**PSG COLLEGE OF TECHNOLOGY**

COIMBATORE – 641 004

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**15I711-DISTRIBUTED COMPONENTS LABORATORY**

****

NOV-2020

CERTIFIED BONAFIDE RECORD OF WORK DONE BY **VISHNUVARDHAN S [ 17I361 ]**

--------------------

Faculty In-charge

Submitted for the practical examination held on **16.11.2020**

**---------------------- ------------------------**

**Internal Examiner External Examiner**

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **DATE** | **TOPIC** | **PAGE NO** |
| **1.** | 04.08.20 | Socket Programming - Prelab | 3 |
| **2.** | 31.08.20 | Socket Programming in C – Exercise | 11 |
| **3.** | 08.09.20 | Socket programming in C – Postlab | 28 |
| **4.** | 14.09.20 | Remote Procedure Call (RPC) – Prelab | 35 |
| **5.** | 21.09.20 | Remote Procedure Call (RPC)- Exercise | 42 |
| **6.** | 05.10.20 | Java Remote Method Invocation (RMI) – Prelab | 54 |
| **7.** | 12.10.20 | Java Remote Method Invocation (RMI) -Exercise | 59 |
| **8.** | 02.11.20 | Enterprise Java Beans (EJB) - Prelab | 79 |
| **9.** | 07.11.20 | Enterprise Java Beans (EJB) - Exercise | 85 |
| **10.** | 10.11.20 | Mini Project – Medical Image Sharing using Blockchain | 93 |

**SOCKET PROGRAMMING - PRELAB**

**What are the various methods of Inter Process Communication? Pipes:**

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

**Message Passing:**

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

Send (message)- message size fixed or variable

Received (message)

**Message Queues:**

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

**Direct Communication:**

In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

**Indirect Communication:**

Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.

**Shared Memory:**

Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.

**FIFO:**

Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.

# 2. Execute any command line argument program in C Code:

# 

# Output:

# 

**3. What are sockets? Explain Types of sockets**

**Sockets** are seen as the end of the two-way communication between two processes; they allow processes to communicate with each other using a file descriptor and are commonly used in client-server applications that allow for communication between multiple applications.

There are four types of sockets available to the users.

**Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

**Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you

build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

**Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

**Sequenced Packet Sockets** − They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications.

# 4. List out the client and server-side system calls used in socket programming along with explanations

**Socket**

socket (struct proc ∗p, struct socket\_args ∗uap, int retval) In the socket system call:

p is a pointer to the proc structure of the process that makes the socket call.

uap is a pointer to the socket\_args structure that contains the arguments passed to the process in the socket system call.

retval is the return value of the system call.

The socket system call creates a new socket by assigning a new descriptor. The new descriptor is returned to the calling process. Any subsequent system calls are identified with the created socket. The socket system call also assigns the protocol to the created socket descriptor.

# Bind

bind (struct proc ∗p, struct bind\_args ∗uap, int ∗retval) In the bind system call function:

s is the socket descriptor.

name is the pointer to the buffer that contains the network transport address.

namelen is the size of the buffer.

The bind system call associates a local network transport address with a socket. For a client process, it is not mandatory to issue a bind call. The kernel takes care of doing an implicit binding when the client process issues the [connect](https://developer.ibm.com/technologies/systems/articles/au-tcpsystemcalls/#connect) system call. It is often necessary for a server process to issue an explicit bind request before it can accept connections or start communication with clients.

**Connect**

connect (struct proc ∗p, struct connect\_args ∗uap, int ∗retval); In the connect system call:

s is the socket descriptor.

name is the pointer to the buffer that has the foreign IP/port address pair.

namelen is the length of the buffer.

The connect system call is normally called by the client process to connect to the server process. If the client process has not explicitly issued a bind system call before initiating the connection, implicit binding on the local socket is taken care of by the stack.

**Accept**

accept(struct proc ∗p, struct accept\_args ∗uap, int ∗retval); In the accept system call:

s is the socket descriptor.

name is a buffer (an OUT parameter), which contains the network transport address of the foreign host.

anamelen is the size of the name buffer.

The accept system call is a blocking call that waits for incoming connections. Once a connection request is processed, a new socket descriptor is returned by accept. This new socket is connected to the client and the other socket s remains in LISTEN state to accept further connections.

**Listen**

listen (struct proc ∗p, struct listen\_args ∗uap, int ∗retval) In the listen system call:

s is the socket descriptor.

backlog is the queue limit for the number of connections on a socket.

The listen call indicates to the protocol that the server process is ready to accept any new incoming connections on the socket. There is a limit on the number of connections that can be queued up, after which any further connection requests are ignored.

**Close**

soo\_close(struct file ∗fp , struct proc ∗p);

In the close system call:

fp is the pointer to the file structure.

p is the pointer to the proc structure of the calling process.

The close system call closes or aborts any pending connections on the socket.

# 5. Explain TCP and UDP

**Transmission Control Protocol (TCP)** is a transport layer protocol used to create a connection between applications so that they can exchange data over a communication network.

TCP provides a process for communication. Port numbers are appended in the header of the packet to enable this.

A three-way handshaking process takes place to open a connection in TCP; hence, it is called a connection-oriented protocol.

A packet (called a segment in case of TCP) contains a checksum field used for error control.

TCP prevents the sender from sending in more data than the receiver can handle with the window size provided by the receiver.

It guarantees the reliable delivery of segments. Incase of segment loss or corruption, segments are re-transmitted.

It reassembles the packets in the correct order at the receiver side.

It uses a congestion avoidance algorithm to avoid the network from being congested.

TCP provides communication between an application program and the Internet Protocol (they are frequently written as TCP/IP.) An application does not need to required packet fragmentation on the transmission medium or other mechanisms for sending data in order to be sent via TCP. [While IP handles actual delivery of the data,](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) TCP keeps track of 'segments' - the individual units of data transmission that a message is divided into for efficient routing through the network.

Due to unpredictable network behaviour, IP packets can be lost or delivered out of order; TCP detects and minimizes these issues by reordering packet data or requesting redelivery. This accuracy comes with a trade-off in speed. TCP is known more for reliability, but this accuracy comes from trading speed, sometimes coming with a delay of several seconds.

# User Datagram Protocol (UDP)

**User Datagram Protocol (UDP)** is a transport layer protocol that is used to create a connection between applications running on hosts that are connected via a network.

UDP provides a process to process communication. Port numbers are appended in the header of the packet to enable this.

There is no handshaking required before sending a message; hence, UDP is referred to as a connection-less protocol.

UDP is a best-effort delivery service – it does not transmit any lost or corrupt packets.

A packet (called a datagram in the case of UDP) contains a checksum field that is used to detect a corrupted datagram. However, it has relatively limited error detection abilities.

# 6. What is called a socket descriptor?

A socket is an abstraction of a communication endpoint. Just as they would use file descriptors to access a file, applications use socket descriptors to access sockets. Socket descriptors are implemented as file descriptors in the UNIX System. Indeed, many of the functions that deal with file descriptors, such as read and write, will work with a socket descriptor.

To create a socket, we call the socket function.

# #include <sys/socket.h>

**int socket(int *domain*, int *type*, int *protocol*); Returns: file (socket) descriptor if OK, 1 on error**

**7. Give detailed explanations on**

**Multiple clients involved in socket Programming**

**Socket programming where client and server running in same and different systems Multiple clients involved in socket Programming:**

The simple way to handle multiple clients would be to spawn new thread for every new client connected to the server. This method is strongly not recommended because of various disadvantages, namely:

Threads are difficult to code, debug and sometimes they have unpredictable results.

Overhead switching of context

Not scalable for large number of clients

Deadlocks can occur

# Select()

A better way to handle multiple clients is by using **select()** linux command.

Select command allows to monitor multiple file descriptors, waiting until one of the file descriptors become active.

For example, if there is some data to be read on one of the sockets select will provide that information.

**Select** works like an interrupt handler, which gets activated as soon as any file descriptor sends any data.

# Socket programming where client and server running in same and different systems

When Server daemon is running on a host, it is listening to its port and handles all requests sent by clients to the port on the host(server socket). A client must know IP and port of the server (Server socket) to send a request to it. Client’s port is often provided by OS kernel when client starts communication with the server and is freed when communication is over.

# Result:

# The basics socket programming is learnt successfully.

**SOCKET PROGRAMMING - EXERCISE**

**1. Echo client server (one server and multiple clients) in C**

**Algorithm:**

**SERVER SIDE:**

* **Step 1:** Create a stream socket with internet domain using socket () system call which returns a socket descriptor.
* **Step 2:** Create a structure that holds the acceptable IP address, address family and port number to be connected. Assign the address, family and port to the structure.
* **Step 3:** Bind the address structure to the socket created using bind () system call.
* **Step 4:** Call the listen () function by passing the socket descriptor and length of the queue as arguments. The server starts listening for connection from client.
* **Step 5:** The accept () system call is used to accept the request from the client. It returns the socket descriptor of client.
* **Step 6:** The server responds with the same string that is given from the user.
* **Step 7:** There will be a single server ready to respond the client. Multiple clients can request to the server.

**CLIENT SIDE:**

* **Step 1**: Create a stream socket with internet domain using socket () system call which returns a socket descriptor.
* **Step 2:** Create a structure that holds the client’s IP address, address family and port number to be connected. Assign the address, family and port to the structure.
* **Step 3:** Create a connection with the server using connect () system call that takes the socket descriptor, address structure and size of the structure as arguments. The port address is got from the user.
* **Step 4:** Get the input from the user and send it to the server using write () system call.
* **Step 5:** Use read () system call to get the string from the server. Print the string to the user. The server responds whatever client sent and this is called as echo.

**Program:**

**SERVER SIDE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 4444

int main(){

    int sockfd, ret;

     struct sockaddr\_in serverAddr;

    int newSocket;

    struct sockaddr\_in newAddr;

    socklen\_t addr\_size;

    char buffer[1024];

    pid\_t childpid;

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if(sockfd < 0){

        printf("[-]Error in connection.\n");

        exit(1);

    }

    printf("[+]Server Socket is created.\n");

    memset(&serverAddr, '\0', sizeof(serverAddr));

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_port = htons(PORT);

    serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    ret = bind(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

    if(ret < 0){

        printf("[-]Error in binding.\n");

        exit(1);

    }

    printf("[+]Bind to port %d\n", 4444);

    if(listen(sockfd, 10) == 0){

        printf("[+]Listening....\n");

    }else{

        printf("[-]Error in binding.\n");

    }

    while(1){

        newSocket = accept(sockfd, (struct sockaddr\*)&newAddr, &addr\_size);

        if(newSocket < 0){

            exit(1);

        }

        printf("Connection accepted from %s:%d\n", inet\_ntoa(newAddr.sin\_addr), ntohs(newAddr.sin\_port));

        if((childpid = fork()) == 0){

            close(sockfd);

            while(1){

                recv(newSocket, buffer, 1024, 0);

                if(strcmp(buffer, ":exit") == 0){

                    printf("Disconnected from %s:%d\n", inet\_ntoa(newAddr.sin\_addr), ntohs(newAddr.sin\_port));

                    break;

                }else{

                    printf("Client: %s\n", buffer);

                    send(newSocket, buffer, strlen(buffer), 0);

                    bzero(buffer, sizeof(buffer));

                }

            }

        }

    }

    close(newSocket);

    return 0;

}

**CLIENT SIDE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 4444

int main(){

    int clientSocket, ret;

    struct sockaddr\_in serverAddr;

    char buffer[1024];

    clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

    if(clientSocket < 0){

        printf("[-]Error in connection.\n");

        exit(1);

    }

    printf("[+]Client Socket is created.\n");

    memset(&serverAddr, '\0', sizeof(serverAddr));

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_port = htons(PORT);

    serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    ret = connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

    if(ret < 0){

        printf("[-]Error in connection.\n");

        exit(1);

    }

    printf("[+]Connected to Server.\n");

    while(1){

        printf("Client: \t");

        scanf("%s", &buffer[0]);

        send(clientSocket, buffer, strlen(buffer), 0);

        if(strcmp(buffer, ":exit") == 0){

            close(clientSocket);

            printf("[-]Disconnected from server.\n");

            exit(1);

        }

        if(recv(clientSocket, buffer, 1024, 0) < 0){

            printf("[-]Error in receiving data.\n");

        }else{

            printf("Server: \t%s\n", buffer);

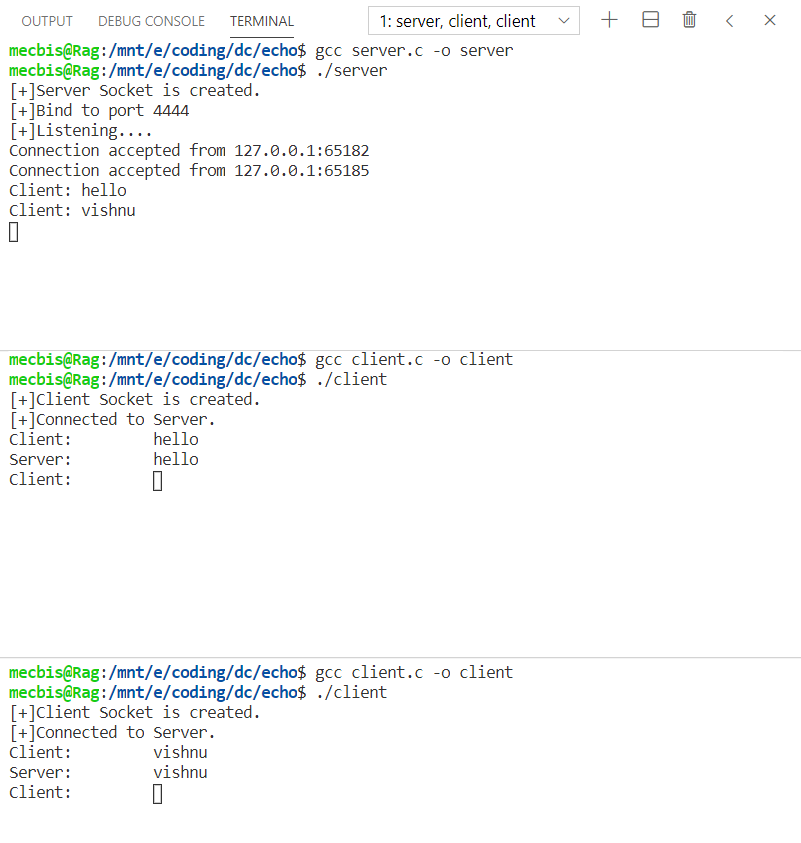
        }

    }

    return 0;

}

**Output:**

****

**2. Chat Application (multiple servers and multiple clients) in C**

**Algorithm:**

**SERVER SIDE:**

* **Step 1:** Create a stream socket with internet domain using socket () system call which returns a socket descriptor.
* **Step 2:** Create a structure that holds the acceptable IP address, address family and port number to be connected. Assign the address, family and port to the structure.
* **Step 3:** Bind the address structure to the socket created using bind () system call.
* **Step 4:** Call the listen () function by passing the socket descriptor and length of the queue as arguments. The server starts listening for connection from client.
* **Step 5:** The accept () system call is used to accept the request from the client. It returns the socket descriptor of client.
* **Step 6:** The server responds with the same string that is given from the user.
* **Step 7:** There will be multiple number of servers ready to respond the client. The client can request for a particular server.

**CLIENT SIDE:**

* **Step 1**: Create a stream socket with internet domain using socket () system call which returns a socket descriptor.
* **Step 2:** Create a structure that holds the client’s IP address, address family and port number to be connected. Assign the address, family and port to the structure.
* **Step 3:** Create a connection with the server using connect () system call that takes the socket descriptor, address structure and size of the structure as arguments. The port address is got from the user.
* **Step 4:** Get the input from the user and send it to the server using write () system call.
* **Step 5:** Use read () system call to get the string from the server. Print the string to the user.

**Program:**

**SERVER SIDE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 4444

int main(){

    int sockfd, ret;

     struct sockaddr\_in serverAddr;

    int newSocket;

    struct sockaddr\_in newAddr;

    socklen\_t addr\_size;

    char buffer[1024];

    pid\_t childpid;

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if(sockfd < 0){

        printf("[-]Error in connection.\n");

        exit(1);

    }

    printf("[+]Server Socket is created.\n");

    memset(&serverAddr, '\0', sizeof(serverAddr));

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_port = htons(PORT);

    serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    ret = bind(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

    if(ret < 0){

        printf("[-]Error in binding.\n");

        exit(1);

    }

    printf("[+]Bind to port %d\n", 4444);

    if(listen(sockfd, 10) == 0){

        printf("[+]Listening....\n");

    }else{

        printf("[-]Error in binding.\n");

    }

    while(1){

        newSocket = accept(sockfd, (struct sockaddr\*)&newAddr, &addr\_size);

        if(newSocket < 0){

            exit(1);

        }

        printf("Connection accepted from %s:%d\n", inet\_ntoa(newAddr.sin\_addr), ntohs(newAddr.sin\_port));

        if((childpid = fork()) == 0){

            close(sockfd);

            while(1){

                recv(newSocket, buffer, 1024, 0);

                if(strcmp(buffer, ":exit") == 0){

                    printf("Disconnected from %s:%d\n", inet\_ntoa(newAddr.sin\_addr), ntohs(newAddr.sin\_port));

                    break;

                }else{

                    printf("Client: %s\n", buffer);

                    send(newSocket, buffer, strlen(buffer), 0);

                    bzero(buffer, sizeof(buffer));

                }

            }

        }

    }

    close(newSocket);

    return 0;

}

**CLIENT SIDE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int PORT;

int main(){

    int clientSocket, ret;

    struct sockaddr\_in serverAddr;

    char buffer[1024];

    printf("Enter the port number to connect: ");

    scanf("%d",&PORT);

    clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

    if(clientSocket < 0){

        printf("[-]Error in connection.\n");

        exit(1);

    }

    printf("[+]Client Socket is created.\n");

    memset(&serverAddr, '\0', sizeof(serverAddr));

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_port = htons(PORT);

    serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    ret = connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

    if(ret < 0){

        printf("[-]Error in connection.\n");

        exit(1);}

    printf("[+]Connected to Server.\n");

    while(1){

        printf("Client: \t");

        scanf("%s", &buffer[0]);

        send(clientSocket, buffer, strlen(buffer), 0);

        if(strcmp(buffer, ":exit") == 0){

            close(clientSocket);

            printf("[-]Disconnected from server.\n");

            exit(1);}

        if(recv(clientSocket, buffer, 1024, 0) < 0){

            printf("[-]Error in receiving data.\n");

        }else{

            printf("Server: \t%s\n", buffer);

         }

}

return 0;

}

**output:**

****

**3. Client Server Calculator**

**SERVER SIDE:**

* **Step 1:** Create a stream socket with internet domain using socket () system call which returns a socket descriptor.
* **Step 2:** Create a structure that holds the acceptable IP address, address family and port number to be connected. Assign the address, family and port to the structure.
* **Step 3:** Bind the address structure to the socket created using bind () system call.
* **Step 4:** Call the listen () function by passing the socket descriptor and length of the queue as arguments. The server starts listening for connection from client.
* **Step 5:** The accept () system call is used to accept the request from the client. It returns the socket descriptor of client. Using the descriptor read the operator and operands using read ().
* **Step 6:** Calculate the answer and store the result in a variable.
* **Step 7:** Send the result using write () system call.

