**Name: Isha Moykhede**

**Roll No: 50**

**DS Assignment No: 1**

**Title: Implement multi-threaded client/server Process communication using RMI**.

**Codes:**

**File name: Adder.java**

import java.rmi.\*;

public interface Adder extends Remote

{

public int add(int x,int y)throws RemoteException;

}

**File name: AdderRemote.java**

import java.rmi.\*;

import java.rmi.server.\*;

public class AdderRemote extends UnicastRemoteObject implements Adder

{

AdderRemote()throws RemoteException

{

super();

}

public int add(int x,int y){return x+y;}

}

**File name: MyServer.java**

import java.rmi.\*;

import java.rmi.registry.\*;

public class MyServer

{

public static void main(String args[])

{

Try

{

Adder stub=new AdderRemote();

Naming.rebind("rmi://localhost:5000/sonoo",stub);

}

catch(Exception e)

{System.out.println(e);}

}

}

**File name: MyClient.java**

import java.rmi.\*;

public class MyClient{

public static void main(String args[]){

try{

Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");

System.out.println(“Addition is :”);

System.out.println(stub.add(34,4));

}

catch(Exception e)

{System.out.println(e);}

}

}

**OUTPUT:**

**Command prompt 1**

Microsoft Windows [Version 10.0.19044.1889]

(c) Microsoft Corporation. All rights reserved.

C:\Users\isham>cd C:\Users\isham\Downloads\rmi1\1

C:\Users\isham\Downloads\rmi1\1>javac \*.java

C:\Users\isham\Downloads\rmi1\1>rmic AdderRemote

Warning: generation and use of skeletons and static stubs for JRMP  
is deprecated. Skeletons are unnecessary, and static stubs have  
been superseded by dynamically generated stubs. Users are  
encouraged to migrate away from using rmic to generate skeletons and static  
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\Users\isham\Downloads\rmi1\1>rmiregistry 5000

**Command prompt 2**

Microsoft Windows [Version 10.0.19044.1889]

(c) Microsoft Corporation. All rights reserved.

C:\Users\isham>cd C:\Users\isham\Downloads\rmi1\1

C:\Users\isham\Downloads\rmi1\1>java MyServer

**Command prompt 3**

Microsoft Windows [Version 10.0.19044.1889]

(c) Microsoft Corporation. All rights reserved.

C:\Users\isham>cd C:\Users\isham\Downloads\rmi1\1

C:\Users\isham\Downloads\rmi1\1>Java MyClient

38

C:\Users\isham\Downloads\rmi1\1>

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

CODES EXPLAINAITION:

Sure! I'll provide a line-by-line explanation of the code you provided, explaining each part in simple language.

File name: Adder.java

1. `import java.rmi.\*;` - This line imports the necessary RMI (Remote Method Invocation) classes.

3. `public interface Adder extends Remote {` - This line declares an interface named `Adder` that extends the `Remote` interface. It defines a remote method `add()` that can be invoked by remote clients.

1. `public int add(int x, int y) throws RemoteException;` - This line defines the `add()` method signature, which takes two integer parameters `x` and `y` and returns an integer sum. It also throws a `RemoteException` in case of remote communication errors.

File name: AdderRemote.java

1. `import java.rmi.\*;` - This line imports the necessary RMI classes.

3. `import java.rmi.server.\*;` - This line imports the necessary RMI server classes.

5. `public class AdderRemote extends UnicastRemoteObject implements Adder {` - This line defines a class named `AdderRemote` that extends `UnicastRemoteObject` and implements the `Adder` interface. It serves as the implementation of the remote interface.

7. `AdderRemote() throws RemoteException {` - This line defines the constructor for the `AdderRemote` class, which throws a `RemoteException`. It ensures that the superclass constructor is called.

10. `public int add(int x, int y) { return x + y; }` - This line implements the `add()` method defined in the `Adder` interface. It simply adds the two integer parameters `x` and `y` and returns the sum.

File name: MyServer.java

1. `import java.rmi.\*;` - This line imports the necessary RMI classes.

3. `import java.rmi.registry.\*;` - This line imports the necessary RMI registry classes.

5. `public class MyServer {` - This line defines a class named `MyServer`.

7. `public static void main(String args[]) {` - This line defines the `main()` method of the `MyServer` class.

9. `Adder stub = new AdderRemote();` - This line creates an instance of the `AdderRemote` class and assigns it to the `Adder` interface reference `stub`.

10. `Naming.rebind("rmi://localhost:5000/sonoo", stub);` - This line binds the remote object `stub` to the RMI registry with the specified name ("rmi://localhost:5000/sonoo").

File name: MyClient.java

1. `import java.rmi.\*;` - This line imports the necessary RMI classes.

3. `public class MyClient {` - This line defines a class named `MyClient`.

5. `public static void main(String args[]) {` - This line defines the `main()` method of the `MyClient` class.

7. `Adder stub = (Adder)Naming.lookup("rmi://localhost:5000/sonoo");` - This line performs a lookup in the RMI registry for the remote object with the specified name ("rmi://localhost:5000/sonoo") and assigns it to the `Adder` interface reference `stub`.

8. `System.out.println(stub.add(34, 4));` - This line invokes the `add()` method on the remote object `stub` with the arguments 34 and 4, and prints the result.

I hope this explanation helps you understand the code better! Let me know if you have any further questions.

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

**Name: Isha Moykhede**

**Roll No: 50**

**DS Assignment No: 3**

**Title: Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors.**.

**Codes:**

**Hello world program**

nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ gedit hello.c

#include <stdio.h>

#include "mpi.h"

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

{

int rank, size, len;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

printf("Hello, world, I am %d of %d\n",rank, size);

MPI\_Finalize();

return 0;

}

**Program to transfer data from core 0 to core 1**.

nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ gedit assi03.c

#include <stdio.h>

#include "mpi.h"

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

{

int rank, size, len;

int num=10;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if(rank == 0)

{

printf("Sending message containing: %d from rank %d\n", num,rank);

MPI\_Send(&num, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

}

else

{

printf(" at rank %d\n",rank);

MPI\_Recv(&num, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Received message containing: %d at rank %d\n", num,rank);

}

MPI\_Finalize();

return 0;

}

**Assignment program: Add 20 numbers in an array using 4 cores**

nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ gedit addition.c

#include <stdio.h>

#include "mpi.h"

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

{

int rank, size;

int num[20]; //N=20, n=4

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

for(int i=0;i<20;i++)

num[i]=i+1;

if(rank == 0){

int s[4];

printf("Distribution at rank %d \n", rank);

for(int i=1;i<4;i++)

