EXPERIMENT NO.-01

AIM-

Write a Program in Java to implement RPC

THEORY-

Remote Procedure Call (RPC) is a mechanism that allows a program to execute procedures or functions located on another system or process as if they were local.
It hides the details of the network communication so that the programmer can simply call a function without worrying about how the request reaches the remote machine.

In Java, RPC can be implemented using Java RMI (Remote Method Invocation).
RMI lets objects invoke methods running on another JVM, which may be on a different computer.

Main components of RPC using Java RMI:

1. Remote Interface – declares the methods that can be called remotely.
2. Server – implements the remote interface and registers the service with the RMI registry.
3. Client – looks up the remote service and calls the method.
4. RMI Registry – acts like a name service that allows clients to locate remote objects.

CODE-

```
//AddInterface.java

import java.rmi.*;

public interface AddInterface extends Remote {
   public int addNumbers(int a, int b) throws RemoteException;
}

//        AddServer.java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class AddServer extends UnicastRemoteObject implements AddInterface {

   AddServer() throws RemoteException {
      super();
   }

   public int addNumbers(int a, int b) throws RemoteException {
      System.out.println("Server: Adding " + a + " + " + b);
      return a + b;
```

```
    }

    public static void main(String[] args) {
        try {
            AddServer server = new AddServer();
            LocateRegistry.createRegistry(1099);
            Naming.rebind("AddService", server);
            System.out.println("Server is ready...");
        } catch (Exception e) {
            System.out.println("Server error: " + e);
        }
    }
}

//AddClient.java
import java.rmi.*;

public class AddClient {
    public static void main(String[] args) {
        try {
            AddInterface stub = (AddInterface) Naming.lookup("rmi://localhost/AddService");
            int result = stub.addNumbers(15, 25);
            System.out.println("Client: The sum is " + result);
        } catch (Exception e) {
            System.out.println("Client error: " + e);
        }
    }
}
```

OUTPUT-

```
Server is ready...
Server: Adding 15 + 25
```

```
Client: The sum is 40

Process finished with exit code 0
```

LEARNING OUTCOME-

EXPERIMENT NO.-02

AIM-

Implement the concept of Remote Method Invocation in Java.

THEORY-

Remote Method Invocation (RMI) in Java allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM.
It provides a simple and direct way to implement Remote Procedure Calls (RPC) in Java while maintaining object-oriented principles.

Important Concepts

1. **Remote Interface**
   Declares methods that can be called remotely. It must extend java.rmi.Remote and each method must throw RemoteException.
2. **Remote Object (Server Implementation)**
   Implements the remote interface and defines the logic for the methods. It must extend UnicastRemoteObject to make it available for remote communication.
3. **RMI Registry**
   A name service where remote objects are registered so that clients can locate them by name.
4. **Client**
   Looks up the remote object in the registry and invokes the remote methods as if they were local methods.
5. **Communication**
   RMI uses stubs (client-side proxies) and skeletons (server-side dispatchers) to handle all network communication automatically.

CODE-

```
//AddInterface.java

import java.rmi.*;

public interface AddInterface extends Remote {
    int addNumbers(int a, int b) throws RemoteException;
}

//AddServer.java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class AddServer extends UnicastRemoteObject implements AddInterface {

   protected AddServer() throws RemoteException {
```

```
        super();
    }

    public int addNumbers(int a, int b) throws RemoteException {
        System.out.println("Server adding: " + a + " + " + b);
        return a + b;
    }

    public static void main(String[] args) {
        try {
            AddServer server = new AddServer();
            LocateRegistry.createRegistry(1099); // Start RMI registry
            Naming.rebind("AddService", server);
            System.out.println("Server is ready...");
        } catch (Exception e) {
            System.out.println("Server error: " + e);
        }
    }
}
//AddClient.java
import java.rmi.Naming;

public class AddClient {
    public static void main(String[] args) {
        try {
            AddInterface stub = (AddInterface) Naming.lookup("rmi://localhost/AddService");
            int result = stub.addNumbers(15, 25);
            System.out.println("Client: The sum is " + result);
        } catch (Exception e) {
            System.out.println("Client error: " + e);
        }
    }
}
```

OUTPUT-

```
Server is ready...
Server: Adding 15 + 25
```

```
Client: The sum is 40

Process finished with exit code 0
```

LEARNING OUTCOME-

EXPERIMENT NO.-03

AIM-

Write a java program to implement Lamport's Logical clock

THEORY-

In a distributed system, there is no global clock, and different processes may have different local times.
To maintain the correct order of events that occur across processes, Lamport introduced a logical clock mechanism.The main idea is to assign a numerical timestamp to each event in such a way that if one event happens before another, the timestamp reflects this ordering.

Rules for Lamport's logical clock:

1. Each process maintains a local logical clock. Initially, all clocks are set to zero.
2. Whenever an event occurs in a process, it increments its clock by 1.
3. When a message is sent from one process to another, the message carries the sender's timestamp.
4. When a process receives a message, it sets its clock to be greater than both its current clock value and the received timestamp, then increments it by 1.

This ensures that if event A happens before event B, then Clock(A) < Clock(B).

Lamport's clock only preserves the order of causally related events, not the exact physical time.