**CLIENT SIDE:**

* **Step 1**: Create a stream socket with internet domain using socket () system call which returns a socket descriptor.
* **Step 2:** Create a structure that holds the client’s IP address, address family and port number to be connected. Assign the address, family and port to the structure.
* **Step 3:** Create a connection with the server using connect () system call that takes the socket descriptor, address structure and size of the structure as arguments.
* **Step 4:** Get the operands and operator from the user and send it to the server using write () system call.
* **Step 5:** Use read () system call to get the answer from the server. Print the result to the user.

**Program:**

**SERVER SIDE:**

#include<sys/types.h>

#include<sys/socket.h>

#include<stdio.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

#include<arpa/inet.h>

void main()

{

        int sockfd,connfd,sin\_size;

        char operator;

        int op1,op2,result;

        if((sockfd=socket(AF\_INET,SOCK\_STREAM,0))>0)

        {

                printf("Socket created\n");

        }

        struct sockaddr\_in servaddr;

        struct sockaddr\_in clientaddr;

        servaddr.sin\_family=AF\_INET;

        servaddr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

        servaddr.sin\_port=htons(6006);

        if((bind(sockfd, (struct sockaddr \*)&servaddr,sizeof(servaddr)))==0)

        {

                printf("Bind successful\n");

        }

        if((listen(sockfd,5))==0)

        {

                printf("Server listening\n");

        }

        sin\_size=sizeof(struct sockaddr\_in);

        if((connfd=accept(sockfd, (struct sockaddr \*)&clientaddr,&sin\_size))>0)

        {

                printf("Client accepted\n");

        }

        read(connfd,&operator,10);

        read(connfd,&op1,sizeof(op1));

        read(connfd,&op2,sizeof(op2));

        if(operator=='+')

        {

                result=op1 + op2;

        }

        else if(operator=='-')

        {

                result=op1 - op2;

        }

        else if(operator=='\*')

        {

                result=op1 \* op2;

        }

        else if(operator=='/')

        {

                result=op1 / op2;

        }

        else

        {

                printf("Invalid operator\n");

                result=-1;

        }

        write(connfd,&result,sizeof(result));

        close(sockfd);

}

**CLIENT SIDE:**

#include<sys/types.h>

#include<sys/socket.h>

#include<stdio.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

#include<strings.h>

#include<arpa/inet.h>

void main()

{

        int sockfd,sin\_size,con;

        char operat;

        int op1,op2,result;

        if((sockfd=socket(AF\_INET,SOCK\_STREAM,0))>0)

        {

                printf("Socket created!\n");

        }

        struct sockaddr\_in servaddr;

        servaddr.sin\_family=AF\_INET;

        servaddr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

        servaddr.sin\_port=htons(6006);

        sin\_size=sizeof(struct sockaddr\_in);

        if((con=connect(sockfd,(struct sockaddr \*) &servaddr, sin\_size))==0)

        {

                printf("Connected to server\n");

        }

        printf("Enter operands and operation to perform\n");

        scanf("%d %d %c",&op1,&op2,&operat);

        write(sockfd,&operat,10);

        write(sockfd,&op1,sizeof(op1));

        write(sockfd,&op2,sizeof(op2));

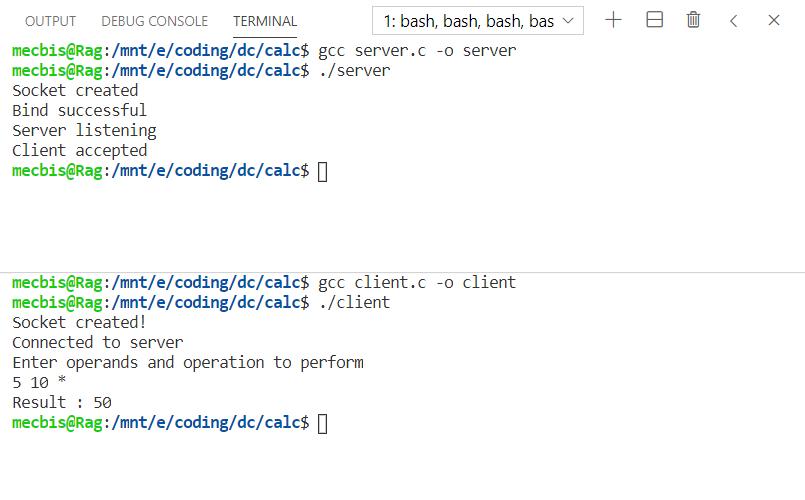
        read(sockfd,&result,sizeof(result));

        printf("Result : %d\n",result);

        close(sockfd);

}

**Output:**

****

**4. File Transfer**

**Algorithm:**

* **Step 1:** The server starts and waits for filename.
* **Step 2:** The client sends a filename.
* **Step 3:** The server receives filename. If file is present, server starts reading file

and continues to send a buffer filled with file contents encrypted until file-end is reached.

* **Step 4:** End is marked by EOF.
* **Step 5:** File is received as buffers until EOF is
* received. Then it is decrypted.
* **Step 6:** If Not present, a file not found is sent.

**Program:**

**SERVER SIDE:**

#include<unistd.h>

#include<stdio.h>

#include<sys/socket.h>

#include<stdlib.h>

#include<netinet/in.h>

#include<string.h>

#define PORT 4444

int main(int argc, char const \*argv[])

{

        int server\_fd, new\_socket, valread;

        struct sockaddr\_in address;

        int opt=1;

        int addrlen = sizeof(address);

        char buf[1024]={0};

        if((server\_fd=socket(AF\_INET, SOCK\_STREAM, 0))==0)

        {

                perror("Socket failed");

                exit(EXIT\_FAILURE);

        }

        address.sin\_family=AF\_INET;

        address.sin\_addr.s\_addr=INADDR\_ANY;

        address.sin\_port=htons(PORT);

        if(bind(server\_fd,(struct sockaddr \*)&address, sizeof(address))<0)

        {

                perror("Bind failed");

                exit(EXIT\_FAILURE);

        }

        if(listen(server\_fd, 3) <0)

        {

                perror("Listen");

                exit(EXIT\_FAILURE);

        }

        if((new\_socket=accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

        {

                perror("Accept");

                exit(EXIT\_FAILURE);

        }

        FILE \*fp;

        read(new\_socket,buf,100);

        fp=fopen("servermessage.txt","w");

        fprintf(fp,"%s",buf);

        printf("File recieved\n");

        return 0;

}

**CLIENT SIDE:**

#include<stdio.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<unistd.h>

#include<string.h>

#define PORT 4444

int main(int argc, char const \*argv[])

{

        int sock=0, valread;

        struct sockaddr\_in serv\_addr;

        char buf[1024]={0};

        if((sock=socket(AF\_INET, SOCK\_STREAM, 0))<0)

        {

                printf("\nSocket creation error\n");

                return -1;

        }

        serv\_addr.sin\_family = AF\_INET;

        serv\_addr.sin\_port = htons(PORT);

        if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

        {

                printf("\nInvalid address\n");

                return -1;

        }

        if(connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr))<0)

        {

                printf("\nConnection failed\n");

                return -1;

        }

        FILE \*f;

        f=fopen("clientmessage.txt","r");

        fscanf(f,"%s",buf);

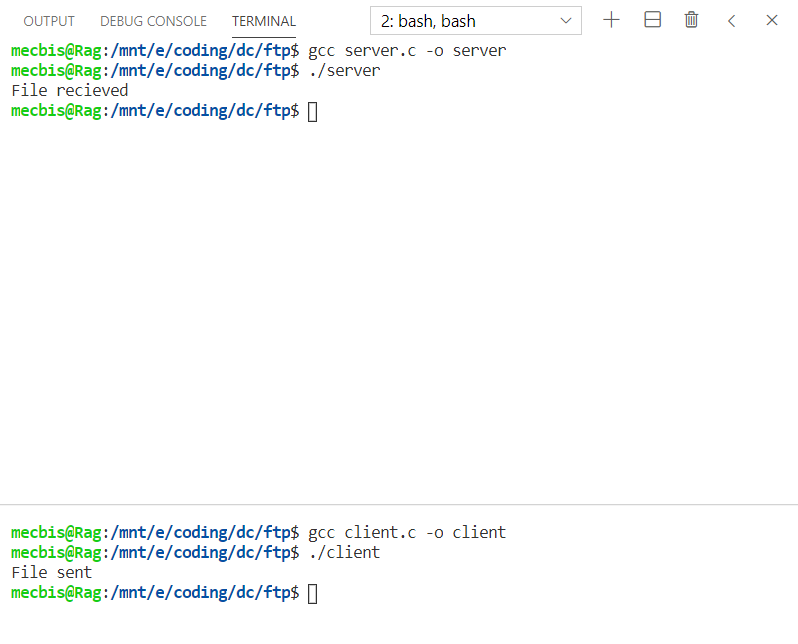
        write(sock,buf,100);

        printf("File sent\n");

        return 0;

}

**Output:**

****

**Result:**

The socket programming is done successfully using the C Programming language.

**SOCKET PROGRAMMING - POSTLAB**

**Describe the following core socket functions *(with syntax, data type, parameters and return value)* required to write a complete client and server program in C.**

**1. socket ()**

**Description:**

To perform network I/O, the first thing a process must do is, call the socket function, specifying the type of communication protocol desired and protocol family, etc.

**Syntax:**

int socket (int family, int type, int protocol);

**Parameters:**

**family** − It specifies the protocol family

**type** − It specifies the kind of socket you want

**protocol** − The argument should be set to the specific protocol or 0 to select the system's default for the given combination of family and type

**Return Type:**

This call returns a socket descriptor that you can use in later system calls or -1 on error.

**2. bind()**

**Description:**

The *bind* function assigns a local protocol address to a socket. With the Internet protocols, the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number. This function is called by TCP server only.

**Syntax:**

int bind(int sockfd, struct sockaddr \*my\_addr,int addrlen);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**my\_addr** − It is a pointer to struct sockaddr that contains the local IP address and port.

**addrlen** − Set it to sizeof(struct sockaddr).

**3. listen()**

**Description:**

The *listen* function is called only by a TCP server and it performs two actions −

The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.

The second argument to this function specifies the maximum number of connections the kernel should queue for this socket.

**Syntax:**

int listen(int sockfd,int backlog);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**backlog** − It is the number of allowed connections.

**Return Type:**

This call returns 0 on success, otherwise it returns -1 on error.

**4. accept()**

**Description:**

The *accept* function is called by a TCP server to return the next completed connection from the front of the completed connection queue.

**Syntax:**

int accept (int sockfd, struct sockaddr \*cliaddr, socklen\_t \*addrlen);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**cliaddr** − It is a pointer to struct sockaddr that contains client IP address and port.

**addrlen** − Set it to sizeof(struct sockaddr).

**Return Type:**

This call returns a non-negative descriptor on success, otherwise it returns -1 on error.

**5. read()**

**Description:**

The *read* function attempts to read nbyte bytes from the file associated with the buffer, fildes, into the buffer pointed to by buf.

**Syntax:**

int read(int fildes, const void \*buf, int nbyte);

**Parameters:**

**fildes** − It is a socket descriptor returned by the socket function.

**buf** − It is the buffer to read the information into.

**nbyte** − It is the number of bytes to read.

**Return Type:**

Upon successful reading, it returns the number of bytes read, and -1 when there is any error.

**6. write()**

**Description:**

The *write* function attempts to write nbyte bytes from the buffer pointed by *buf* to the file associated with the open file descriptor, *fildes*.

**Syntax:**

int write(int fildes, const void \*buf, int nbyte);

**Parameters:**

**fildes** − It is a socket descriptor returned by the socket function.

**buf** − It is a pointer to the data you want to send.

**nbyte** − It is the number of bytes to be written. If nbyte is 0, write() will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.

**Return Type:**

Upon successful completion, write() returns the number of bytes actually written to the file associated with fildes. This number is never greater than nbyte. Otherwise, -1 is returned.

**7. connect()**

**Description:**

The *connect* function is used by a TCP client to establish a connection with a TCP server.

**Syntax:**

int connect(int sockfd, struct sockaddr \*serv\_addr, int addrlen);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**serv\_addr** − It is a pointer to struct sockaddr that contains destination IP address and port.

**addrlen** − Set it to sizeof(struct sockaddr).

**Return Type:**

This call returns 0 if it successfully connects to the server, otherwise it returns -1 on error.

**8. close()**

**Description:**

The *close* function is used to close the communication between the client and the server.

**Syntax:**

int close(int sockfd);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**Return Type:**

This call returns 0 on success, otherwise it returns -1 on error.

**9. send()**

**Description:**

The *send* function is used to send data over stream sockets or CONNECTED datagram sockets. If you want to send data over UNCONNECTED datagram sockets, you must use sendto() function.

**Syntax:**

int send(int sockfd, const void \*msg, int len, int flags);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**msg** − It is a pointer to the data you want to send.

**len** − It is the length of the data you want to send (in bytes).

**flags** − It is set to 0.

**Return Type:**

This call returns the number of bytes sent out, otherwise it will return -1 on error.

**10. recv ()**

**Description:**

The *recv* function is used to receive data over stream sockets or CONNECTED datagram sockets. If you want to receive data over UNCONNECTED datagram sockets you must use recvfrom().

**Syntax:**

int recv(int sockfd, void \*buf, int len, unsigned int flags);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**buf** − It is the buffer to read the information into.

**len** − It is the maximum length of the buffer.

**flags** − It is set to 0.

**Return Type:**

This call returns the number of bytes read into the buffer, otherwise it will return -1 on error.

**11. recvfrom ()**

**Description:**

The *recvfrom* function is used to receive data from UNCONNECTED datagram sockets.

**Syntax:**

int recvfrom(int sockfd, void \*buf, int len, unsigned int flags struct sockaddr \*from, int \*fromlen;

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**buf** − It is the buffer to read the information into.

**len** − It is the maximum length of the buffer.

**flags** − It is set to 0.

**from** − It is a pointer to struct sockaddr for the host where data has to be read.

**fromlen** − It is set it to sizeof(struct sockaddr).

**Return Type:**

This call returns the number of bytes read into the buffer, otherwise it returns -1 on error.

**12. sendto()**

**Description:**

The *sendto* function is used to send data over UNCONNECTED datagram sockets.

**Syntax:**

int sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**msg** − It is a pointer to the data you want to send.

**len** − It is the length of the data you want to send (in bytes).

**flags** − It is set to 0.

**to** − It is a pointer to struct sockaddr for the host where data has to be sent.

**tolen** − It is set it to sizeof(struct sockaddr).

**Return Type:**

This call returns the number of bytes sent, otherwise it returns -1 on error.

**13. shutdown:**

**Description:**

The *shutdown* function is used to gracefully close the communication between the client and the server. This function gives more control in comparison to the *close* function.

**Syntax:**

int shutdown(int sockfd, int how;

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**how** − Put one of the numbers −

**0** − indicates that receiving is not allowed,

**1** − indicates that sending is not allowed, and

**2** − indicates that both sending and receiving are not allowed. When *how* is set to 2, it's the same thing as close()

**Return Type:**

This call returns 0 on success, otherwise it returns -1 on error.

**14. select()**

**Description:**

The *select* function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending.

**Syntax:**

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*errorfds, struct timeval \*timeout;

**Parameters:**

**nfds** − It specifies the range of file descriptors to be tested. The select() function tests file descriptors in the range of 0 to nfds-1

**readfds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for being ready to read, and on output, indicates which file descriptors are ready to read. It can be NULL to indicate an empty set.

**writefds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for being ready to write, and on output, indicates which file descriptors are ready to write. It can be NULL to indicate an empty set.

**exceptfds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for error conditions pending, and on output indicates, which file descriptors have error conditions pending. It can be NULL to indicate an empty set.

**timeout** − It points to a timeval struct that specifies how long the select call should poll the descriptors for an available I/O operation. If the timeout value is 0, then select will return immediately. If the timeout argument is NULL, then select will block until at least one file/socket handle is ready for an available I/O operation. Otherwise *select* will return after the amount of time in the timeout has elapsed OR when at least one file/socket descriptor is ready for an I/O operation.

**Return Type:**

The return value from select is the number of handles specified in the file descriptor sets that are ready for I/O. If the time limit specified by the timeout field is reached, select return 0.

**Result:**

The above functions have been explored and studied successfully.

**REMOTE PROCEDURE CALL - PRELAB**

**1. RPC is not suitable to pass pointers to remote procedures. Justify**

Values like integers, Booleans are easy since they can be copied into the message and sent without problems.

When we need to pass pointers, we will have a confusion whether the pointer is to be passed or the value. What happens if the pointer is pointing to something in the middle of a complex structure with other pointers inside? Here a new problem arises whether we need to send the whole structure? Or the parties should interact with each other asking for values from each pointer the remote function uses?

Since we have so many conflicts when we pass pointers to remote procedures, RPC is not suitable.

**2. Differentiate between SUN RPC and MS RPC:**

|  |  |
| --- | --- |
| **SUN RPC** | **MS RPC** |
| Originally developed by Sun, but now widely available on other platforms (including Digital Unix). Also known as Open Network Computing (ONC). | MSRPC was originally derived from open source software but has been developed further and copyrighted by Microsoft. |
| Sun RPC package has an RPC compiler (rpcgen) that automatically generates the client and server stubs. | The client stub code retrieves the required parameters from the client address space and delivers them to the client runtime library, which then translates the parameters into a standard Network Data Representation format to transmit to the server. |
| RPC package uses XDR (eXternal Data Representation) to represent data sent between client and server stubs. | The client stub then calls functions in the RPC client runtime library to send the request and parameters to the server.  If the server is located remotely, the runtime library specifies an appropriate transport protocol and engine and passes the RPC to the network stack for transport to the server. |

**3. Differentiate between Socket and RPC**

|  |  |
| --- | --- |
| **Socket** | **RPC** |
| Also commonly referred to as a "socket", used to describe an IP address and port, is a handle to a communication chain that can be used to communicate between different virtual machines or different computers. | RPC (Remote Procedure Call) is built on top of Socket. For an analogous desire, the main program running on one machine can call the subroutine prepared on another machine. |
| Each service opens a Socket and is bound to a port, with different ports corresponding to different services. | The basic principle of the RPC method is to ignore the specific details of the communication with the simplicity of the module call. |
| Sockets could be used to make the connection to the server | RPC is implemented on the basis of Socket. |

**4. Write down the functionalities of stub**

RMI (Remote Method Invocation) uses stub and skeleton object for communication with the remote object.

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

It initiates a connection with remote Virtual Machine (JVM),

It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),

It waits for the result

It reads (unmarshals) the return value or exception, and

It finally, returns the value to the caller.