MPI\_Send(&num[i\*5], 5, MPI\_INT, i, 1, MPI\_COMM\_WORLD); //N/n i.e. 20/4=5

int sum=0, local\_sum=0;

for(int i=0;i<5;i++)

{

local\_sum=local\_sum+num[i];

}

for(int i=1;i<4;i++)

{

MPI\_Recv(&s[i], 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

printf("local sum at rank %d is %d\n", rank,local\_sum);

sum=local\_sum;

for(int i=1;i<4;i++)

sum=sum+s[i];

printf("final sum = %d\n\n",sum);

}

else

{

int k[5];

MPI\_Recv(k, 5, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

int local\_sum=0;

for(int i=0;i<5;i++)

{

local\_sum=local\_sum+k[i];

}

printf("local sum at rank %d is %d\n", rank, local\_sum);

MPI\_Send(&local\_sum, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

**Output:**

student@student-HP-Pro-3330-MT:~$ cd opt  
student@student-HP-Pro-3330-MT:~/opt$ cd openmpi  
student@student-HP-Pro-3330-MT:~/opt/openmpi$ cd bin  
**student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ mpicc hello.c  
student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ mpirun -np N ./a.out**Hello, world, I am 0 of 1  
**student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ mpicc assi03.c  
student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ mpirun -np 4 ./a.out**at rank 3Sending message containing: 10 from rank 0  
 at rank 1  
 at rank 2  
Received message containing: 10 at rank 1  
^Z  
[1]+  Stopped                 mpirun -np 4 ./a.out  
**student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ mpicc addition.c  
student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ mpirun -np 4 ./a.out**  
Distribution at rank 0  
local sum at rank 1 is 40  
local sum at rank 2 is 65  
local sum at rank 3 is 90  
local sum at rank 0 is 15  
final sum = 210  
  
student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$ ^C  
student@student-HP-Pro-3330-MT:~/opt/openmpi/bin$

CODES EXPLAINATION:

Sure! I'll provide a line-by-line explanation of the code you provided, explaining each part in simple language.

Code: hello.c

1. `#include <stdio.h>` - This line includes the standard input/output library.

2. `#include "mpi.h"` - This line includes the MPI (Message Passing Interface) library.

4. `int main(int argc, char\* argv[])` - This line declares the main function, which is the entry point of the program.

6. `int rank, size, len;` - This line declares variables `rank`, `size`, and `len` to store the rank of the process, total number of processes, and length of the message.

8. `MPI\_Init(&argc, &argv);` - This line initializes the MPI environment.

9. `MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);` - This line retrieves the rank of the current process.

10. `MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);` - This line retrieves the total number of processes.

12. `printf("Hello, world, I am %d of %d\n", rank, size);` - This line prints the "Hello, world, I am [rank] of [size]" message, where `[rank]` is the rank of the process and `[size]` is the total number of processes.

14. `MPI\_Finalize();` - This line finalizes the MPI environment.

16. `return 0;` - This line returns 0 to indicate successful program execution.

Code: assi03.c

1. `#include <stdio.h>` - This line includes the standard input/output library.

2. `#include "mpi.h"` - This line includes the MPI library.

5. `int main(int argc, char\* argv[])` - This line declares the main function.

7. `int rank, size, len;` - This line declares variables `rank`, `size`, and `len`.

9. `int num = 10;` - This line initializes the variable `num` with the value 10.

11. `MPI\_Init(&argc, &argv);` - This line initializes the MPI environment.

12. `MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);` - This line retrieves the rank of the current process.

13. `MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);` - This line retrieves the total number of processes.

15. `if (rank == 0)` - This line checks if the rank of the process is 0.

17. `printf("Sending message containing: %d from rank %d\n", num, rank);` - This line prints the message "Sending message containing: [num] from rank [rank]" where `[num]` is the value of the `num` variable and `[rank]` is the rank of the process.

18. `MPI\_Send(&num, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);` - This line sends the value of `num` as a message to the process with rank 1 using MPI\_Send function.

21. `else` - This line executes if the rank of the process is not 0.

23. `printf(" at rank %d\n", rank);` - This line prints the message "at rank [rank]" where `[rank]` is the rank of the process.

24. `MPI\_Recv(&num, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);` - This line receives a message of type MPI\_INT from the process with rank 0 using MPI\_Recv function and stores it in the `num` variable.

26. `printf("

Received message containing: %d at rank %d\n", num, rank);` - This line prints the message "Received message containing: [num] at rank [rank]" where `[num]` is the value of the `num` variable and `[rank]` is the rank of the process.

29. `MPI\_Finalize();` - This line finalizes the MPI environment.

31. `return 0;` - This line returns 0 to indicate successful program execution.

Code: addition.c

1. `#include <stdio.h>` - This line includes the standard input/output library.

2. `#include "mpi.h"` - This line includes the MPI library.

5. `int main(int argc, char\* argv[])` - This line declares the main function.

7. `int rank, size;` - This line declares variables `rank` and `size`.

8. `int num[20]; // N=20, n=4` - This line declares an array `num` with a size of 20 to store the numbers.

11. `MPI\_Init(&argc, &argv);` - This line initializes the MPI environment.

12. `MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);` - This line retrieves the rank of the current process.

13. `MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);` - This line retrieves the total number of processes.

14. `for (int i = 0; i < 20; i++) num[i] = i + 1;` - This line initializes the `num` array with values from 1 to 20.

17. `if (rank == 0)` - This line checks if the rank of the process is 0.

19. `int s[4];` - This line declares an array `s` to store the sums calculated by each process.

21. `printf("Distribution at rank %d\n", rank);` - This line prints the message "Distribution at rank [rank]" where `[rank]` is the rank of the process.

22. `for (int i = 1; i < 4; i++) MPI\_Send(&num[i \* 5], 5, MPI\_INT, i, 1, MPI\_COMM\_WORLD);` - This line sends a subset of the `num` array (5 elements) to each process (rank 1 to 3) using MPI\_Send function.

24. `int sum = 0, local\_sum = 0;` - This line declares variables `sum` and `local\_sum` to store the overall sum and local sum.

25. `for (int i = 0; i < 5; i++) local\_sum = local\_sum + num[i];` - This line calculates the local sum by summing the first 5 elements of the `num` array.

28. `for (int i = 1; i < 4; i++) MPI\_Recv(&s[i], 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);` - This line receives the local sums from each process (rank 1 to 3) and stores them in the `s` array using MPI\_Recv function.

31. `printf("local sum at rank %d is %d\n", rank, local\_sum);` - This line prints the message "local sum at rank [rank] is [local\_sum]" where `[rank]` is the rank of the process and `[local\_sum]` is the local sum calculated by the process.

32. `sum = local\_sum;` - This line assigns the value of `local\_sum` to `sum`.

