DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS

FACULTY OF SCIENCE AND HUMANITIES

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

VADAPALANI CAMPUS

RECORD NOTE

NAME OF THE STUDENT :

REGISTER NUMBER :

NAME OF THE COURSE : MCA / Computer Applications

SEMESTER & YEAR : I & I

SUBJECT CODE : PCA20C0J

SUBJECT NAME : Operating System Laboratory

NOV/DEC – 2022

DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS

FACULTY OF SCIENCE AND HUMANITIES

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

VADAPALANI CAMPUS

CERTIFICATE

Certified to be the Bonafide record of Practical work done by

NAME OF THE STUDENT :

REGISTER NUMBER :

NAME OF THE COURSE : MCA / Computer Applications

SEMESTER & YEAR : I & I

SUBJECT CODE : PCA20C0J

SUBJECT NAME : Operating System Laboratory

In SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, DEPARTMENT OF COMPUTER APPLICATIONS Laboratory during the Academic Year2022-2024 and Submitted for M.C.A degree practical examination held on __/_/2022.

STAFF **HOD**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

CONTENT

S.NO	DATE	NAME OF THE EXERCISE	PAGE NO	SIGNATURE
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				

1. CPU SCHEDULING

A. FIRST COME FIRST SERVE (FCFS):

```
import java.util.*;

public class FCFS {
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
System.out.println("enter no of process: ");
int n = sc.nextInt();
int pid[] = new int[n];   // process ids
int ar[] = new int[n];     // arrival times
int bt[] = new int[n];     // burst or execution times
int ct[] = new int[n];     // completion times
int ta[] = new int[n];     // turn around times
int wt[] = new int[n];     // waiting times
int temp;
float avgwt=0,avgta=0;

for(int i = 0; i < n; i++)
{
System.out.println("enter process " + (i+1) + " arrival time: ");
ar[i] = sc.nextInt();
System.out.println("enter process " + (i+1) + " brust time: ");
bt[i] = sc.nextInt();
pid[i] = i+1;
}

//sorting according to arrival times
for(int i = 0 ; i <n; i++)
{
for(int  j=0;  j < n-(i+1) ; j++)
{
if( ar[j] > ar[j+1] )
{
temp = ar[j];
ar[j] = ar[j+1];
ar[j+1] = temp;
temp = bt[j];
bt[j] = bt[j+1];
bt[j+1] = temp;
temp = pid[j];
pid[j] = pid[j+1];
pid[j+1] = temp;
}
}
}
```

```
// finding completion times
for(int  i = 0 ; i < n; i++)
{
if( i == 0)
{
ct[i] = ar[i] + bt[i];
}
else
{
if( ar[i] > ct[i-1])
{
ct[i] = ar[i] + bt[i];
}
else
ct[i] = ct[i-1] + bt[i];
}
ta[i] = ct[i] - ar[i] ;          // turnaround time= completion time- arrival time
wt[i] = ta[i] - bt[i] ;          // waiting time= turnaround time- burst time
avgwt += wt[i] ;                 // total waiting time
avgta += ta[i] ;                // total turnaround time
}
System.out.println("\npid  arrival  brust  complete turn waiting");
for(int  i = 0 ; i< n;  i++)
{
System.out.println(pid[i] + "  \t " + ar[i] + "\t" + bt[i] + "\t" + ct[i] + "\t" + ta[i] + "\t"  + wt[i] ) ;
}
sc.close();
System.out.println("\naverage waiting time: "+ (avgwt/n));     // printing average waiting time.
System.out.println("average turnaround time:"+(avgta/n));    // printing average turnaround time.
}
}
```

OUTPUT

```
java -cp /tmp/bqX1eUxDmf FCFS
enter no of process: 3
enter process 1 arrival time:
0
enter process 1 burst time:
9
enter process 2 arrival time:
1
enter process 2 burst time:
4
enter process 3 arrival time:
2
enter process 3 burst time:
9
pid     arrival    burst   complete  turn     waiting
1          0         9         9         9         0
2          1         4         13        12        8
3          2         9         22        20        11
average waiting time: 6.3333335
average turnaround time:13.666667
```