**5. How does the procedure-oriented middleware differ from Object oriented middleware? Give an example**

|  |  |
| --- | --- |
| **Procedure Oriented Middleware** | **Object Oriented Middleware** |
| Procedure-oriented middleware is characterized by a client converting the parameters of a procedure into a message, sending this message to the server (or host), who then converts the message back into the parameters. | Object-Oriented Middleware evolved more or less directly from the idea of remote procedure calls |
| Good examples of this type of middleware are Open Network Computing (ONC) from Sun Microsystems, Inc. and Distributed Computing Environment (DCE) from Open Software Foundation (OSF). | The first of these systems was OMG’s Common Object Request Broker Architecture (CORBA). |
| It supports exceptions by returning a message that a failure occurred. | The idea here is to make object-oriented principles available for the development of distributed systems. |

**6. List out all the issues in RPC and also mention how it is resolved**

**i) Single Threaded Servers:**

RPC “forces” the choice of a multi-threaded server instead of single threaded. This happens because the RPC model doesn’t allow the server to return without serving a response to the client. If the data requested by the client is not immediately available the server has to wait and can’t start serving new requests. The most obvious choice then is to make the server multi-threaded.

**Solution:**

Nowadays is hard, although not impossible, to imagine single-threaded servers being built.  Finagle is built around *services* which are asynchronous functions and gRPC offers both synchronous and asynchronous function calls.

**ii) The Two Army Problem**

This is also known as “*Two Generals’ Problem*” and it states that is impossible for two processes to agree on a decision over an unreliable network.

**Solution:**

This might be the main reason for timeouts when using RPC. Since the system can’t know that the other party received the message and it could wait indefinitely for a response it chooses to timeout and then act properly by canceling the request, as seen above, retrying after some time or some other approach suitable for the application in questions.

**iii) Exception Handling**

When a procedure is executed locally it either completes or fails entirely. Remote procedures introduce new errors regarding the communication over the network and also when one party fails.

**Solution:**

The communication should be stable so that the procedure calls will not introduce any new errors.

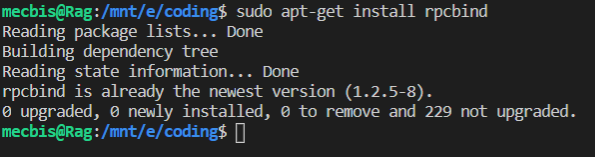
**7. What is marshalling / unmarshalling?**

**Marshalling** is the process of transforming the memory representation of an object to a data format suitable for the storage and transmission.

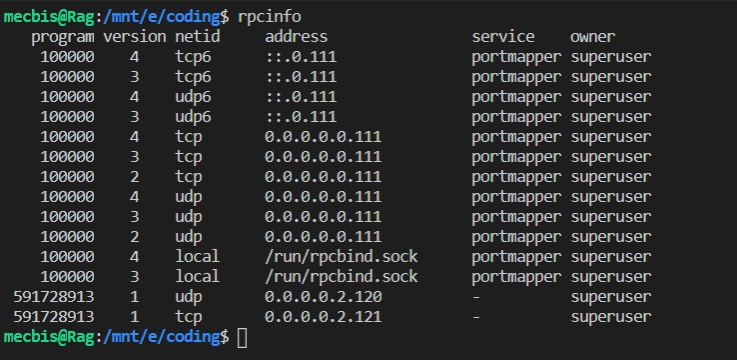
**Unmarshalling** refers to the process of transforming a representation of an object that is used for storage or transmission to a representation of the object that is executable.

**8. Execute a simple calculator program using rpcgen**

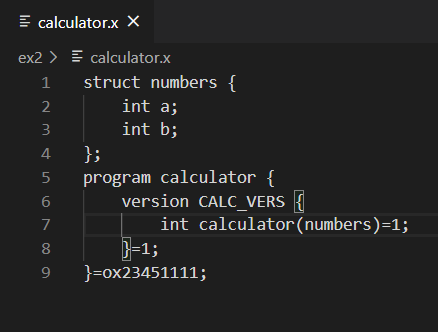
**Step 1:** The rpcbind package is installed in the first place.



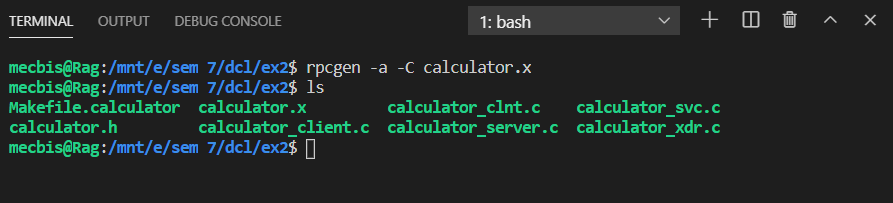
**Step 2:** Check if the package is working by using rpcinfo command



**Step 3:** Create a file with .x extension for the calculator program.

****

**Step 4:** Use the rpcgen command to generate the required files according to the calculator.x file

****

**Step 5:** Once the editing is done for the calculator\_client.c and calculator\_server.c, they are compiled using the make command.

**CALCULATOR\_SERVER.C:**

#include "calculator.h"

int \*

calculator\_1\_svc(numbers \*argp, struct svc\_req \*rqstp)

{

    static int  result;

    printf("The given choice is: %c\n",argp->c);

    printf("The given numbers are: %d %d\n",argp->a,argp->b);

    if(argp->c=='+') {

        result = argp->a + argp->b;

    }

    else if(argp->c=='-') {

        result = argp->a - argp->b;

    }

    else if(argp->c=='\*') {

        result = argp->a \* argp->b;

    }

    else {

        result = argp->a / argp->b;

    }

    return &result;

}

**CALCULATOR\_CLIENT.C:**

#include "calculator.h"

void

calculator\_prog\_1(char \*host,int num1,int num2,char operator)

{

    CLIENT \*clnt;

    int  \*result\_1;

    numbers  calculator\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, CALCULATOR\_PROG, CALC\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif  /\* DEBUG \*/

    calculator\_1\_arg.a = num1;

    calculator\_1\_arg.b = num2;

    calculator\_1\_arg.c = operator;

    result\_1 = calculator\_1(&calculator\_1\_arg, clnt);

    if (result\_1 == (int \*) NULL) {

        clnt\_perror (clnt, "call failed");

    }

    else

    {

        printf("The result is: %d\n",\*result\_1);

    }

#ifndef DEBUG

    clnt\_destroy (clnt);

#endif   /\* DEBUG \*/

}

int

main (int argc, char \*argv[])

{

    char \*host;

    if (argc < 5) {

        printf ("usage: %s server\_host\n", argv[0]);

        exit (1);

    }

    host = argv[1];

    int num1 = atoi(argv[2]);

    int num2 = atoi(argv[3]);

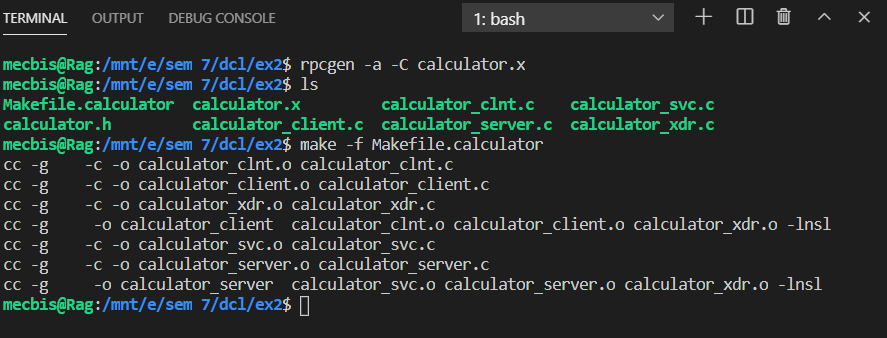
    char operator = argv[4][0];

    calculator\_prog\_1 (host,num1,num2,operator);

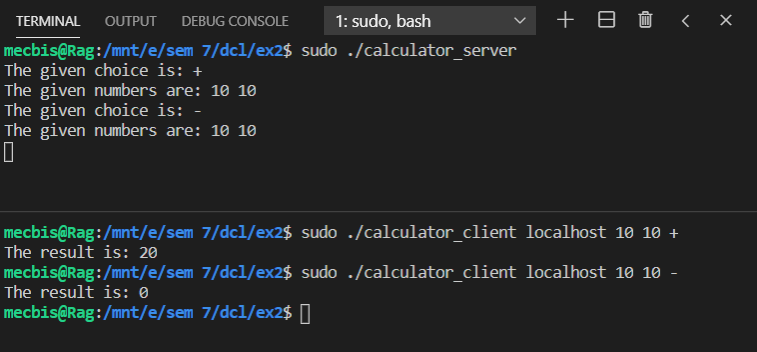
exit (0);

}

**Output:**

****

**Step 6:** Run the calculator\_server and calculator\_client to perform the calculator operations.

****

**Result:**

The Remote Procedure Call is understood successfully with the help of a program using rpcgen.

**REMOTE PROCEDURE CALL - EXERCISE**

**Aim:**

To write programs using SUN - RPCGEN package available in Linux operating system.

**1. Implement a factorial procedure**

**.X CODE:**

program fact {

    version FACT\_VERS {

        int factorial(int)=1;

    }=1;

}=0x23451111;

**SERVER CODE:**

#include "fact.h"

int \*

factorial\_1\_svc(int \*argp, struct svc\_req \*rqstp)

{

    static int result = 1;

    int facter = 1;

    printf("Computing factorial...\n");

    for(int i=1;i<=\*argp;i++) {

        facter = facter \* i;

    }

    result = facter;

    return &result;

}

**CLIENT CODE:**

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "fact.h"

void

fact\_1(char \*host)

{

    CLIENT \*clnt;

    int  \*result\_1;

    int  factorial\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, fact, FACT\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif  /\* DEBUG \*/

    while(361) {

        int choice;

        printf("Enter a choice:\n1.Compute factorial\n2.Exit\n");

        scanf("%d",&choice);

        if(choice==1) {

            int num;

            printf("Enter a number: ");

            scanf("%d",&num);

            factorial\_1\_arg = num;

            result\_1 = factorial\_1(&factorial\_1\_arg, clnt);

            if (result\_1 == (int \*) NULL) {

                clnt\_perror (clnt, "call failed");

            }

            else {

                printf("The factorial of the number is: %d\n",\*result\_1);

            }

        }

        else {

            exit(0);

        }

    }

#ifndef DEBUG

    clnt\_destroy (clnt);

#endif   /\* DEBUG \*/

}

int

main (int argc, char \*argv[])

{

    char \*host;

    if (argc < 2) {

        printf ("usage: %s server\_host\n", argv[0]);

        exit (1);

    }

    host = argv[1];

    fact\_1 (host);

exit (0);

}

**OUTPUT:**



**2. Develop a distributed application that implements dictionary lookup operation.**

**Server should maintain a dictionary of words and client interface should keep a cache of recently referenced words and searches the cache before making a remote procedure call. The dictionary may be maintained using string arrays.**

**When the client submits a word, it is checked in the client cache and if it is not found, a request is made to the server to fetch the meaning. Cache may be implemented using string array in the client side.**

**.X CODE:**

struct word {

    char s[100];

};

program dict {

    version DICT\_VERS {

        word search(word)=1;

    }=1;

}=0x23451111;

**SERVER CODE:**

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "dict.h"

word \*

search\_1\_svc(word \*argp, struct svc\_req \*rqstp)

{

    static word result;

    char words[100][100] = {"socket","blockchain","queue","database","UPS"};

    char meaning[100][100] = {"Programming in networks","Decentralized network","A data structure","Storehouse of data","Backup battery"};

    int index = -1;

    printf("Searching for the word...%s\n",argp->s);

    for(int i=0;i<100;i++) {

        if(strcmp(words[i],argp->s)==0) {

            index = i;

            break;

        }

    }

    if(index==-1) {

        printf("Word not found!\n");

        char ret[100] = "NF";

        strcpy(result.s,ret);

    }

    else {

        printf("Word found!\n");

        strcpy(result.s,meaning[index]);

        printf("The meaning returned is: %s\n",meaning[index]);

    }

    return &result;

}

**CLIENT CODE:**

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "dict.h"

void

dict\_1(char \*host)

{

    CLIENT \*clnt;

    word  \*result\_1;

    word  search\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, dict, DICT\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif  /\* DEBUG \*/

    char cache[100][100];

    char meaning[100][100];

    int cache\_size = 0;

    while(361) {

        int choice;

        printf("1. Search for a word\n2. Exit\n");

        scanf("%d",&choice);

        if(choice==1) {

            int flag = 0;

            char finder[100];

            printf("Enter the word to be searched\n");

            scanf("%s",finder);

            printf("The word searched is: %s\n",finder);

            for(int i=0;i<cache\_size;i++) {

                if(strcmp(cache[i],finder)==0) {

                    printf("Word found in cache\n");

                    printf("The meaning is: %s\n",meaning[i]);

                    flag = 1;

                    break;

                }

            }

            if(flag==0) {

                strcpy(search\_1\_arg.s,finder);

                result\_1 = search\_1(&search\_1\_arg, clnt);

                if (result\_1 == (word \*) NULL) {

                clnt\_perror (clnt, "call failed");

                }

                else {

                    printf("Word not found in cache!\n");

                    if(strcmp(result\_1->s,"NF")==0) {

                        printf("Word was not found!\n");

                        continue;

                    }

                    printf("The meaning retrieved from server is: %s\n",result\_1->s);

                    printf("Length is: %ld\n",strlen(result\_1->s));

                    strcpy(cache[cache\_size],finder);

                    strcpy(meaning[cache\_size],result\_1->s);

                    cache\_size++;

                    if(cache\_size==101) {

                        printf("Cache full!!\n");

                        cache\_size -= 5;

                        printf("Deleting last 5 value!\n");

                        printf("Freed up last 5 values successfully!\n");

                    }

                }

            }

        }

        if(choice==2) {

            exit(0);

        }

    }

#ifndef DEBUG

    clnt\_destroy (clnt);

#endif   /\* DEBUG \*/

}

int

main (int argc, char \*argv[])

{

    char \*host;

    if (argc < 2) {

        printf ("usage: %s server\_host\n", argv[0]);

        exit (1);

}

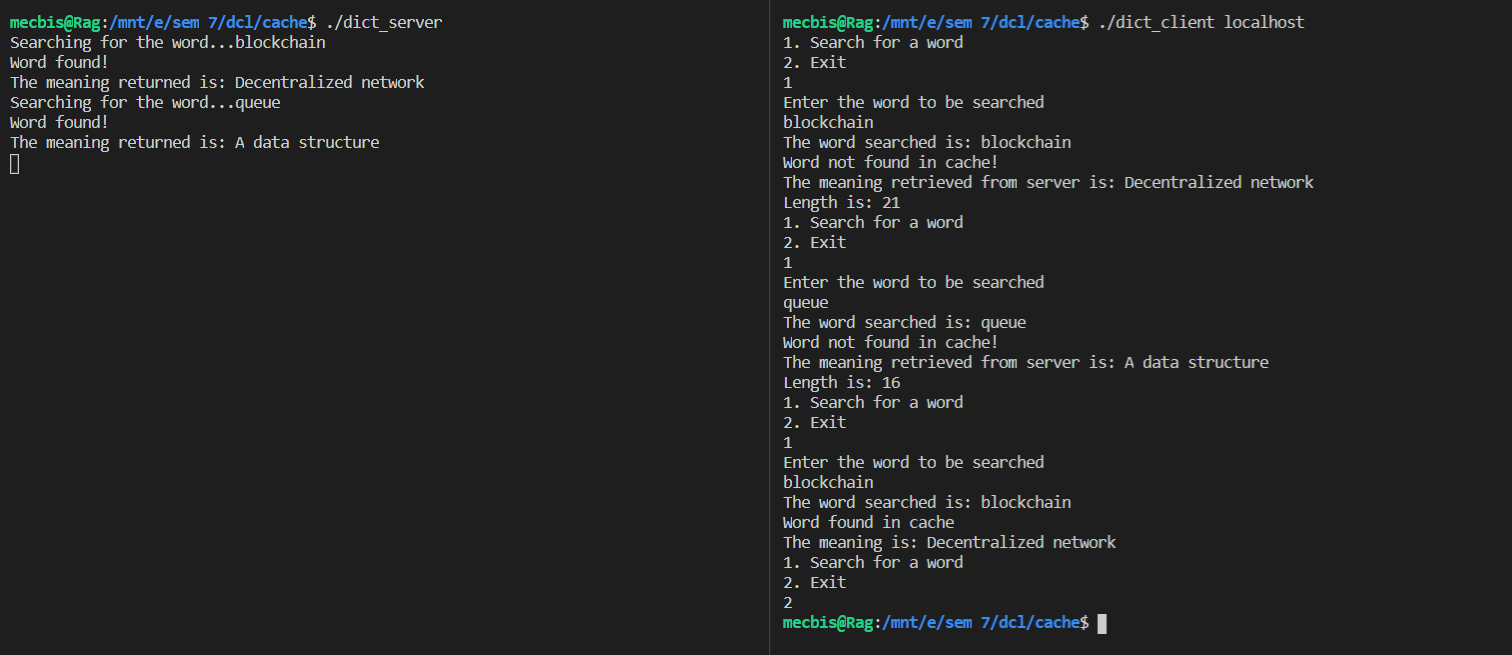
host = argv[1];

    dict\_1 (host);

exit (0);

}

**OUTPUT:**



**3. Online Test**

**.X CODE:**

struct question {

    char q[1000];

};

struct leader {

    char names[100];

    int mark;

};

struct qanda {

    int q;

    int a;

};

struct name {

    char n[100];

};

program test {

    version TEST\_VERS {

        void intialize(name)=1;

        question request(int)=2;

        void judge(qanda)=3;

        int score(name)=4;

        leader leaderboard(int)=5;

    }=1;

}=0x23451111;

**SERVER CODE:**

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "test.h"

#define N 3

char names[100][100];

int scores[100] = {0};

int countPpl = 0;

char questions[100][1000] = {"Where is PSG Tech?\n1.CBE\n2.MAS\n3.PUD\n4.KAN","Where is SVMHSS?\n1.UDT\n2.CBE\n3.MAS\n4.POC","Who is who?\n1.W\n2.L\n3.R\n4.W"};

int answers[100] = {0};

void \*

intialize\_1\_svc(name \*argp, struct svc\_req \*rqstp)

{

    static char \* result;

    answers[0] = 0;

    answers[1] = 0;

    answers[2] = 1;

    printf("Questions initialized and ready to be served!\n");

    strcpy(names[countPpl],argp->n);

    return (void \*) &result;

}

question \*

request\_1\_svc(int \*argp, struct svc\_req \*rqstp)

{

    printf("Sending a question!\n");

    static question  result;

    strcpy(result.q,questions[\*argp]);

    return &result;

}

void \*

judge\_1\_svc(qanda \*argp, struct svc\_req \*rqstp)

{

    static char \* result;

    if(answers[argp->q]==argp->a) {

        scores[countPpl]++;

    }

    if(argp->q==N-1) {

        countPpl++;