33. `for (int i = 1;

i < 4; i++) sum = sum + s[i];` - This line calculates the final sum by adding the local sums calculated by each process.

35. `printf("final sum = %d\n\n", sum);` - This line prints the final sum calculated by adding the local sums of all processes.

38. `else` - This line executes if the rank of the process is not 0.

40. `int k[5];` - This line declares an array `k` to receive the subset of the `num` array sent by process 0.

42. `MPI\_Recv(k, 5, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);` - This line receives the subset of the `num` array sent by process 0 and stores it in the `k` array using MPI\_Recv function.

44. `int local\_sum = 0;` - This line declares the variable `local\_sum` to store the local sum.

45. `for (int i = 0; i < 5; i++) local\_sum = local\_sum + k[i];` - This line calculates the local sum by summing the 5 elements received from process 0.

47. `printf("local sum at rank %d is %d\n", rank, local\_sum);` - This line prints the message "local sum at rank [rank] is [local\_sum]" where `[rank]` is the rank of the process and `[local\_sum]` is the local sum calculated by the process.

48. `MPI\_Send(&local\_sum, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);` - This line sends the local sum to process 0 using MPI\_Send function.

51. `MPI\_Finalize();` - This line finalizes the MPI environment.

53. `return 0;` - This line returns 0 to indicate successful program execution.

Please note that understanding and executing MPI programs require a working knowledge of parallel programming and the MPI library.

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

**Name: Isha Moykhede**

**Roll No: 50**

**DS Assignment No: 4**

**Title: Implement Berkeley algorithm for clock synchronization.**

**Codes:**

**File name: Server.cpp**

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#include <arpa/inet.h>

#include <vector>

#include <cstdlib>

#include <ctime>

#define PORT 8080

using namespace std;

// function for string delimiter

vector<string> split(string s, string delimiter) {

    size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

    string token;

    vector<string> res;

    while ((pos\_end = s.find (delimiter, pos\_start)) != string::npos) {

        token = s.substr (pos\_start, pos\_end - pos\_start);

        pos\_start = pos\_end + delim\_len;

        res.push\_back (token);

    }

    res.push\_back (s.substr (pos\_start));

    return res;

}

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

{

    // /\* deal with input arguments\*/

    // std::cout << "print arguments:\nargc == " << argc << '\n';

    // for(int ndx{}; ndx != argc; ++ndx) {

    //     std::cout << "argv[" << ndx << "] == " << argv[ndx] << '\n';

    // }

    // std::cout << "argv[" << argc << "] == "

    //           << static\_cast<void\*>(argv[argc]) << '\n';

    srand((unsigned int)time(NULL)); // avoid always same output of rand()

    float server\_local\_clock = rand() % 10; // range from 0 to 9

    vector<float> clients\_local\_clocks;

    printf("Sever starts. Server pid is %d \n", getpid());

    printf("Server local clock is %f \n\n", server\_local\_clock);

    // Socket Cite: https://www.geeksforgeeks.org/socket-programming-cc/?ref=lbp

    int server\_socket\_fd, new\_socket, valread;

    vector<int> client\_sockets;

    vector<string> client\_ips;

    vector<int> client\_ports;

    struct sockaddr\_in server\_address;

    server\_address.sin\_family = AF\_INET;    // IPv4

    server\_address.sin\_addr.s\_addr = INADDR\_ANY; // localhost

    server\_address.sin\_port = htons( PORT ); // 8080

    int opt = 1; // for setsockopt

    // Creating socket file descriptor (IPv4, TCP, IP)

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

    {

        perror("Server: socket failed");

        exit(EXIT\_FAILURE);

    }

    // Optional: it helps in reuse of address and port. Prevents error such as: “address already in use”.

    if (setsockopt(server\_socket\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

                                                  &opt, sizeof(opt)))

    {

        perror("Server: setsockopt");

        exit(EXIT\_FAILURE);

    }

    // Forcefully attaching socket to the port 8080

    if (bind(server\_socket\_fd, (struct sockaddr \*)&server\_address,

                                 sizeof(server\_address))<0)

    {

        perror("Server: bind failed");

        exit(EXIT\_FAILURE);

    }

    // Putting the server socket in a passive mode, waiting for the client to approach the server to make a connection

    // The backlog=7, defines the maximum length to which the queue of pending connections for sockfd may grow.

    // If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

    if (listen(server\_socket\_fd, 7) < 0)

    {

        perror("Server: listen");

        exit(EXIT\_FAILURE);

    }

    printf("Server: server is listening ...\n\nYou can open one or multiple new terminal windows now to run ./client\n");

    int clients\_ctr = 0;

    // Setting up buffer for receiving msg

    char recv\_buf[65536];

    memset(recv\_buf, '\0', sizeof(recv\_buf));

    int in\_client\_enough = 0;

    while ( in\_client\_enough == 0) { // block on accept() until positive fd or error

        struct sockaddr\_in client\_addr;

        socklen\_t length = sizeof(client\_addr);

        // Extracting the first connection request on the queue of pending connections for the listening socket (server\_socket\_fd)

        // Creates a new connected socket, and returns a new file descriptor referring to that socket

        if ((new\_socket = accept(server\_socket\_fd, (struct sockaddr \*)&client\_addr,

                        (socklen\_t\*)&length))<0)

        {

            perror("Server: accept");

            exit(EXIT\_FAILURE);

        }

        clients\_ctr ++;

        printf("\nYou have connected %d client(s) now.", clients\_ctr);

        // converting the network address structure src in the af address family into a character string.

        char client\_ip[INET\_ADDRSTRLEN] = "";

        inet\_ntop(AF\_INET, &client\_addr.sin\_addr, client\_ip, INET\_ADDRSTRLEN);

        printf("Server: new client accepted. client ip and port: %s:%d\n", client\_ip, ntohs(client\_addr.sin\_port));

        // store new client connection into array

        client\_sockets.push\_back(new\_socket);

        client\_ips.push\_back(client\_ip);

        client\_ports.push\_back(ntohs(client\_addr.sin\_port));

        printf("current connected clients amount is %d \n", int(client\_sockets.size()) );

        cout << "Do you have enought clients? (please input '1' for yes, '0' for no):" ;

        cin >> in\_client\_enough;

        if (in\_client\_enough == 0){

            cout << "OK. Please continute opening one or multiple new terminal windows to run ./client\n" << endl;

        }else if (in\_client\_enough != 1){

            cout << "Unrecognized input has been considered as 0. You can create one more client.\n" << endl;

            in\_client\_enough = 0;

