**Laboratory Manual**

For

# Distributed Computing

**(IT 717)**

B.Tech (IT)

SEM VII

June 2019

Faculty of Technology
Dharmsinh Desai University
Nadiad.
www.ddu.ac.in

## Table of Contents

**LABWORK BEYOND CURRICULA**

**Sample experiment**

**1 AIM**: Implement concurrent echo client-server application.

**2 TOOLS/ APPARATUS:** Unix/Linux C Programming Environment

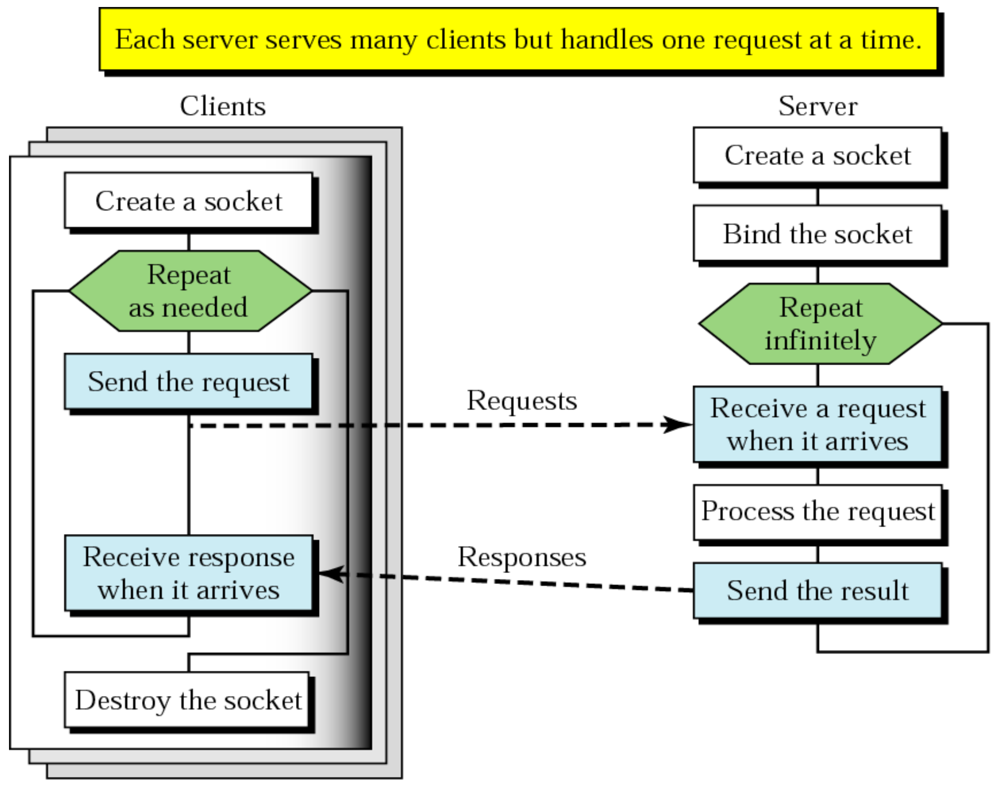
**3 STANDARD PROCEDURES**

**3.1 Analyzing the problem**

Whether we want our server to process one request at a time (iterative server) or whether to process each request in parallel (concurrent server). There are several ways we can implement concurrent server:

$$\begin{tikzcd}
& \text{Concurrent Connection-oriented Server} \arrow[dl] \arrow[d] \\
\text{Using fork()} & \text{Multi-threaded server (We will not study this one)}
\end{tikzcd}$$

**3.2 Designing the solution**



## 3.3 Implementing the solution

1) Write a server (TCP) C Program that opens a listening socket and waits to serve client
2) Write a client (TCP) C Program that connects with the server program knowing IP address and port number.
3) Get the input string from console on client and send it to server, server echoes back that string to client.