    }

    return (void \*) &result;

}

int \*

score\_1\_svc(name \*argp, struct svc\_req \*rqstp)

{

    printf("Score released!\n");

    static int  result;

    int index = -1;

    for(int i=0;i<countPpl;i++) {

        if(strcmp(names[i],argp->n)==0) {

            index = i;

            break;

        }

    }

    if(index==-1) {

        result = -1;

    }

    else {

        result = scores[index];

    }

    return &result;

}

leader \*

leaderboard\_1\_svc(int \*argp, struct svc\_req \*rqstp)

{

    static leader  result;

    if(\*argp==0) {

        printf("Leaderboard released!\n");

    }

    if(\*argp >= countPpl) {

        result.mark = -1;

    }

    else {

        strcpy(result.names,names[\*argp]);

        result.mark = scores[\*argp];

    }

    return &result;

}

**CLIENT CODE;**

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "test.h"

#define N 3

void

test\_1(char \*host)

{

    CLIENT \*clnt;

    void  \*result\_1;

    name  intialize\_1\_arg;

    question  \*result\_2;

    int  request\_1\_arg;

    void  \*result\_3;

    qanda  judge\_1\_arg;

    int  \*result\_4;

    name  score\_1\_arg;

    leader  \*result\_5;

    int  leaderboard\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, test, TEST\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif

    while(361) {

        printf("1.Start your test\n2.Show Leaderboard\n3.Exit\n");

        int choice;

        scanf("%d",&choice);

        if(choice==1) {

            printf("Enter your sweet name\n");

            scanf("%s",intialize\_1\_arg.n);

            result\_1 = intialize\_1(&intialize\_1\_arg, clnt);

            if (result\_1 == (void \*) NULL) {

                clnt\_perror (clnt, "call failed");

            }

            else {

                printf("Starting your test\n");

                for(int i=0;i<N;i++) {

                    request\_1\_arg = i;

                    result\_2 = request\_1(&request\_1\_arg, clnt);

                    if (result\_2 == (question \*) NULL) {

                        clnt\_perror (clnt, "call failed");

                    }

                    else {

                        int option;

                        printf("\nYour question %d\n",i+1);

                        printf("%s",result\_2->q);

                        printf("\nEnter your choice: ");

                        scanf("%d",&option);

                        judge\_1\_arg.q = i;

                        judge\_1\_arg.a = option%4;

                        result\_3 = judge\_1(&judge\_1\_arg, clnt);

                        if (result\_3 == (void \*) NULL) {

                            clnt\_perror (clnt, "call failed");

                        }

                    }

                }

                printf("Your test is completed now\n");

                strcpy(score\_1\_arg.n,intialize\_1\_arg.n);

                result\_4 = score\_1(&score\_1\_arg, clnt);

                if (result\_4 == (int \*) NULL) {

                    clnt\_perror (clnt, "call failed");

                }

                printf("Your score is: %d\n",\*result\_4);

                printf("You answered %d out of %d questions\n",\*result\_4,N);

                int sc = \*result\_4;

                printf("Your accuracy is: %f\n",(float)sc/(float)N);

            }

        }

        else if(choice==2) {

            int cur = 0;

            leaderboard\_1\_arg = cur;

            printf("\nName\tScore\n");

            while(361) {

                result\_5 = leaderboard\_1(&leaderboard\_1\_arg, clnt);

                if (result\_5 == (leader \*) NULL) {

                    clnt\_perror (clnt, "call failed");

                }

                if(result\_5->mark==-1) {

                    break;

                }

                else {

                    printf("%s\t%d\n",result\_5->names,result\_5->mark);

                    leaderboard\_1\_arg = ++cur;

                }

            }

            printf("\n");

        }

        else {

            printf("Thank you!\n");

            exit(0);

        }

    }

#ifndef DEBUG

    clnt\_destroy (clnt);

#endif   /\* DEBUG \*/

}

int

main (int argc, char \*argv[])

{

    char \*host;

    if (argc < 2) {

        printf ("usage: %s server\_host\n", argv[0]);

        exit (1);

    }

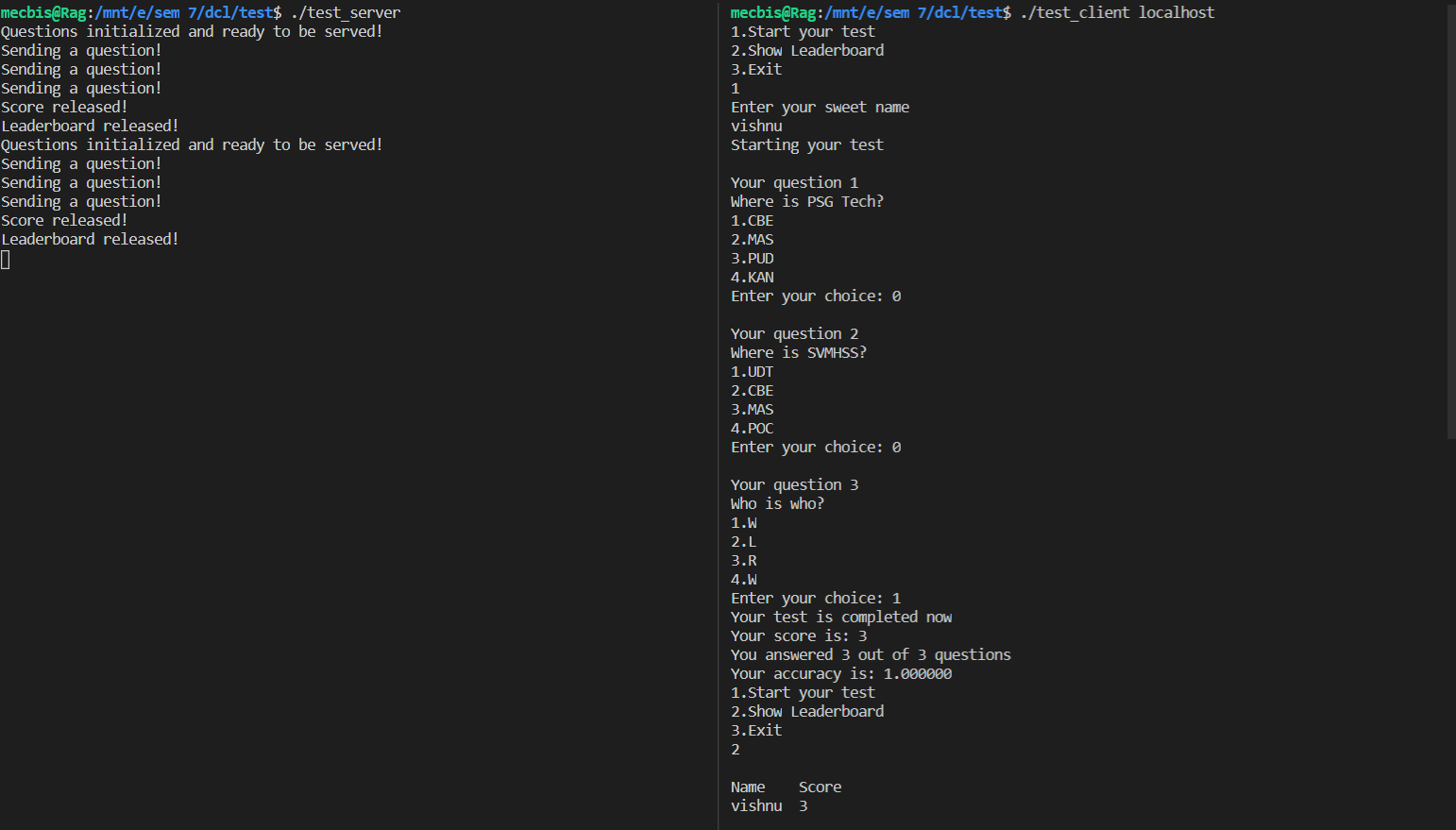
    host = argv[1];

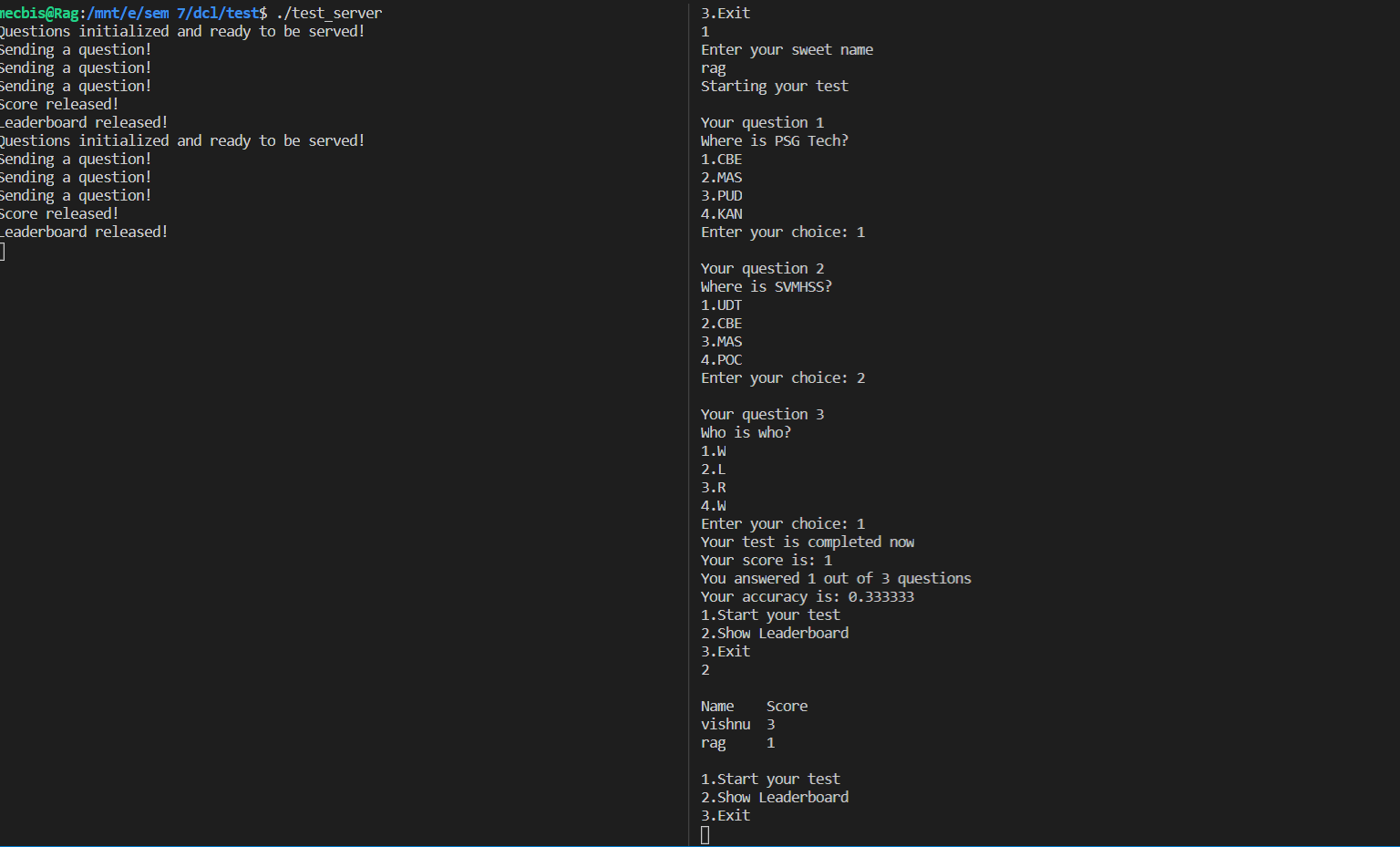
    test\_1 (host);

exit (0);

}

**OUTPUT:**

****

****

**Result:**

The programs are done successfully using the Sun RPCGEN.

**JAVA REMOTE METHOD INVOCATION - PRELAB**

**Aim:**

To study about the concepts of JAVA RMI and to implement a simple application using the same.

**1. How does the Java RMI differ from CORBA?**

|  |  |
| --- | --- |
| **JAVA RMI** | **COBRA** |
| Java only | Language independent Can access foreign language objects |
| Interfaces specified in Java | Interfaces specified externally in IDL |
| Pass objects by remote reference or by value (making local copy) | Pass objects by reference (by value possible but awkward) |
| Distributed garbage collection integrated with local collectors | Remote references manually released |
| Generally simpler to use | More complicated |

**2. List out the differences between RPC and RMI.**

|  |  |  |
| --- | --- | --- |
| **BASIS** | **RPC** | **RMI** |
| Supports | Procedural programming | Object-oriented programming |
| Parameters | Ordinary data structures are passed to remote procedures. | Objects are passed to remote methods. |
| Efficiency | Lower than RMI | More than RPC and supported by modern programming approach (i.e. Object-oriented paradigms) |
| Overheads | More | Less comparatively |
| In-Out parameters are mandatory | Yes | Not necessarily needed |
| Provision of ease of programming | High | Low |

**3. What does the following statement indicate? Naming.rebind ("//rmi:3000/ HelloRegistry ", object1);**

The code Naming.rebind(“RMI\_INSTANCE”, implclass); informs RMI registry that impl class object is available with the name “RMI\_INSTANCE”

**About the rebind function:**

public static void rebind([String](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.lang.String.html#_top_) name, [Remote](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.Remote.html#_top_) obj) throws [RemoteException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.RemoteException.html#_top_), [MalformedURLException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.net.MalformedURLException.html#_top_), [UnknownHostException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.UnknownHostException.html#_top_)

Rebind the name to a new object; replaces any existing binding.

**Throws:** [RemoteException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.RemoteException.html#_top_) If registry could not be contacted.

**4. What is the default port number used by RMI Registry?**

By default, the RMI Registry uses port **1099**. Client and server (stubs, remote objects) communicate over random ports unless a fixed port has been specified when exporting a remote object. The communication is started via a socket factory which uses 0 as starting port, which means use any port that's available between 1 and 65535.

**5. What is the difference between bind () and rebind () methods of Naming Class?**

**bind** method (String name) binds the specified name to a remote object while **rebind** (String

name) method rebinds the specified name to a new remote object, Any existing binding

for the name is replaced.

About the function:

**Bind:**

public static void bind([String](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.lang.String.html#_top_) name, [Remote](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.Remote.html#_top_) obj) throws [AlreadyBoundException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.AlreadyBoundException.html#_top_), [MalformedURLException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.net.MalformedURLException.html#_top_), [UnknownHostException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.UnknownHostException.html#_top_), [RemoteException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.RemoteException.html#_top_)

Binds the name to the specified remote object.

**Throws:** [RemoteException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.RemoteException.html#_top_)

If registry could not be contacted.

**Throws:** [AlreadyBoundException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.AlreadyBoundException.html#_top_)

If name is already bound.

**Rebind:**

public static void rebind([String](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.lang.String.html#_top_) name, [Remote](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.Remote.html#_top_) obj) throws [RemoteException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.RemoteException.html#_top_), [MalformedURLException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.net.MalformedURLException.html#_top_), [UnknownHostException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.UnknownHostException.html#_top_)

Rebind the name to a new object; replaces any existing binding.

**Throws:** [RemoteException](https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.rmi.RemoteException.html#_top_)

If registry could not be contacted.

**6. Implement Print Hello Program using Java RMI and include the screenshot**

**Java code:**

**RMI\_Interface.java:**

public interface RMI\_Interface extends java.rmi.Remote

{

    public String print() throws java.rmi.RemoteException;

}

**RMI\_Impl.java:**

import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

public class RMI\_Impl extends UnicastRemoteObject implements RMI\_Interface

{

    private static final long serialVersionUID = 1L;

    public RMI\_Impl() throws RemoteException

    {

    }

    public String print()

    {

        return new String("Helloworld!");

    }

}

**RMI\_Server.java:**

import java.rmi.\*;

public class RMI\_Server {

    public static void main(String args[])

    {

        try

        {

            RMI\_Impl implclass=new RMI\_Impl();

            Naming.rebind("RMI\_INSTANCE",implclass);

        }

        catch (Exception e)

        {

            System.out.println("Exception occured:" + e);

        }

    }

}

**RMI\_Client.java:**

import java.rmi.\*;

public class RMI\_Client

{

    public static void main(String[] argv)

    {

        if (argv.length !=1)

        {

            System.out.println("Usage: java RMI\_Client &lt;IP addr of host running RMI Server");

            System.exit(0);

        }

        String serverName = argv[0];

        try

        {

            RMI\_Interface myserverobject = (RMI\_Interface)Naming.lookup("rmi://"+serverName+"/RMI\_INSTANCE");

            String s = myserverobject.print();

            System.out.println("Server says: " + s);

        }

        catch(Exception e)

        {

            System.out.println("Exception Occured " + e);

            System.exit(0);

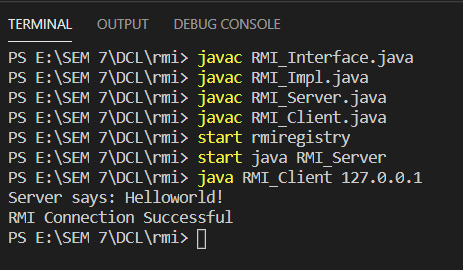
        }

        System.out.println("RMI Connection Successful");

    }

}

**OUTPUT:**

****

**Result:**

The concepts about JAVA RMI is studied and a simple application is executed using that successfully.

**JAVA – REMOTE METHOD INVOCATION - EXERCISE**

**Aim:**

To implement client sever programs using JAVA RMI.

