Distributed System Lab

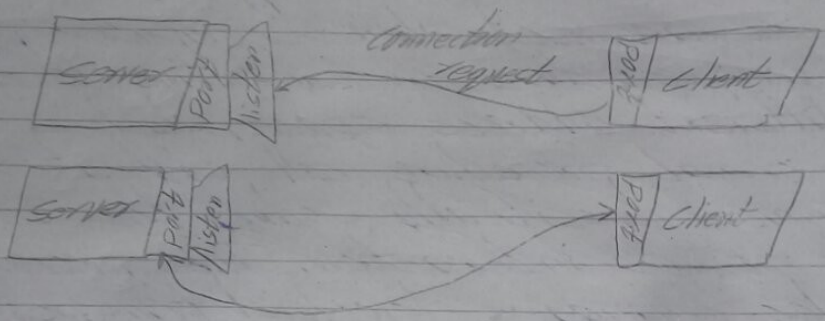
Theory

1) RPC

It is an interprocess communication technique, which is used for client-server applications. Its mechanisms are used when a computer program causes a procedure or subroutine to execute in a different address space, which is coded as a normal procedure call without the programmer specifically coding the details for the remote interaction. This procedure calls also manages low-level transport protocol, such as User Datagram Protocol, TCP/IP, etc. It is used for carrying the message data between programs.

ii) Sockets

It is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.



iii) Lamport's Logical Clock

The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed computer system. As different node or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead, and conceptually provide a starting point for the more advanced vector clock method.

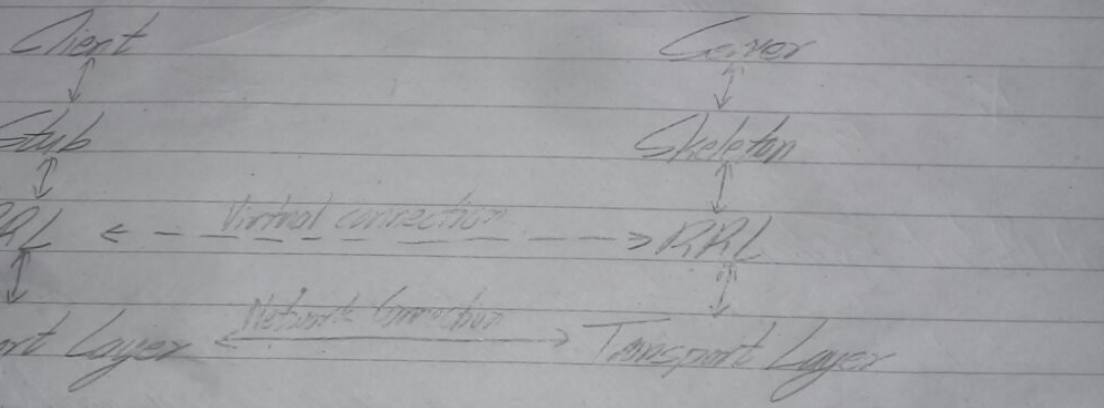
The algorithm follows some simple rules:-

→ A process increments its counter before

each event in that process
→ When a process sends a message, it includes its counter value with the message.
→ On receiving a message, the counter of the recipient is updated, if necessary to the greater of its current counter and the timestamp in the received message. The counter is the incremented by 1 before the message is considered received.

iv) RMI

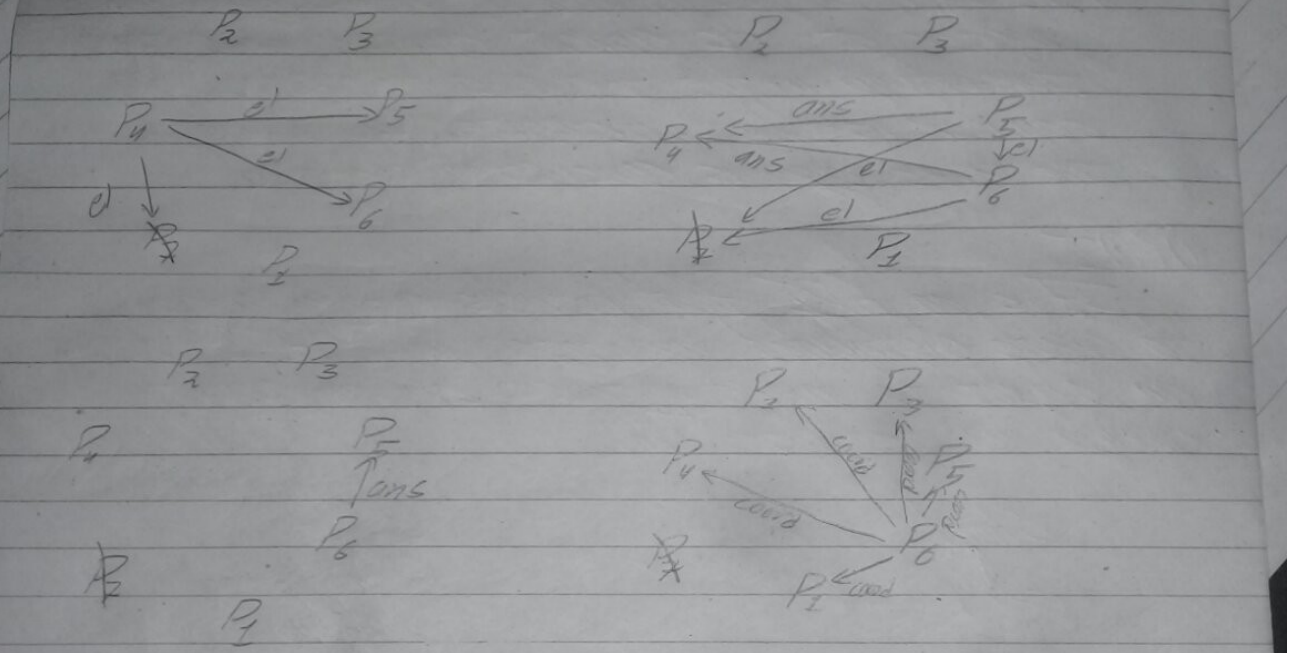
It is an API that provides a mechanism to create distributed application in JAVA. It allows an object to invoke methods on an object running in another JVM.