### 3.3.1 Writing the source code

**Client.c**

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), send(), and recv() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define RCVBUFSIZE 32   /* Size of receive buffer */

void DieWithError(char *errorMessage);  /* Error handling function */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket descriptor */
    struct sockaddr_in echoServAddr; /* Echo server address */
    unsigned short echoServPort;     /* Echo server port */
    char *servIP;                    /* Server IP address (dotted quad) */
    char *echoString;                /* String to send to echo server */
    char echoBuffer[RCVBUFSIZE];     /* Buffer for echo string */
    unsigned Int echoStringLen;      /* Length of string to echo */
    int bytesRcvd, totalBytesRcvd;   /* Bytes read in single recv()
                                        and total bytes read */

    if ((argc < 3) || (argc > 4))    /* Test for correct number of arguments */
    {
        fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n",
        argv[0]);
        exit(1);
   }

    servIP = argv[1];             /* First arg: server IP address (dotted quad) */
    echoString = argv[2];         /* Second arg: string to echo */

    if (argc == 4)
```

```
        echoServPort = atoi(argv[3]); /* Use given port, if any */
    else
        echoServPort = 7;  /* 7 is the well-known port for the echo service */

    /* Create a reliable, stream socket using TCP */
    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");

    /* Construct the server address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));     /* Zero out structure */
    echoServAddr.sin_family      = AF_INET;             /* Internet address family */
    echoServAddr.sin_addr.s_addr = inet_addr(servIP);   /* Server IP address */
    echoServAddr.sin_port        = htons(echoServPort); /* Server port */

    /* Establish the connection to the echo server */
    if (connect(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
        DieWithError("connect() failed");

    echoStringLen = strlen(echoString);          /* Determine input length */

    /* Send the string to the server */
    if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
        DieWithError("send() sent a different number of bytes than expected");

    /* Receive the same string back from the server */
    totalBytesRcvd = 0;
    printf("Received: ");                /* Setup to print the echoed string */
    while (totalBytesRcvd < echoStringLen)
    {
        /* Receive up to the buffer size (minus 1 to leave space for
       a null terminator) bytes from the sender */
        if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1, 0)) <= 0)
            DieWithError("recv() failed or connection closed prematurely");
        totalBytesRcvd += bytesRcvd;   /* Keep tally of total bytes */
        echoBuffer[bytesRcvd] = '\0';  /* Terminate the string! */
        printf(echoBuffer);            /* Print the echo buffer */
    }

    printf("\n");    /* Print a final linefeed */

    close(sock);
    exit(0);
}

void DieWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```

**Server.c**

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), bind(), and connect() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_ntoa() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define MAXPENDING 5    /* Maximum outstanding connection requests */
#define RCVBUFSIZE 32   /* Size of receive buffer */

void DieWithError(char *errorMessage);  /* Error handling function */
void HandleTCPClient(int clntSocket);   /* TCP client handling function */

int main(int argc, char *argv[])
{
    int servSock;                    /* Socket descriptor for server */
    int clntSock;                    /* Socket descriptor for client */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoClntAddr; /* Client address */
    unsigned short echoServPort;     /* Server port */
    unsigned int clntLen;            /* Length of client address data structure */

    if (argc != 2)     /* Test for correct number of arguments */
    {
        fprintf(stderr, "Usage:  %s <Server Port>\n", argv[0]);
        exit(1);
    }
    echoServPort = atoi(argv[1]);  /* First arg:  local port */

    /* Create socket for incoming connections */
    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");

    /* Construct local address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));   /* Zero out structure */
    echoServAddr.sin_family = AF_INET;                /* Internet address family */
    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any incoming interface */
    echoServAddr.sin_port = htons(echoServPort);      /* Local port */

    /* Bind to the local address */
    if (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
```

```
            DieWithError("bind() failed");

      /* Mark the socket so it will listen for incoming connections */
      if (listen(servSock, MAXPENDING) < 0)
            DieWithError("listen() failed");

      for (;;) /* Run forever */
      {
            /* Set the size of the in-out parameter */
            clntLen = sizeof(echoClntAddr);

