Implementation of Round Robin Task Scheduling in Both Time Shared and Space Shared CPU

AIM:

To implement the round robin task scheduling in both time shared and space shared CPU using CloudSim.

PROCEDURE:

1. Create a new project by selecting java console line application template and JDK 18.

2. Open project settings from the file menu of the options window.

3. Navigate to project dependencies and select on add external jars and then click on 'Browse' to open the path where you have unzipped the Cloudsim Jars and click on apply.

4. Create a java file with the cloudsim code to implement the round robin scheduling algorithm.

5. Run the application as a java file to see the output in the console below.

CODE: import   org.cloudbus.cloudsim.*;

```
import
org.cloudbus.cloudsim.core.CloudSim;
import java.util.*;
public class RoundRobinScheduler
{
  public static void main(String[] args) {
    try {
```

```java
        int numUser = 1; // number of cloud users
        Calendar calendar = Calendar.getInstance();
        boolean traceFlag = false; // mean trace
        events

        CloudSim.init(numUser, calendar, traceFlag);

        Datacenter              datacenter0              =
createDatacenter("Datacenter_0");

        DatacenterBroker broker = createBroker();
        int brokerId = broker.getId();

        List<Vm>    vmList    =    new
        ArrayList<>(); int vmId = 0; int
        mips = 1000; long size = 10000; //
        image size (MB) int ram = 512; //
        vm memory (MB) long bw = 1000;
        int pesNumber = 1; // number of CPUs
        String vmm = "Xen"; // VMM name

        for (int i = 0; i < 3; i++) {
    vmList.add(new Vm(vmId++, brokerId, mips, pesNumber, ram,
bw, size, vmm, new CloudletSchedulerTimeShared()));
        }
 broker.submitVmList(vmList);
        List<Cloudlet> cloudletList =
        new ArrayList<>(); int
        cloudletId = 0; long length =
        40000; long fileSize = 300;
        long outputSize = 300;
```

```java
            UtilizationModel      utilizationModel      =      new
      UtilizationModelFull();

      for (int i = 0; i < 6; i++) {
        Cloudlet cloudlet = new Cloudlet(cloudletId++, length,
pesNumber,        fileSize,        outputSize,        utilizationModel,
utilizationModel, utilizationModel); cloudlet.setUserId(brokerId);
cloudletList.add(cloudlet);
      }

      broker.submitCloudletList(cloudletList);

      CloudSim.startSimulation();

      List<Cloudlet> newList =

      broker.getCloudletReceivedList();

      CloudSim.stopSimulation(); printCloudletList(newList);

    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  private static Datacenter createDatacenter(String name) {
    List<Host> hostList = new ArrayList<>();
```

```java
int mips = 1000; int ram = 2048; // host
    memory    (MB)    long    storage    =
    1000000; // host storage int bw =
    10000;

for (int i = 0; i < 2; i++)
    {
        List<Pe> peList = new ArrayList<>();
        peList.add(new Pe(0, new PeProvisionerSimple(mips)));

    hostList.add(new Host(i, new RamProvisionerSimple(ram),
new    BwProvisionerSimple(bw),    storage,    peList,    new
VmSchedulerTimeShared(peList)));
    }

    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;
    DatacenterCharacteristics
    characteristics    =    new
    DatacenterCharacteristics(arc
    h,    os,    vmm,    hostList,
    time_zone, cost, costPerMem,
    costPerStorage, costPerBw);
```

```java
        Datacenter  datacenter  =  null;
        try {
            datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), new LinkedList<Storage>(),
0);
        } catch (Exception e) {
            e.printStackTrace();
        }


        return datacenter;
    }


    private static DatacenterBroker createBroker()
        { DatacenterBroker broker = null; try {
            broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }        return
        broker;
    }


    private static void printCloudletList(List<Cloudlet> list) {
        String indent = " ";
        System.out.println();
    System.out.println("========== OUTPUT ==========");
```

```java
        System.out.println("Cloudlet  ID"  +  indent  +  "STATUS"  +
        indent +
 "Data center ID" + indent + "VM ID" + indent + "Time" + indent
+ "Start Time" + indent + "Finish Time");

    for (Cloudlet cloudlet : list) {
       System.out.print(indent + cloudlet.getCloudletId() + indent
+ indent);
if (cloudlet.getStatus() == Cloudlet.SUCCESS) {
       System.out.print("SUCCESS");

       System.out.println(indent          +          indent          +
cloudlet.getResourceId()  +  indent  +  indent  +  indent  +
cloudlet.getVmId()        +       indent       +       indent       +
cloudlet.getActualCPUTime()    +    indent    +    indent    +
cloudlet.getExecStartTime()     +     indent     +     indent     +
cloudlet.getFinishTime());
       }
     }
   }
}
```
OUTPUT:

```
.0: Broker: Trying to Create VM #0 in Datacenter_0
.0: Broker: Trying to Create VM #1 in Datacenter_0
.0: Broker: Trying to Create VM #2 in Datacenter_0
VmScheduler.vmCreate] Allocation of VM #2 to Host #0 failed by MIPS
VmScheduler.vmCreate] Allocation of VM #2 to Host #1 failed by MIPS
.1: Broker: VM #0 has been created in Datacenter #2, Host #0
.1: Broker: VM #1 has been created in Datacenter #2, Host #1
.1: Broker: Creation of VM #2 failed in Datacenter #2
.1: Broker: Sending cloudlet 0 to VM #0
.1: Broker: Sending cloudlet 1 to VM #1
.1: Broker: Sending cloudlet 2 to VM #0
.1: Broker: Sending cloudlet 3 to VM #1
.1: Broker: Sending cloudlet 4 to VM #0
.1: Broker: Sending cloudlet 5 to VM #1
20.09800000000001: Broker: Cloudlet 0 received
20.09800000000001: Broker: Cloudlet 2 received
20.09800000000001: Broker: Cloudlet 4 received
20.09800000000001: Broker: Cloudlet 1 received
20.09800000000001: Broker: Cloudlet 3 received
20.09800000000001: Broker: Cloudlet 5 received
20.09800000000001: Broker: All Cloudlets executed. Finishing...
20.09800000000001: Broker: Destroying VM #0
20.09800000000001: Broker: Destroying VM #1
roker is shutting down...
imulation: No more future events
loudInformationService: Notify all CloudSim entities for shutting down.
atacenter_0 is shutting down...
roker is shutting down...
imulation completed.
imulation completed.

========= OUTPUT ==========
loudlet ID    STATUS    Data center ID    VM ID     Time      Start Time    Finish Time
    0         SUCCESS        2              0     119.99800000000002      0.1      120.09800000000001
    2         SUCCESS        2              0     119.99800000000002      0.1      120.09800000000001
    4         SUCCESS        2              0     119.99800000000002      0.1      120.09800000000001
    1         SUCCESS        2              1     119.99800000000002      0.1      120.09800000000001
    3         SUCCESS        2              1     119.99800000000002      0.1      120.09800000000001
    5         SUCCESS        2              1     119.99800000000002      0.1      120.09800000000001
```

RESULT:

Thus, to implement the round robin task scheduling using CloudSim is done successfully.