

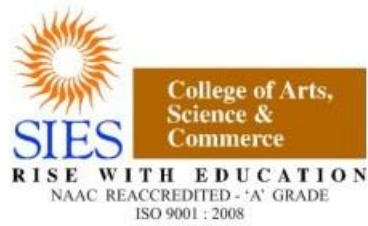


OS PRACTICAL JOURNAL

SCS2223057



Mayank Padmakar



S.I.E.S College of Arts, Science and Commerce Sion(W),
Mumbai – 400 022.

CERTIFICATE

This is to certify that **MR.MAYANK KISHOR KUMAR PADMAKAR** Roll No. **SCS2223057** has successfully completed the necessary course of experiments in the subject of Operating System during the academic year 2022 - 2023 complying with the requirements of University of Mumbai, for the course of S.Y.BSc. Computer Science [Semester-3]

Prof. In-Charge
Maya Nair
(Operating System)

Examination Date

Examiner's Signature & Date:

**Head of the
Department**
Prof. Manoj Singh

**College
Seal
And Date**

Sr.no	Aim
1	Program to implement the concept of multi-threading.
2	Program to implement RMI programming a. RMI SERVER (SQRT) b. RMI SERVER (GCD)
3	Program to implement bounded buffer to solve producer-consumer without semaphore
4	Program to stimulate the FCFS algorithm
5	Program to stimulate the SJF algorithm
6	Program to implement bounded buffer to solve producer-consumer with semaphore
7	Program to implement the Banker's Algorithm

Practical 1:

Aim: Program to implement the concept of multi-threading.

Code:

1. Sum of number:

```
import threading
def summation(num):
    print(f'Current thread running is {threading.current_thread().name}')
    sum=0;
    for i in range(1,num+1):
        sum+=i
    print(f'summation of {num} is {sum}')
t1=threading.Thread(target=summation,args=(3,),name="t1")
t2=threading.Thread(target=summation,args=(2,),name="t2")

print(f'Current thread running is {threading.current_thread().name}')
t1.start()
t2.start()
t1.join()
t2.join()
print("Done!")
```

Output:

```
Current thread running is MainThread
Current thread running is t1
summation of 3 is 6
Current thread running is t2
summation of 2 is 3
Done!
```

2. Square of number:

```
import threading
def square(num):
    print(f'Current thread running is {threading.current_thread().name}')
    print(f'square of {num} is {num*num}')

print(f'Current thread running is {threading.current_thread().name}')

t1=threading.Thread(target=square,args=(10,),name="t1")
t2=threading.Thread(target=square,args=(20,),name="t2")

t1.start()
t2.start()

t1.join()
t2.join()
print("Done!!!")
```

Output:

```
Current thread running is MainThread
Current thread running is t1
square of 10 is 100
Current thread running is t2
square of 20 is 400
Done!!!
```

3. Prime numbers:

```
import threading
def showprime(num1,num2):
    print(f'Following are the prime numbers between {num1} and {num2}')
    for num in range(num1,num2+1):
        if num > 1:
            for i in range(2,num):
                if (num % i) == 0:
```

```

        break
    else:
        print(num)

```

```

t1=threading.Thread(target=showprime,args=(10,20),name="t1")
t2=threading.Thread(target=showprime,args=(50,100),name="t2")

print(f'Current thread running is {threading.current_thread().name}')
t1.start()
t2.start()
t1.join()
t2.join()
print("Done!")

```

Output:

```

Current thread running is MainThread
Following are the prime numbers between 10 and 20
11
13Following are the prime numbers between 50 and 100
53

1759
61
67

19
71
73
79
83
89
97
Done!

```

Practical 2A:

Aim: Program to implement RMI programming (SQRT)

Code:

RMI Server:

Server code :

```

import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
public class RMIServer extends UnicastRemoteObject implements MethodImpl
{
    public RMIServer() throws RemoteException
    {
        System.out.println("The server is instantiated");
    }

    public double getSqrt(double dbl)
    {
        return Math.sqrt(dbl);
    }
    public static void main(String[] args)
    {
        try
        {
            RMIServer server = new RMIServer();
            Naming.rebind("//localhost/call1", server);
        }
        catch (Exception exce)
        {
            System.out.println("Error -- " +exce.toString());
            exce.printStackTrace();
        }
    }
}

```

```
}}}
```

RMI Client:

```
import java.rmi.*;
import java.rmi.registry.*;
public class RMIClient
{
    public static void main(String[] arguments)
    {
        try
        {
            MethodImpl mthdIp = (MethodImpl)Naming.lookup("//localhost/call1");
            double dbl = mthdIp.getSqrt(100);
            System.out.println("SQRT: " + dbl);
        }
        catch (Exception exec)
        {
            System.out.println("Error--" + exec.toString());
            exec.printStackTrace();
        }
    }
}
```