        }

    }

    printf("\nClients creation finished! There are totally %d connected clients.\n", int(client\_sockets.size()) );

    printf("Asking all clients to report their local clock value ... \n\n\n");

    for (int i = 0; i < client\_sockets.size(); i++){

        // sending a message to client

        const char \*msg = "Hello from server, please tell me your local clock value.";

        send(client\_sockets[i] , msg , strlen(msg) , 0 );

        printf("Server: sent to client(%s:%d): '%s'\n", client\_ips[i].c\_str(), client\_ports[i], msg);

        // receiving

        while(recv(client\_sockets[i], recv\_buf, sizeof(recv\_buf), 0) > 0 ){

            printf("Server: recv from client(%s:%d): '%s' \n", client\_ips[i].c\_str(), client\_ports[i], recv\_buf);

            // convert char array to string

            string recv\_msg = string(recv\_buf);

            if (recv\_msg.find("Hello from client, my local clock value is") != string::npos){

                string substr\_after\_last\_space;

                vector<string> split\_str = split(recv\_msg, " ");

                substr\_after\_last\_space = split\_str[ split\_str.size() - 1 ];

                cout << "Server: received client local clock (string) is " << substr\_after\_last\_space << endl;

                float substr\_after\_last\_space\_f = stof(substr\_after\_last\_space);

                cout << "Server: received client local clock (float) is " << substr\_after\_last\_space\_f << endl;

                clients\_local\_clocks.push\_back(substr\_after\_last\_space\_f);

            }

            memset(recv\_buf, '\0', strlen(recv\_buf));

            break;

        }

    }

    printf("\n\n");

    // average clock values

    float all\_clock\_sum = server\_local\_clock;

    for (int i = 0; i < clients\_local\_clocks.size(); i++){

        all\_clock\_sum += clients\_local\_clocks[i];

    }

    float avg\_clock = all\_clock\_sum / (client\_sockets.size() + 1);

    // tell clients how to adjust

    for (int i = 0; i < client\_sockets.size(); i++){

        // prepare msg

        float offset = clients\_local\_clocks[i] - avg\_clock;

        string operation;

        if (offset >= 0){

            operation = "minus";

        }else{

            operation = "add";

            offset = 0 - offset;

        }

        string msg\_str = "From server, your clock adjustment offset is " + operation + " " + to\_string(offset);

        char msg\_char\_array[msg\_str.length() + 1];

        strcpy(msg\_char\_array, msg\_str.c\_str());

        // sending a message to client

        send(client\_sockets[i] , &msg\_char\_array , strlen(msg\_char\_array) , 0 );

        printf("Server: sent to client(%s:%d): '%s'\n", client\_ips[i].c\_str(), client\_ports[i], msg\_char\_array);

    }

    // adjust self

    server\_local\_clock += avg\_clock - server\_local\_clock;

    printf("\n\nServer new local clock is %f \n\n", server\_local\_clock);

    printf("Server: server stopped. \n");

    close(server\_socket\_fd);

    return 0;

}

**File name: Client.cpp**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <iostream>

#include <stdlib.h>     /\* srand, rand \*/

#include <cstdlib>

#include <ctime>

#include <vector>

#define PORT 8080

using namespace std;

// function for string delimiter

vector<string> split(string s, string delimiter) {

    size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

    string token;

    vector<string> res;

    while ((pos\_end = s.find (delimiter, pos\_start)) != string::npos) {

        token = s.substr (pos\_start, pos\_end - pos\_start);

        pos\_start = pos\_end + delim\_len;

        res.push\_back (token);

    }

    res.push\_back (s.substr (pos\_start));

    return res;

}

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

{

    srand((unsigned int)time(NULL)); // avoid always same output of rand()

    float client\_local\_clock = rand() % 10; // range from 0 to 9

    printf("Client starts. Client pid is %d \n", getpid());

    printf("Client local clock is %f \n\n", client\_local\_clock);

    int client\_socket\_fd, valread;

    char client\_read\_buffer[1024] = {0};

    struct sockaddr\_in server\_addr;

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

    // server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]); // hardcode to 127.0.0.1

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

    // Creating socket file descriptor (IPv4, TCP, IP)

    if ((client\_socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

    {

        printf("\n Client: Socket creation error \n");

        return -1;

    }

    // Converting IPv4 and IPv6 addresses from text to binary form,

    //   from character string src into a network

    //   address structure in the af address family, then copies the

    //   network address structure to dst.

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

    {

        printf("\nClient: Invalid address/ Address not supported \n");

        return -1;

    }

    // Connecting server, return 0 with success, return -1 with error

    if (connect(client\_socket\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

    {

        printf("\nClient: Connection Failed \n");

        return -1;

    }

    char server\_ip[INET\_ADDRSTRLEN]="";

    inet\_ntop(AF\_INET, &server\_addr.sin\_addr, server\_ip, INET\_ADDRSTRLEN);

    printf("Client: connected server(%s:%d). \n", server\_ip, ntohs(server\_addr.sin\_port));

    printf("\n\n");

    //

    // first round communicattion

    //

    // receiving form server

    valread = read( client\_socket\_fd , client\_read\_buffer, 1024);

    printf("Client: read: '%s'\n",client\_read\_buffer );

    // convert char array to string

    string recv\_msg = string(client\_read\_buffer);

    // reply according to what client receive

    if (strcmp(client\_read\_buffer, "Hello from server, please tell me your local clock value.") == 0) {

        // prepare msg

        string msg\_str = "Hello from client, my local clock value is " + to\_string(client\_local\_clock);

        char msg\_char\_array[msg\_str.length() + 1];

        strcpy(msg\_char\_array, msg\_str.c\_str());

        // sending a message to server

        send(client\_socket\_fd , &msg\_char\_array , strlen(msg\_char\_array) , 0 );

        printf("Client: sent message: '%s'\n", msg\_char\_array);

    }

    //

    // second round communicattion

    //

    // receiving form server

    valread = read( client\_socket\_fd , client\_read\_buffer, 1024);

    printf("Client: read: '%s'\n",client\_read\_buffer );

    // convert char array to string

    recv\_msg = string(client\_read\_buffer);

    if (recv\_msg.find("From server, your clock adjustment offset is") != string::npos){ // if latter is a substring of former

        string substr\_after\_lastbutone\_space;

        string substr\_after\_last\_space;

        vector<string> split\_str = split(recv\_msg, " ");

        substr\_after\_lastbutone\_space = split\_str[ split\_str.size() - 2 ];

        substr\_after\_last\_space = split\_str[ split\_str.size() - 1 ];

        cout << "Client: received local clock adjustment offset (string) is " << substr\_after\_lastbutone\_space << " " << substr\_after\_last\_space << endl;

        float substr\_after\_last\_space\_f = stof(substr\_after\_last\_space);

        cout << "Client: received local clock adjustment offset (float) is " << substr\_after\_lastbutone\_space << " " << substr\_after\_last\_space\_f << endl;

        char oper\_char\_array[substr\_after\_lastbutone\_space.length() + 1];

        strcpy(oper\_char\_array, substr\_after\_lastbutone\_space.c\_str());

        if (strcmp(oper\_char\_array, "add") == 0 ){

            client\_local\_clock += substr\_after\_last\_space\_f;