B. SHORTEST JOB FIRST (PREEMPTIVE)

```
import java.util.*;

public class SRTF {
public static void main (String args[])
{
Scanner sc=new Scanner(System.in);
System.out.println ("enter no of process:");
int n= sc.nextInt();
int pid[] = new int[n]; // it takes pid of process
int at[] = new int[n]; // at means arrival time
int bt[] = new int[n]; // bt means burst time
int ct[] = new int[n]; // ct means complete time
int ta[] = new int[n];// ta means turn around time
int wt[] = new int[n];  // wt means waiting time
int f[] = new int[n];  // f means it is flag it checks process is completed or not
int k[]= new int[n];   // it is also stores brust time
    int i, st=0, tot=0;
    float avgwt=0, avgta=0;

    for (i=0;i<n;i++)
    {
     pid[i]= i+1;
     System.out.println ("enter process " +(i+1)+ " arrival time:");
     at[i]= sc.nextInt();
     System.out.println("enter process " +(i+1)+ " burst time:");
     bt[i]= sc.nextInt();
     k[i]= bt[i];
     f[i]= 0;
    }

    while(true){
     int min=99,c=n;
     if (tot==n)
     break;

     for ( i=0;i<n;i++)
     {
     if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))
     {
     min=bt[i];
     c=i;
     }
     }

     if (c==n)
     st++;
```

```
    else
    {
    bt[c]--;
    st++;
    if (bt[c]==0)
    {
    ct[c]= st;
    f[c]=1;
    tot++;
    }
    }
   }

   for(i=0;i<n;i++)
   {
    ta[i] = ct[i] - at[i];
    wt[i] = ta[i] - k[i];
    avgwt+= wt[i];
    avgta+= ta[i];
   }

   System.out.println("pid  arrival  burst  complete turn waiting");
   for(i=0;i<n;i++)
   {
    System.out.println(pid[i] +"\t"+ at[i]+"\t"+ k[i] +"\t"+ ct[i] +"\t"+ ta[i] +"\t"+ wt[i]);
   }

   System.out.println("\naverage tat is "+ (float)(avgta/n));
   System.out.println("average wt is "+ (float)(avgwt/n));
   sc.close();
}
}
```

OUTPUT:

```
enter no of process:4
enter process 1 arrival time:
2
enter process 1 burst time:
3
enter process 2 arrival time:
1
enter process 2 burst time:
2
enter proceAss 3 arrival time:3
enter process 3 burst time:
4
enter process 4 arrival time:
5
enter process 4 burst time:6
process id    arrival    burst    complete    turn    waiting
1                2          3          6          4          1
2                1          2          3          2          0
3                3          4          10         7          3
4                5          6          16         11         5
average tat is 6.0
average wt is 2.25
```

C. SHORTEST JOB FIRST (NON-PREMPTIVE)

```
 import java.util.*;

public class SJF {
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
System.out.println ("enter no of process:");
int n = sc.nextInt();
int pid[] = new int[n];
int at[] = new int[n]; // at means arrival time
int bt[] = new int[n]; // bt means burst time
int ct[] = new int[n]; // ct means complete time
int ta[] = new int[n]; // ta means turn around time
int wt[] = new int[n];  //wt means waiting time
int f[] = new int[n];  // f means it is flag it checks process is completed or not
int st=0, tot=0;
float avgwt=0, avgta=0;

for(int i=0;i<n;i++)
{
System.out.println ("enter process " + (i+1) + " arrival time:");
at[i] = sc.nextInt();
System.out.println ("enter process " + (i+1) + " brust time:");
bt[i] = sc.nextInt();
pid[i] = i+1;
f[i] = 0;
}
boolean a = true;
while(true)
{
int c=n, min=999;
if (tot == n) // total no of process = completed process loop will be terminated
break;
for (int i=0; i<n; i++)
{
/*
* If i'th process arrival time <= system time and its flag=0 and burst<min
* That process will be executed first
*/
if ((at[i] <= st) && (f[i] == 0) && (bt[i]<min))
{
min=bt[i];
c=i;
}
}
```

```
/* If c==n means c value can not updated because no process arrival time< system time so we increase the system time */
if (c==n)
st++;
else
{
ct[c]=st+bt[c];
st+=bt[c];
ta[c]=ct[c]-at[c];
wt[c]=ta[c]-bt[c];
f[c]=1;
tot++;
}
}
System.out.println("\npid  arrival brust  complete turn waiting");
for(int i=0;i<n;i++)
{
avgwt+= wt[i];
avgta+= ta[i];
System.out.println(pid[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+ta[i]+"\t"+wt[i]);
}
System.out.println ("\naverage tat is "+ (float)(avgta/n));
System.out.println ("average wt is "+ (float)(avgwt/n));
sc.close();
}
}
```