Interface:

```
import java.rmi.*;
interface MethodImpl extends Remote
{
    double getSqrt(double dbl) throws RemoteException;
}
```

Output:

```
\
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.2075]
(c) Microsoft Corporation. All rights reserved.

D:\SYCS\OS\RMI 2>set path="C:\Program Files\Java\java\jdk1.8.0_341\bin"

D:\SYCS\OS\RMI 2>javac *.java

D:\SYCS\OS\RMI 2>rmic RMIServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

D:\SYCS\OS\RMI 2>start rmiregistry

D:\SYCS\OS\RMI 2>start java RMIServer

D:\SYCS\OS\RMI 2>java RMIClient
SQRT: 10.0

D:\SYCS\OS\RMI 2>
```

Practical 2B:

Aim: Program to implement RMI programming (GCD)

Code:

RMI Server:

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
public class RMIServer extends UnicastRemoteObject implements gcdinterface
{
    public RMIServer() throws RemoteException
    {
        System.out.println("The server is instantiated");
    }
}
```

```

    }
    public int gcdinterface(int a,int b)
    {
    int min,gcd=0;
    if(a>b)
    min=b;
    else
    min=a;
    for(int i=1;i<=min;i++)
    {
    if (a%i==0 && b%i==0)
    gcd=i; //overwrite the earlier variables
    }
    return gcd;
    }
    public static void main(String[] args)
    {
    try
    {
    RMIServer server = new RMIServer();
    Naming.rebind("//localhost/call1", server);
    }
    catch (Exception exce)
    {
    System.out.println("Error -- " +exce.toString());
    exce.printStackTrace();
    }}}

```

RMI Client:

```

import java.rmi.*;
import java.rmi.registry.*;
import java.util.Scanner;
public class RMIClient
{
    public static void main(String[] arguments)
    {
    try
    {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter First number: ");
    int num1=sc.nextInt();
    System.out.println("Enter Second number: ");
    int num2=sc.nextInt();
    gcdinterface gcd = (gcdinterface)Naming.lookup("//localhost/call1");
    int res = gcd.gcdinterface(num1,num2);
    System.out.println("GCD: " + res);
    }
    catch (Exception exec)
    {
    System.out.println("Error--" + exec.toString());
    exec.printStackTrace();
    }}}

```

Interface:

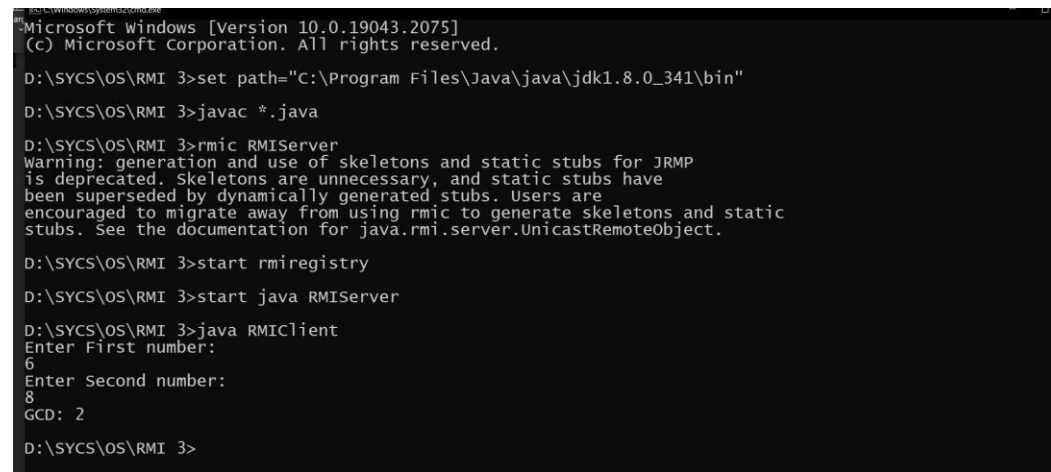
```

import java.rmi.*;
interface gcdinterface extends Remote
{

```

```
int gcdinterface(int a, int b) throws RemoteException;
}
```

Output:



```
Microsoft Windows [Version 10.0.19043.2075]
(c) Microsoft Corporation. All rights reserved.

