Laboratory Manual

For

Distributed Computing

(IT 715)

B.Tech (IT)

SEM VII

June 2010

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

Table of Contents

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

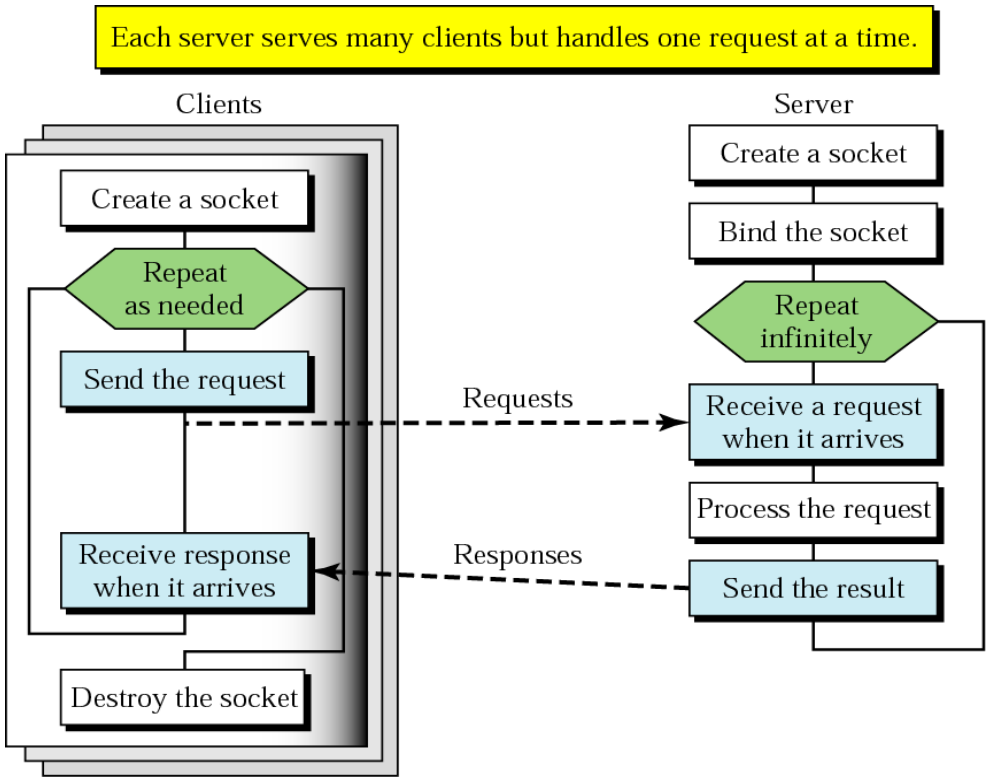
Whether we want our server to process one request at a time (iterative server) or whether to process each request in parallel (concurrent server). There are several ways we can implement concurrent server:

Concurrent Connection-oriented Server

Using fork()

Multi-threaded server (We will not study this one)

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

TCP Socket Programming (UNIX)

Aim: Implement concurrent echo client-server application.

Tools/ Apparatus: Unix/Linux C Programming Environment

Procedure:

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

EXPERIMENT-2

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-3

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-4

Shared Memory, Semaphore

Aim: Incrementing a counter in shared memory.

Tools/ Apparatus: Unix/Linux C Programming Environment

Procedure:

1) Write a server C Program using Shared memory and semaphore (server increments counter between sem_wait and sem_post). Create shared memory using mmap.
2) Write a client C Program that reads counter value between sem_wait and sem_post. Access shared memory using open.

EXPERIMENT-5

CORBA

Aim: Create CORBA based server-client application.

Tools/ Apparatus: CORBA Runtime: inbuilt in Java Standard Edition Runtime JDK

Procedure:

1) Define the remote interface : Write IDL file
2) Compile the remote interface : Mapping IDL file to platform specific files
   C:\...>idlj -fall Hello.idl
3) Implement the server
4) Implement Client
5) Start name service
   start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
6) Start server
   start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
7) Start client
   java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost

EXPERIMENT-6

XML document validation

Aim: Design XML Schema and XML instance document

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-7

Web service, WSDL based, from Java source

Aim: WSDL based: Implement ArithmeticService that implements add, and subtract operations.
Aim: Java based: Implement TrigonometricService that implements sin, and cos operations.

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

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.

EXPERIMENT-8

Configuring WS-standards for Web service

Aim: Configuring reliability and security options.

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

EXPERIMENT-9

Monitoring SOAP packets using TCPMonitor/WS-Monitor

Aim: 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:

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-10

BPEL

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:

5. Create a BPEL module
6. Add WSDL for BPEL Process
7. Using BPEL designer, design BPEL process
8. Clean and build BPEL process
9. Create a SOA composite application
10. Add BPEL module in JBI
11. Test the application using BPEL test cases

EXPERIMENT-11

ESB

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 Publish/Subscribe Paradigm using Web Services, ESB and JMS

Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0, JMS, ESB: WSO2

Procedure:

1. Create user interface in .NET environment – 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-13

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

References

Text Books:

- UNIX Network Programming by W. Richard Stevens, Prentice Hall Publication
- Distributed Computing : Concepts & Applications : by M.L.Liu Addison Wiselly
- SOA in Practice: The Art of Distributed System Design by Nicolai M. Josuttis, Prentice Hall Publication

Reference Books

- SOA: Principles of Service Design by Thomas Erl
- Distributed Operating Systems: Concepts and Design by Pradeep K. Sinha, PHI Publication
- Distributed Systems: Concepts