OUTPUT:

```
enter no of process:
3
enter process 1 arrival time:
0
enter process 1 brust time:
3
0enter process 2 arrival time:
0
enter process 2 brust time:
1
enter process 3 arrival time:
0
enter process 3 brust time:2
process id   arrival        brust  complete turn   waiting
1                    0         3         6         6         3
2                    0         1         1         1         0
3                    0         2         3         3         1
average tat is 3.3333333
average wt is 1.3333334
```

D. ROUND ROBIN SCHEDULING

```
import java.util.Scanner;

public class RoundRobin

{

public static void main(String args[])

{

int n,i,qt,count=0,temp,sq=0,bt[],wt[],tat[],rem_bt[];

float awt=0,atat=0;

bt = new int[10];

wt = new int[10];

tat = new int[10];

rem_bt = new int[10];

Scanner s=new Scanner(System.in);

System.out.print("Enter the number of process (maximum 10) = ");

n = s.nextInt();

System.out.print("Enter the burst time of the process\n");

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

{

System.out.print("P"+i+" = ");

bt[i] = s.nextInt();

rem_bt[i] = bt[i];

}

System.out.print("Enter the quantum time: ");
```

```
qt = s.nextInt();
while(true)
{
for (i=0,count=0;i<n;i++)
{
temp = qt;
if(rem_bt[i] == 0)
{
count++;
continue;
}
if(rem_bt[i]>qt)
rem_bt[i]= rem_bt[i] - qt;
else
if(rem_bt[i]>=0)
{
temp = rem_bt[i];
rem_bt[i] = 0;
}
sq = sq + temp;
tat[i] = sq;
}
if(n == count)
break;
```

```
}
System.out.print("---------------------------------------------------------------------------");
System.out.print("\nProcess\t      Burst Time\t       Turnaround Time\t          Waiting Time\n");
System.out.print("---------------------------------------------------------------------------");
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
awt=awt+wt[i];
atat=atat+tat[i];
System.out.print("\n "+(i+1)+"\t "+bt[i]+"\t\t "+tat[i]+"\t\t "+wt[i]+"\n");
}
awt=awt/n;
atat=atat/n;
System.out.println("\nAverage waiting Time = "+awt+"\n");
System.out.println("Average turnaround time = "+atat);
}
}
```

OUTPUT:

Enter the number of process (maximum 10) = 6
Enter the burst time of the process
P0 = 6
P1 = 34
P2 = 23
P3 = 45
P4 = 90
P5 = 12
Enter the quantum time: 5

Process	Burst Time	Turnaround Time	Waiting Time
1	6	31	25
2	34	135	101
3	23	106	83
4	45	160	115
5	90	210	120
6	12	78	66

Average waiting Time = 85.0

Average turnaround time = 120.0

2. PRODUCER CONSUMER PROBLEM

```
public class ProducerConsumer
{
    public static void main(String[] args)
    {
        Shop c = new Shop();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}
class Shop
{
    private int materials;
    private boolean available = false;
    public synchronized int get()
    {
        while (available == false)
        {
            try
            {
                wait();
            }
            catch (InterruptedException ie)
            {
            }
        }
        available = false;
        notifyAll();
        return materials;
    }
    public synchronized void put(int value)
    {
        while (available == true)
        {
            try
            {
                wait();
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
        materials = value;
        available = true;
```

```
            notifyAll();
        }
    }
    class Consumer extends Thread
    {
        private Shop Shop;
        private int number;
        public Consumer(Shop c, int number)
        {
            Shop = c;
            this.number = number;
        }
        public void run()
        {
            int value = 0;
            for (int i = 0; i < 10; i++)
            {
                value = Shop.get();
                System.out.println("Consumed value " + this.number+ " got: " + value);
            }
        }
    }
    class Producer extends Thread
    {
        private Shop Shop;
        private int number;

        public Producer(Shop c, int number)
        {
            Shop = c;
            this.number = number;
        }
        public void run()
        {
            for (int i = 0; i < 10; i++)
            {
                Shop.put(i);
                System.out.println("Produced value " + this.number+ " put: " + i);
                try
                {
                    sleep((int)(Math.random() * 100));
                }
                catch (InterruptedException ie)
                {
                    ie.printStackTrace();
                }
            }
        }
    }
```