        }else if (strcmp(oper\_char\_array, "minus") == 0 ){

            client\_local\_clock -= substr\_after\_last\_space\_f;

        }

        printf("Client local clock is %f \n\n", client\_local\_clock);

    }

    close(client\_socket\_fd);

    return 0;

}

**Output**

**Terminal Output 1**

student@student-HP-Pro-3330-MT:~$ cd /documents/berk  
student@student-HP-Pro-3330-MT:~$ cd berk  
student@student-HP-Pro-3330-MT:~/berk$ make  
g++ server.cpp -o server -std=c++11  
g++ client.cpp -o client -std=c++11  
  
student@student-HP-Pro-3330-MT:~/berk$  
student@student-HP-Pro-3330-MT:~/berk$ ./server  
Sever starts. Server pid is 6329  
Server local clock is 3.000000  
  
Server: server is listening ...  
  
You can open one or multiple new terminal windows now to run ./client  
  
You have connected 1 client(s) now.Server: new client accepted. client ip and port: [127.0.0.1:54186](http://127.0.0.1:54186/" \l "inbox/_blank" \t "https://mail.google.com/mail/u/0/?tab=rm&ogbl)  
current connected clients amount is 1  
Do you have enought clients? (please input '1' for yes, '0' for no):1  
  
Clients creation finished! There are totally 1 connected clients.  
Asking all clients to report their local clock value …

**Terminal Output 2**

student@student-HP-Pro-3330-MT:~$ ./client  
bash: ./client: No such file or directory  
student@student-HP-Pro-3330-MT:~$ cd berk  
student@student-HP-Pro-3330-MT:~/berk$ ./client  
Client starts. Client pid is 6348  
Client local clock is 2.000000  
  
Client: connected server([127.0.0.1:8080](http://127.0.0.1:8080/" \l "inbox/_blank" \t "https://mail.google.com/mail/u/0/?tab=rm&ogbl)).  
  
  
Client: read: 'Hello from server, please tell me your local clock value.'  
Client: sent message: 'Hello from client, my local clock value is 2.000000'  
Client: read: 'From server, your clock adjustment offset is add 0.500000'  
Client: received local clock adjustment offset (string) is add 0.500000  
Client: received local clock adjustment offset (float) is add 0.5  
Client local clock is 2.500000

CODES EXPLAINATION:

Sure! Here's a line-by-line explanation of the code:

1. The code starts by including necessary libraries and headers.

3. The `split` function is defined, which takes a string `s` and a delimiter string as input and splits the string into a vector of substrings based on the delimiter.

19-26. The `main` function begins here. It initializes variables for the server's local clock and vectors to store clients' local clocks, client sockets, client IP addresses, and client ports. It also prints the server's process ID and the initial server local clock value.

31-43. Socket setup code is implemented here. It creates a socket file descriptor, sets socket options, binds the socket to a specific port, and starts listening for client connections.

47-57. A loop is set up to accept client connections until the desired number of clients is reached. It accepts a new client connection, prints client information, and stores the client socket, IP address, and port in the respective vectors.

62-77. After all clients have connected, the server sends a message to each client requesting their local clock values. It then receives the responses from each client and extracts the local clock values from the received messages, storing them in the `clients\_local\_clocks` vector.

82-85. The code calculates the average clock value by summing the server's local clock and the client local clocks.

88-98. The server sends adjustment instructions to each client based on the calculated average clock value. It calculates the offset between each client's local clock and the average clock, determines whether to add or subtract the offset, and sends the adjustment message to the respective client.

101-103. The server adjusts its own local clock by adding or subtracting the difference between its current local clock and the average clock.

105-106. The final server local clock value is printed, and the server stops listening and closes the socket.

That's the explanation of the code. It implements a basic server that synchronizes clocks with multiple clients using the Berkeley algorithm.

Certainly! Here's a line-by-line explanation of the code in `Client.cpp`:

1. The first set of lines includes various header files that provide necessary functionality for the program, such as socket programming, string manipulation, and input/output operations.

3. `#define PORT 8080` defines a constant value `PORT` with the value `8080`. It represents the port number used for the client-server communication.

5. `using namespace std;` declares that the program will use the `std` namespace, which allows you to directly access standard library functions and objects without explicitly specifying the namespace.

8. The `split` function is defined. It takes a string `s` and a delimiter as input and splits the string into substrings based on the delimiter. The substrings are stored in a vector and returned.

18. The `main` function is the entry point of the program. It takes command-line arguments `argc` (argument count) and `argv` (argument vector).

22-24. `srand` and `rand` functions are used to generate a random number. The random number is assigned to `client\_local\_clock`, which represents the local clock value of the client. The `printf` statements print the client's process ID (`getpid()`) and the local clock value.

28-30. Variables `client\_socket\_fd`, `valread`, and `client\_read\_buffer` are declared. `client\_socket\_fd` holds the client's socket file descriptor, `valread` stores the number of bytes read, and `client\_read\_buffer` is an array used to store data read from the socket.

33-35. A `sockaddr\_in` structure named `server\_addr` is defined to hold the server's address information. `AF\_INET` represents the address family (IPv4), and `htons` converts the port number to network byte order.

39-46. The `socket` function is used to create a socket. The `AF\_INET` parameter specifies the address family, `SOCK\_STREAM` indicates a TCP socket, and `0` is the protocol value. If the socket creation fails, an error message is printed, and the program returns `-1`.

50-57. The program attempts to convert the server's IP address from text to binary form using `inet\_pton`. If the conversion fails, an error message is printed, and the program returns `-1`.

61-68. The `connect` function is called to establish a connection with the server using the client socket, server address, and port number. If the connection fails, an error message is printed, and the program returns `-1`.

72-77. The server's IP address and port number are converted from binary form to a string using `inet\_ntop`, and the information is printed.

82-86. The program reads data from the socket into the `client\_read\_buffer` using the `read` function, and the received message is printed.

89-100. The received message is converted from a character array to a string. Depending on the received message, the client prepares and sends a response to the server.

104-109. The program reads data from the socket into the `client\_read\_buffer` again, and the received message is printed.

112-131. The received message is processed to extract the clock adjustment offset value. The offset is applied to the client's local clock based on the received instructions.