v) Bully Algorithm

It is an algorithm which is used to select coordinator when existing coordinator fails. It can be

done by election process. By sending information to processes with higher priority, for each information recieved, the process with highest priority can be determined



vii) Mutual Exclusion

In order to access resources, Processes need to enter critical section and when other enter this section at the same time it can cause concurrency problem. Hence, to allow only one process to enter critical section at a time, Mutual Exclusion is used.

RPC program output:-

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
Enter any Number
5
Enter another Number
2
10.0

Process finished with exit code 0
```

This program was built using sockets. Following is the code for finding possible combinations:-

```
class Combination{
    private int factorial(int n) {
        int temp = n, prod = 1;
        while (temp != 1) {
            prod *= temp;
            temp--;
        }
        return prod;
    }
    public float combinationCalc(int n, int r) { return (float)factorial(n)/(factorial( n: n-r) * factorial(r)); }

}
```

This is server side code for combination finding.

```
System.out.println("Server is Ready!");
Scanner sc = new Scanner(so.getInputStream());
number1 = sc.nextInt();  // Stream input from client
number2 = sc.nextInt();

Combination cb = new Combination();

result = cb.combinationCalc(number1, number2);    // manipulation to result

//output to stream
PrintStream p = new PrintStream(so.getOutputStream());
p.println(result);
```

This is RMI program's output:-

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
Enter two numbers to Multiply
5
2
Combination is 10.0
Result is 10

Process finished with exit code 0
```

This is how registry is implemented in client side

```
Registry reg = LocateRegistry.getRegistry( s: "localhost",  i: 8000);
multiply mul = (multiply) reg.lookup( s: "server");
System.out.println("Combination is " + mul.combination(num1, num2));
System.out.println("Result is " + mul.multiply(num1, num2));
```