OUTPUT:

```
Consumed value 1 got: 0
Produced value 1 put: 0
Produced value 1 put: 1
Consumed value 1 got: 1
Produced value 1 put: 2
Consumed value 1 got: 2
Produced value 1 put: 3
Consumed value 1 got: 3
Produced value 1 put: 4
Consumed value 1 got: 4
Produced value 1 put: 5
Consumed value 1 got: 5
Produced value 1 put: 6
Consumed value 1 got: 6
Produced value 1 put: 7
Consumed value 1 got: 7
Produced value 1 put: 8
Consumed value 1 got: 8
Produced value 1 put: 9
Consumed value 1 got: 9
```

3. DINING PHILOSOPHER PROBLEM :

PROGRAMS:

```
import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;

public class Main {

   static int philosopher = 5;
   static philosopher philosophers[] = new philosopher[philosopher];
   static chopstick chopsticks[] = new chopstick[philosopher];

   static class chopstick {

      public Semaphore mutex = new Semaphore(1);

      void grab() {
         try {
            mutex.acquire();
         }
         catch (Exception e) {
            e.printStackTrace(System.out);
         }
      }

      void release() {
         mutex.release();
      }

      boolean isFree() {
         return mutex.availablePermits() > 0;
      }

   }

   static class philosopher extends Thread {

      public int number;
      public chopstick leftchopstick;
      public chopstick rightchopstick;

      philosopher(int num, chopstick left, chopstick right) {
         number = num;
         leftchopstick = left;
         rightchopstick = right;
      }
```

```
    public void run(){

        while (true) {
            leftchopstick.grab();
            System.out.println("philosopher " + (number+1) + " grabs left chopstick.");
            rightchopstick.grab();
            System.out.println("philosopher " + (number+1) + " grabs right chopstick.");
            eat();
            leftchopstick.release();
            System.out.println("philosopher " + (number+1) + " releases left chopstick.");
            rightchopstick.release();
            System.out.println("philosopher " + (number+1) + " releases right chopstick.");
        }
    }

    void eat() {
        try {
            int sleepTime = ThreadLocalRandom.current().nextInt(0, 1000);
            System.out.println("philosopher " + (number+1) + " eats for " + sleepTime);
            Thread.sleep(sleepTime);
        }
        catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }

}

public static void main(String argv[]) {

    for (int i = 0; i < philosopher; i++) {
        chopsticks[i] = new chopstick();
    }

    for (int i = 0; i < philosopher; i++) {
        philosophers[i] = new philosopher(i, chopsticks[i], chopsticks[(i + 1) % philosopher]);
        philosophers[i].start();
    }

    while (true) {
        try {
            // sleep 1 sec
            Thread.sleep(1000);

            // check for deadlock
            boolean deadlock = true;
            for (chopstick f : chopsticks) {
                if (f.isFree()) {
                    deadlock = false;
                    break;
                }
```

```
            }
            if (deadlock) {
                Thread.sleep(1000);
                System.out.println("Everyone Eats");
                break;
            }
        }
        catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }

    System.out.println("Exit The Program!");
    System.exit(0);
  }

}
```