136-138. The client socket is closed using the `close` function, and the program terminates with a return value of `0`.

I hope this explanation helps clarify the code for you. Let me know if you have any further questions!

**Name: Isha Moykhede**

**Roll No: 50**

**DS Assignment No: 5**

**Title: Implement token ring based mutual exclusion algorithm.**

**Code**

**Filename: TokenRing.java**

import java.util.Scanner;

public class TokenRing {

private int numNodes;

private int[] nodes;

private int token;

public TokenRing(int numNodes) {

this.numNodes = numNodes;k

this.nodes = new int[numNodes];

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

nodes[i] = i;

}

this.token = 0;

}

public void send(int sender, int receiver, int data) {

System.out.print("Token passing: ");

for (int i = token, j = token; (i % numNodes) != sender; i++, j = (j + 1) % numNodes) {

System.out.print(j + "->");

}

System.out.println(sender);

System.out.println("Sender " + sender + " sending data: " + data);

for (int i = sender + 1; i != receiver; i = (i + 1) % numNodes) {

System.out.println("data " + data + " forwarded by " + i);

}

System.out.println("Receiver " + receiver + " received data: " + data);

token = sender;

}

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the num of nodes:");

int numNodes = scan.nextInt();

TokenRing algorithm = new TokenRing(numNodes);

int sender, receiver, data, choice;

do {

System.out.println("Enter sender:");

sender = scan.nextInt();

System.out.println("Enter receiver:");

receiver = scan.nextInt();

System.out.println("Enter data:");

data = scan.nextInt();

algorithm.send(sender, receiver, data);

System.out.println("Do you want to send again? Enter 1 for Yes and 0 for No:");

choice = scan.nextInt();

} while (choice == 1);

}

}

**OutPut :**

Enter the num of nodes:

4

Enter sender:

0

Enter receiver:

3

Enter data:

100

Token passing: 1->2->3

Sender 0 sending data: 100

data 100 forwarded by 1

data 100 forwarded by 2

Receiver 3 received data: 100

Do you want to send again? Enter 1 for Yes and 0 for No:

1

Enter sender:

1

Enter receiver:

2

Enter data:

50

Token passing: 3->0->1

Sender 1 sending data: 50

data 50 forwarded by 2

Receiver 2 received data: 50

Do you want to send again? Enter 1 for Yes and 0 for No:

0

CODES EXPLAINANTION:

Certainly! Here's a line-by-line explanation of the code in `TokenRing.java`:

1. The code begins by importing the `Scanner` class from `java.util` to allow user input.

3. The `TokenRing` class is defined, representing the token ring algorithm.

4. The class has three instance variables: `numNodes` to store the number of nodes in the ring, `nodes` to store the node IDs, and `token` to represent the current token holder.

7-14. The constructor `TokenRing` is defined. It initializes the `numNodes` and `nodes` array based on the provided number of nodes. Each node is assigned an ID from 0 to `numNodes-1`, and the `token` is set to 0 initially.

17-30. The `send` method is defined to simulate the passing of the token and data between nodes. It takes three parameters: `sender` (the ID of the sending node), `receiver` (the ID of the receiving node), and `data` (the data to be transmitted).

19-25. The method begins by printing a message indicating the token passing process. It iterates from the current token position (`i`) to the sender's position (`sender`) in the ring, printing the IDs of the nodes involved in the token passing. Q

27-28. The sender's ID and the data being sent are printed.

29-35. The method continues by iterating from the next node after the sender (`sender + 1`) until the receiver is reached. It prints the data being forwarded and the ID of the forwarding node.

37-38. Finally, the method prints a message indicating that the receiver has received the data, along with the receiver's ID and the data itself. The token is updated to the sender's ID.

41-56. The `main` method is the entry point of the program.

43-45. It prompts the user to enter the number of nodes in the ring.

46-47. An instance of the `TokenRing` class is created, using the provided number of nodes.

49-54. Inside a do-while loop, the program prompts the user to enter the sender, receiver, and data for a message. It then calls the `send` method of the `algorithm` object to simulate sending the message.

56. The program asks the user if they want to send another message. If the user enters `1`, the loop continues; otherwise, it terminates.

I hope this explanation helps you understand the code. Let me know if you have any further questions!

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

**Name: Isha Moykhede**

**Roll No: 50**

**DS Assignment No: 6**

**Title: Implement Bully and Ring algorithm for leader election.**

**Codes**

**Bully algorithm**

**Filename: BullyAlgorithm.java**

import java.util.\*;

public class BullyAlgorithm {

private int numProcesses;

private boolean[] processes;

private int leader;

public BullyAlgorithm(int numProcesses) {

this.numProcesses = numProcesses;

this.processes = new boolean[numProcesses];

Arrays.fill(processes, true);

this.leader = numProcesses - 1;

}

public void startElection(int failedProcess) {

System.out.println("Process " + failedProcess + " has failed!");

for (int i = failedProcess + 1; i < numProcesses; i++) {

if (processes[i]) {

System.out.println("Sending election message from " + i + " to " + leader);

if (sendMessage(i, leader)) {

leader = i;

}

}

}

System.out.println("Process " + leader + " is the new leader!");

}

public boolean sendMessage(int source, int destination) {

System.out.println("Sending message from " + source + " to " + destination);

return processes[destination];

}

public static void main(String[] args) {

int numProcesses = 5;

BullyAlgorithm algorithm = new BullyAlgorithm(numProcesses);

algorithm.startElection(2);

}

}

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

**Filename: RingAlgorithm.java**

import java.util.\*;

public class RingAlgorithm {

private int numProcesses;

private int[] processes;

private int leader;

public RingAlgorithm(int numProcesses) {

this.numProcesses = numProcesses;

this.processes = new int[numProcesses];

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

processes[i] = i;

}

this.leader = numProcesses - 1;

}

public void startElection(int failedProcess) {

System.out.println("Process " + failedProcess + " has failed!");

int sender = (failedProcess + 1) % numProcesses;

boolean leaderFound = false;

while (!leaderFound) {

System.out.println("Sending election message from " + sender + " to " + (sender + 1) % numProcesses);

int receiver = (sender + 1) % numProcesses;

if (sendMessage(sender, receiver)) {

sender = receiver;

} else {

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

if (i != sender && sendMessage(sender, i)) {

sender = i;

}

}

leaderFound = true;

}

}

System.out.println("Process " + leader + " is the new leader!");

}

public boolean sendMessage(int source, int destination) {

System.out.println("Sending message from " + source + " to " + destination);

if (destination == leader) {

return true;

}

return false;

}

public static void main(String[] args) {

int numProcesses = 5;

RingAlgorithm algorithm = new RingAlgorithm(numProcesses);

algorithm.startElection(2);

}

}

**Output**

**Output of Bully.java**

Process 2 has failed!