CODE-

```
import java.util.Scanner;
public class LamportClock {
   private int clock;

   public LamportClock() {
      clock = 0;
   }
   // internal event
   public void event() {
      clock++;
      System.out.println("Internal event occurred. Clock = " + clock);
   }
   // send message
   public int sendMessage() {
      clock++;
      System.out.println("Message sent. Clock = " + clock);
      return clock;
   }
```

```
    // receive message
    public void receiveMessage(int receivedTime) {
        clock = Math.max(clock, receivedTime) + 1;
        System.out.println("Message received. Updated Clock = " + clock);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LamportClock process1 = new LamportClock();
        LamportClock process2 = new LamportClock();

        System.out.println("Initial clocks -> P1: " + 0 + "  P2: " + 0);

        System.out.println("\nP1 performs an internal event:");
        process1.event();

        System.out.println("\nP1 sends a message to P2:");
        int msgTime = process1.sendMessage();

        System.out.println("\nP2 receives the message:");
        process2.receiveMessage(msgTime);

        System.out.println("\nP2 performs an internal event:");
        process2.event();

        System.out.println("\nFinal Clock values:");
        System.out.println("P1 clock: " + process1.clock);
        System.out.println("P2 clock: " + process2.clock);
    }
}
```

OUTPUT-

```
Initial clocks -> P1: 0  P2: 0

P1 performs an internal event:
Internal event occurred. Clock = 1

P1 sends a message to P2:
Message sent. Clock = 2

P2 receives the message:
Message received. Updated Clock = 3

P2 performs an internal event:
Internal event occurred. Clock = 4

Final Clock values:
P1 clock: 2
P2 clock: 4

Process finished with exit code 0
```

LEARNING OUTCOME-

EXPERIMENT NO.-04

AIM-

Implement mutual exclusion service using Lamport's Mutual Exclusion Algorithm

THEORY-

In a distributed system, multiple processes may need to access a shared resource such as a file or printer.
Mutual exclusion ensures that only one process can use the shared resource at a time.

Lamport's Mutual Exclusion Algorithm is a distributed approach based on message passing and Lamport's logical clocks.
It guarantees that all processes agree on the order of requests for the critical section without using a centralized coordinator.

The algorithm works as follows:

1. Each process maintains a logical clock.
2. When a process wants to enter its critical section, it sends a request message containing its process ID and current clock value to all other processes.
3. Each process that receives the request:
    - Places the request in its local request queue.
    - Sends an acknowledgment message back to the requesting process.
4. A process can enter its critical section only when:
    - It has received an acknowledgment from every other process, and
    - Its request is the smallest (earliest timestamp) in its queue.
5. After leaving the critical section, the process sends a release message to all others, and they remove that request from their queues.

This algorithm ensures mutual exclusion, fairness, and absence of deadlock.

CODE-

```
import java.util.*;

class LamportProcess implements Comparable<LamportProcess> {
    int processId;
    int timestamp;

    LamportProcess(int processId, int timestamp) {
        this.processId = processId;
        this.timestamp = timestamp;
    }

    @Override
    public int compareTo(LamportProcess other) {
        if (this.timestamp == other.timestamp)
```

```
            return this.processId - other.processId;
        return this.timestamp - other.timestamp;
    }
}

public class LamportMutualExclusion {
    private int clock;
    private int processId;
    private PriorityQueue<LamportProcess> requestQueue;

    public LamportMutualExclusion(int processId) {
        this.clock = 0;
        this.processId = processId;
        this.requestQueue = new PriorityQueue<>();
    }

    // simulate sending a request
    public void requestCS() {
        clock++;
        LamportProcess request = new LamportProcess(processId, clock);
        requestQueue.add(request);
        System.out.println("Process " + processId + " sends request with timestamp " + clock);
    }

    // simulate receiving request from another process
    public void receiveRequest(int senderId, int senderTime) {
        clock = Math.max(clock, senderTime) + 1;
        requestQueue.add(new LamportProcess(senderId, senderTime));
        System.out.println("Process " + processId + " received request from P" + senderId + "
(time=" + senderTime + ")");
    }

    // check if this process can enter the critical section
    public boolean canEnterCS(int totalProcesses) {
        LamportProcess first = requestQueue.peek();
        return (first != null && first.processId == processId && requestQueue.size() ==
totalProcesses);
    }

    // release critical section
    public void releaseCS() {
        requestQueue.poll();
        clock++;
        System.out.println("Process " + processId + " releases critical section (clock=" + clock +
")");
    }
```

```
    public static void main(String[] args) {
        int totalProcesses = 3;

        LamportMutualExclusion p1 = new LamportMutualExclusion(1);
        LamportMutualExclusion p2 = new LamportMutualExclusion(2);
        LamportMutualExclusion p3 = new LamportMutualExclusion(3);

        System.out.println("=== Lamport Mutual Exclusion Simulation ===\n");

        // P1 requests critical section
        p1.requestCS();
        p2.receiveRequest(1, p1.clock);
        p3.receiveRequest(1, p1.clock);

        // P2 requests after some time
        p2.requestCS();
        p1.receiveRequest(2, p2.clock);
        p3.receiveRequest(2, p2.clock);

        // P3 requests after some time
        p3.requestCS();
        p1.receiveRequest(3, p3.clock);
        p2.receiveRequest(3, p3.clock);

        System.out.println("\nChecking who can enter critical section...\n");

        if (p1.canEnterCS(totalProcesses)) {
            System.out.println("Process 1 enters Critical Section");
            p1.releaseCS();
        }
        if (p2.canEnterCS(totalProcesses)) {
            System.out.println("Process 2 enters Critical Section");
            p2.releaseCS();
        }
        if (p3.canEnterCS(totalProcesses)) {
            System.out.println("Process 3 enters Critical Section");
            p3.releaseCS();
        }
    }
}
```