            /* Wait for a client to connect */
            if ((clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr,
                              &clntLen)) < 0)
                  DieWithError("accept() failed");

            /* clntSock is connected to a client! */

            printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));

            //fork this process into a child and parent
            processid = fork();

            if (processid == 0){
            /*Child Process*/
            close(servSock);
            HandleTCPClient(clntSock);
            }
            close(clntSock);
     }
}
void DieWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}

void DieWithError(char *errorMessage);  /* Error handling function */

void HandleTCPClient(int clntSocket)
{
    char echoBuffer[RCVBUFSIZE];        /* Buffer for echo string */
    int recvMsgSize;                    /* Size of received message */

      while(1)
      {
        /* Receive message from client */
        if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
            DieWithError("recv() failed");

        /* Send received string and receive again until end of transmission */
        while (recvMsgSize > 0)      /* zero indicates end of transmission */
        {
            /* Echo message back to client */
            if (send(clntSocket, echoBuffer, recvMsgSize, 0) != recvMsgSize)
                DieWithError("send() failed");

            /* See if there is more data to receive */
            if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
                DieWithError("recv() failed");
        }
    }
        close(clntSocket);    /* Close client socket */
}
```

### 3.3.2 Compilation and Running the Solution

1) Compilation

Compile client and server programs using gcc.

```
gcc server.c –o server
gcc client.c –o client
```

2) Executing the solution

Execute server program : ./server

Execute client program by passing arguments : ServerIP, EchoString, PortNo.

