**FCFS:**

#include <stdio.h>

int main() {

int n, i, j, temp;

int p[10], at[10], bt[10], ct[10], tat[10], wt[10];

float awt = 0, atat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Process IDs:\n");

for (i = 0; i < n; i++) {

scanf("%d", &p[i]);

}

printf("Enter Arrival Times:\n");

for (i = 0; i < n; i++) {

scanf("%d", &at[i]);

}

printf("Enter Burst Times:\n");

for (i = 0; i < n; i++) {

scanf("%d", &bt[i]);

}

for (i = 0; i < n - 1; i++) {

for (j = 0; j < n - i - 1; j++) {

if (at[j] > at[j + 1]) {

temp = at[j];

at[j] = at[j + 1];

at[j + 1] = temp;

temp = bt[j];

bt[j] = bt[j + 1];

bt[j + 1] = temp;

temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

ct[0] = at[0] + bt[0];

for (i = 1; i < n; i++) {

if (at[i] > ct[i - 1]) {

ct[i] = at[i] + bt[i];

} else {

ct[i] = ct[i - 1] + bt[i];

}

}

for (i = 0; i < n; i++) {

tat[i] = ct[i] - at[i];

wt[i] = tat[i] - bt[i];

atat += tat[i];

awt += wt[i];

}

atat /= n;

awt /= n;

printf("\nP\tAT\tBT\tCT\tTAT\tWT\n");

for (i = 0; i < n; i++) {

printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], ct[i], tat[i], wt[i]);

}

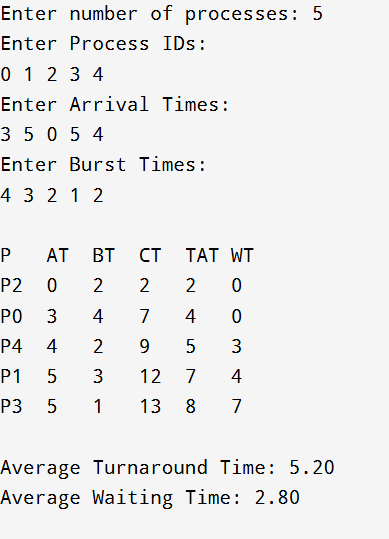
printf("\nAverage Turnaround Time: %.2f\n", atat);

printf("Average Waiting Time: %.2f\n", awt);

return 0;

}

**OUTPUT:**

****

**SJF:**

#include <stdio.h>

int main() {

int n, i, time = 0, done = 0, p[20], at[20], bt[20], ct[20], tat[20], wt[20], completed[20] = {0};

float atat = 0, awt = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

for (i = 0; i < n; i++) {

printf("Enter PID, Arrival Time & Burst Time for P%d: ", i);

scanf("%d%d%d", &p[i], &at[i], &bt[i]);

}

while (done < n) {

int min = -1;

for (i = 0; i < n; i++)

if (!completed[i] && at[i] <= time && (min == -1 || bt[i] < bt[min]))

min = i;

if (min == -1)

time++;

else {

time += bt[min];

ct[min] = time;

tat[min] = ct[min] - at[min];

wt[min] = tat[min] - bt[min];

atat += tat[min];

awt += wt[min];

completed[min] = 1;

done++;

}

}

printf("\nP\tAT\tBT\tCT\tTAT\tWT\n");

for (i = 0; i < n; i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], ct[i], tat[i], wt[i]);

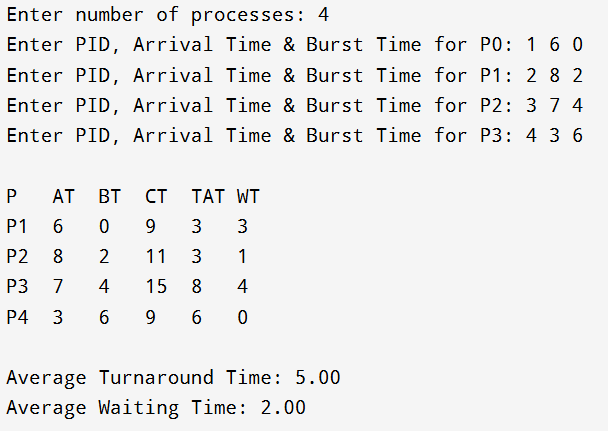
printf("\nAverage Turnaround Time: %.2f\n", atat / n);

printf("Average Waiting Time: %.2f\n", awt / n);

return 0;

}

**OUTPUT:**

****

**Priority:**

#include <stdio.h>

int main() {

int n, i, time = 0, done = 0;

int p[20], at[20], bt[20], pr[20], ct[20], tat[20], wt[20], completed[20] = {0};

float atat = 0, awt = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

for (i = 0; i < n; i++) {

printf("Enter PID, Arrival Time, Burst Time, and Priority for P%d: ", i);

scanf("%d%d%d%d", &p[i], &at[i], &bt[i], &pr[i]);