OUTPUT-

```
=== Lamport Mutual Exclusion Simulation ===

Process 1 sends request with timestamp 1
Process 2 received request from P1 (time=1)
Process 3 received request from P1 (time=1)
Process 2 sends request with timestamp 3
Process 1 received request from P2 (time=3)
Process 3 received request from P2 (time=3)
Process 3 sends request with timestamp 5
Process 1 received request from P3 (time=5)
Process 2 received request from P3 (time=5)

Checking who can enter critical section...

Process 1 enters Critical Section
Process 1 releases critical section (clock=7)

Process finished with exit code 0
```

LEARNING OUTCOME-

EXPERIMENT NO.-05

AIM-

Install Hadoop on Windows

THEORY-

Hadoop is an open-source framework developed by Apache that allows for distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from a single server to thousands of machines, each offering local computation and storage.

Hadoop consists mainly of two components:

1. Hadoop Distributed File System (HDFS): Provides high-throughput access to application data.
2. MapReduce: A programming model for processing large data sets in parallel.

To install Hadoop on Windows, we set up a single-node pseudo-distributed cluster. This means Hadoop runs on a single machine but simulates distributed behaviour.

Steps to Install Hadoop on Windows:-

1. **Install Java**
   • Download and install JDK (Java 8 or Java 11).
   • Set environment variables:
   JAVA_HOME = C:\Program Files\Java\jdk1.8.0_xx
   PATH = %JAVA_HOME%\bin

2. **Download Hadoop**
   • Visit https://hadoop.apache.org/releases.html
   • Download the latest stable Hadoop binary.
   • Extract the folder to:
   C:\hadoop