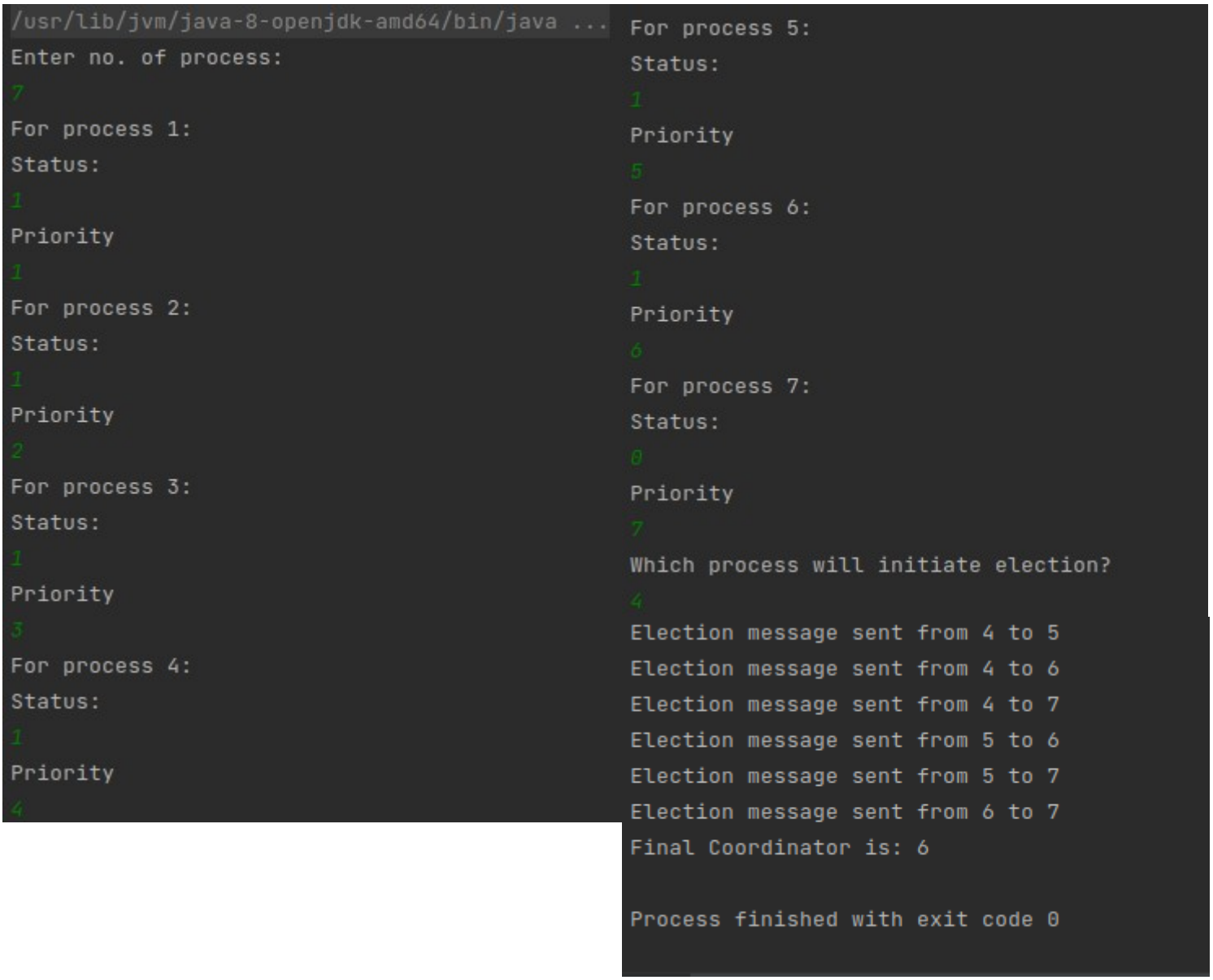
Following is the code for implementation of tasks

```
public int multiply(int n1, int n2) throws RemoteException{
    return n1 * n2;
}

public int factorial(int n1) throws RemoteException{
    int temp = n1, prod = 1;
    while(temp != 1){
        prod *= temp;
        temp--;
    }
    return prod;
}
```

```
public float combination(int n, int r) throws RemoteException {
    return (float)factorial(n)/(factorial( n1: n-r) * factorial(r));
}
```

Bully Algorithm Output



Mutual Exclusion in C program used library such as dos.h and conio.h which is not applicable for linux hence it was implemented in python language

```
from time import sleep
from threading import Thread
class mutExec:
    def __init__(self):
        self.cs = 0
        self.pro = 0
        self.run = 5.0
        self.key = 'a'
        self.inp = False
        self.newInp = True
```

```
        self.time = 1
        print("Press a key except q to enter to CS.")
        print("Press q to exit.\n")
    def processStart(self):
        while(not self.inp):
            if not self.cs == 0:
                self.t2 = self.time
                if(self.t2-self.t1 > self.run):
                    print("Process ", self.pro-1)
                    print("Exit CS.")
                    self.cs = 0
                    if( self.key=='q' or self.key=='Q'):
                        break
                else:
                    print("Running process", self.pro-1)
                    print(self.t2-self.t1, "Times done of ", self.run)
            if(self.key!='q' and self.newInp):
                if(self.cs!=0):
                    print("Error Another process in CS")
                    print("Please.....................")
                else:
                    print("Process ", self.pro)
                    print("Entered CS")
                    self.cs = 1
                    self.pro+=1
                    self.t1 = self.time
                    self.newInp=False
            self.time += 1
            sleep(5)
    def keypress(self):
        inp = input()
        if inp:
            self.inp = True
            self.key = inp
            self.newInp = True

    def runProgram(self):
        while(self.key!='q'):
            process = Thread(target=self.processStart)
            interrupt = Thread(target=self.keypress)
            self.inp = False
            process.start()
            interrupt.start()
            process.join()
            interrupt.join()

if __name__ == "__main__":
    main = mutExec()
    main.runProgram()
```

Mutual Exclusion algorithm output:-

```
Press a key except q to enter to CS.
Press q to exit.

Process  0
Entered CS
Running process 0
1 Times done of  5.0
w
Running process 0
2 Times done of  5.0
Error Another process in CS
Please.....................
Running process 0
3 Times done of  5.0
Error Another process in CS
Please.....................
e
Running process 0
4 Times done of  5.0
Error Another process in CS
Please.....................
Running process 0
5 Times done of  5.0
Error Another process in CS
Please.....................
Process  0
Exit CS.
Process  1
Entered CS
Running process 1
1 Times done of  5.0
q
```

As for Lamport's clock,

```
enter the events : 4 4
enter the dependency matrix:
        enter 1 if e1->e2
        enter -1, if e2->e1
        else enter 0

       e21     e22     e23     e24
 e11   0       1       1       1

 e12   -1      1       1       1

 e13   -1      0       1       1

 e14   -1      -1      -1      -1

P1 : 1 2 3 6
P2 : 1 3 4 5

Process returned 0 (0x0)   execution time : 44.466 s
Press ENTER to continue.
```