**Question 1:**

**Develop a simple client/server banking service application using Java RMI. Whenever a user-logins, the user is allowed to perform the following operations:**

**Deposit operation: one can deposit some amount to an existing account**

**Withdraw operation: one can withdraw some amount from an existing account. While withdrawing, it should check for minimum balance. The minimum balance for an account is 500.**

**Balance Enquiry: one can check the balance of an account**

**Transfer operation: one can transfer some amount to another existing account. While Transferring check for Minimum balance.**

**If there isn't sufficient money in the account, then an error message should be returned. Check the existence of the account(s) in all the operations and return an error message if it doesn’t exist.**

**Code:**

**RMI\_Interface.java**

public interface RMI\_Interface extends java.rmi.Remote

{

    public String create(String accno,String password) throws java.rmi.RemoteException;

    public String login(String accno, String password) throws java.rmi.RemoteException;

    public String deposit(String accno, int amount) throws java.rmi.RemoteException;

    public String withdraw(String accno, int amount) throws java.rmi.RemoteException;

    public String balance(String accno) throws java.rmi.RemoteException;

    public String transfer(String sender, String reciever, int amount) throws java.rmi.RemoteException;

}

**RMI\_Impl.java:**

import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

import java.util.\*;

class Account

{

    public String name;

    public String password;

    public int balance;

    Account(String name,String password)

    {

        this.name = name;

        this.password = password;

        this.balance = 0;

    }

}

public class RMI\_Impl extends UnicastRemoteObject implements RMI\_Interface

{

    private static final long serialVersionUID = 1L;

    ArrayList<Account> accounts = new ArrayList<>();

    public RMI\_Impl() throws RemoteException

    {

    }

    public String create(String accno,String password)

    {

        int flag = -1;

        for(int i=0;i<accounts.size();i++)

        {

            if(accounts.get(i).name.equals(accno))

            {

                flag = 361;

                break;

            }

        }

        if(flag==361)

        {

            return "Account already exits!";

        }

        else

        {

            accounts.add(new Account(accno,password));

            return "Account created successfully!";

        }

    }

    public String login(String accno, String password)

    {

        int flag = -1;

        for(int i=0;i<accounts.size();i++)

        {

            if(accounts.get(i).name.equals(accno) && accounts.get(i).password.equals(password))

            {

                flag = 361;

                break;

            }

        }

        if(flag==361)

        {

            return "SUCCESS";

        }

        else

        {

            return "FAIL";

        }

    }

    public String deposit(String accno, int amount)

    {

        int flag = -1;

        for(int i=0;i<accounts.size();i++)

        {

            if(accounts.get(i).name.equals(accno))

            {

                flag = i;

                break;

            }

        }

        if(flag!=-1)

        {

            accounts.get(flag).balance += amount;

            return "Deposit successful! Current balance: " + Integer.toString(accounts.get(flag).balance);

        }

        else

        {

            return "No account exists!";

        }

    }

    public String withdraw(String accno, int amount)

    {

        int flag = -1;

        for(int i=0;i<accounts.size();i++)

        {

            if(accounts.get(i).name.equals(accno))

            {

                flag = i;

                break;

            }

        }

        if(flag!=-1)

        {

            int curBalance = accounts.get(flag).balance - amount;

            if(curBalance < 500)

            {

                return "Transaction not allowed!";

            }

            else

            {

                accounts.get(flag).balance -= amount;

                return "Withdraw successful! Current balance: " + Integer.toString(accounts.get(flag).balance);

            }

        }

        else

        {

            return "No account exists!";

        }

    }

    public String balance(String accno)

    {

        int flag = -1;

        for(int i=0;i<accounts.size();i++)

        {

            if(accounts.get(i).name.equals(accno))

            {

                flag = i;

                break;

            }

        }

        if(flag!=-1)

        {

            return "Your balance is: " + Integer.toString(accounts.get(flag).balance);

        }

        else

        {

            return "No account exists!";

        }

    }

    public String transfer(String sender, String reciever, int amount)

    {

        int flag1 = -1, flag2 = -1;

        for(int i=0;i<accounts.size();i++)

        {

            if(accounts.get(i).name.equals(sender))

            {

                flag1 = i;

            }

            if(accounts.get(i).name.equals(reciever))

            {

                flag2 = i;

            }

        }

        if(flag1==-1 || flag2==-1)

        {

            return "Sender/Reciever account do no exists!";

        }

        else

        {

            int curBalance = accounts.get(flag1).balance - amount;

            if(curBalance < 500)

            {

                return "Transaction not allowed!";

            }

            else

            {

                accounts.get(flag1).balance -= amount;

                accounts.get(flag2).balance += amount;

                return "Transfer successful! Current balance: " + Integer.toString(accounts.get(flag1).balance);

            }

        }

    }

}

**RMI\_Server.java:**

import java.rmi.\*;

public class RMI\_Server {

    public static void main(String args[])

    {

        try

        {

            RMI\_Impl implclass=new RMI\_Impl();

            Naming.rebind("RMI\_INSTANCE",implclass);

        }

        catch (Exception e)

        {

            System.out.println("Exception occured:" + e);

        }

    }

}

**RMI\_Client.java:**

import java.rmi.\*;

import java.util.\*;

public class RMI\_Client

{

    public static void main(String[] argv)

    {

        if (argv.length !=1)

        {

            System.out.println("Usage: java RMI\_Client &lt;IP addr of host running RMI Server");

            System.exit(0);

        }

        String serverName = argv[0];

        try

        {

            RMI\_Interface obj = (RMI\_Interface)Naming.lookup("rmi://"+serverName+"/RMI\_INSTANCE");

            Scanner sc = new Scanner(System.in);

            boolean loginFlag = false;

            while(true)

            {

                System.out.println("1. Create account\n2. Login\n3. Deposit\n4. Withdraw\n5. Balance\n6. Transfer\n7.Exit\n");

                int choice = sc.nextInt();

                if(choice==1)

                {

                    System.out.println("Enter your account number: ");

                    String accno = sc.next();

                    System.out.println("Enter your password: ");

                    String password = sc.next();

                    String output = obj.create(accno, password);

                    System.out.println(output);

                }

                else if(choice==2)

                {

                    System.out.println("Enter your account number: ");

                    String accno = sc.next();

                    System.out.println("Enter your password: ");

                    String password = sc.next();

                    String output = obj.login(accno, password);

                    if(output.equals("SUCCESS"))

                    {

                        loginFlag = true;

                    }

                }

                else if(choice==3)

                {

                    if(loginFlag==true)

                    {

                        System.out.println("Enter your account number: ");

                        String accno = sc.next();

                        System.out.println("Enter amount to deposit: ");

                        int amount = sc.nextInt();

                        String output = obj.deposit(accno, amount);

                        System.out.println(output);

                    }

                    else

                    {

                        System.out.println("Login first!");

                    }

                }

                else if(choice==4)

                {

                    if(loginFlag==true)

                    {

                        System.out.println("Enter your account number: ");

                        String accno = sc.next();

                        System.out.println("Enter amount to withdraw: ");

                        int amount = sc.nextInt();

                        String output = obj.withdraw(accno, amount);

                        System.out.println(output);

                    }

                    else

                    {

                        System.out.println("Login first!");

                    }

                }

                else if(choice==5)

                {

                    if(loginFlag==true)

                    {

                        System.out.println("Enter your account number: ");

                        String accno = sc.next();

                        String output = obj.balance(accno);

                        System.out.println(output);

                    }

                    else

                    {

                        System.out.println("Login first!");

                    }

                }

                else if(choice==6)

                {

                    if(loginFlag==true)

                    {

                        System.out.println("Enter your account number: ");

                        String accno1 = sc.next();

                        System.out.println("Enter your friend's number: ");

                        String accno2 = sc.next();

                        System.out.println("Enter amount to transfer: ");

                        int amount = sc.nextInt();

                        String output = obj.transfer(accno1,accno2,amount);

                        System.out.println(output);

                    }

                    else

                    {

                        System.out.println("Login first!");

                    }

                }

                else if(choice==7)

                {

                    break;

                }

            }

            sc.close();

        }

        catch(Exception e)

        {

            System.out.println("Exception Occured " + e);

            System.exit(0);

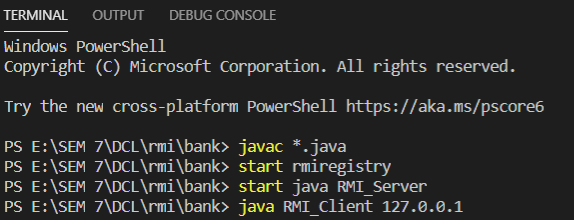
        }

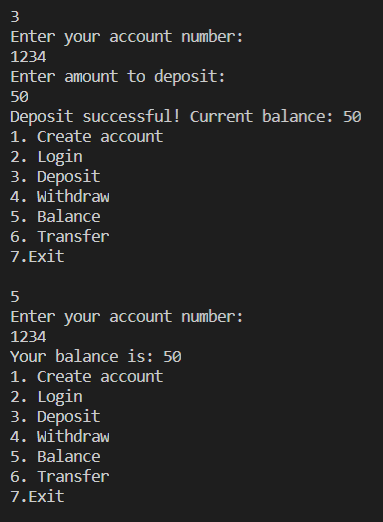
        System.out.println("Connection successful!");

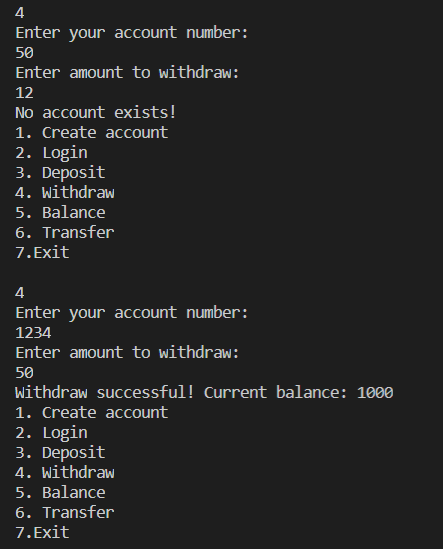
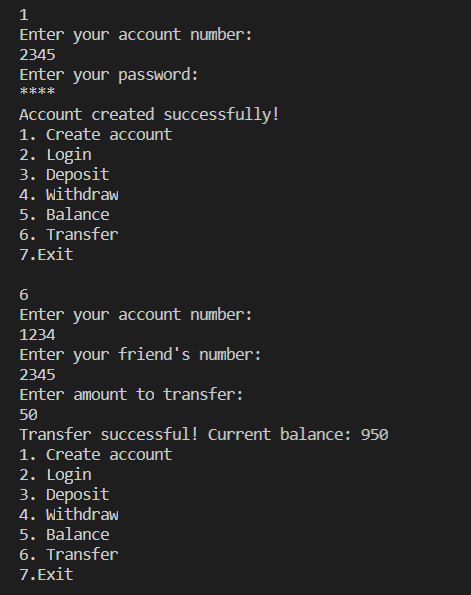
    }

}

**Output:**

****

****

****

**Question 2:**

**Develop a Patient Information system using Java RMI. This information system keeps track of the patient’s information and treatment details in a hospital or clinic. The Patient Information system offers following operations:**

**a.     Register patient details**

**b.    Search for Patient details by name**

**Use java swing or frames for the user interface.**

**Code:**

**RMI\_Interface.java:**

public interface RMI\_Interface extends java.rmi.Remote

{

    public String register(String patitenid, String name,String phoneNumber,int age,String entryDate, String dept, String disease, String doctor) throws java.rmi.RemoteException;

    public String search(String name) throws java.rmi.RemoteException;

}

**RMI\_Impl.java:**

import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

import java.util.\*;

class Patient

{

    String patientid;

    String name;

    String phoneNumber;

    int age;

    String entryDate;

    String dept;

    String disease;

    String doctor;

    Patient(String patitenid, String name,String phoneNumber,int age,String entryDate,String dept,String disease, String doctor)

    {

        this.patientid = patitenid;

        this.name = name;

        this.phoneNumber = phoneNumber;

        this.age = age;

        this.entryDate = entryDate;

        this.dept = dept;

        this.disease = disease;

        this.doctor = doctor;

    }

    public String toString()

    {

        return this.patientid + "," + this.name + "," + this.phoneNumber + "," + this.age + "," + this.entryDate + "," + this.dept + "," + this.disease + "," + this.doctor;

    }

}

public class RMI\_Impl extends UnicastRemoteObject implements RMI\_Interface

{

    private static final long serialVersionUID = 1L;

    ArrayList<Patient> patients = new ArrayList<>();

    public RMI\_Impl() throws RemoteException

    {

    }

    public String register(String patitenid, String name,String phoneNumber,int age,String entryDate,String dept,String disease, String doctor)

    {

        Patient p = new Patient(patitenid,name,phoneNumber,age,entryDate,dept,disease,doctor);

        patients.add(p);

        return "SUCCESS";

    }

    public String search(String name)

    {

        for(int i=0;i<patients.size();i++)

        {

            if(patients.get(i).name.equals(name))

            {

                return patients.get(i).toString();

            }

        }

        return "NOT FOUND";

    }

}

**RMI\_Server.java:**

import java.rmi.\*;

public class RMI\_Server {

    public static void main(String args[])

    {

        try

        {

            RMI\_Impl implclass=new RMI\_Impl();

            Naming.rebind("RMI\_INSTANCE",implclass);

        }

        catch (Exception e)

        {

            System.out.println("Exception occured:" + e);

        }

    }

}

**RMI\_Client.java:**

import java.rmi.\*;

import java.awt.\*;

import javax.swing.\*;

import java.awt.event.\*;

import javax.swing.JFrame;

import javax.swing.border.Border;

public class RMI\_Client

{

    public static void main(String[] argv)

    {

        if (argv.length !=1)

        {

            System.out.println("Usage: java RMI\_Client &lt;IP addr of host running RMI Server");

            System.exit(0);

        }

        String serverName = argv[0];

        try

        {

            RMI\_Interface myobj = (RMI\_Interface)Naming.lookup("rmi://"+serverName+"/RMI\_INSTANCE");

            JFrame jf = new JFrame();

            jf.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

            jf.setExtendedState(JFrame.MAXIMIZED\_BOTH);

            JButton register = new JButton("REGISTER");

            JButton search = new JButton("SEARCH");

            Label l1 = new Label("HOSPITAL MANAGEMENT SYSTEM");

            Font myFont = new Font("Times New Roman",Font.BOLD,20);

            Border border = BorderFactory.createLineBorder(new Color(255, 77, 77), 2);

            l1.setFont(myFont);

            l1.setBounds(650,120,350,40);

            register.setBounds(710,200,220,40);

            search.setBounds(710,280,220,40);

            register.addActionListener(new ActionListener()

            {

                public void actionPerformed(ActionEvent e)

                {

                    try

                    {

                        JFrame jf\_r = new JFrame();

                        jf\_r.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

                        jf\_r.setExtendedState(JFrame.MAXIMIZED\_BOTH);

                        Label center = new Label("HOSPITAL MANAGEMENT SYSTEM");

                        Label l1 = new Label("Patient ID");

                        Label l2 = new Label("Name");

                        Label l3 = new Label("Phone Number");

                        Label l4 = new Label("Age");

                        Label l5 = new Label("Entry Date");

                        Label l6 = new Label("Department");

                        Label l7 = new Label("Status");

                        Label l8 = new Label("Doctor");

                        JTextField jt1 = new JTextField("");

                        JTextField jt2 = new JTextField("");

                        JTextField jt3 = new JTextField("");

                        JTextField jt4 = new JTextField("");

                        JTextField jt5 = new JTextField("");

                        JTextField jt6 = new JTextField("");

                        JTextField jt7 = new JTextField("");

                        JTextField jt8 = new JTextField("");

                        JButton register\_button = new JButton("REGISTER");

                        center.setFont(myFont);

                        l1.setFont(myFont);

                        l2.setFont(myFont);

                        l3.setFont(myFont);

                        l4.setFont(myFont);

                        l5.setFont(myFont);

                        l6.setFont(myFont);

                        l7.setFont(myFont);

                        l8.setFont(myFont);

                        jt1.setFont(myFont);

                        jt2.setFont(myFont);

                        jt3.setFont(myFont);

                        jt4.setFont(myFont);

                        jt5.setFont(myFont);

                        jt6.setFont(myFont);

                        jt7.setFont(myFont);

                        jt8.setFont(myFont);

                        jt1.setBorder(border);

                        jt2.setBorder(border);

                        jt3.setBorder(border);

                        jt4.setBorder(border);

                        jt5.setBorder(border);

                        jt6.setBorder(border);

                        jt7.setBorder(border);

                        jt8.setBorder(border);

                        center.setBounds(650,120,350,40);

                        l1.setBounds(650,200,180,40);

                        l2.setBounds(650,250,180,40);

                        l3.setBounds(650,300,180,40);

                        l4.setBounds(650,350,180,40);

                        l5.setBounds(650,400,180,40);

                        l6.setBounds(650,450,180,40);

                        l7.setBounds(650,500,180,40);

                        l8.setBounds(650,550,180,40);

                        jt1.setBounds(850,200,220,40);

                        jt2.setBounds(850,250,220,40);

                        jt3.setBounds(850,300,220,40);

                        jt4.setBounds(850,350,220,40);

                        jt5.setBounds(850,400,220,40);

                        jt6.setBounds(850,450,220,40);

                        jt7.setBounds(850,500,220,40);

                        jt8.setBounds(850,550,220,40);

                        register\_button.setBounds(700,650,220,40);

                        register\_button.addActionListener(new ActionListener()

                        {

                            public void actionPerformed(ActionEvent e)

                            {

                                try

                                {

                                    String result = myobj.register(jt1.getText(), jt2.getText(), jt3.getText(),Integer.parseInt(jt4.getText()),jt5.getText(),jt6.getText(),jt7.getText(),jt8.getText());

                                    JOptionPane.showMessageDialog(null,result,"Message",JOptionPane.PLAIN\_MESSAGE);

                                }

                                catch(Exception d1)

                                {

                                    System.out.println(d1);

                                }

                            }

                        });

                        jf\_r.add(center);

                        jf\_r.add(l1);

                        jf\_r.add(l2);

                        jf\_r.add(l3);

                        jf\_r.add(l4);

                        jf\_r.add(l5);

                        jf\_r.add(l6);

                        jf\_r.add(l7);

                        jf\_r.add(l8);

                        jf\_r.add(jt1);

                        jf\_r.add(jt2);

                        jf\_r.add(jt3);

                        jf\_r.add(jt4);

                        jf\_r.add(jt5);

                        jf\_r.add(jt6);

                        jf\_r.add(jt7);

                        jf\_r.add(jt8);

                        jf\_r.add(register\_button);

                        jf\_r.setLayout(null);

                        jf\_r.setVisible(true);

                    }

                    catch(Exception e1)

                    {

                        System.out.println(e1);

                    }

                }

            });

            search.addActionListener(new ActionListener()

            {

                public void actionPerformed(ActionEvent e)

                {

                    try

                    {

                        JFrame jf\_s = new JFrame();

                        jf\_s.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

                        jf\_s.setExtendedState(JFrame.MAXIMIZED\_BOTH);

                        Label center = new Label("HOSPITAL MANAGEMENT SYSTEM");

                        Label l1 = new Label("Patient name");

                        JTextField jt1 = new JTextField("");

                        JButton search\_button = new JButton("SEARCH");

                        center.setFont(myFont);

                        l1.setFont(myFont);

                        jt1.setFont(myFont);

                        search\_button.setFont(myFont);

                        center.setBounds(650,120,350,40);

                        l1.setBounds(650,200,180,40);

                        jt1.setBounds(850,200,220,40);

                        search\_button.setBounds(700,280,220,40);

                        search\_button.addActionListener(new ActionListener()

                        {

                            public void actionPerformed(ActionEvent e)

                            {

                                try

                                {

                                    String result = myobj.search(jt1.getText());

                                    String unpack[] = result.split(",");

                                    if(unpack.length<8)

                                    {

                                        JOptionPane.showMessageDialog(null,"Not enough values!","Error",JOptionPane.ERROR\_MESSAGE);

                                    }

                                    Label l1 = new Label("Patient ID: " + unpack[0]);

                                    Label l2 = new Label("Name: " + unpack[1]);

                                    Label l3 = new Label("Phone Number: " + unpack[2]);

                                    Label l4 = new Label("Age: " + unpack[3]);

                                    Label l5 = new Label("Entry date: " + unpack[4]);

                                    Label l6 = new Label("Department: " + unpack[5]);

                                    Label l7 = new Label("Disease: " + unpack[6]);