3. **Set Hadoop Environment Variables**
   Open System Properties → Environment Variables → Add:
   HADOOP_HOME = C:\hadoop
   PATH = %HADOOP_HOME%\bin
   Then edit the file:
   C:\hadoop\etc\hadoop\hadoop-env.cmd
   Add this line:
   set JAVA_HOME=C:\Program Files\Java\jdk1.8.0_xx
4. **Configure Hadoop XML Files**
   Open files inside `C:\hadoop\etc\hadoop` and make the following changes:
   <configuration>
   <property>

```
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>

<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/C:/hadoop/data/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/C:/hadoop/data/datanode</value>
    </property>
</configuration>

<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>

<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

5. **Format the Hadoop NameNode**

   Open Command Prompt and run:

   hdfs namenode –format

6. **Start Hadoop Services**

   Run the following commands one by one:

   start-dfs.cmd

   start-yarn.cmd

EXPERIMENT NO.-06

AIM-

Run a simple application on single node Hadoop Cluster.

THEORY-

Hadoop is an open source framework used to store and process large datasets using a distributed computing model. A single node Hadoop cluster means that all the Hadoop daemons such as NameNode, DataNode, ResourceManager, and NodeManager run on a single machine. This is useful for learning, testing, and development.

To run a simple application on a single node Hadoop cluster, we first need to set up Hadoop and configure it for single node operation. Then we can write a simple MapReduce program in Java. MapReduce is a programming model used for processing large datasets in parallel. It consists of two main functions: Mapper and Reducer. Mapper processes input data and produces key-value pairs, while Reducer aggregates the values corresponding to each key.

CODE-

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.util.StringTokenizer;

public class WordCount {

   public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable> {

      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(Object key, Text value, Context context
      ) throws IOException, InterruptedException {
         StringTokenizer itr = new StringTokenizer(value.toString());
         while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
         }
      }
   }
```

```
    public static class IntSumReducer
           extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context
        ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

OUTPUT-

Input.txt:
hello world hello hadoop

output:
hadoop 1
hello 2
world 1

LEARNING OUTCOME-

EXPERIMENT NO.-07

AIM-

Install Google App Engine / AWS and develop a simple web application.

THEORY-

Google App Engine (GAE) and AWS (Amazon Web Services) provide cloud platforms to deploy and run applications without managing physical servers. They allow scalable, reliable, and managed hosting for web applications.

A simple web application can be built using Python (Flask) or Java (Spring Boot) and deployed on GAE or AWS. The workflow is:

1. Develop the application locally using your preferred framework.
2. Set up the cloud environment (GAE or AWS Elastic Beanstalk / AWS Lambda).
3. Deploy the application to the cloud using the provided CLI tools.
4. Access the application via a public URL.

Google App Engine supports standard and flexible environments. AWS Elastic Beanstalk automates deployment, scaling, and monitoring.

STEPS-

Step 1: Log in and Navigate to S3

1. Log in to your AWS Management Console.
2. In the main search bar at the top, you would type "S3" and press Enter.
3. You would click on the "S3" service link that appears.

Step 2: Create Your S3 Bucket

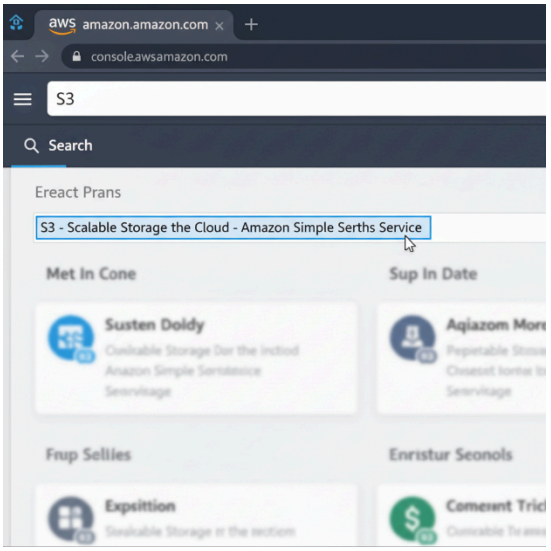
This bucket will hold all your website's files.

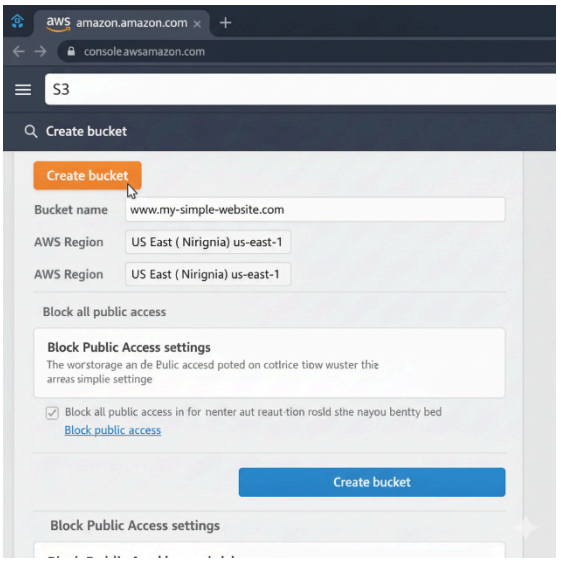
1. On the S3 console page, you'd see a big orange button labeled "Create bucket". You would click it.
2. Configure the bucket:
    - Bucket name**:** You'd type in a globally unique name. A best practice is to name it exactly like your domain name (e.g., `www.my-simple-website.com`).