Sending election message from 3 to 3

Sending election message from 4 to 3

Process 4 is the new leader!

**Output of Ring.java**

Process 2 has failed!

Sending election message from 3 to 4

Sending message from 3 to 4

Sending election message from 4 to 0

Sending message from 4 to 0

Sending message from 4 to 0

Sending message from 4 to 1

Sending message from 4 to 2

Sending message from 4 to 3

Process 4 is the new leader!

CODES EXPLAINATION:

Sure! Here's a detailed explanation of the code for the Bully and Ring algorithms for leader election:

\*\*BullyAlgorithm.java:\*\*

1. Import the `java.util` package for utility classes.

3. Start defining the `BullyAlgorithm` class.

5. Declare the instance variables: `numProcesses`, `processes`, and `leader`.

8-16. Define the constructor `BullyAlgorithm`, which takes the number of processes as a parameter. It initializes `numProcesses`, creates an array `processes` of size `numProcesses`, and sets all processes to active by setting the corresponding elements in the `processes` array to `true`. The initial leader is set as the process with the highest ID (equal to `numProcesses - 1`).

18-30. Define the `startElection` method to simulate the leader election process. It takes the ID of the failed process as a parameter.

20. Print the ID of the failed process.

22-29. Iterate over the processes with higher IDs, starting from `failedProcess + 1`. For each process, if it is active (`processes[i]` is `true`), simulate sending an election message from that process to the current leader (`leader`). If a response is received (indicating that the process with the higher ID is active), update the `leader` variable to the process sending the message.

31. Print the ID of the new leader.

33-37. Define the `sendMessage` method to simulate sending a message between processes. It takes the ID of the source process (`source`) and the ID of the destination process (`destination`) as parameters. In this implementation, the method simply checks if the destination process is active (`processes[destination]`). If it is active, it returns `true`; otherwise, it returns `false`.

39-44. Define the `main` method as the program entry point.

46. Set the number of processes (in this case, `5`).

47. Create an instance of the `BullyAlgorithm` class, passing the number of processes as an argument.

48. Start the leader election process by invoking the `startElection` method and passing the ID of the failed process (`2` in this case).

\*\*RingAlgorithm.java:\*\*

1. Import the `java.util` package for utility classes.

3. Start defining the `RingAlgorithm` class.

5. Declare the instance variables: `numProcesses`, `processes`, and `leader`.

8-15. Define the constructor `RingAlgorithm`, which takes the number of processes as a parameter. It initializes `numProcesses`, creates an array `processes` of size `numProcesses`, and assigns IDs to the processes in ascending order, starting from `0`. The initial leader is set as the process with the highest ID (equal to `numProcesses - 1`).

17-29. Define the `startElection` method to simulate the leader election process. It takes the ID of the failed process as a parameter.

31. Print the ID of the failed process.

33-34. Initialize the `sender` variable based on the failed process. In this implementation, the process with the ID next to the failed process (`(failedProcess + 1) % numProcesses`) becomes the initial sender.

35. Set a flag `leaderFound` to track if the leader has been determined.

37-44. Execute a loop until the leader is found. Inside the loop, the process identified by `sender` sends an election message to the next process in the ring (`(sender + 1) % numProcesses`). The variable `receiver` stores the ID of the next receiver.

39-41. Print the current sender and the next receiver.

42

. Calculate the next receiver in the ring using the modulo operator to ensure the value wraps around to the beginning if it exceeds the number of processes.

43-49. Send messages to the next receiver and check for a response. If a response is received (indicating that the next receiver is active), update the `sender` variable to the next receiver.

51-55. If no response is received, it means that the current sender has reached the highest ID process without finding a leader. In this case, the process iterates over all the other processes (excluding the current sender) and sends them messages. If a response is received from any of the other processes, update the `sender` variable to that process, indicating that it is the new sender. Finally, set the `leaderFound` flag to true, indicating that the leader has been determined.

57. Print the ID of the new leader.

59-64. Define the `sendMessage` method to simulate sending a message between processes. It takes the ID of the source process (`source`) and the ID of the destination process (`destination`) as parameters. In this implementation, the method checks if the destination process is the current leader. If it is the leader, it returns `true`; otherwise, it returns `false`.

66-71. Define the `main` method as the program entry point.

73. Set the number of processes (in this case, `5`).

74. Create an instance of the `RingAlgorithm` class, passing the number of processes as an argument.

75. Start the leader election process by invoking the `startElection` method and passing the ID of the failed process (`2` in this case).

I hope this provides a detailed and understandable explanation of each line of code in the Bully and Ring algorithms for leader election. Let me know if you have any further questions!

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

**Name: Isha Moykhede**

**Roll No: 50**

**DS Assignment No: 8**

**Title: To develop any distributed application through implementing client-server communication programs based on Java Sockets.**

**Codes**

**Filename: MyServer.java**

import java.net.\*;

import java.io.\*;

public class MyServer {

public static void main(String[] args) throws Exception{

//Creating a port for communication

ServerSocket ss = new ServerSocket(5555);

System.out.println("Server Initiated, Waiting for Client to Connect...");

//Binding Client and Server on port 5555

Socket s = ss.accept();

System.out.println("Client Connected");

//Reading input from KeyBoard

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

//OutputStream object to write to clients

OutputStream ostream = s.getOutputStream();

//PrintWriter object to send the data to the outputstream

PrintWriter pw = new PrintWriter(ostream,true);

//InputStream objects to recieve from Client

InputStream istream = s.getInputStream();

//Reading receieved message from client

BufferedReader recieve = new BufferedReader(new InputStreamReader(istream));

//Client Message and Server Message objects

String servermessage = "";

String clientmessage = "";

while(true)

{

//Read the inputstream of the client from the socket

clientmessage = recieve.readLine();

System.out.println("Client: "+clientmessage);

//if the message is bye end the communication here

if(clientmessage.equals("bye"))

{

break;

}

//Server writing its message

System.out.print("Server: ");

servermessage = br.readLine();

//print writer object sending the message to the socket through outputstream

pw.println(servermessage);

if(servermessage.equals("bye"))

{

break;

}

}

//closing all the streams and sockets

s.close();

ss.close();

istream.close();

ostream.close();

System.out.println("Connection Terminated");

}

}

**Filename: MyClient.java**

import java.net.\*;

import java.io.\*;

public class MyClient {

public static void main(String[] args) throws Exception{

//The socket object takes ip and port number of the server which client wants to connect

Socket s = new Socket("127.0.0.1",5555);

System.out.println("Connected to Server, Please type your message and hit Enter to send");