```
./client 192.168.31.10 Hello 8089
```

## 4 CONCLUSIONS

The client is getting echostring as a command-line argument and sending it to server. The server echo back that string to client.

## EXPERIMENT-1

**a) TCP Socket Programming (UNIX)**

Aim: Implement concurrent echo client-server application.

Tools/ Apparatus: Unix/Linux C Programming Environment

Procedure:

1. Write a server (TCP) C Program that opens a listening socket and waits to serve client
2. Write a client (TCP) C Program that connects with the server program knowing IP address and port number.
3. Get the input string from console on client and send it to server, server echoes back that string to client.

**b) UDP Socket Programming (UNIX)**

Aim: Implement concurrent day-time client-server application.

Tools/ Apparatus: Unix/Linux C Programming Environment

Procedure:

1. Write a server(UDP) C Program that waits in recvfrom
2. Write a client(UDP) C Program that calls sendto to send string to server program knowing IP address and port number.
3. Server replies current date and time (using time, and ctime calls) to client.

## EXPERIMENT-2

**Configuring Socket options**

Aim: Configure following options on server socket and tests them: SO_KEEPALIVE, SO_LINGER, SO_SNDBUF, SO_RCVBUF, TCP_NODELAY

Tools/ Apparatus: Unix/Linux C Programming Environment

Procedure:

1) Write a server(TCP) C Program that sets the socket options using setsockopt on server one by one and displays the information using getsockopt.

## EXPERIMENT-3

**Configuring Socket options**

Aim: Data Representation and Data Validation: XML Schema and XML instance document, JSON.

Tools/ Apparatus:GUI-IDE Tool NetBeans 6.0

Procedure:

1. Design a schema for student list. A student has information such as name, semester, roll no, email-ids, phone-nos, etc.
2. Write an XML instance document for the designed schema and validate this instance document against the schema.

## EXPERIMENT-4

Aim: WSDL based webservice and its monitoring: Implement ArithmeticService that implements add and subtract operations / Java based: Implement TrigonometricService that implements sin, and cos operations. Monitor SOAP request and response packets. Analyze parts of it and compare them with the operations (java functions) headers.

Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0, WSMonitor

Procedure:

Steps for creation of ArithmeticService web service:

1. Create a Project of type Web application. Give it name Arithmetic.
2. Right click on Project folder, select New, select a Web service. A dialog box will appear. Specify Name of Web service (ArithmeticService), package name(websvc), and select option “Create Web service from scratch”
3. .java source file can be seen in Source view or Design view. From design view, you will be able to add operations. While adding operations, you have to specify name of operation, return type, names and types of input parameters.
4. Go in source view, and provide definition of Web service operations.

Steps for creation of web service Client:

1. Create a new project of type Java application. Give it name ArithmeticClient.
2. Right click on project folder and select New Web service client.
3. A dialog box will appear asking location of WSDL and client. For WSDL specify http://localhost:8080/Arithmetic/ArithmeticServiceService?WSDL and for client specify websvcclient in package option. Make sure Style is JAX-WS.
4. Right click in source code of Main class. Select option “Web service client resource” ☾ Call web service operation.
5. A new dialog box will appear asking for selecting name of operation. Select “add” operation.

Steps for Monitoring SOAP Messages:

1. Start WS-Monitor on port 4040 and forwarding port 8080
2. Change the service target port from 8080 to 4040
3. WS-Monitor will capture SOAP request and SOAP response.
4. Study request and response packets and try to relate them with operation name, service name, namespace, etc.

## EXPERIMENT-5

Aim: Design and test BPEL module that composes ArithmeticService and TrignometricService.

Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0

Procedure:

1. Create two webservices as per the described steps in experiment – 4, deploy them and test them.
2. Create a BPEL module
3. Add WSDL for BPEL Process
4. Using BPEL designer, design BPEL process
5. Clean and build BPEL process
6. Create a SOA composite application
7. Add BPEL module in JBI
8. Test the application using BPEL test cases

## EXPERIMENT-6

Aim: Deployment of a HADOOP cluster and monitoring status of its components.
Tools/ Apparatus: GUI-IDE Tool NetBeans 6.0, Hadoop Common**,** Hadoop Distributed File System, Hadoop YARN, Hadoop MapReduce, Ambari
Procedure:

- Install the appropriate version of java for your Hadoop.
- ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.
- Download the appropriate hadoop file system from link given below. http://www.apache.org/dyn/closer.cgi/hadoop/common/"
- Unpack the downloaded Hadoop distribution. In the distribution, edit the file etc/hadoop/hadoop-env.sh to define some parameters as follows: " #export JAVA_HOME=/usr/java/latest".
- The following example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

```
 $ mkdir input
 $ cp etc/hadoop/*.xml input
 $ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples- 2.9.2.jar
grep input output 'dfs[a-z.]+'
 $ cat output/*
```

- Now check that you can ssh to the localhost without a passphrase:

```
$ ssh localhost
```

- If you cannot ssh to localhost without a passphrase, execute the following commands:

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

**Execution**

- The following instructions are to run a MapReduce job locally.

1. Format the filesystem:

```
$ bin/hdfs namenode -format
```

2. Start NameNode daemon and DataNode daemon:

```
$ sbin/start-dfs.sh
```

- The hadoop daemon log output is written to the $HADOOP_LOG_DIR directory (defaults to $HADOOP_HOME/logs).

- Browse the web interface for the NameNode; by default it is available at: NameNode - http://localhost:50070/

- Make the HDFS directories required to execute MapReduce jobs:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/<username>
```

- Copy the input files into the distributed filesystem:

```
$ bin/hdfs dfs -put etc/hadoop input
```

- Run some of the examples provided:

```
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.2.jar grep input
      output 'dfs[a-z.]+'
```

- Examine the output files: Copy the output files from the distributed file-system to the local filesystem and examine them:

```
$ bin/hdfs dfs -get output output
$ cat output/*
```

**OR**

- View the output files on the distributed filesystem:

```
$ bin/hdfs dfs -cat output/*
```

- When you’re done, stop the daemons with:

```
$ sbin/stop-dfs.sh
```

## EXPERIMENT-7

Aim: Perform data intensive computing using map-reduce based programming on a HADOOP cluster.