    - AWS Region: You'd select a region from the dropdown (e.g., `us-east-1` or one close to your users).
    - Block Public Access settings: This is the most important part. By default, all public access is blocked. For a public website, you must uncheck the box labeled "Block all public access".
    - You'd see a warning a new checkbox to acknowledge that you are turning off this setting. You would check that box.
3. You would scroll to the bottom and click the "Create bucket" button.

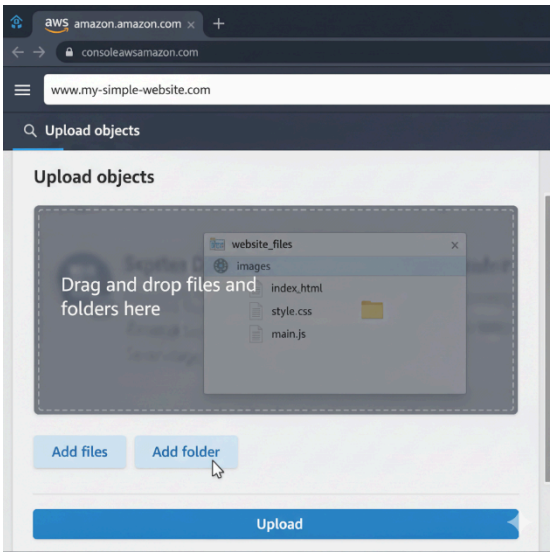
Step 3: Upload Your Website Files

1. You'd now see your new bucket in the list. You would click on its name.
2. Inside the bucket, you'd see an "Upload" button. You would click it.
3. A new screen would appear. You could either drag-and-drop your website files directly onto the page or click "Add files" and "Add folder".
4. Once your files are listed, you would scroll down and click the "Upload" button.

OUTPUT-







LEARNING OUTCOME-

EXPERIMENT NO.-08

AIM-

Launch Web application using Google App Engine / AWS

THEORY-

Cloud computing provides on-demand access to computing resources such as servers, storage, databases, and networking over the Internet. Two popular platforms for deploying web applications are Google App Engine (GAE) and Amazon Web Services (AWS).

1. Google App Engine (GAE):
   GAE is a Platform as a Service (PaaS) that allows developers to build and deploy web applications without managing the underlying infrastructure. It automatically handles scaling, load balancing, and monitoring. Developers can deploy applications written in Python, Java, Node.js, PHP, Go, and other supported languages. The deployment is done using the Google Cloud SDK and a configuration file (app.yaml) that defines the runtime and environment. Once deployed, the application is hosted on Google's infrastructure and can be accessed using a public URL.

2. Amazon Web Services (AWS):
   AWS provides several services for web app deployment, such as:

- Amazon S3 for hosting static websites
- Amazon EC2 for running applications on virtual servers
- Elastic Beanstalk for easy deployment of full-stack applications (supports Node.js, Java, Python, and more)
  Developers can deploy applications using the AWS Management Console, AWS Command Line Interface (CLI), or Elastic Beanstalk CLI. AWS automatically manages resource provisioning, load balancing, and auto-scaling to ensure efficient performance.

Key Concepts:
Cloud Deployment refers to making an application available on cloud servers.
Scalability means the ability of an application to handle increased load automatically.
PaaS (Platform as a Service) provides a platform where developers focus on writing code while the service provider manages the infrastructure.
Pay-as-you-go means users are charged only for the resources they consume.

STEPS-

Step 1: Enable Static Website Hosting

Now you tell S3 that this bucket should act like a website.

1. Click on your bucket's name to go back to its main page (if you're not already there).
2. You would click on the "Properties" tab.
3. You would scroll all the way down to the bottom, where you'd find a section labeled "Static website hosting".
4. You would click the "Edit" button in that section.
5. A new screen would appear. You would select "Enable" for Static website hosting.

6. Two new fields would appear. In the "Index document" field, you would type the name of your main file, which is almost always `index.html`.
7. You would click the "Save changes" button.

Step 2: Make the Files Publicly Viewable

This is the final, critical step. Even though you enabled website hosting, no one can see the files yet. You need to create a "Bucket Policy" (a small piece of JSON code) to grant read-only access to the public.

1. You would click on the "Permissions" tab for your bucket.
2. You'd see a section called "Bucket policy". You would click the "Edit" button.
3. A text editor would open. You would paste in the following JSON policy.

   JSON

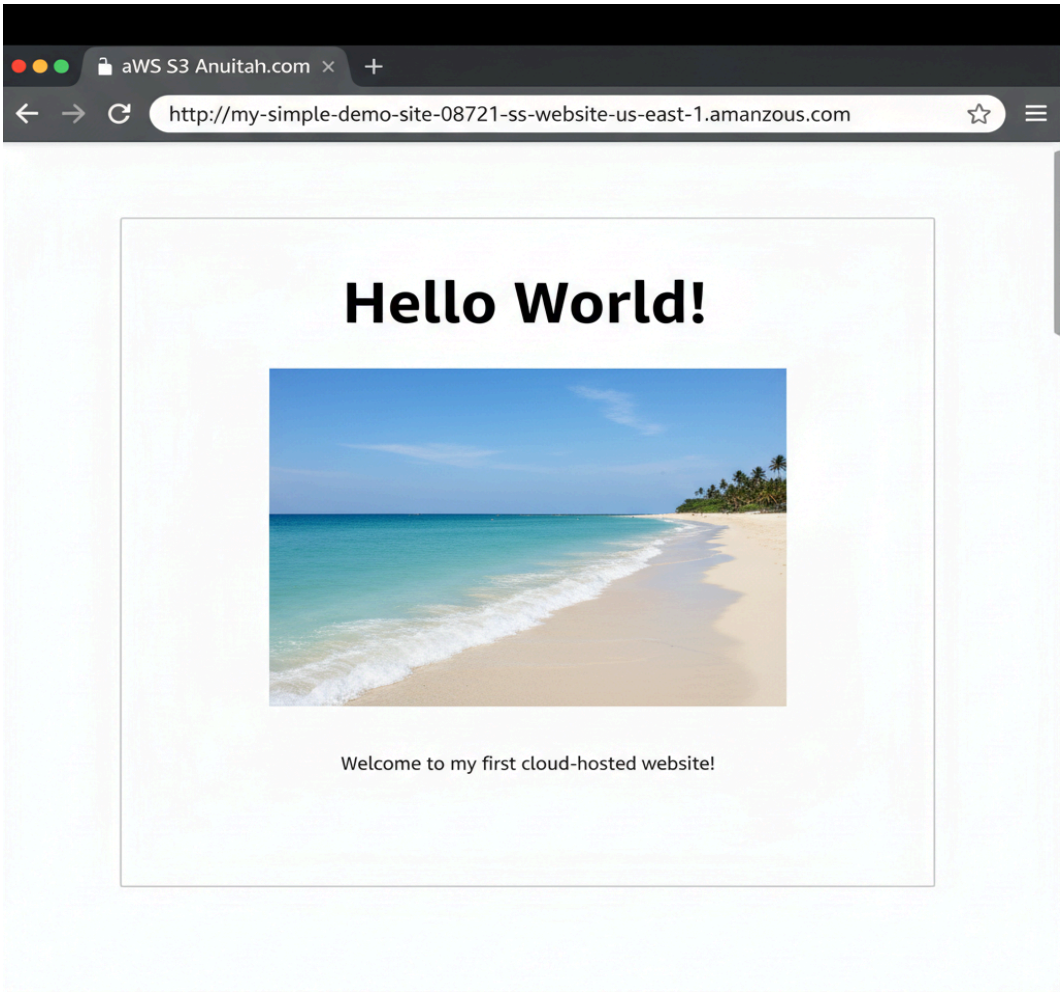
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::YOUR-BUCKET-NAME-HERE/*"
    }
  ]
}
```