//Reading input from KeyBoard

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

//OutputStream object to write to Server

OutputStream ostream = s.getOutputStream();

//PrintWriter object to send the data to the outputstream

PrintWriter pw = new PrintWriter(ostream, true);

//InputStream objects to recieve from Server

InputStream istream = s.getInputStream();

//Reading receieved message from Server

BufferedReader recieve = new BufferedReader(new InputStreamReader(istream));

//Client Message and Server Message objects

String clientmessage = "";

String servermessage = "";

while(true)

{

//Input Message to be sent to Server

System.out.print("Client: ");

clientmessage = br.readLine();

//print writer object sending the message to the socket through outputstream

pw.println(clientmessage);

//if the message is bye end the communication here

if(clientmessage.equals("bye"))

{

break;

}

//Read the inputstream of the server from the socket

servermessage = recieve.readLine();

System.out.println("Server: "+servermessage);

//if the message is bye end the communication here

if(servermessage.equals("bye"))

{

break;

}

}

//closing all the streams and sockets

s.close();

istream.close();

ostream.close();

System.out.println("Connection Terminated");

}

}

**Output**

**CommandPrompt1: MyServer.java**

C:\Users\student>cd C:\Users\student\LP-V\_Program\1.Socket  
  
C:\Users\student\LP-V\_Program\1.Socket>javac MyServer.java  
  
C:\Users\student\LP-V\_Program\1.Socket>java MyServer  
Server Initiated, Waiting for Client to Connect...  
Client Connected  
Client: hi  
Server: hello  
Client: how are you?  
Server: i am fine  
Client: bye  
Connection Terminated  
  
C:\Users\student\LP-V\_Program\1.Socket>

**CommandPrompt2: MyClient.java**

Microsoft Windows [Version 6.3.9600]  
(c) 2013 Microsoft Corporation. All rights reserved.  
  
  
C:\Users\student>cd C:\Users\student\LP-V\_Program\1.Socket  
  
C:\Users\student\LP-V\_Program\1.Socket>javac MyClient.java  
  
C:\Users\student\LP-V\_Program\1.Socket>java MyClient  
Connected to Server, Please type your message and hit Enter to send  
Client: hi  
Server: hello  
Client: how are you?  
Server: i am fine  
Client: bye  
Connection Terminated  
  
C:\Users\student\LP-V\_Program\1.Socket>

CODES EXPLAINATION:

Certainly! Here's a line-by-line explanation of the code for implementing client-server communication using Java sockets:

\*\*MyServer.java:\*\*

1. Import the required packages for networking and I/O operations.

4. Start defining the `MyServer` class.

6-9. In the `main` method, create a `ServerSocket` object `ss` that listens for client connections on port `5555`.

10. Print a message to indicate that the server is initiated and waiting for a client to connect.

12. `accept()` method is called on the `ServerSocket` object to block until a client connection is made. When a client connects, a `Socket` object `s` is created to represent the connection.

13. Print a message to indicate that the client is connected.

16. Create a `BufferedReader` object `br` to read input from the keyboard.

19. Create an `OutputStream` object `ostream` from the `Socket` object `s` to write data to the client.

22. Create a `PrintWriter` object `pw` to send data to the output stream `ostream`. Setting `true` as the second argument enables automatic flushing of the stream.

25. Create an `InputStream` object `istream` from the `Socket` object `s` to receive data from the client.

28. Create a `BufferedReader` object `receive` to read the received message from the input stream `istream`.

31-32. Define `servermessage` and `clientmessage` variables to store the messages from the server and client, respectively.

34. Enter an infinite loop to handle the communication between the server and client.

37. Read the input stream of the client from the socket and store the message in `clientmessage`.

39. Print the client's message.

42-45. If the client's message is "bye," break out of the loop to end the communication.

48. Prompt the server to enter its message.

51. Read the server's message from the keyboard and store it in `servermessage`.

54. Use the `PrintWriter` object `pw` to send the server's message to the client through the output stream.

57-60. If the server's message is "bye," break out of the loop to end the communication.

63-70. Close all the streams and sockets to release the resources.

73. Print a message to indicate that the connection has been terminated.

\*\*MyClient.java:\*\*

1. Import the required packages for networking and I/O operations.

4. Start defining the `MyClient` class.

6-9. In the `main` method, create a `Socket` object `s` to establish a connection to the server with IP address "127.0.0.1" (localhost) and port `5555`.

10. Print a message to indicate that the client is connected to the server.

13. Create a `BufferedReader` object `br` to read input from the keyboard.

16. Create an `OutputStream` object `ostream` from the `Socket` object `s` to write data to the server.

19. Create a `PrintWriter` object `pw` to send data to the output stream `ostream`. Setting `true` as the second argument enables automatic flushing of the stream.

22. Create an `InputStream` object `istream` from the `Socket` object `s` to receive data from the server.

25. Create a `BufferedReader` object `receive` to read the received message from the input stream `istream`.

28-29. Define `clientmessage` and `servermessage` variables to store the messages from the client and server, respectively.

32. Enter an infinite loop to handle the communication between the client and server.

35

. Prompt the client to enter its message.

38. Read the client's message from the keyboard and store it in `clientmessage`.

41. Use the `PrintWriter` object `pw` to send the client's message to the server through the output stream.

44-47. If the client's message is "bye," break out of the loop to end the communication.

50. Read the input stream of the server from the socket and store the message in `servermessage`.

52. Print the server's message.

55-58. If the server's message is "bye," break out of the loop to end the communication.

61-67. Close all the streams and sockets to release the resources.

70. Print a message to indicate that the connection has been terminated.

I hope this explanation helps you understand the code for implementing client-server communication using Java sockets. Let me know if you have any further questions!

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

Berkely Algorithm in c

#include <stdio.h>

#define SERVER\_COUNT 4

// Function to calculate the average time

int calculateAverageTime(int serverTimes[], int count) {

int sum = 0;

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

sum += serverTimes[i];

}

int average = sum / count;

return average;

}

// Function to adjust the local time

void adjustLocalTime(int \*localTime, int offset) {

\*localTime += offset;

}

int main() {

int localTime = 0;

int serverTimes[SERVER\_COUNT] = {10, 8, 12, 6};

int offsets[SERVER\_COUNT];

// Calculate the offset for each server

for (int i = 0; i < SERVER\_COUNT; i++) {

offsets[i] = serverTimes[i] - localTime;

}

// Calculate the average offset

int averageOffset = calculateAverageTime(offsets, SERVER\_COUNT);

// Adjust the local time using the average offset

adjustLocalTime(&localTime, averageOffset);

// Print the adjusted time

printf("Adjusted Time: %d\n", localTime);

return 0;

}