                                    Label l8 = new Label("Doctor: " + unpack[7]);

                                    l1.setFont(myFont);

                                    l2.setFont(myFont);

                                    l3.setFont(myFont);

                                    l4.setFont(myFont);

                                    l5.setFont(myFont);

                                    l6.setFont(myFont);

                                    l7.setFont(myFont);

                                    l8.setFont(myFont);

                                    l1.setBounds(650,360,220,40);

                                    l2.setBounds(650,400,220,40);

                                    l3.setBounds(650,440,220,40);

                                    l4.setBounds(650,480,220,40);

                                    l5.setBounds(650,520,220,40);

                                    l6.setBounds(650,560,220,40);

                                    l7.setBounds(650,600,220,40);

                                    l8.setBounds(650,640,220,40);

                                    jf\_s.add(l1);

                                    jf\_s.add(l2);

                                    jf\_s.add(l3);

                                    jf\_s.add(l4);

                                    jf\_s.add(l5);

                                    jf\_s.add(l6);

                                    jf\_s.add(l7);

                                    jf\_s.add(l8);

                                }

                                catch(Exception d2)

                                {

                                    System.out.println(d2);

                                }

                            }

                        });

                        jf\_s.add(center);

                        jf\_s.add(l1);

                        jf\_s.add(jt1);

                        jf\_s.add(search\_button);

                        jf\_s.setLayout(null);

                        jf\_s.setVisible(true);

                    }

                    catch(Exception d1)

                    {

                        System.out.println(d1);

                    }

                }

            });

            jf.add(l1);

            jf.add(register);

            jf.add(search);

            jf.setLayout(null);

            jf.setVisible(true);

        }

        catch(Exception e)

        {

            System.out.println("Exception Occured " + e);

            System.exit(0);

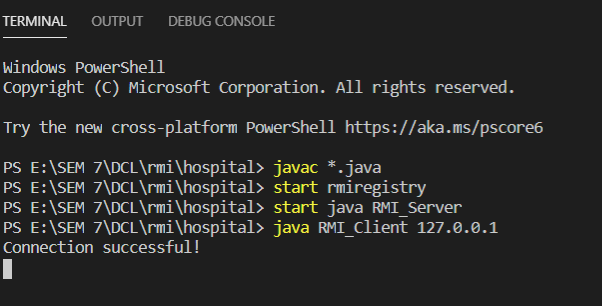
        }

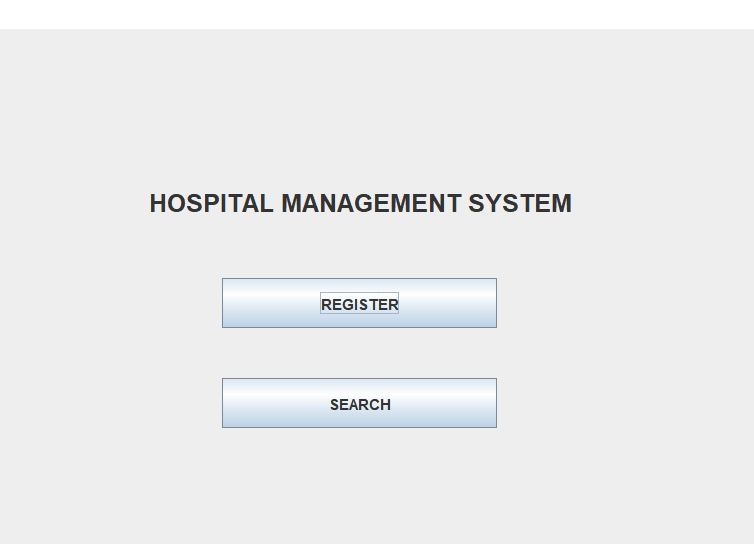
        System.out.println("Connection successful!");

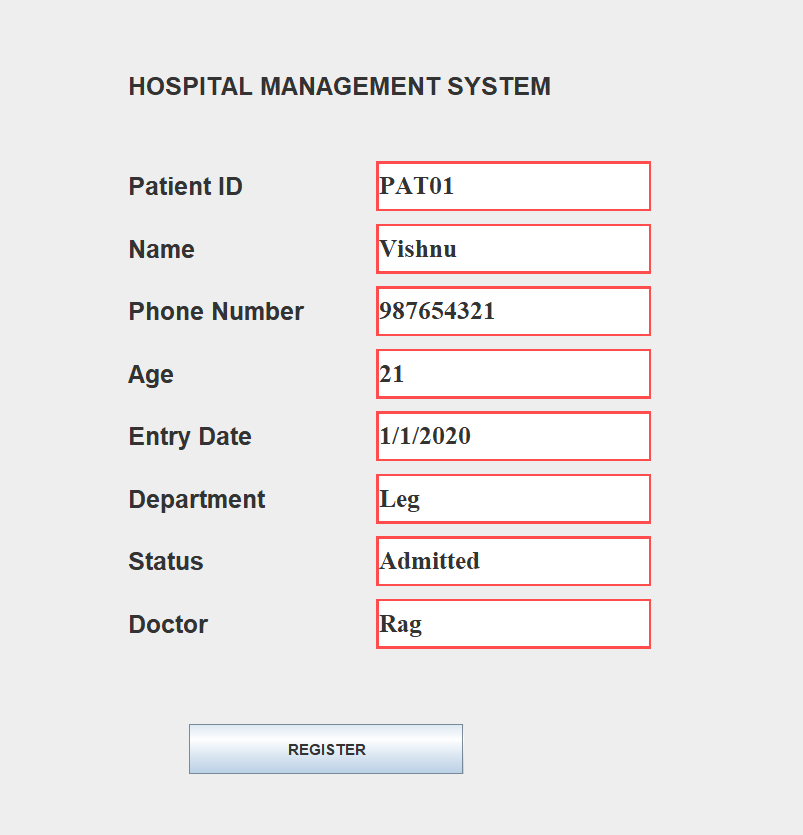
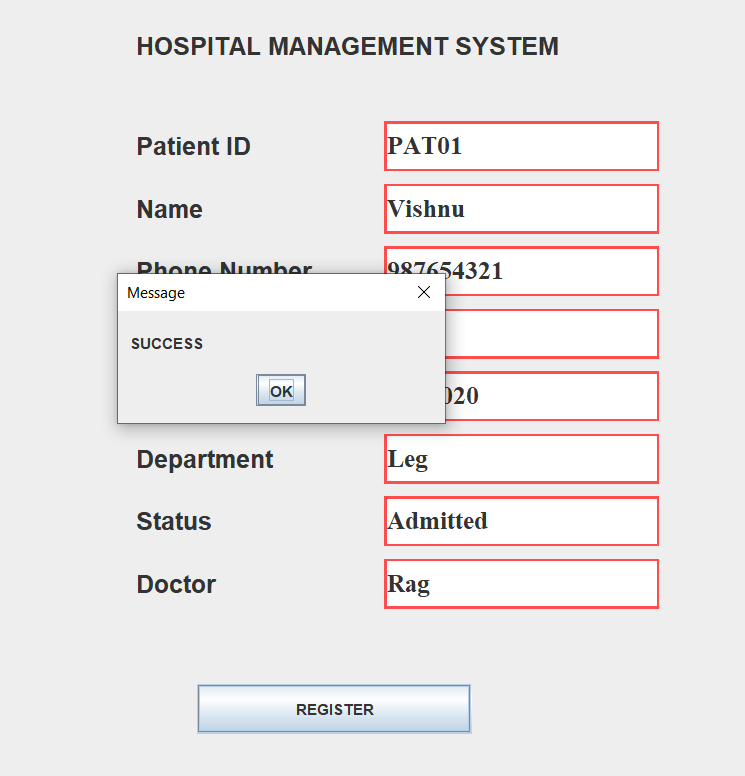
    }

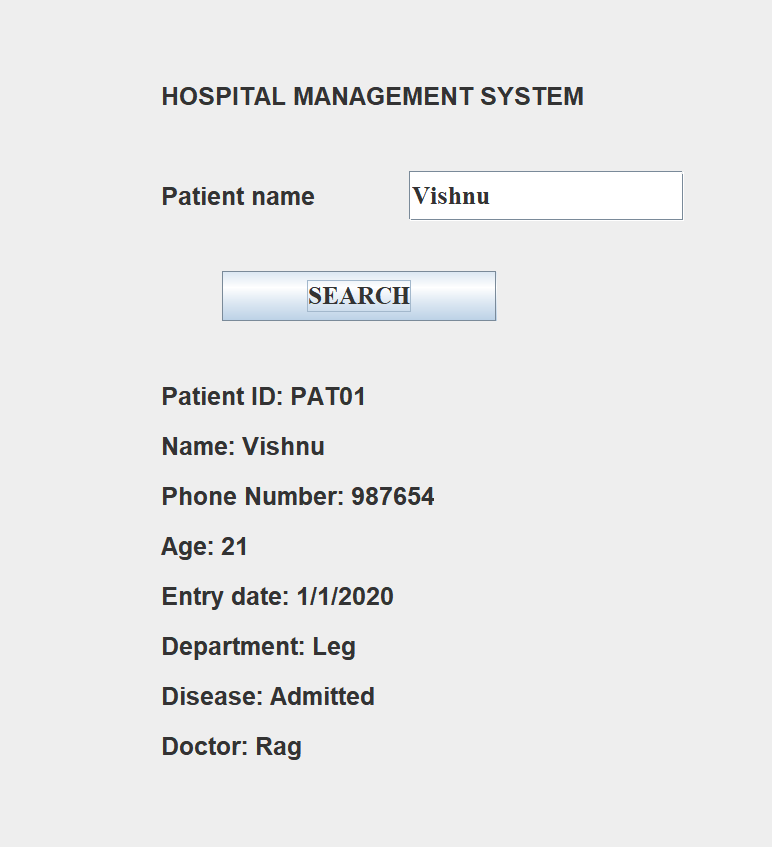
}

**Output:**

****





**Result:**

The given applications have been successfully implemented using JAVA RMI.

**ENTERPRISE JAVA BEAN - PRELAB**

**AIM:**

To study the concepts of EJB.

**QUESTIONS AND ANSWERS:**

1. **Comparison of EJB versions with examples.**

Major difference between EJB 2.0 and 3.0 are listed below in theory

* 1. Elimination of home interface gives simpler lookup process in EJB 3.0 than in EJB 2.0.
  2. Employment of annotations makes EJB 3.0 a latest technology while the EJB 2.0 has home and remote interfaces instead of annotations.
  3. EJB 2.0 deploys entity beans for accessing database. EJB 3.0 has Java Persistence API to access data which is generalized to address issues of portability.
  4. EJB 3.0 performs better because it uses POJOs along with metadata annotation, a new introduction. The POJOs are faster than XML Descriptor and JNDI deployed by EJB 2.0 for reference of objects. EJB 2.0 does not implement POJO.
  5. EJB 2.0 is heavy when it comes to writing home and remote interfaces. EJB 3.0 has no restriction on using any of the standard interfaces. It is a well configured and simple POJO which doesn’t need to execute container call back methods such as ejbActive, ejbStore, etc which are used by EJB 2.0.

1. **Explain EJB Types and its need**

EJB are of 3 different types as follows:

**Session EJB**

A session EJB is a nonpersistent object: Its lifetime is the duration of a particular interaction between the client and the EJB. The client normally creates an EJB, calls methods on it, and then removes it. If the client fails to remove it, the EJB container will remove it after a certain period of inactivity. Session EJBs are subdivided into ‘stateful’ and ‘stateless’ types. A stateless session EJB is shared amongst a number of clients, while a stateful session EJB is created for a specific client and not used by any others. The use of stateless EJBs offers efficiency advantages but, of course, it is not always possible to use them.

**Entity EJBs**

Entity EJBs represent persistent objects: Their lifetimes are not related to the duration of interaction with clients. In nearly all cases, entity EJBs are synchronized with relational

databases. This is how persistence is achieved. Entity EJBs are always shared amongst clients: A client cannot get an entity EJB to itself. Thus, entity EJBs are nearly always used as a scheme for mapping relational databases into object-oriented applications.

Whereas a client normally creates a session EJB and removes it after use, clients normally look up an existing entity EJB. Creation of an entity EJB corresponds to adding new data items to the application (e.g., adding rows to database tables).

An important feature of entity EJBs is that they have identity—that is, one can be distinguished from another. This is implemented by assigning a primary key to each instance of the EJB, where ‘primary key’ has the same meaning as it does for database management. Primary keys that identify EJBs can be of any type, including programmer-defined classes.

**Message-driven EJBs**

A message-driven bean acts as a consumer of asynchronous messages: It cannot be called directly by clients, but is activated by the container when a message arrives. Clients interact with these EJBs by sending messages to the queues or topics to which they are listening. Although a message-driven EJB cannot be called directly by clients, it can call other EJBs itself.

Message-driven EJBs are the only type that do not follow a strict request-response interaction with clients.

1. **Stateful vs stateless sessions**

Stateless session beans do not maintain state associated with any client. Each stateless session bean can server multiple clients.

Stateful session beans maintain the state associated with a client. Each stateful session bean serves exactly one client.

Stateless session beans are intended to be simple and lightweight; that is, they are easy to develop with low runtime resource requirements on the server. If required, any state is maintained by the client, and thereby makes the server highly scalable. Because no state is maintained in this enterprise bean type, stateless session beans aren't tied to any specific client. Therefore, any available instance of a stateless session bean can be used to service another client.

The container creates an implicit identity for a stateful session bean to manage its passivation and activation phases. On the other hand, the container doesn't create any identity for a stateless session bean.

The number of stateful session beans is equal to the number of active clients, whereas a small number of stateless session beans can be used to satisfy a large number of clients.

1. **Explain EJB annotations. What does EJB annotations do?**

Annotations were introduced in Java 5.0. The purpose of having annotations is to attach additional information in the class or a meta-data of a class within its source code. In EJB 3.0, annotations are used to describe configuration meta-data in EJB classes. By this way, EJB 3.0 eliminates the need to describe configuration data in configuration XML files.

EJB container uses compiler tool to generate required artifacts like interfaces, deployment descriptors by reading those annotations. Following is the list of commonly used annotations.

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Name** | **Description** |
| 1 | javax.ejb.Stateless | Specifies that a given EJB class is a stateless session bean.  **Attributes**   * **name** − Used to specify name of the session bean. * **mappedName** − Used to specify the JNDI name of the session bean. * **description** − Used to provide description of the session bean. |
| 2 | javax.ejb.Stateful | Specifies that a given EJB class is a stateful session bean.  **Attributes**   * **name** − Used to specify name of the session bean. * **mappedName** − Used to specify the JNDI name of the session bean. * **description** − Used to provide description of the session bean. |
| 3 | javax.ejb.MessageDrivenBean | Specifies that a given EJB class is a message driven bean.  **Attributes**   * **name** − Used to specify name of the message driven bean. * **messageListenerInterface** − Used to specify message listener interface for the message driven bean. * **activationConfig** − Used to specify the configuration details of the message- driven bean in an operational environment of the message driven bean. * **mappedName** − Used to specify the JNDI |

|  |  |  |
| --- | --- | --- |
|  |  | name of the session bean.   * **description** − Used to provide description of the session bean. |
| 4 | javax.ejb.EJB | Used to specify or inject a dependency as EJB instance into another EJB.  **Attributes**   * **name** − Used to specify name, which will be used to locate the referenced bean in the environment. * **beanInterface** − Used to specify the interface type of the referenced bean. * **beanName** − Used to provide name of the referenced bean. * **mappedName** − Used to specify the JNDI name of the referenced bean. * **description** − Used to provide description of the referenced bean. |
| 5 | javax.ejb.Local | Used to specify Local interface(s) of a session bean. This local interface states the business methods of the session bean (which can be stateless or stateful).  This interface is used to expose the business methods to local clients, which are running in the same deployment/application as EJB.  **Attributes**   * **value** − Used to specify the list of local interfaces as an array of interfaces. |
| 6 | javax.ejb.Remote | Used to specify Remote interface(s) of a session bean. This remote interface states the business methods of the session bean (which can be stateless or stateful).  This interface is used to expose the business methods to remote clients, which are running in different deployment/application as EJB.  **Attributes**   * **value** − Used to specify the list of remote |

|  |  |  |
| --- | --- | --- |
|  |  | interfaces as an array of interfaces. |
| 7 | javax.ejb.Activation ConfigProperty | Used to specify properties required for a message driven bean. For example, end point, destination, message selector etc.  This annotation is passed as a parameter to activationConfig attribute of javax.ejb.MessageDrivenBean annotation.  **Attributes**   * **propertyName** − name of the property. * **propertyValue** − value of the property. |
| 8 | javax.ejb.PostActivate | Used to specify callback method of EJB lifecycle. This method will be called when EJB container just activated/reactivated the bean instance.  This interface is used to expose the business methods to local clients, which are running in same deployment/application as EJB. |

**5. Implement a simple session bean to add two numbers SERVER:**

package addejb;

import javax.ejb.Stateless; @Stateless

public class SessionAddition implements SessionAdditionRemote { @Override

public int addMethod(int num1, int num2) { return num1+num2;

}

}

**CLIENT:**

package additionclient;

import addejb.SessionAdditionRemote; import javax.ejb.EJB;

import java.util.\*; public class Main { @EJB

private static SessionAdditionRemote sessionAddition; public static void main(String[] args) {

Scanner s=new Scanner(System.in);

System.out.println("Enter Num1 and Num2"); int num1=s.nextInt();

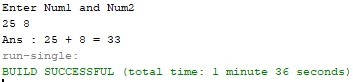
int num2=s.nextInt();

int ans=sessionAddition.addMethod(num1,num2); System.out.println("Ans : "+num1+" + "+num2+" = "+ans);

}

}

**Output:**



**Result:**

The concepts of EJB has been successfully studied and an addition program is made using simple session.

**ENTERPRISE JAVA BEAN - EXERCISE**

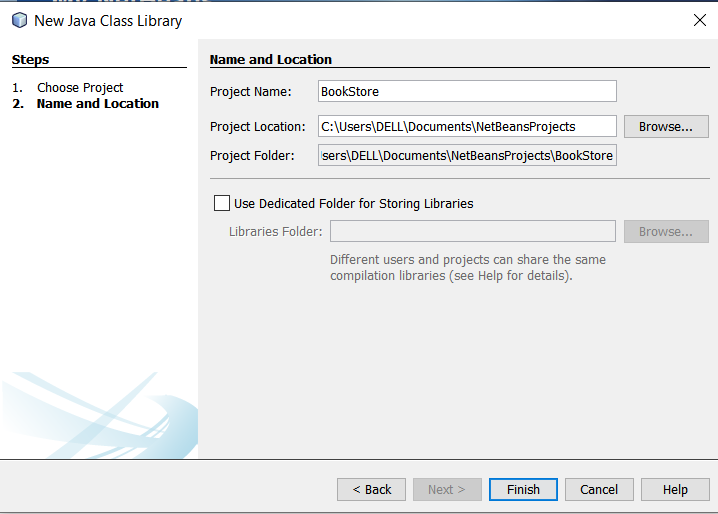
**Aim:**

To create Java Session Beans and implement the logic using the same for the given application.

