SIES College of Arts, Science & Commerce
RISE WITH EDUCATION
NAAC REACCREDITED - 'A' GRADE
ISO 9001 : 2008

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

CERTIFICATE

This is to certify that Mr. / Miss. Lingraj Namal
Roll No. SCS2223054 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

INDEX

	algorithm and calculate average waiting time and average turnaround time		
5	Write a program to simulate SJF algorithm (Non Pre-emptive) and calculate average waiting time and average turnaround time	18	15/9/22
6	Write a program to implement Bounded Buffer to solve Producer - Consumer with Semaphore.	20	22/9/22
7	Write a Python program that implements the banker's algorithm.	23	29/9/22

Practical 1

Aim: -

Programs to implement the concept of Multithreading

a. Program to display Summation of numbers using thread

b. Program to display the prime numbers using thread

c. Program to display the Fibonacci series using thread

Code: -

a. **For Summation: -**

```
import threading
def submission(num):
        sum1=0
        for i in range(1, num+1):
            sum1+=i
        print(f"The sum of {num} is {sum1}")
        print(f"current thread runnning is
{threading.current_thread().name}")
t3=threading.Thread(target=submission,args=(10,),name="t3
")
t4=threading.Thread(target=submission,args=(20,),name="t4
")
print(f"current thread running is
{threading.current_thread().name}")
t3.start()
t4.start()
t3.join()
t4.join()
print("finish!!")
```

b. **<u>For Prime: -</u>**

```
import threading
def showprime(num1,num2):
   print(f"current thread running is
(threading.current_thread().name)")
   print(f"prime numbers between {num1} and {num2}:")
   for i in range(num1,num2+1):
      count=0
      for j in range(1,i+1):
         if i%j==0:
            count+=1
      if count==2:
         print(i)
t3=threading.Thread(target=showprime,args=(10,20),name=
"t3")
t4=threading.Thread(target=showprime,args=(1,100),name=
"t4")
print(f"current thread running is
{threading.current_thread().name}")
t3.start()
t4.start()
t3.join()
t4.join()
print("finish!!")
```

c. **<u>For Factorial: -</u>**

```
import threading
def showfact(num):
      if num < 0:
         print("Please enter number great than 1");
      elif num == 0 or num == 1:
```

```
        print("Please enter number great than 1");
    else:
        fact = 1
        while(num > 1):
            fact *= num
            num -= 1
    print(f"The factorial of {num} is {fact}");
    print(f"current thread runnning is
{threading.current_thread().name}")
t3=threading.Thread(target=showfact,args=(5,),name="t3")
t4=threading.Thread(target=showfact,args=(9,),name="t4")
print(f"current thread running is
{threading.current_thread().name}")
t3.start()
t4.start()
t3.join()
t4.join()
```

Output: -

i. **For Summation: -**

```
current thread running is MainThread
The sum of 10 is 55
current thread runnning is t3
The sum of 20 is 210
current thread runnning is t4
finish!!
```

ii. **For Prime: -**

```
current thread running is MainThread
current thread running is (threading.current_thread().name)
prime numbers between 10 and 20:
11
13
17
19
current thread running is (threading.current_thread().name)
prime numbers between 1 and 100:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
finish!!
```

iii. <u>For Factorial: -</u>

```
current thread running is MainThread
The factorial of 1 is 120
current thread runnning is t3
The factorial of 1 is 362880
current thread runnning is t4
```

Practical 2

Aim: -

Write a program to implement the concept of Remote Method Invocation(RMI)

a. Program to display square root of a number with RMI

b. Program to display factorial of a number with RMI

Code: -

For Square Root: -

```
import java.rmi.*;

interface MethodImpl extends Remote

{

double getSqrt(double dbl) throws RemoteException;

}
```

- **Client: -**

```
import java.rmi.*;
import java.rmi.registry.*;
public class RMIClient
{
public static void main(String args[])
{
try
{
MethodImpl
mthdIp=(MethodImpl)Naming.lookup("//localhost/call1");
double dbl=mthdIp.getSqrt(100);
```

```
System.out.println("SQRT:" +dbl);
}
catch (Exception exce)
{
System.out.println("Error--" + exce.toString());
exce.printStackTrace();
}
}
}
```

- **<u>Server: -</u>**

```
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();
}
}
}
```

For Factorial: -

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

- **Server**

```
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 getFact(double dbl)
{
```

```
int f=1, n=1;
while(n<=dbl)
{
f = f*n;
n++;
}
return(f);
}
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();
}
}
}
```

- **Client**

```
import java.rmi.*;
import java.rmi.registry.*;
public class RMIClient
{
public static void main(String args[])
{
try
```

```
{
MethodImpl
mthdlp=(MethodImpl)Naming.lookup("//localhost/call1");
double dbl = mthdlp.getFact(5);
System.out.println("Factorial: "+dbl);
}
catch (Exception exce)
{
System.out.println("Error--" + exce.toString());
exce.printStackTrace();
}
}
}
```

Output: -

For Square Root: -

```
D:\Ajay Programmers\OS\RMI>start rmiregistry

D:\Ajay Programmers\OS\RMI>start java RMIServer
```

```
D:\Ajay Programmers\OS\RMI>java RMIClient
SQRT:10.0
```

For Factorial: -

```
D:\Ajay Programmers\OS\RMI_>start rmiregistry

D:\Ajay Programmers\OS\RMI_>start java RMIServer

D:\Ajay Programmers\OS\RMI_>java RMIClient
Factorial: 120.0
```