4. Crucially, you would replace `YOUR-BUCKET-NAME-HERE` with the exact name of your bucket.
5. You would then click the "Save changes" button. You might see a "public" tag appear next to your bucket name.

Step 3: Find Your Website!

1. Go back to the "Properties" tab.
2. Scroll down to the "Static website hosting" section again.
3. Now, you would see a URL listed at the bottom of that section. It would look something like this: http://your-bucket-name.s3-website.your-region.amazonaws.com
4. You would copy this URL, paste it into your web browser, and your simple website would be live on the internet.

OUTPUT-



LEARNING OUTCOME-

EXPERIMENT NO.-09

AIM-

Install Virtualbox / VMware Workstation with different flavours of linux on windows.

THEORY-

Virtualization is a technology that allows multiple operating systems to run on a single physical machine by creating virtual machines (VMs). Each virtual machine functions as an independent computer with its own operating system and resources such as CPU, memory, and storage. This helps in testing, learning, and running multiple environments without affecting the host system.

VirtualBox and VMware Workstation are two popular virtualization tools used for this purpose.

- VirtualBox is a free and open-source virtualization software developed by Oracle. It supports various operating systems like Windows, Linux, and macOS as both host and guest systems.
- VMware Workstation is a commercial virtualization product that offers advanced features such as better hardware integration, snapshot management, and enhanced performance.

Different flavours of Linux such as Ubuntu, Fedora, Kali Linux, and CentOS can be installed on these virtual machines. Each flavour provides a different environment and set of tools useful for learning system administration, networking, and software development.

Advantages of virtualization include better utilization of hardware resources, easier software testing, isolation between environments, and the ability to run multiple OS platforms simultaneously.

STEPS-

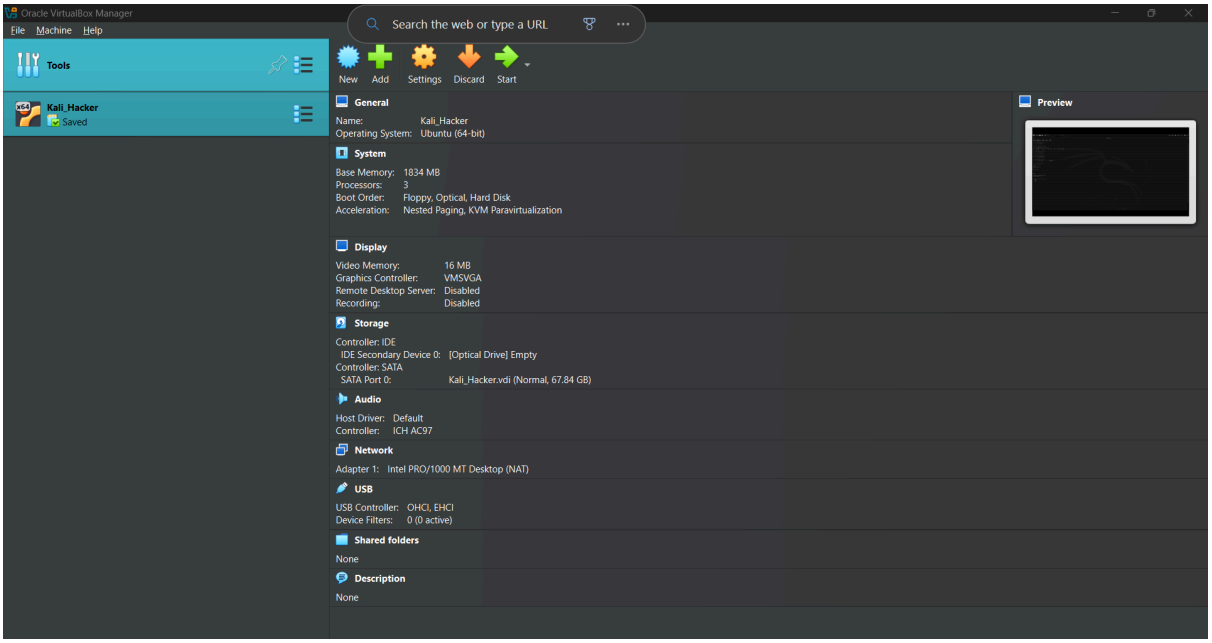
Steps to Download and Install VirtualBox or VMware Workstation:

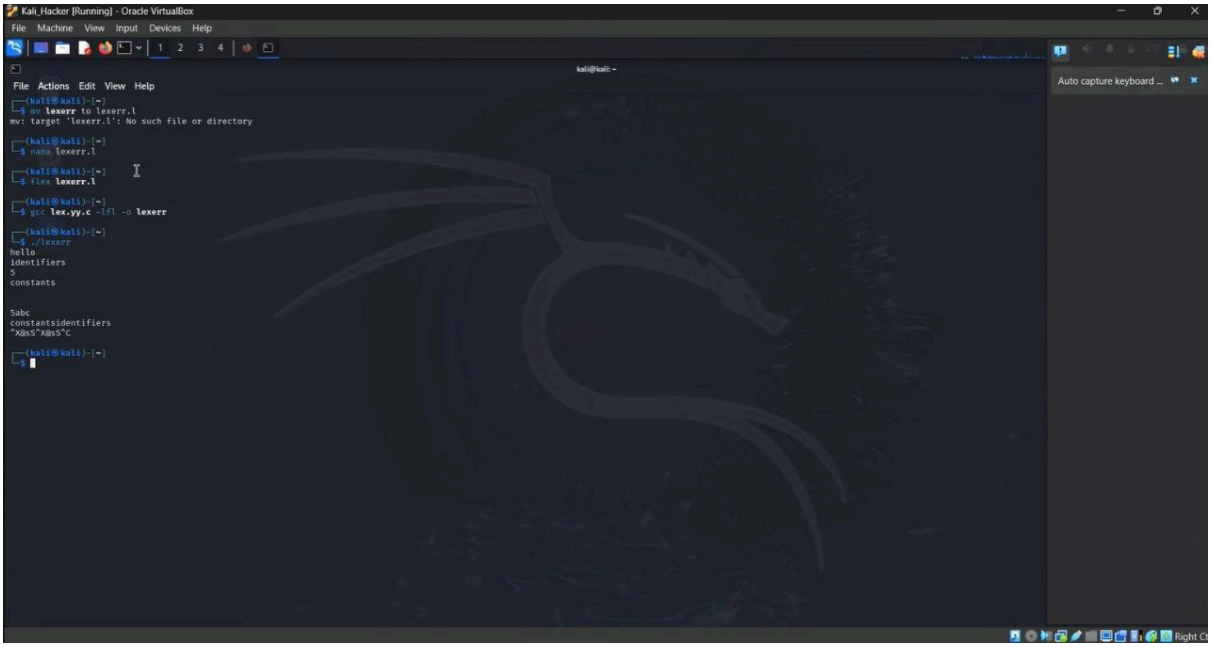
1. Open any web browser on the Windows system.
2. For VirtualBox:
    - Go to the official website https://www.virtualbox.org
    - Click on Downloads
    - Choose Windows hosts and download the installer
    - Run the setup file and follow on-screen instructions to complete installation
3. For VMware Workstation:
    - Visit https://www.vmware.com
    - Navigate to the Products section and select VMware Workstation Player
    - Download the Windows version and install it using the default settings
4. After installation, download a Linux ISO image from the official website of the chosen distribution such as:

- Ubuntu: https://ubuntu.com/download
- Fedora: https://getfedora.org
- Kali Linux: https://www.kali.org/get-kali
- CentOS: https://www.centos.org/download

5. Open VirtualBox or VMware Workstation and create a new virtual machine.
6. Allocate the desired amount of RAM, CPU cores, and disk space.
7. Select the downloaded Linux ISO file as the boot image.
8. Start the virtual machine and follow the installation steps of the selected Linux distribution.
9. Once the installation is complete, log in to the Linux system and verify that it runs properly inside the virtual machine.

OUTPUT-





LEARNING OUTCOME-

EXPERIMENT NO.-10

AIM-

Simulate a cloud scenario using CloudSim and run a scheduling algorithm

THEORY-

1. **CloudSim:** A Java-based simulation library. It's used to model cloud computing environments. We use it to test scheduling policies and resource allocation strategies without needing real hardware.
2. **Core Components:**
    - **Datacenter:** Represents the physical infrastructure (e.g., a data center building). It contains Hosts.
    - **Host:** A single physical server. It has defined resources like CPU cores (called Pe), RAM, and storage.
    - **VM (Virtual Machine):** A virtual instance that runs on a Host. We define its specs, like MIPS (processing power), RAM, etc.
    - **DatacenterBroker:** An entity that acts for the cloud user. It requests VMs from the Datacenter and assigns tasks (Cloudlets) to them.
    - **Cloudlet:** The actual task (a job or application) that needs to be executed. Its main property is its length in Million Instructions (MI).
3. Scheduling Algorithm: CloudletSchedulerTimeShared

This is the policy inside the Virtual Machine that decides how to run the tasks assigned to it.

- **How it works:** It's a "parallel" or "concurrent" scheduler.
- Instead of running one task at a time (like a queue), it runs all its assigned tasks at once.
- It divides the VM's total processing power (MIPS) equally among all the tasks that are currently running.
- **Example:** If a VM has 500 MIPS and 10 tasks are running, each task gets $500 / 10 = 50$ MIPS.