D:\SYCS\OS\RMI 3>set path="C:\Program Files\Java\java\jdk1.8.0_341\bin"

D:\SYCS\OS\RMI 3>javac *.java

D:\SYCS\OS\RMI 3>rmic RMIServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

D:\SYCS\OS\RMI 3>start rmiregistry

D:\SYCS\OS\RMI 3>start java RMIServer

D:\SYCS\OS\RMI 3>java RMIclient
Enter First number:
6
Enter Second number:
8
GCD: 2

D:\SYCS\OS\RMI 3>
```

Practical 3:

Aim: Program to implement bounded buffer to solve producer-consumer without semaphore

Code:

from threading import Thread

import time

class buffer:

```
    def __init__(self,size):
        self.size=size
        self.b=[0]*size
        self.into=0
        self.out=0
        self.counter=0
```

```
    def getvalue(self):
        x=self.b[self.out]
        self.out=(self.out+1)%self.size
        self.counter-=1
        return x
```

```
    def putvalue(self,value):
        self.b[self.into]=value
        self.out=(self.into+1) %self.size
        self.counter+=1
```

class Producer(Thread):

```
    def __init__(self,b):
        super(Producer,self).__init__()
        self.b=b
```

```
    def run(self):
        i=0
        while self.b.counter < self.b.size:
            i=i+1
            self.b.putvalue(i)
            print(f"{i} put in buffer\n")
```



```

        print(f"Buffer contents after production: {self.b.counter}")
        time.sleep(5)

class Consumer(Thread):
    def __init__(self,b):
        super(Consumer,self).__init__()
        self.b=b

    def run(self):
        while self.b.counter!=0:
            value=self.b.getvalue()
            print(f"{value} consumed from buffer\n")
            print(f"Buffer contents after consumption: {self.b.counter}\n")
            time.sleep(10)

b=buffer(5)
p=Producer(b)
c=Consumer(b)
p.start()
c.start()
c.join()
p.join()

```

Output:

```

1 put in buffer

Buffer contents after production: 1
0 consumed from buffer

Buffer contents after consumption: 0

2 put in buffer

Buffer contents after production: 1
0 consumed from buffer

Buffer contents after consumption: 0

3 put in buffer

Buffer contents after production: 1
4 put in buffer

Buffer contents after production: 2
0 consumed from buffer

Buffer contents after consumption: 1

5 put in buffer

Buffer contents after production: 2

```

Practical 4:

Aim: Program to stimulate the FCFS algorithm

Code:

```

def getwt(n,bt,at,wt):
    st=[0]*n
    for i in range(1,n):
        st[i]=st[i-1]+bt[i-1]
        wt[i]=st[i]-at[i]

def gettat(n,bt,at,wt,tat):
    for i in range(n):
        tat[i]=wt[i]+bt[i]

def getaverage(n,processes,bt,at):
    wt=[0]*n
    tat=[0]*n
    getwt(n,bt,at,wt)
    gettat(n,bt,at,wt,tat)

```

```

totalwt=0
totaltat=0
print("Processes\tAT\tBT\tWT\tTAT")
for i in range(n):
    totalwt=totalwt+wt[i]
    totaltat=totaltat+tat[i]
    print(f'\tP{processes[i]}\t{bt[i]}\t{at[i]}\t{wt[i]}\t{tat[i]}')
avgwt=totalwt/n
avgtat=totaltat/n
print(f"The average waiting time is {round(avgwt,2)}")
print(f"The average turn around time is {round(avgtat,2)}")

```

```

n=int(input("Enter no. of process "))
processes=list(map(int,input(f"Enter {n} process ids seperated by space ").split()))
at=list(map(int,input(f"Enter arrival time of {n} processes seperated by space ").split()))
bt=list(map(int,input(f"Enter brust time for {n} processes seperated by space ").split()))
getaverage(n,processes,bt,at)

```

Output:

```

Enter no. of process 3
Enter 3 process ids seperated by space 1 2 3
Enter arrival time of 3 processes seperated by space 0 2 0
Enter brust time for 3 processes seperated by space 4 8 3
Processes    AT    BT    WT    TAT
P1           4     0     0     4
P2           8     2     2    10
P3           3     0    12    15
The average waiting time is 4.67
The average turn around time is 9.67

```

Practical 5:

Aim: Program to simulate the SJF algorithm

Code:

```

def getwt(n,plist):
    st=[0]*n
    for i in range(1,n):
        st[i]=st[i-1]+plist[i-1][1]
        plist[i][2]=st[i]

def gettat(n,plist):
    for i in range(n):
        plist[i][3]=plist[i][1]+plist[i][2]

def getaverage(n,plist):
    getwt(n,plist)
    gettat(n,plist)
    print("Process\t BT\t WT\t TAT \n")
    tot_wt=0
    tot_tat=0
    for i in range (n):
        tot_wt+=plist[i][2]
        tot_tat+=plist[i][3]
        print(f'p{plist[i][0]}\t{plist[i][1]}\t{plist[i][2]}\t{plist[i][3]}')
    avg_wt=tot_wt/n
    avg_tat=tot_tat/n
    print(f"Average waiting time is {avg_wt}")
    print(f"Average turn around time is {avg_tat}")

```