Tools/ Apparatus: GUI-IDE Tool NetBeans 6.0, Hadoop Common**,** Hadoop Distributed File System, Hadoop YARN, Hadoop MapReduce, Ubuntu

Procedure:

*Prerequisites*

1. Installation and Configuration of Single node Hadoop :
2. Prepare your computer network (Decide no of nodes to set up cluster) :
3. Basic installation and configuration :

3.1 configure etc/hosts for master and slaves nodes

```
 $  sudo gedit /etc/hosts
# Add following hostname and their ip in host table
192.168.2.14    HadoopMaster
192.168.2.15    HadoopSlave1
192.168.2.16    HadoopSlave2
```

3.2 Create hadoop as group and hduser as user in all Machines (if not created !!).

```
DDU@HadoopMaster:~$ sudo addgroup hadoop
DDU@HadoopMaster:~$ sudo adduser --ingroup hadoop hduser

sudo usermod -a -G sudo hduser
```

OR

Add following line in /etc/sudoers/

```
hduser    ALL=(ALL:ALL) ALL
```

3.3 Install rsync for sharing hadoop source with rest all Machines, and reboot all the machine.

```
$ sudo apt-get install rsync
```

3.4 To make above changes reflected, we need to reboot all of the Machines.

```
$ sudo reboot
```

4. Applying Common Hadoop Configuration

4.1 *Update core-site.xml*

*Update this file by changing hostname from localhost to HadoopMaster*

## To edit file, fire the below given command

hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit core-site.xml

## Paste these lines into <configuration> tag OR Just update it by replacing localhost with master

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://HadoopMaster:9000</value>
 </property>
```

4.2 Update hdfs-site.xml

*Update this file by updating repliction factor from 1 to 3.*

## To edit file, fire the below given command

hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit hdfs-site.xml

## Paste/Update these lines into <configuration> tag

```
<property>
   <name>dfs.replication</name>
   <value>3</value>
 </property>
```

4.3 Update yarn-site.xml

Update this file by updating the following three properties by updating hostname from localhost to HadoopMaster

## To edit file, fire the below given command

hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit yarn-site.xml

## Paste/Update these lines into <configuration> tag

```
      <property>
      <name>yarn.resourcemanager.resource-tracker.address</name>
      <value>HadoopMaster:8025</value>
</property>
<property>
      <name>yarn.resourcemanager.scheduler.address</name>
      <value>HadoopMaster:8035</value>
</property>

<property>
```

```
          <name>yarn.resourcemanager.address</name>
          <value>HadoopMaster:8050</value>
      </property>
```

4.4 Update Mapred-site.xml

Update this file by updating and adding following properties,

## To edit file, fire the below given command

```
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit mapred-site.xml
```

## Paste/Update these lines into <configuration> tag

```
      <property>
          <name>mapreduce.job.tracker</name>
          <value>HadoopMaster:5431</value>
      </property>
      <property>
          <name>mapred.framework.name</name>
          <value>yarn</value>
      </property>
```

4.5 Update masters

Update the directory of master nodes of Hadoop cluster

## To edit file, fire the below given command

```
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit masters
```

## Add name of master nodes

```
   HadoopMaster
```

4.6 Update slaves

Update the directory of slave nodes of Hadoop cluster

## To edit file, fire the below given command

```
  hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit slaves
```

## Add name of slave nodes

```
      HadoopSlave1
      HadoopSlave2
```

5. Copying/Sharing/Distributing Hadoop config files to rest all nodes – master/slaves

5.1 Use rsync for distributing configured Hadoop source among rest of nodes via network.