OUTPUT:

philosopher 1 grabs left chopstick.
philosopher 2 grabs left chopstick.
philosopher 2 grabs right chopstick.
philosopher 2 eats for 92
philosopher 4 grabs left chopstick.
philosopher 5 grabs left chopstick.
philosopher 2 releases left chopstick.
philosopher 1 grabs right chopstick.
philosopher 1 eats for 246
philosopher 3 grabs left chopstick.
philosopher 2 releases right chopstick.
philosopher 5 grabs right chopstick.
philosopher 5 eats for 217
philosopher 1 releases left chopstick.
philosopher 2 grabs left chopstick.
philosopher 1 releases right chopstick.
philosopher 5 releases left chopstick.
philosopher 4 grabs right chopstick.
philosopher 4 eats for 180
philosopher 1 grabs left chopstick.
philosopher 5 releases right chopstick.
philosopher 4 releases left chopstick.
philosopher 3 grabs right chopstick.
philosopher 5 grabs left chopstick.
philosopher 4 releases right chopstick.
philosopher 3 eats for 109
philosopher 3 releases left chopstick.
philosopher 2 grabs right chopstick.
philosopher 3 releases right chopstick.
philosopher 4 grabs left chopstick.
philosopher 2 eats for 581
philosopher 2 releases left chopstick.
philosopher 1 grabs right chopstick.
philosopher 3 grabs left chopstick.
philosopher 1 eats for 531
philosopher 2 releases right chopstick.
philosopher 5 grabs right chopstick.
philosopher 5 eats for 440
philosopher 1 releases left chopstick.
philosopher 1 releases right chopstick.
philosopher 2 grabs left chopstick.
Everyone Eats
Exit The Program!

4.(A) CONTIGUOUS MEMORY ALLOCATION – BEST FIT

```
public class GFG
{

   static void bestFit(int blockSize[], int m, int processSize[],
                                          int n)
   {
      int allocation[] = new int[n];

      for (int i = 0; i < allocation.length; i++)
         allocation[i] = -1;

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

         int bestIdx = -1;
         for (int j=0; j<m; j++)
         {
            if (blockSize[j] >= processSize[i])
            {
               if (bestIdx == -1)
                  bestIdx = j;
               else if (blockSize[bestIdx] > blockSize[j])
                  bestIdx = j;
            }
         }

         if (bestIdx != -1)
         {

            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
         }
      }

      System.out.println("\nProcess No.\tProcess Size\tBlock no.");
      for (int i = 0; i < n; i++)
      {
         System.out.print("   " + (i+1) + "\t\t" + processSize[i] + "\t\t");
         if (allocation[i] != -1)
            System.out.print(allocation[i] + 1);
         else
```

```
                System.out.print("Not Allocated");
            System.out.println();
        }
    }


    public static void main(String[] args)
    {
        int blockSize[] = {100, 500, 200, 300, 600};
        int processSize[] = {212, 417, 112, 426};
        int m = blockSize.length;
        int n = processSize.length;

        bestFit(blockSize, m, processSize, n);
    }
}
```

Output:

Process No.	Process Size	Block no.
1	212	4
2	417	2
3	112	3
4	426	5

4. (b)CONTIGUOUS MEMORY ALLOCATION – worst FIT

```
public class GFG
{

        static void worstFit(int blockSize[], int m, int processSize[],

        int n)
        {

                int allocation[] = new int[n];

                for (int i = 0; i < allocation.length; i++)
                        allocation[i] = -1;

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

                        int wstIdx = -1;
                        for (int j=0; j<m; j++)
                        {
                                if (blockSize[j] >= processSize[i])
                                {
                                        if (wstIdx == -1)
                                                wstIdx = j;
                                        else if (blockSize[wstIdx] < blockSize[j])
                                                wstIdx = j;
                                }
                        }


                        if (wstIdx != -1)
                        {

                                allocation[i] = wstIdx;

                                blockSize[wstIdx] -= processSize[i];
                        }
                }
```

```
            System.out.println("\nProcess No.\tProcess Size\tBlock no.");
            for (int i = 0; i < n; i++)
            {
                System.out.print(" " + (i+1) + "\t\t" + processSize[i] +
"\t\t");
                if (allocation[i] != -1)
                    System.out.print(allocation[i] + 1);
                else
                    System.out.print("Not Allocated");
                System.out.println();
            }
        }


        public static void main(String[] args)
        {
            int blockSize[] = {100, 500, 200, 300, 600};
            int processSize[] = {212, 417, 112, 426};
            int m = blockSize.length;
            int n = processSize.length;

            worstFit(blockSize, m, processSize, n);
        }
}
```

Output

```
Process No.    Process Size    Block no.
   1                212         5
   2                417         2
   3                112         5
   4                426         Not Allocated
```