Practical 3

Aim: -

Write a 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.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+=1
            self.buf.empty.acquire() #similar to wait()
            self.buf.mutex.acquire()
            self.buf.putvalue(i)
            self.buf.mutex.release()
            self.buf.full.release()
            print(f"{i} out 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()
            values=self.buf.getvalue()
            self.buf.mutex.release()
```

```
        self.buf.empty.release()
        print(f"{values} is consumed from Buffer\n")
        time.sleep(10)

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

Output: -

```
1 out in Buffer
1 is consumed from Buffer
2 out in Buffer
2 is consumed from Buffer
3 out in Buffer
4 out in Buffer
```

Practical 4

Aim: - Write a program to simulate FCFS algorithm and calculate average waiting time and average turnaround time

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,wt,tat):
   for i in range(n):
      tat[i]=wt[i]+bt[i]
def getaverage(n,p,bt,at):
   wt=[0]*n
   tat=[0]*n
   getwt(n,bt,at,wt)
   gettat(n,bt,wt,tat)
   totalwt=0
   totat=0
   print("Processes\tBT\tAT\tWT\tTAT")
   for i in range(n):
      totalwt+=wt[i]
      totat+=tat[i]
      print(f"\tP{p[i]}\t{bt[i]}\t{at[i]}\t{wt[i]}\t{tat[i]}")
```

```
    avgwt=totalwt/n
    avgtat=totat/n
    print(f"Average waiting time is {round(avgwt,2)}")
    print(f"Average Turnaround is {round(avgtat,2)}")

n=int(input("Enter the no. of processes: "))
processes=list(map(int,input(f"Enter {n} processes number seperated by space: ").split()))
bursttime=list(map(int,input("Enter the burst time of each processes separated by space: ").split()))
arrivaltime=list(map(int,input("Enter the arrival time of each processes separated by space: ").split()))
getaverage(n,processes,bursttime,arrivaltime)
```

Output: -

```
Enter the no. of processes: 4
Enter 4 processes number seperated by space: 1 2 3 4
Enter the burst time of each processes separated by space: 10 15 4 2
Enter the arrival time of each processes separated by space: 0 1 2 3
Processes       BT      AT      WT      TAT
        P1      10      0       0       10
        P2      15      1       9       24
        P3      4       2       23      27
        P4      2       3       26      28
Average waiting time is 14.5
Average Turnaround is 22.25
```

Practical 5

Aim: - Write a program to simulate SJF algorithm (Non Pre-emptive) and calculate average waiting time and average turnaround time

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 BT WT 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]}\t")
```

```
    avg_wt=tot_wt/n
    avg_tat=tot_tat/n
    print(f"Average Waiting time: {avg_wt}")
    print(f"Average Turn around time: {avg_tat}")

process_list = []
n=int(input("Enter the number of  processes: "))
for i in range(n):
    process=list(map(int,input("Enter process no and burst time separated by space: ").split()))
    process.extend([0,0])#waiting time and turnaround time [if,bt,wt,tat]
    process_list.append(process)
process_list=sorted(process_list,key=lambda x:x[1])
print(process_list)
getaverage(n,process_list)
```

Output: -

```
Enter the number of  processes: 4
Enter process no and burst time separated by space: 1 18
Enter process no and burst time separated by space: 2 6
Enter process no and burst time separated by space: 3 12
Enter process no and burst time separated by space: 4 20
[[2, 6, 0, 0], [3, 12, 0, 0], [1, 18, 0, 0], [4, 20, 0, 0]]
Process BT WT TAT

P2      6       0       6
P3      12      6       18
P1      18      18      36
P4      20      36      56
Average Waiting time: 15.0
Average Turn around time: 29.0
```

Practical 6

Aim: - Write a program to implement Bounded Buffer to solve Producer -Consumer with 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.into=(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+=1
            self.b.putvalue(i)
            print(f"{i} out in Buffer\n")
            print(f"Buffer size 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 size 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 out in Buffer

Buffer size after production: 1
1 consumed from Buffer

Buffer size after consumption: 0

2 out in Buffer

Buffer size after production: 1
3 out in Buffer

Buffer size after production: 2
2 consumed from Buffer

Buffer size after consumption: 1

4 out in Buffer

Buffer size after production: 2
```

Practical 7

Aim: - Write a Python program that implements the banker's algorithm.

Code: -

```
n=int(input("Enter the no. of processor: - "))
m=int(input("Enter the no. 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)
```

```
#Calculate need matrix
for i in range(n):
   rowinput=[]
   for j in range(m):
      x=Max[i][j]-Allocation[i][j]
      rowinput.append(x)
   Need.append(rowinput)
Resources=[]
print("Enter the total resources: - ")
for i in range(m):
   x=int(input())
   Resources.append(x)

Available=[]
for j in range(m):
   x=0
   for i in range(n):
      x+=Allocation[i][j]
   x=Resources[j]-x
   Available.append(x)

#Safety Algorithmn
Work=Available.copy()
finish=[0]*n
```

```
Sequence=[]
alldone=False
attempt=0
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 no. of processor: - 5
Enter the no. of resources: - 3
Enter the Allocation Matrix: -
0
1
0
2
0
0
3
0
2
2
1
1
0
0
2
Enter the Max Matrix: -
7
5
3
3
2
2
9
0
2
2
2
2
4
3
3
Enter the total resources: -
10
5
```

```
7
System is in Safe State
The sequence is  [1, 3, 4, 0, 2]
```