```
# In HadoopSlave1 machine
$ sudo rsync -avxP /usr/local/hadoop/ hduser@HadoopSlave1:/usr/local/hadoop/

# In HadoopSlave2 machine
$ sudo rsync -avxP /usr/local/hadoop/ hduser@HadoopSlave2:/usr/local/hadoop/
```

6 Applying Master node specific Hadoop configuration: (Only for master nodes)

6.1 Remove existing Hadoop_data folder (which was created while single node hadoop setup.)

```
$ sudo rm -rf /usr/local/hadoop_tmp/
```

6.2 : Make same (/usr/local/hadoop_tmp/hdfs) directory and create NameNode ( /usr/local/hadoop_tmp/hdfs/namenode) directory

```
$ sudo mkdir -p /usr/local/hadoop_tmp/
$ sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
```

6.3 : Make hduser as owner of that directory.

```
$ sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/
```

7 Applying Slave node specific Hadoop configuration : (Only for slave nodes)

7.1 Remove existing Hadoop_data folder (which was created while single node hadoop setup)

```
$ sudo rm -rf /usr/local/hadoop_tmp/hdfs/
```

7.2 Creates same (/usr/local/hadoop_tmp/) directory/folder, an inside this folder again Create DataNode (/usr/local/hadoop_tmp/hdfs/namenode) directory/folder

```
$ sudo mkdir -p /usr/local/hadoop_tmp/
$ sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
```

*Step 7C :* Make hduser as owner of that directory

```
sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/
```

8 Copying ssh key for Setting up passwordless ssh access from Master to Slave node :

```
hduser@HadoopMaster: ~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub
hduser@HadoopSlave1
```

```
hduser@HadoopMaster: ~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@HadoopSlave2
```

9. Format Namenonde (Run on MasterNode) :

```
# Run this command from Masternode
 hduser@HadoopMaster: usr/local/hadoop/$ hdfs namenode -format
```

10. Starting up Hadoop cluster daemons : (Run on MasterNode)

    Start HDFS daemons:

```
hduser@HadoopMaster:/usr/local/hadoop$ start-dfs.sh
```

11. Start MapReduce daemons:

```
hduser@HadoopMaster:/usr/local/hadoop$ start-yarn.sh
```

12. Instead both of these above command you can also use start-all.sh, but its now deprecated so its not recommended to be used for better Hadoop operations.
13. Track/Monitor/Verify Hadoop cluster : (Run on any Node)
    Verify Hadoop daemons on Master and slaves(All slave):

```
hduser@HadoopMaster: jps
```

## EXPERIMENT-8

Aim: Create Restful Webservice and test it using Postman.
Tools/ Apparatus: Web service, GlassFish Server, GUI-IDE Tool:NetBeans 6.0
Procedure:

1. In NetBeans IDE, create a simple web application. This example creates a very simple "Hello, World" web application.
2. From Categories, select Java Web. From Projects, select Web Application. Click Next.
3. Type a project name, **`HelloWorldApplication`**, and click Next.
4. Make sure that the Server is GlassFish Server (or similar wording.) Click Finish.
5. Right-click the project and select New; then select RESTful Web Services from Patterns.
6. Select Simple Root Resource and click Next.
7. Type a Resource Package name, such as helloWorld.
8. Type helloworld in the Path field. Type HelloWorld in the Class Name field. For MIME Type, select text/html. Click Finish.
9. The REST Resources Configuration page appears. Click OK.
10. A new resource, HelloWorld.java, is added to the project and appears in the Source pane. This file provides a template for creating a RESTful web service.
11. In HelloWorld.java, find the getHtml() method. Replace the //TODO comment and the exception with the following text, so that the finished product resembles the following method.