4. (C)CONTIGUOUS MEMORY ALLOCATION – FIRST FIT

```
class GFG
{
   static void firstFit(int blockSize[], int m,
                        int processSize[], int n)
   {
      int allocation[] = new int[n];
      for (int i = 0; i < allocation.length; i++)
         allocation[i] = -1;
      for (int i = 0; i < n; i++)
      {
         for (int j = 0; j < m; j++)
         {
            if (blockSize[j] >= processSize[i])
            {
               allocation[i] = j;
               blockSize[j] -= processSize[i];

               break;
            }
         }
      }

      System.out.println("\nProcess No.\tProcess Size\tBlock no.");
      for (int i = 0; i < n; i++)
      {
         System.out.print(" " + (i+1) + "\t\t" +
                          processSize[i] + "\t\t");
         if (allocation[i] != -1)
            System.out.print(allocation[i] + 1);
         else
            System.out.print("Not Allocated");
         System.out.println();
      }
   }

   public static void main(String[] args)
   {
      int blockSize[] = {100, 500, 200, 300, 600};
      int processSize[] = {212, 417, 112, 426};
      int m = blockSize.length;
      int n = processSize.length;

      firstFit(blockSize, m, processSize, n);
   }
}
```

Output :

```
Process No.    Process Size    Block no.
1              212             2
2              417             5
3              112             2
4              426             Not Allocated
```

5.(A) Page replacement algorithm-FIFO

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;

class Test
{
   static int pageFaults(int pages[], int n, int capacity)
   {
HashSet<Integer> s = new HashSet<>(capacity);
      Queue<Integer> indexes = new LinkedList<>() ;

      int page_faults = 0;
      for (int i=0; i<n; i++)
      {
         if (s.size() < capacity)
         {
            if (!s.contains(pages[i]))
            {
               s.add(pages[i]);
               page_faults++;
               indexes.add(pages[i]);
            }
         }

         else
         {
            if (!s.contains(pages[i]))
            {
               int val = indexes.peek();
               indexes.poll();
               s.remove(val);
               s.add(pages[i]);
               indexes.add(pages[i]);
                        page_faults++;
            }
         }
      }

      return page_faults;
   }

   public static void main(String args[])
   {
      int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,
                     2, 3, 0, 3, 2};

      int capacity = 4;
```

```
System.out.println(pageFaults(pages, pages.length, capacity));
```

OUTPUT:
7

5 (B) Page replacement algorithm-LRU

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;

class Test
{
   static int pageFaults(int pages[], int n, int capacity)
   {
      HashSet<Integer> s = new HashSet<>(capacity);
      HashMap<Integer, Integer> indexes = new HashMap<>();
      int page_faults = 0;
      for (int i=0; i<n; i++)
      {
         if (s.size() < capacity)
         {
            if (!s.contains(pages[i]))
            {
               s.add(pages[i]);
               page_faults++;
            }
            indexes.put(pages[i], i);
         }

         else
         {
            if (!s.contains(pages[i]))
            {
               int lru = Integer.MAX_VALUE, val=Integer.MIN_VALUE;

               Iterator<Integer> itr = s.iterator();

               while (itr.hasNext()) {
                  int temp = itr.next();
                  if (indexes.get(temp) < lru)
                  {
                     lru = indexes.get(temp);
                     val = temp;
                  }
               }
               s.remove(val);
              indexes.remove(val);
               s.add(pages[i]);
               page_faults++;
            }
            indexes.put(pages[i], i);
         }
      }
      return page_faults;
```

```
    }

    public static void main(String args[])
    {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
        int capacity = 4;
        System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
```