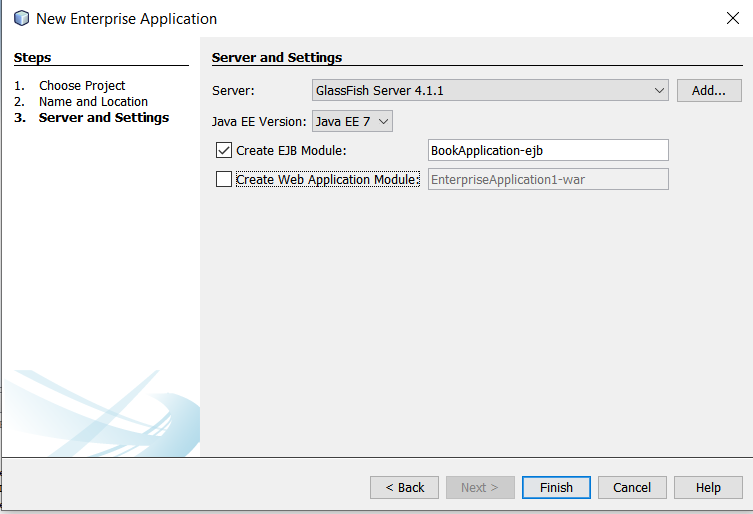
**Algorithm:**

**Step 1:** Start

**Step 2:** Create Java Class Library

****

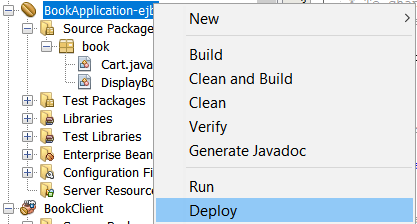
**Step 3:** Create Enterprise Application



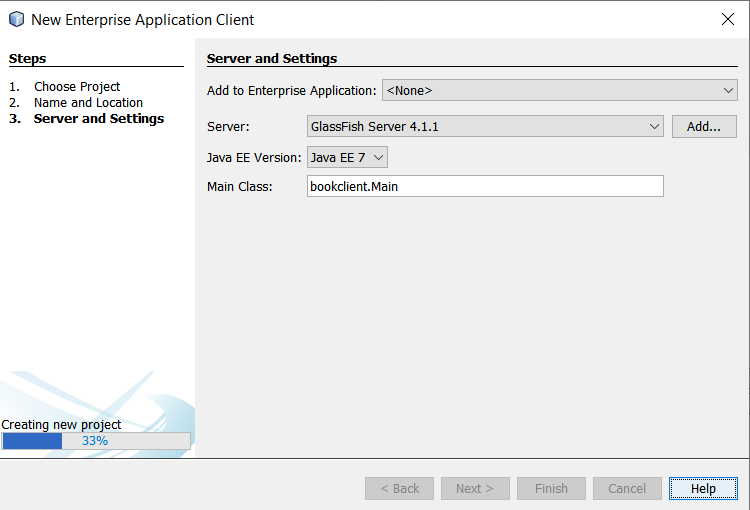
**Step 4:** Add the necessary beans



**Step 5:** Deploy the beans



**Step 6:** Create the Client



**Step 7**: Use the beans to implement the application

**Step 8:** Stop

**CODE:**

**DISPLAYBOOK.JAVA:**

/\*

 \* To change this license header, choose License Headers in Project Properties.

 \* To change this template file, choose Tools | Templates

 \* and open the template in the editor.

 \*/

package book;

import javax.ejb.Stateless;

import java.util.\*;

/\*\*

 \*

 \* @author DELL

 \*/

@Stateless

public class DisplayBook implements DisplayBookRemote {

    HashMap<String,Integer> books = new HashMap<>();

    public DisplayBook() {

        books.put("Java",100);

        books.put("Cpp",1000);

        books.put("Blockchain", 1999);

        books.put("DCL",1022);

        books.put("Interview",1243);

        books.put("Python",1320);

    }

    @Override

    public String bookDetails(String book) {

        if(books.containsKey(book)) {

            return Integer.toString(books.get(book));

        }

        else {

            return "No book found!";

        }

    }

    // Add business logic below. (Right-click in editor and choose

    // "Insert Code > Add Business Method")

}

**CART.JAVA:**

/\*

 \* To change this license header, choose License Headers in Project Properties.

 \* To change this template file, choose Tools | Templates

 \* and open the template in the editor.

 \*/

package book;

import javax.ejb.Stateless;

import java.util.\*;

/\*\*

 \*

 \* @author DELL

 \*/

class CreditCard {

    String sixteen;

    String cvv;

    String expiry;

    String name;

    CreditCard(String sixteen,String cvv,String expiry,String name) {

        this.sixteen = sixteen;

        this.cvv = cvv;

        this.expiry = expiry;

        this.name = name;

    }

}

@Stateless

public class Cart implements CartRemote {

    HashMap<String,Integer> books = new HashMap<>();

    ArrayList<CreditCard> cards = new ArrayList<>();

    ArrayList<String> cart = new ArrayList<>();

    public Cart() {

        books.put("Java",100);

        books.put("Cpp",1000);

        books.put("Blockchain", 1999);

        books.put("DCL",1022);

        books.put("Interview",1243);

        books.put("Python",1320);

        cards.add(new CreditCard("12345","123","1/2/2020","vishnu"));

        cards.add(new CreditCard("12347","124","1/3/2020","sishnu"));

        cards.add(new CreditCard("12349","126","1/4/2020","gishnu"));

        cards.add(new CreditCard("12348","127","1/5/2020","hishnu"));

        cards.add(new CreditCard("12346","128","1/6/2020","wishnu"));

    }

    @Override

    public String addToCard(String book) {

        cart.add(book);

        return "Added";

    }

    // Add business logic below. (Right-click in editor and choose

    // "Insert Code > Add Business Method")

    @Override

    public String removeFromCart(String book) {

        cart.remove(book);

        return "Removed!";

    }

    @Override

    public int bill() {

        int amount = 0;

        for(int i=0;i<cart.size();i++) {

            amount += books.get(cart.get(i));

        }

        return amount;

    }

    @Override

    public String payment() {

        return null;

    }

    @Override

    public String pay(String number, String cvv, String date, String name) {

        int flag = 0;

        for(int i=0;i<cards.size();i++) {

            if(cards.get(i).sixteen.equals(number) &&

               cards.get(i).cvv.equals(cvv) &&

               cards.get(i).expiry.equals(date) &&

               cards.get(i).name.equals(name)) {

                flag = 1;

                break;

            }

        }

        if(flag==1) {

            cart.clear();

            return "Successfully Completed!";

        }

        return "Payment details not found!";

    }

}

**CLIENT SIDE MAIN.JAVA:**

package bookclient;

import book.CartRemote;

import book.DisplayBookRemote;

import java.util.\*;

import javax.ejb.EJB;

/\*\*

 \*

 \* @author DELL

 \*/

public class Main {

    @EJB

    private static DisplayBookRemote displayBook;

    @EJB

    private static CartRemote cart;

    /\*\*

     \* @param args the command line arguments

     \*/

    public static void main(String[] args) {

        // TODO code application logic here

        Scanner sc = new Scanner(System.in);

        String book,output;

        while(true) {

            System.out.println("Enter your choice: \n1. Search book\n2. Add to cart\n3. Remove from cart\n4. Bill\n5. Pay");

            int choice = sc.nextInt();

            switch(choice) {

                    case 1:

                        System.out.println("Enter book name: ");

                        book = sc.next();

                        output = displayBook.bookDetails(book);

                        System.out.println("Book found: " + book + ": " + output);

                        break;

                    case 2:

                        System.out.println("Enter book name: ");

                        book = sc.next();

                        output = cart.addToCard(book);

                        System.out.println(output);

                        break;

                    case 3:

                        System.out.println("Enter book name: ");

                        book = sc.next();

                        output = cart.removeFromCart(book);

                        System.out.println(output);

                        break;

                    case 4:

                        System.out.println("The biil is: " + cart.bill());

                        break;

                    case 5:

                        String number = sc.next();

                        String cvv = sc.next();

                        String expiry = sc.next();

                        String name = sc.next();

                        output = cart.pay(number, cvv, expiry, name);

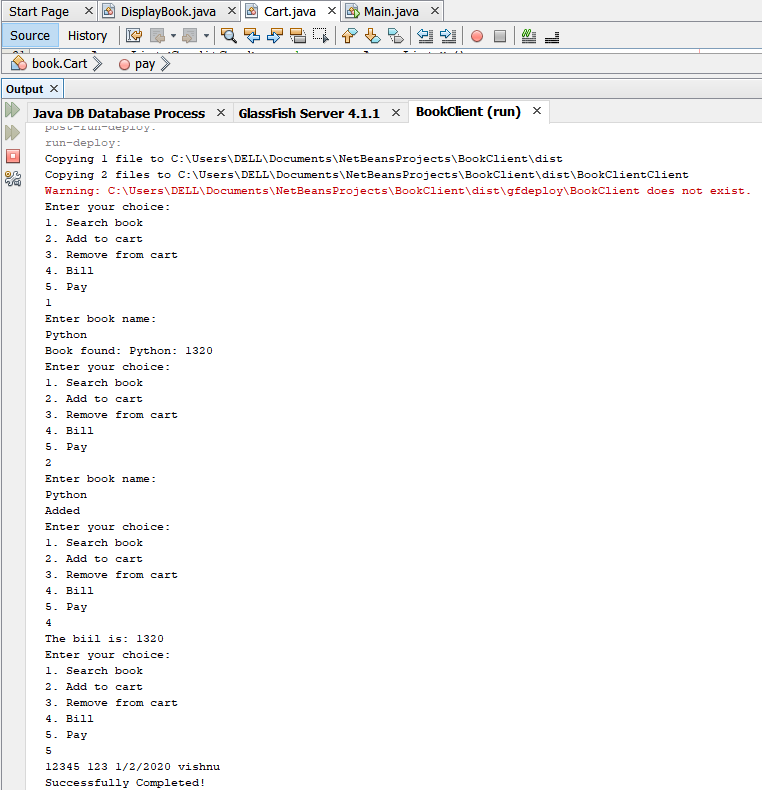
                        System.out.println(output);

                        break;

                    default:

                        break; }}}}

**OUTPUT:**

****

**RESULT:**

The above given application is successfully implemented using Java Beans.

**mINI PROJECT – medical image sharing using blockchain**

**aim:**

To create a secure application to share the electronic health between doctors using Blockchain.

**PROBLEM DESCRIPTION:**

Electronic Health Records are generated by hospitals every day. These records are now shared as hard copies between doctors. To avoid this situation, electronic health records can be shared using the Blockchain application which will make the records immutable and in turns makes the system secure.

The idea of this project is to develop a decentralized application which enables sharing of Electronic Health Records between multiple doctors and track the flow of Electronic Health Records using Blockchain to improve privacy and security.

**SCOPE:**

* + Hospitals where there are no high technological medical devices
  + Consultancy with senior doctors who are abroad
  + A distributed community among doctors.

**TECHNIQUES USED:**

* + Blockchain – Core technology
  + React-Js – for Front-end
  + Meta mask – to import Blockchain account
  + Ganache – for local Blockchain server
  + Truffle – for local deployment of smart contracts

**CODE:**

pragma solidity ^0.5.0;

pragma experimental ABIEncoderV2;

contract HealthCare {

    struct Doctor{

        string name;

        uint age;

        string field;

        uint myPatientsCount;

        mapping(uint=>Patient) myPatients;

    }

    struct Patient {

        address myDocAddress;

        string name;

        uint age;

        string disease;

        string description;

        string ehrhash;

    }

    uint public doctorsCount = 0;

    mapping(address=>Doctor) public doctors;

    mapping(address=>bool) public added;

    mapping(uint=>address) public index;

    function addDoctors(string memory name, uint age, string memory field) public {

        if(added[msg.sender]!=true) {

            doctors[msg.sender] = Doctor(name,age,field,0);

            added[msg.sender] = true;

            index[++doctorsCount] = msg.sender;

        }

    }

   function getDoctorsCount() public view returns (uint) {

        return doctorsCount;

    }

    function getDoctors(uint pos) public view returns (string memory) {

        return doctors[index[pos]].name;

    }

    function getMyPatientsCount() public view returns (uint) {

        return doctors[msg.sender].myPatientsCount;

    }

    function getMyPatients(uint pos) public view returns (Patient memory) {

        return doctors[msg.sender].myPatients[pos];

    }

    function addPatients(string memory name, uint age, string memory disease,string memory des,string memory Hash) public {

        doctors[msg.sender].myPatients[++doctors[msg.sender].myPatientsCount] = Patient(msg.sender,name,age,disease,des,Hash);

    }

    function sendDetails(string memory name, uint age, string memory disease, string memory des, string memory Hash, uint pos) public {

        address docAddress = index[pos];

        if(added[docAddress]==true) {

            doctors[docAddress].myPatients[++doctors[docAddress].myPatientsCount] = Patient(docAddress, name, age, disease, des, Hash);

        }

    }

    function check() public view returns (bool) {

        if(added[msg.sender]==true){

            return true;

        }

        return false;

    }

}

**CONNECTION WITH BLOCKCHAIN:**

  async loadBlockchainData() {

    const web3 = window.web3;

    const accounts = await web3.eth.getAccounts()

    this.setState({

      account: accounts[0]

    })

    const networkId = await web3.eth.net.getId()

    const networkData = HealthCare.networks[networkId]

    if(networkData){

      const abi = HealthCare.abi

      const address = networkData.address

      const contract = new web3.eth.Contract(abi, address)

      this.setState({

        contract: contract

      })

      console.log(this.state.contract);

    }else{

      window.alert("Smart contract not deployed to detected network")

    }

    console.log(networkId)

**ROUTES FILE:**

import React from 'react';

import  {BrowserRouter, Switch, Route} from 'react-router-dom';

import Home from './core/Home'

import Dashboard from './user/UserDashboard';

import AdminRoute from './auth/AdminRoute'

import PatientDashboard from './user/AdminDashboard';

import Signin from './auth/index'

import CreatePatient from './Patient'

const Routes = ()=>{

    return (

        <BrowserRouter>

        <Switch>

        <Route path="/" exact component={Home}/>

        <Route path="/signin" exact component={Signin}/>

        <Route path="/user/dashboardhere" exact component={Dashboard} />

            <AdminRoute path="/user/dashboard" exact component={AdminRoute} />

            <Route path="/patient/dashboard" exact component={PatientDashboard}/>

            <Route path="/create/patient" exact component={CreatePatient}/>

        </Switch>

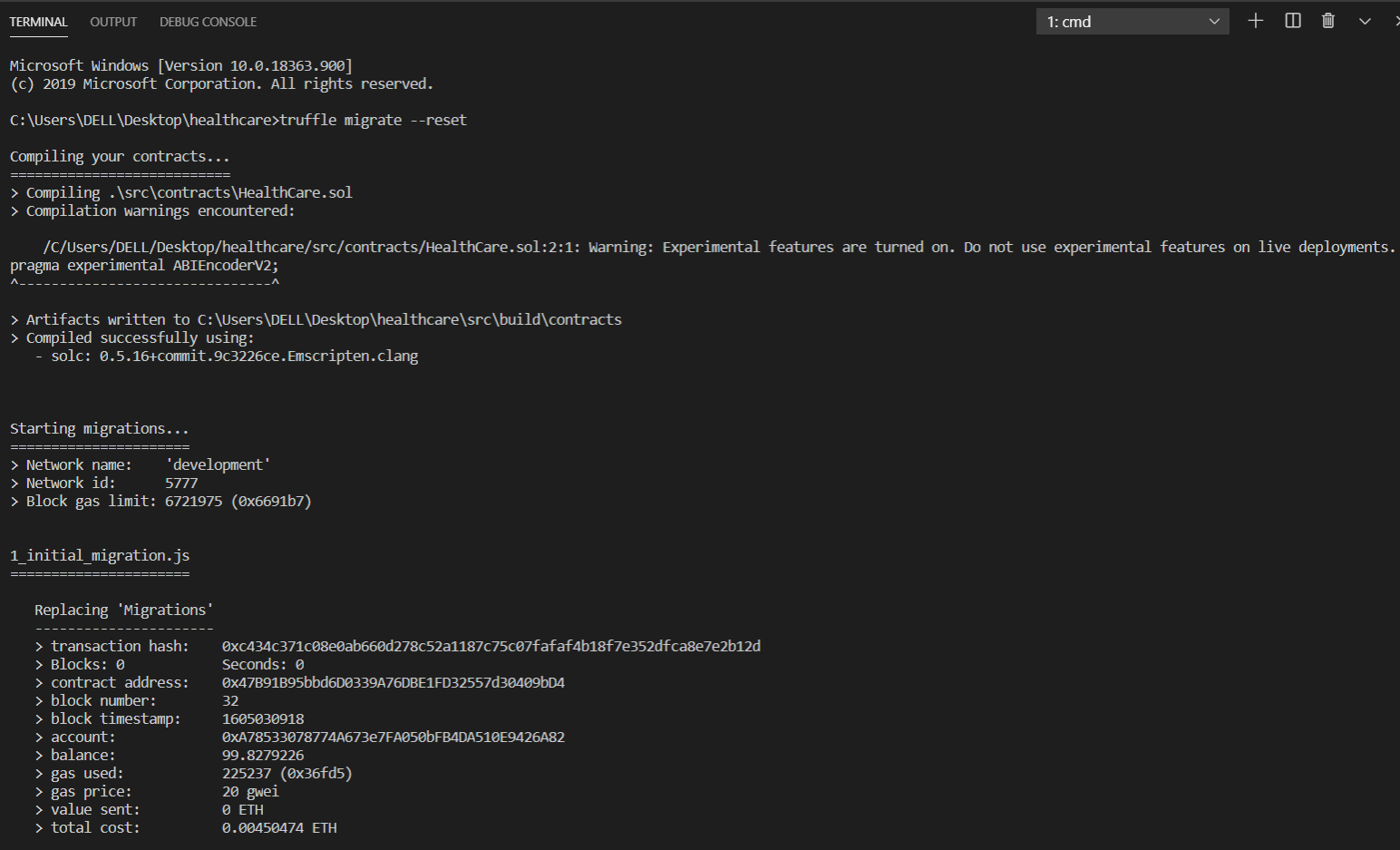
        </BrowserRouter>

        );