CODE-

```
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared; // This is the algorithm we are testing
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
```

```
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared; // This is the Host-level scheduler
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

public class CloudSimBasicExample {

    private static List<Vm> vmList;
    private static List<Cloudlet> cloudletList;

    public static void main(String[] args) {
        Log.printLine("Starting CloudSim Time-Shared Example...");

        try {
            // 1. Initialize CloudSim
            int num_user = 1;
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;
            CloudSim.init(num_user, calendar, trace_flag);

            // 2. Create Datacenter
            Datacenter datacenter0 = createDatacenter("Datacenter_0");

            // 3. Create Broker
            DatacenterBroker broker = createBroker();
            int brokerId = broker.getId();

            // 4. Create 2 Virtual Machines
            vmList = new ArrayList<>();
            int vmid = 0;
            int mips = 500; // VM processing power
            long size = 10000;
            int ram = 512;
```

```
        long bw = 1000;
        int pesNumber = 1; // Number of CPUs for the VM
        String vmm = "Xen";

        // Create VM 1 with the TimeShared scheduler
        Vm vm1 = new Vm(vmid++, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
        vmList.add(vm1);

        // Create VM 2 with the TimeShared scheduler
        Vm vm2 = new Vm(vmid++, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
        vmList.add(vm2);

        broker.submitVmList(vmList); // Submit VMs to the broker

        // 5. Create 20 Cloudlets (Tasks)
        cloudletList = new ArrayList<>();
        int cloudletId = 0;
        long length = 400000; // Task length in Million Instructions (MI)
        long fileSize = 300;
        long outputSize = 300;
        UtilizationModel utilizationModel = new UtilizationModelFull();

        for (int i = 0; i < 20; i++) {
            Cloudlet cloudlet = new Cloudlet(cloudletId++, length, pesNumber, fileSize,
outputSize, utilizationModel, utilizationModel, utilizationModel);
            cloudlet.setUserId(brokerId);

            // Assign tasks to VMs in a round-robin way
            if (i % 2 == 0) {
                cloudlet.setVmId(vm1.getId()); // Even tasks to VM 0
            } else {
                cloudlet.setVmId(vm2.getId()); // Odd tasks to VM 1
            }

            cloudletList.add(cloudlet);
        }
        broker.submitCloudletList(cloudletList); // Submit tasks to the broker

        // 6. Start the simulation
        Log.printLine("Starting simulation...");
        CloudSim.startSimulation();

        // 7. Stop the simulation
        CloudSim.stopSimulation();
```

```
            Log.printLine("Simulation finished!");

            // 8. Print the results
            List<Cloudlet> newList = broker.getCloudletReceivedList();
            printCloudletList(newList);

        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Error during simulation");
        }
    }
    /**
     * Creates a Datacenter.
     */
    private static Datacenter createDatacenter(String name) {
        List<Host> hostList = new ArrayList<>();
        List<Pe> peList = new ArrayList<>();
        int mips = 1000; // Host CPU MIPS
        peList.add(new Pe(0, new PeProvisionerSimple(mips))); // PE 0
        peList.add(new Pe(1, new PeProvisionerSimple(mips))); // PE 1 (Total 2000 MIPS)

        int hostId = 0;
        int ram = 2048;
        long storage = 1000000;
        int bw = 10000;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList,
                        new VmSchedulerTimeShared(peList) // Host scheduler
                )
        );
        // Datacenter config
        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double time_zone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;
        LinkedList<Storage> storageList = new LinkedList<>();
```

```
        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage,
costPerBw);
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }
    /**
     * Creates a DatacenterBroker.
     */
    private static DatacenterBroker createBroker() {
        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return broker;
    }
    /**
     * Prints the Cloudlet execution results.
     */
    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        String indent = "    ";
        Log.printLine();
        Log.printLine("OUTPUT");
        Log.printLine("Cloudlet ID" + indent + "STATUS" + indent
                + "Datacenter ID" + indent + "VM ID" + indent + "Time" + indent
                + "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (int i = 0; i < size; i++) {
            cloudlet = list.get(i);
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);
            if (cloudlet.getStatus() == Cloudlet.SUCCESS) {
                Log.print("SUCCESS");
                Log.printLine(indent + indent + cloudlet.getResourceId()
                        + indent + indent + cloudlet.getVmId()
```

```
                + indent + indent
                + dft.format(cloudlet.getActualCPUTime()) + indent
                + indent + dft.format(cloudlet.getExecStartTime())
                + indent + indent
                + dft.format(cloudlet.getFinishTime()));
        } else {
            Log.print("FAILED");
            Log.printLine();
        }
    }
  }
}
```

OUTPUT-

Output Clear

```
Starting CloudSim Time-Shared Example...
Starting simulation...
Simulation finished!

OUTPUT
Cloudlet ID    STATUS    Datacenter ID    VM ID    Time    Start Time    Finish Time
     0          SUCCESS        2            0       800.00    0.10      800.10
     2          SUCCESS        2            0       800.00    0.10      800.10
     4          SUCCESS        2            0       800.00    0.10      800.10
     6          SUCCESS        2            0       800.00    0.10      800.10
     8          SUCCESS        2            0       800.00    0.10      800.10
     10         SUCCESS        2            0       800.00    0.10      800.10
     12         SUCCESS        2            0       800.00    0.10      800.10
     14         SUCCESS        2            0       800.00    0.10      800.10
     16         SUCCESS        2            0       800.00    0.10      800.10
     18         SUCCESS        2            0       800.00    0.10      800.10
     1          SUCCESS        2            1       800.00    0.10      800.10
     3          SUCCESS        2            1       800.00    0.10      800.10
     5          SUCCESS        2            1       800.00    0.10      800.10
     7          SUCCESS        2            1       800.00    0.10      800.10
     9          SUCCESS        2            1       800.00    0.10      800.10
     11         SUCCESS        2            1       800.00    0.10      800.10
     13         SUCCESS        2            1       800.00    0.10      800.10
     15         SUCCESS        2            1       800.00    0.10      800.10
     17         SUCCESS        2            1       800.00    0.10      800.10
     19         SUCCESS        2            1       800.00    0.10      800.10
```

LEARNING OUTCOME-