OUTPUT:-

6

6. DEADLOCK AVOIDANCE - BANKER'S ALGORITHM

```
import java.util.Scanner;
public class Bankers{
   private int need[][],allocate[][],max[][],avail[][],np,nr;

   private void input(){
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter no. of processes and resources : ");
    np=sc.nextInt();  //no. of process
    nr=sc.nextInt();  //no. of resources
    need=new int[np][nr];  //initializing arrays
    max=new int[np][nr];
    allocate=new int[np][nr];
    avail=new int[1][nr];

    System.out.println("Enter allocation matrix -->");
    for(int i=0;i<np;i++)
         for(int j=0;j<nr;j++)
        allocate[i][j]=sc.nextInt();  //allocation matrix

    System.out.println("Enter max matrix -->");
    for(int i=0;i<np;i++)
         for(int j=0;j<nr;j++)
        max[i][j]=sc.nextInt();  //max matrix
```

```
    System.out.println("Enter available matrix -->");
    for(int j=0;j<nr;j++)
     avail[0][j]=sc.nextInt();  //available matrix

    sc.close();
}

private int[][] calc_need(){
   for(int i=0;i<np;i++)
     for(int j=0;j<nr;j++)  //calculating need matrix
      need[i][j]=max[i][j]-allocate[i][j];

   return need;
}

private boolean check(int i){
   //checking if all resources for ith process can be allocated
   for(int j=0;j<nr;j++)
   if(avail[0][j]<need[i][j])
      return false;

return true;
}

public void isSafe(){
   input();
```

```
    calc_need();
    boolean done[]=new boolean[np];
    int j=0;

    while(j<np){  //until all process allocated
    boolean allocated=false;
    for(int i=0;i<np;i++)
     if(!done[i] && check(i)){  //trying to allocate
        for(int k=0;k<nr;k++)
        avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
      System.out.println("Allocated process : "+i);
      allocated=done[i]=true;
           j++;
          }
       if(!allocated) break;  //if no allocation
    }
    if(j==np)  //if all processes are allocated
     System.out.println("\nSafely allocated");
    else
     System.out.println("All proceess cant be allocated safely");
  }

  public static void main(String[] args) {
    new Bankers().isSafe();
  }
}
```

Output

Enter no. of processes and resources : 3 4

Enter allocation matrix -->

1 2 2 1

1 0 3 3

1 2 1 0

Enter max matrix -->

3 3 2 2

1 1 3 4

1 3 5 0

Enter available matrix -->

3 1 1 2

Allocated process : 0

Allocated process : 1

Allocated process : 2

Safely allocated

7. DEAD LOCK PREVENTION:

PROGRAM:

```
public class DeadlockTest {
  public static void main(String[] args) throws InterruptedException {
    Object obj1 = new Object();
    Object obj2 = new Object();
    Object obj3 = new Object();
    Thread t1 = new Thread(new SyncThread(obj1, obj2), "t1");
    Thread t2 = new Thread(new SyncThread(obj2, obj3), "t2");
    t1.start();
    Thread.sleep(2000);
    t2.start();
    Thread.sleep(2000);
  }
}
class SyncThread implements Runnable {
  private Object obj1;
  private Object obj2;
  public SyncThread(Object o1, Object o2){
    this.obj1=o1;
    this.obj2=o2;
  }
  @Override
  public void run() {
    String name = Thread.currentThread().getName();
    System.out.println(name + " acquiring lock on " + obj1);
    synchronized (obj1) {
      System.out.println(name + " acquired lock on " + obj1);
      work();
    }
    System.out.println(name + " released lock on " + obj1);
    System.out.println(name + " acquiring lock on " + obj2);
    synchronized (obj2) {
      System.out.println(name + " acquired lock on " + obj2);
      work();
    }
    System.out.println(name + " released lock on " + obj2);
    System.out.println(name + " finished execution.");
  }
  private void work() {
    try {
      Thread.sleep(5000);
    } catch (InterruptedException ie) {
      ie.printStackTrace();
    }
  }
}
```

OUTPUT :

```
t1 acquiring lock on java.lang.Object@4686939d
t1 acquired lock on java.lang.Object@4686939d
t2 acquiring lock on java.lang.Object@56b19245
t2 acquired lock on java.lang.Object@56b19245
t1 released lock on java.lang.Object@4686939d
t1 acquiring lock on java.lang.Object@56b19245
t1 acquired lock on java.lang.Object@56b19245
t2 released lock on java.lang.Object@56b19245
t2 acquiring lock on java.lang.Object@587676e3
t2 acquired lock on java.lang.Object@587676e3
t1 released lock on java.lang.Object@56b19245
t1 finished execution.
t2 released lock on java.lang.Object@587676e3
t2 finished execution.
```