```
 /**
 * Retrieves representation of an instance of helloWorld.HelloWorld
 * @return an instance of java.lang.String
 */
@GET
@Produces("text/html")
public String getHtml() {
   return "<html><body><h1>Hello, World!!</body></h1></html>";
}
```

12. Set the Run Properties: Right-click the project node and select Properties.In the dialog, select the Run category.
    Set the Relative URL to the location of the RESTful web service relative to the Context Path, which for this example is resources/helloworld.

13. Right-click the project and select Deploy.
14. Right-click the project and select Run.
15. A browser window opens and displays the return value of Hello, World!!

## EXPERIMENT-9

Aim: Create Microservice based application using Spring Boot.
Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0.
Procedure:

01. Initializing a RESTful Services Project with Spring Boot
02. Understanding the RESTful Services we would create in this course
03. Creating a Hello World Service
04. Enhancing the Hello World Service to return a Bean
05. Quick Review of Spring Boot Auto Configuration and Dispatcher Servlet
06. Enhancing the Hello World Service with a Path Variable
07. Creating User Bean and User Service0
08. Implementing GET Methods for User Resource
09. Implementing POST Method to create User Resource
10. Enhancing POST Method to return correct HTTP Status Code and Location

## EXPERIMENT-10

Aim: : Implementation JMS based application using Publish-Subscribe paradigm.
Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0, JMS, ESB: WSO2
Procedure:

1. Create user interface – for publisher and for subscriber
2. Create two web services - one for subscriber to register themselves in the database and another for the publisher to send message through JMS to Topic for inserting data into to data base.
3. Create two proxies for the two web-services by taking endpoints as the web services. These would provide the functionalities to logs, mediate messages route or transform them as per the settings of ESB (WSO2).
4. Create java message driven bean (JMS) for directing message from the web service to the message driven bean. The message driven bean would in turn through its onMessage() method would take info from the subscribers table and send mail accordingly

## EXPERIMENT-11

Aim: Test open source ESB using web service.
Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0, ESB:WSO2 / OpenESB
Procedure:

1. Install any open ESB
2. Configure the configuration files
3. Start ESB
4. Open Admin console
5. Study various options (mediator, proxy service, statistics of services, Endpoint, etc)

## EXPERIMENT-12

Aim: Implementing Stateful grid services using Globus WS-Core-4.0.3
Tools/ Apparatus: Web service, Globus WS-Core-4.0.3, JNDI, Apache Ant
Procedure:

1. Define the web service's interface. This is done with WSDL.
2. Implement the web service in Java.
3. Define the deployment parameters by using WSDD and JNDI.
4. Compile everything and generate a GAR file using Ant.
5. Deploy service using Globus WS-Core-4.0.3 GT4 tool

## **EXPERIMENT-13**

Aim: Configuring reliability and security options.
Tools/ Apparatus: Web service, GlassFish Server, GUI-IDE Tool:NetBeans 6.0
Procedure:

1. Create a web service, or open an existing web servic.
1.1 In the Projects window, expand the Web Services node.
1.2 Right-click the node for the web service for which you want to configure security options.
1.3 Select Edit Web Service Attributes.
1.4 When the Web Service Attributes Editor is opened, the WSIT options display
2. In the web service's port binding section, the following options for WSIT security are available:
2.1 Optimize Transfer of Binary Data (MTOM)--Select this option to optimize this web service.
2.2 Reliable Message Delivery--Select this option to add reliable messaging to this web service.
2.4 Secure Service--Select this option to add WSIT security for all of the operations of a web service. Security Mechanism--Select one of the options from the list. The mechanisms that use SSL require that you modify the web.xml (or ejb-jar.xml) deployment descriptors.

## References

Text Books:

- UNIX Network Programming by W. Richard Stevens, Prentice Hall Publication
- Distributed Computing : Concepts & Applications : by M.L.Liu Addison Wiselly
- Distributed Computing in Java 9 by Raja Malleswara Rao Pattamsetti, Packt Publishing

Reference Books

- SOA: Principles of Service Design by Thomas Erl , Publisher: Prentice Hall
- Distributed Operating Systems: Concepts and Design by Pradeep K. Sinha, PHI Publication
- Distributed Systems: Concepts and Design, 4th Ed., by: George Coulouris, Jean Dollimore and Tim Kindberg, Publisher: Addison Wesley