4 wants to eat
Philosopher 4 tries to pick left chopstick
Philosopher 4 picks the left chopstick
Philosopher 4 tries to pick the right chopstick
Philosopher 0 tries to pick the right chopstick
Philosopher 2 has finished eating
Philosopher 2 leaves the right chopstick
Philosopher 2 leaves the left chopstick
Philosopher 3 picks the left chopstick
Philosopher 3 tries to pick the right chopstick
Philosopher 1 picks the right chopstick
Philosopher 1 begins to eat
Philosopher 1 has finished eating
Philosopher 1 leaves the right chopstick
Philosopher 1 leaves the left chopstick
Philosopher 0 picks the right chopstick
Philosopher 0 begins to eat
Philosopher 0 has finished eating
Philosopher 0 leaves the right chopstick
Philosopher 0 leaves the left chopstick
Philosopher 4 picks the right chopstick
Philosopher 4 begins to eat
Philosopher 4 has finished eating
Philosopher 4 leaves the right chopstick
Philosopher 4 leaves the left chopstick
Philosopher 3 picks the right chopstick
Philosopher 3 begins to eat
Philosopher 3 has finished eating
Philosopher 3 leaves the right chopstick
Philosopher 3 leaves the left chopstick
```

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;

}
```

```
Producer 2: Insert Item 1649760492 at 0
Consumer 3: Remove Item 1350490027 from 2
Consumer 2: Remove Item 719885386 from 3
Consumer 4: Remove Item 424238335 from 4
Producer 4: Insert Item 1102520059 at 1
Consumer 5: Remove Item 1649760492 from 0
Producer 3: Insert Item 783368690 at 2
Producer 2: Insert Item 2044897763 at 3
Producer 5: Insert Item 596516649 at 4
Consumer 3: Remove Item 1102520059 from 1
Producer 4: Insert Item 1967513926 at 0
Consumer 2: Remove Item 783368690 from 2
Consumer 4: Remove Item 2044897763 from 3
Producer 2: Insert Item 1540383426 at 1
Consumer 5: Remove Item 596516649 from 4
Consumer 3: Remove Item 1967513926 from 0
Producer 3: Insert Item 1365180540 at 2
Producer 2: Insert Item 35005211 at 3
Consumer 4: Remove Item 1540383426 from 1
Consumer 4: Remove Item 1365180540 from 2
Producer 5: Insert Item 304089172 at 4
Producer 5: Insert Item 521595368 at 0
Consumer 3: Remove Item 35005211 from 3
Producer 4: Insert Item 1303455736 at 1
Producer 4: Insert Item 1726956429 at 2
Producer 5: Insert Item 294702567 at 3
Consumer 3: Remove Item 304089172 from 4
Producer 5: Insert Item 336465782 at 4
Consumer 2: Remove Item 521595368 from 0
Consumer 2: Remove Item 1303455736 from 1
Consumer 5: Remove Item 1726956429 from 2
Consumer 5: Remove Item 294702567 from 3
Consumer 4: Remove Item 336465782 from 4
```

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:

```
process 1 tries to enter
process 1 is in critical sectio
process 2 tries to enter
process 2 is in critical section
process 1 is out
process 2 is out
```

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:

```
Processes Burst time Waiting time Turn around time
 1        10         0            10
 2        5          10           15
 3        8          15           23
Average waiting time = 8
Average turn around time = 16 ~/abc$
```

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:

```
~$ make sjfn
g++     sjfn.cpp   -o sjfn
~$ ./sjfn

1. Process p1
waiting time 3
turn around time 8

2. Process p2
waiting time 7
turn around time 8

3. Process p3
waiting time 7
turn around time 9

Average waiting time 5.66667
Average turn around time 8.33333~$
```

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:

```
~$ make sjfp
g++     sjfp.cpp   -o sjfp
./~$ ./sjfp

Process 1
Waiting time: 3
Turn around time: 8

Process 2
Waiting time: 0
Turn around time: 1

Process 3
Waiting time: 0
Turn around time: 2

Average Waiting Time: 1
Average Turn Around Time: 3.66667
```

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:

```
~$ gcc priority.c
~$ ./a.out

Processes       Burst Time      Waiting Time    Turn Around Time

P3              3               0               3
P1              5               3               8
P2              8               8               16
Average Waiting Time = 3.666667
Average Turn Around Time = 9.000000~$
```

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:

```
Process |Turnaround Time|Waiting Time

P[2]    |        3       |        0
P[1]    |        11      |        7
P[3]    |        13      |        7
P[4]    |        13      |        8
```

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:

```
~$ make banker
g++     banker.cpp    -o banker
.~$ ./banker
System is in safe state.
Safe sequence is: P1 P3 P4 P0 P2 ~$
```

14.2RESOURCE ALLOCATION

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

//Resource allocation algorithm when P1 makes the requests
//we need to satisfy three conditions
//req i <= need i
//req i <= available i
//check safety
void res_alloc(int available[],int m,int need[][3],int alloc[][3]){
    int req[1][3]={1,0,2};
    for(int i=0;i<m;i++){
        if(req[0][i]>need[1][i]){
            cout<<"error encountered"<<endl;
            exit(0);
        }
    }
    for(int i=0;i<m;i++){
```

```
        if(req[0][i]>available[i]){
            cout<<"Resources unavailable"<<endl;
            exit(0);
        }
    }
    for(int i=0;i<m;i++){
        available[i]-=req[0][i];
        alloc[1][i]+=req[0][i];
        need[1][i]-=req[0][i];
    }
}

int main(){
    int n=5;
    int m=3;

    int total[3]={10,5,7};//total instances of each resource type
    int available[3]={3,3,2};//available instances

    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

    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

    int need[5][3];

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

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

//applying resource allocation algo for p1
res_alloc(available,3,need,alloc);

int work[3];
for(int i=0;i<m;i++){
    work[i]=available[i];
}

bool finish[n]={0};
int safeseq[n];

int count =0;
while(count<n){
    bool found =false;

    for(int i=0;i<n;i++){
        if(finish[i]==false){
            int a=0;
            while(a<m)
            {
                if(need[i][a]>work[a]){
                break;}
                a++;
            }
            if(a==m){
                for(int k=0;k<m;k++){
                    work[k]+=alloc[i][k];
```

```
            }
            finish[i]=1;
            found=true;
            safeseq[count]=i;
            count++;
          }
        }
      }
      if (found == false)
      {
         cout<< "System is not in safe state";
         cout<<"REQUEST CANNOT BE GRANTED"<<endl;
         return 0;
      }
   }

   cout << "System is in safe state.\nREQUEST CAN BE GRANTED\nSafe sequence is: ";
   for (int i = 0; i < n ; i++)
      cout <<"P"<<safeseq[i]<< " ";

   return 0;

}
```

Output:

```
~$ make res
g++     res.cpp   -o res
.~$ ./res
System is in safe state.
REQUEST CAN BE GRANTED
Safe sequence is: P1 P3 P4 P0 P2 ~$
```