```

process_list=[]
n=int(input("Enter the no. of process :"))
for i in range (n):

```

```

process=list(map(int,input("Enter process no and burst time seperated by space : ").split()))
process.extend([0,0])
process_list.append(process)
process_list=sorted(process_list,key=lambda x:x[1])
print(process_list)
getaverage(n,process_list)

```

Output:

```

Enter the no. of process :3
Enter process no and burst time seperated by space : 1 5
Enter process no and burst time seperated by space : 2 8
Enter process no and burst time seperated by space : 3 9
[[1, 5, 0, 0], [2, 8, 0, 0], [3, 9, 0, 0]]
Process  BT      WT      TAT
p1       5       0       5
p2       8       5      13
p3       9      13      22
Average waiting time is 6.0
Average turn around time is 13.333333333333334

```

Practical 6:

Aim: Program to implement bounded buffer to solve producer-consumer with semaphore

Code:

```

import threading
from threading import Thread
import time

```

```

class Buffer():
    def __init__(self,size):
        self.size=size
        self.buf=[0]*size
        self.into=0
        self.out=0
        self.empty=threading.Semaphore(self.size)
        self.full=threading.Semaphore(0)
        self.mutex=threading.Semaphore(1)

```

```

    def getvalue(self):
        x=self.buf[self.out]
        self.out=(self.out+1)%self.size
        return x

```

```

    def putvalue(self,value):
        self.buf[self.into]=value
        self.into=(self.into+1) %self.size

```

```

class Producer(Thread):
    def __init__(self,b):
        super(Producer,self).__init__()
        self.buf=b

```

```

    def run(self):
        i=0
        while True:
            i=i+1
            self.buf.empty.acquire()
            self.buf.mutex.acquire()
            self.buf.putvalue(i)
            self.buf.mutex.release()
            self.buf.full.release()
            print(f"{i} put in buffer\n")

```

```

        time.sleep(5)

class Consumer(Thread):
    def __init__(self,b):
        super(Consumer,self).__init__()
        self.buf=b

    def run(self):
        while True:
            self.buf.full.acquire()
            self.buf.mutex.acquire()
            value=self.buf.getvalue()
            self.buf.mutex.release()
            self.buf.empty.release()
            print(f"{value} is consumed from buffer\n")
            time.sleep(5)

b=Buffer(5)
p=Producer(b)
c=Consumer(b)
p.start()
c.start()
p.join()
c.join()

```

Output:

```

1 put in buffer
1 is consumed from buffer
2 put in buffer
2 is consumed from buffer

```

Practical 7:

Aim: Program to implement Banker's Algorithm

Code:

```

n=int(input("Enter the number of processes: "))
m=int(input("Enter the number of resources: "))
Allocation=[]
Max=[]
Need=[]
Available=[]
print("Enter the Allocation matrix: ")
for i in range(n):
    rowinput=[]
    for j in range(m):
        x=int(input())
        rowinput.append(x)
    Allocation.append(rowinput)
print("Enter the Max matrix: ")
for i in range(n):
    rowinput=[]
    for j in range(m):
        x=int(input())
        rowinput.append(x)
    Max.append(rowinput)
print("Enter the Need matrix: ")
for i in range(n):
    rowinput=[]

```

```

    for j in range(m):
        x=int(input())
        rowinput.append(x)
    Need.append(rowinput)
Resources=[]
print("Enter the total resources: ")
for i in range(m):
    x=int(input())
    Resources.append(x)
for j in range(m):
    x=0
    for i in range(n):
        x+=Allocation[i][j]
    x=Resources[j]-x
    Available.append(x)
Work=Available.copy()
finish=[0]*n
Sequence=[]
attempt=0
all=False
while alldone==False:
    attempt+=1
    for i in range(n):
        if(finish[i]==0 and Need[i]<=Work):
            for k in range(m):
                Work[k]+=Allocation[i][k]
                finish[i]=1
            Sequence.append(i)
    for i in range(n):
        if(finish[i]==0):
            break
    else:
        alldone==True
    if attempt>2:
        break
if alldone==True:
    print("System is in Safe State. ")
    print("The sequence is ",Sequence)
else:
    print("System is Unsafe. ")

```

Output:

Enter the number of processes: 5

Enter the Allocation matrix:

1

2

2

2

1

1

2

7

2

2

Enter the total resources:

10

7