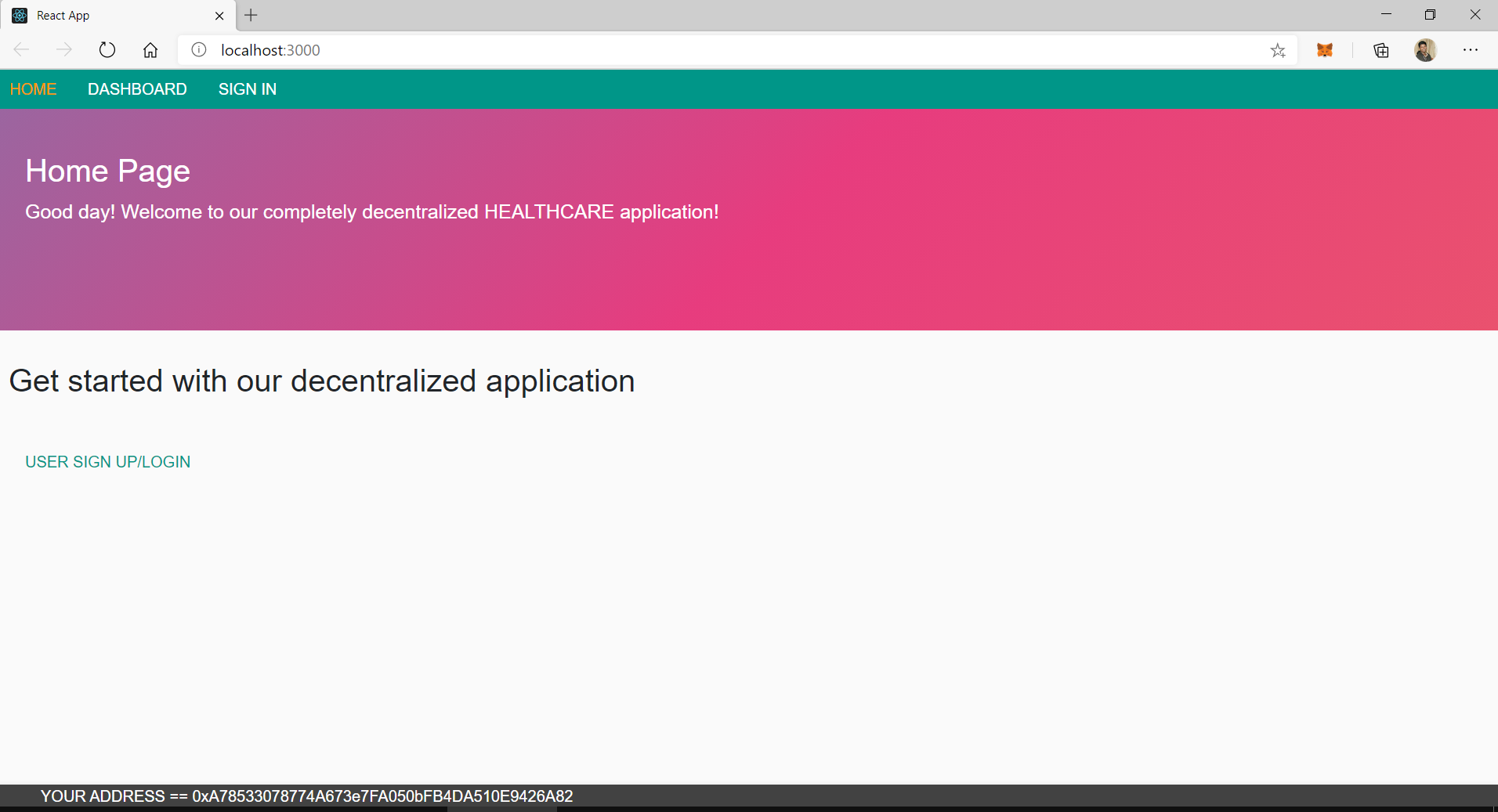
};

export default Routes;

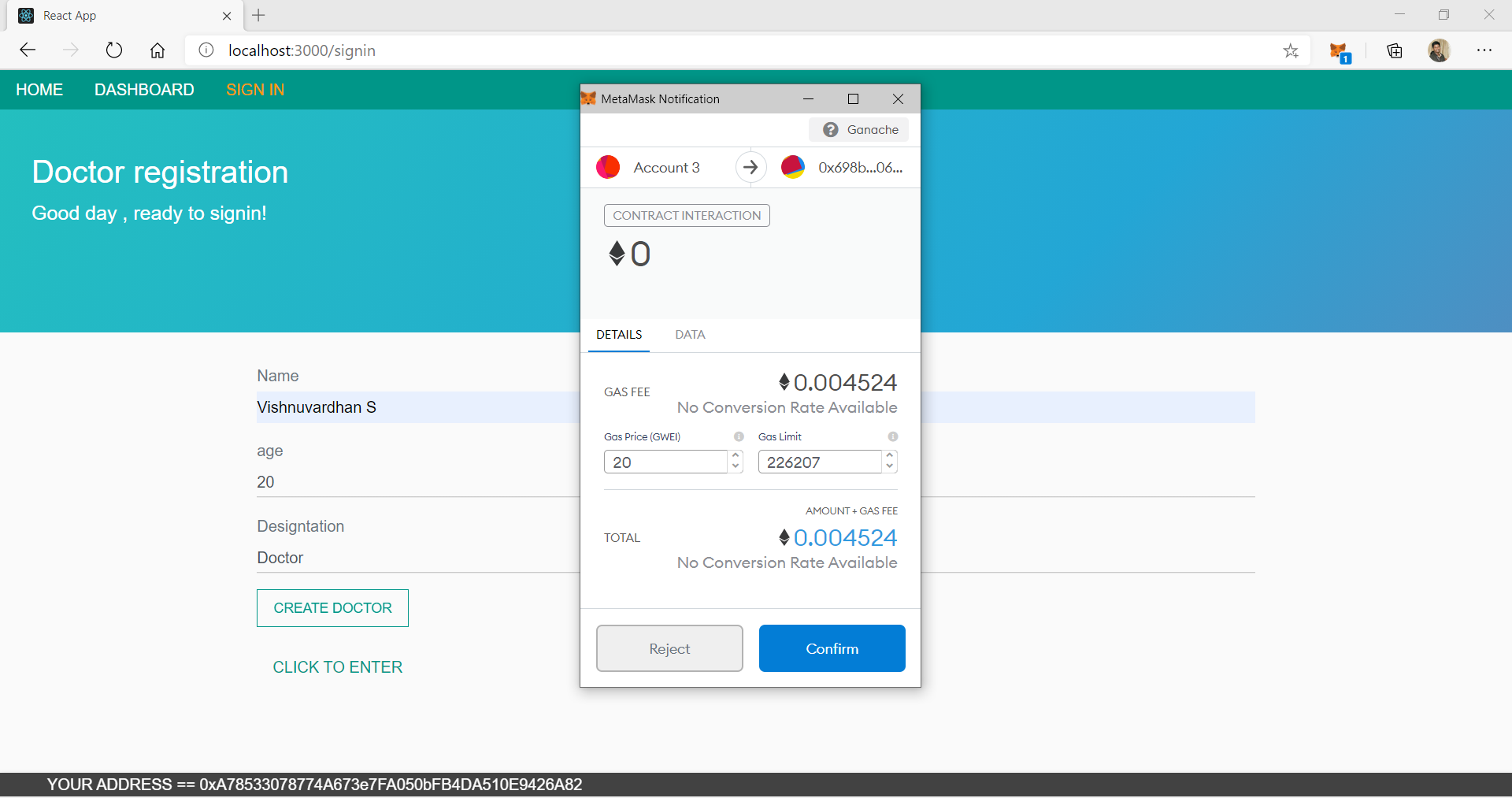
**IMPLEMENTATION SCREENSHOTS:**

****

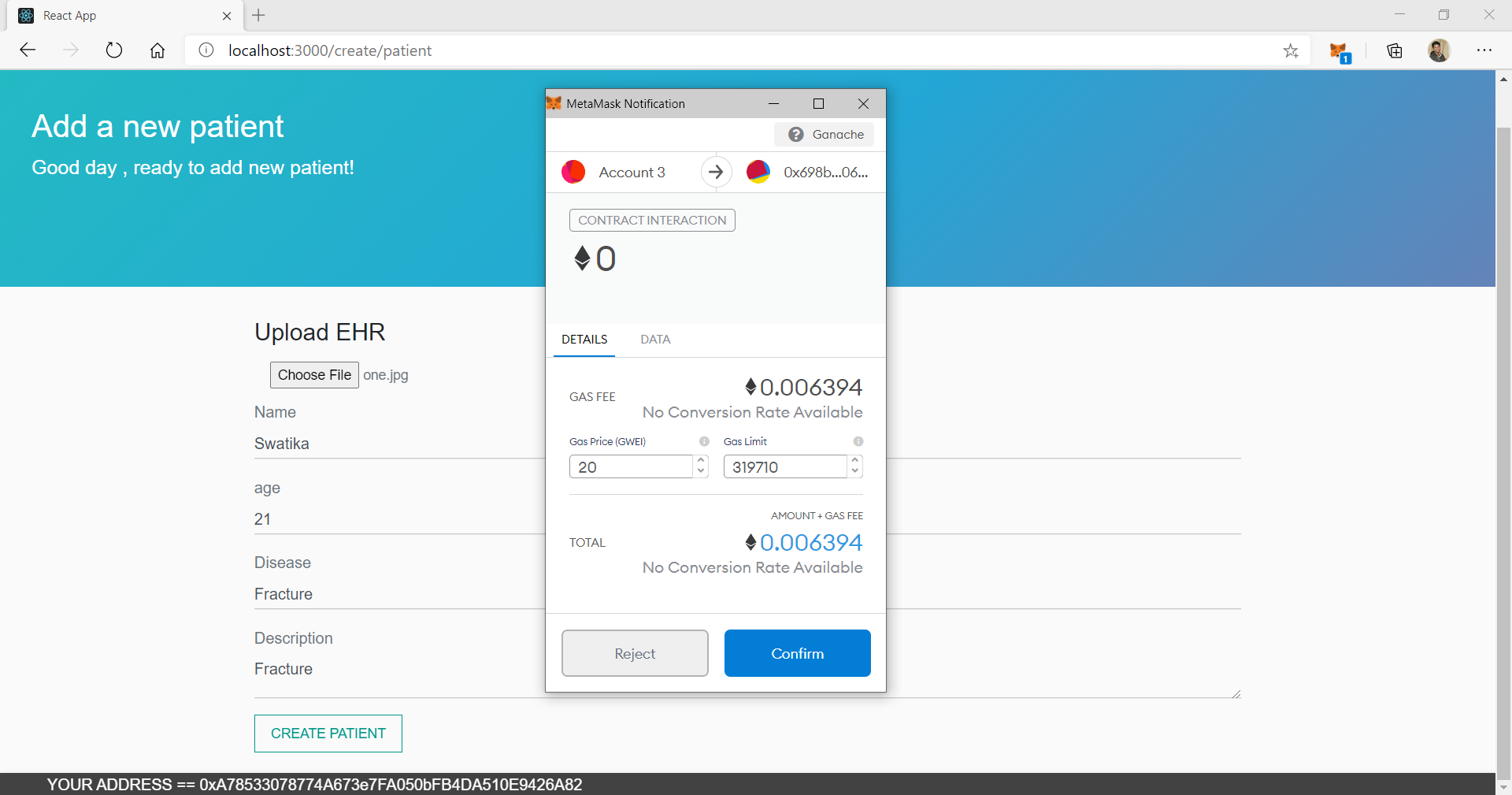
*Figure 1 Deploying contract*



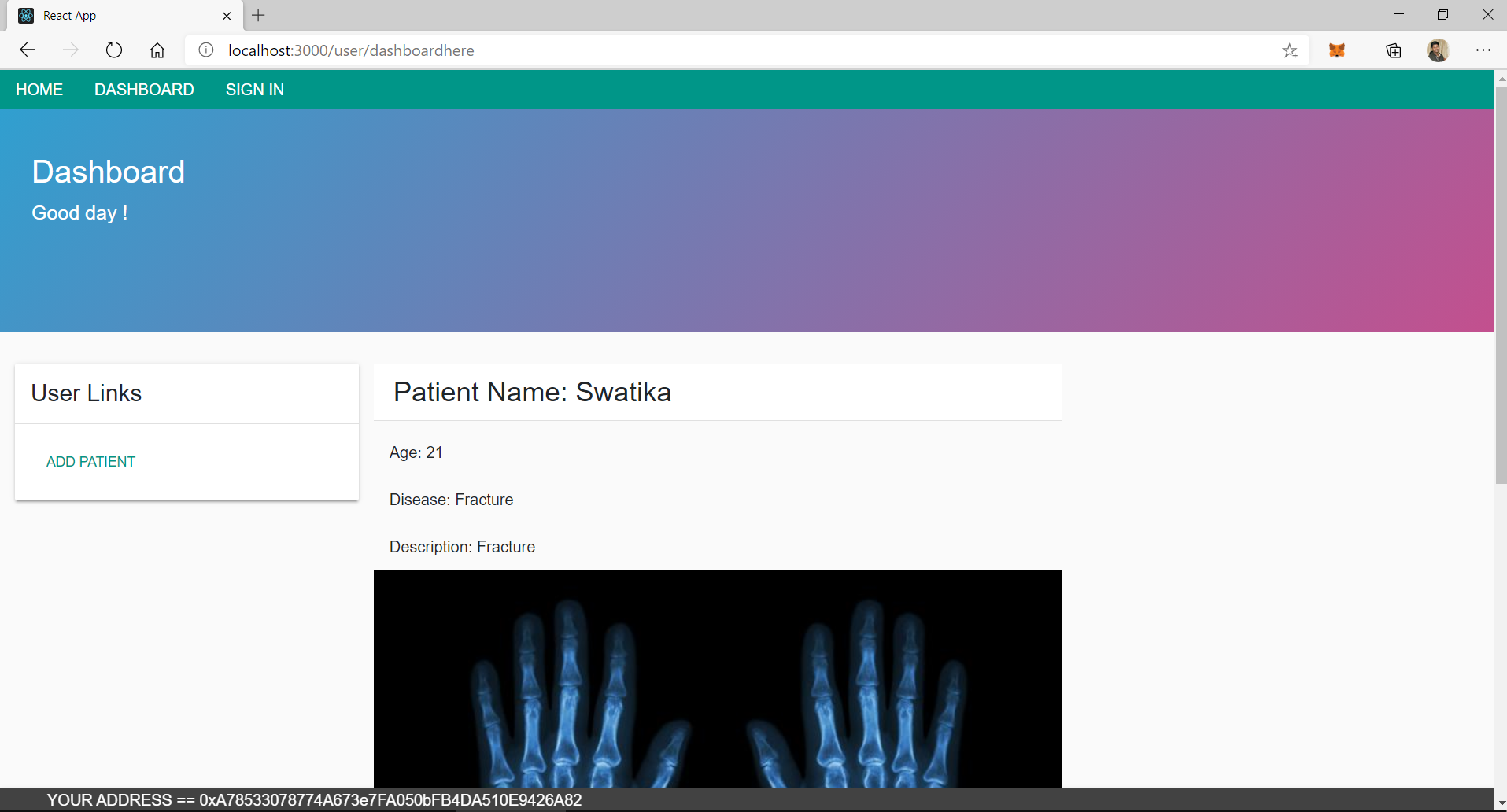
*Figure 2 Home page*



*Figure 3 Adding a doctor*



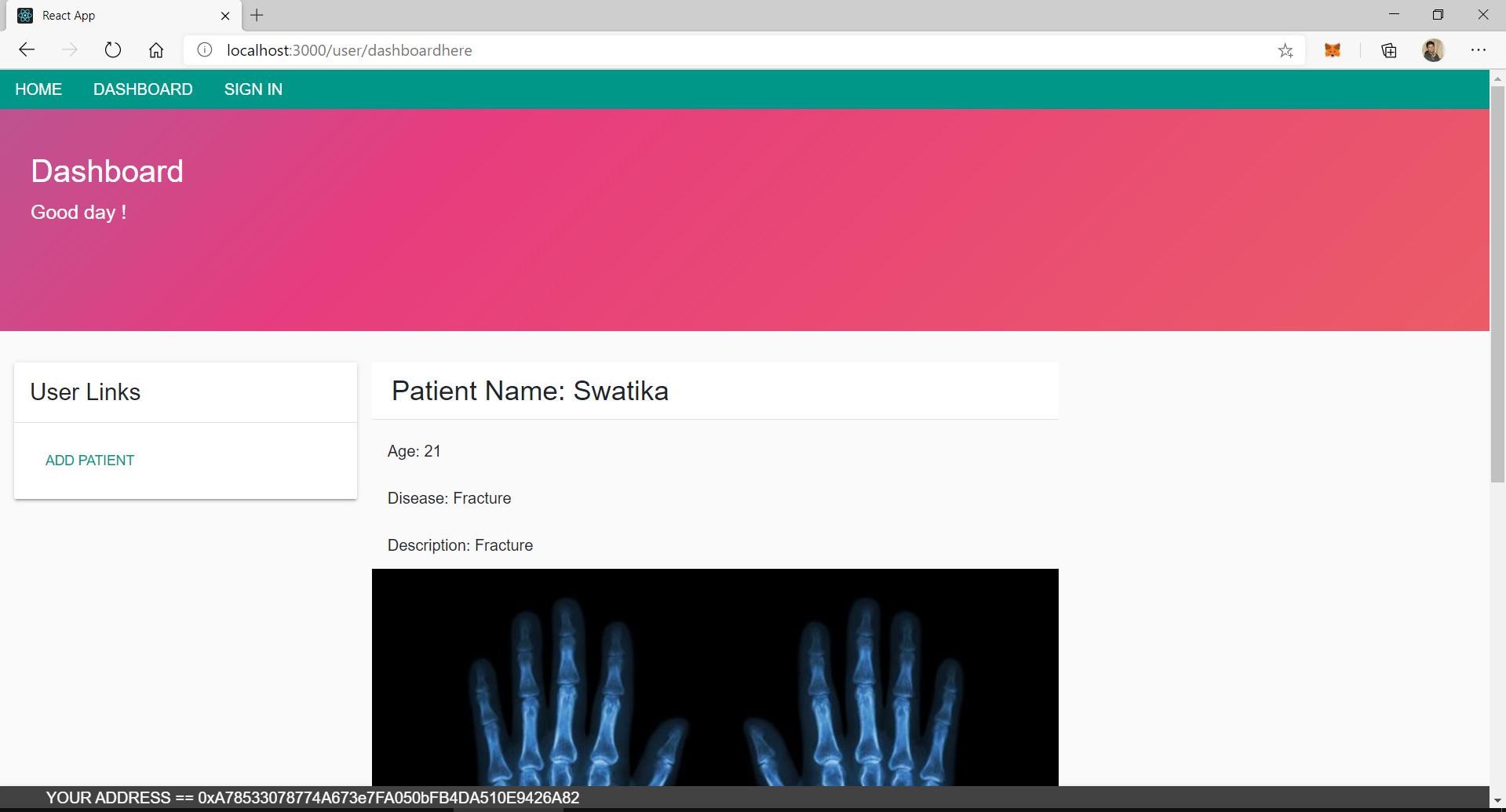
*Figure 4 Inserting a patient*



*Figure 5 Dashboard of the doctor*



*Figure 6 Sending EHR to another doctor*



*Figure 7 The EHR is displayed in the other doctor's account*

**RESULT:**

The application on EHR sharing is implemented successfully using Blockchain.