}

while (done < n) {

int min = -1;

for (i = 0; i < n; i++)

if (!completed[i] && at[i] <= time && (min == -1 || pr[i] < pr[min]))

min = i;

if (min == -1)

time++;

else {

time += bt[min];

ct[min] = time;

tat[min] = ct[min] - at[min];

wt[min] = tat[min] - bt[min];

atat += tat[min];

awt += wt[min];

completed[min] = 1;

done++;

}

}

printf("\nP\tAT\tBT\tPR\tCT\tTAT\tWT\n");

for (i = 0; i < n; i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], pr[i], ct[i], tat[i], wt[i]);

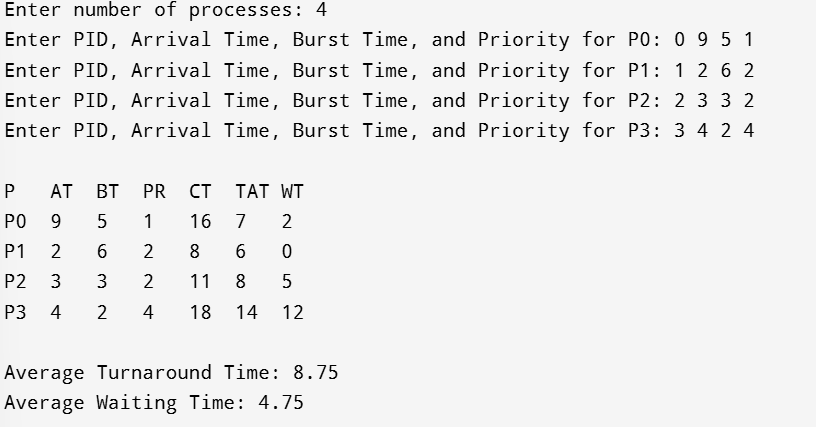
printf("\nAverage Turnaround Time: %.2f\n", atat / n);

printf("Average Waiting Time: %.2f\n", awt / n);

return 0;

}

**OUTPUT:**

****

**ROUND ROBIN:**

#include <stdio.h>

int main() {

int n, i, tq, time = 0, remain, flag = 0;

int at[20], bt[20], rt[20], ct[20], tat[20], wt[20], p[20];

float atat = 0, awt = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

remain = n;

for (i = 0; i < n; i++) {

printf("Enter Process ID, Arrival Time and Burst Time for P%d: ", i);

scanf("%d%d%d", &p[i], &at[i], &bt[i]);

rt[i] = bt[i]; // Copy burst to remaining time

}

printf("Enter Time Quantum: ");

scanf("%d", &tq);

int complete[n], done[n];

for (i = 0; i < n; i++) complete[i] = 0;

while (remain > 0) {

flag = 0;

for (i = 0; i < n; i++) {

if (at[i] <= time && rt[i] > 0) {

flag = 1;

if (rt[i] <= tq) {

time += rt[i];

rt[i] = 0;

ct[i] = time;

tat[i] = ct[i] - at[i];

wt[i] = tat[i] - bt[i];

atat += tat[i];

awt += wt[i];

remain--;

} else {

rt[i] -= tq;

time += tq;

}

}

}

if (flag == 0) time++;

}

printf("\nP\tAT\tBT\tCT\tTAT\tWT\n");

for (i = 0; i < n; i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], ct[i], tat[i], wt[i]);

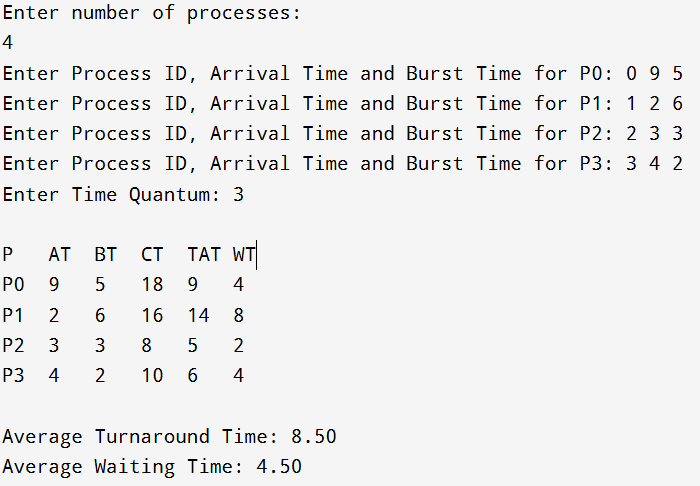
printf("\nAverage Turnaround Time: %.2f\n", atat / n);

printf("Average Waiting Time: %.2f\n", awt / n);

return 0;

}

**OUTPUT:**

****

**PRODUCER-CONSUMER:**

#include <stdio.h>

#include <stdlib.h>

int wait(int s) {

return --s;

}

int signal(int s) {

return ++s;

}

int mutex = 1;

int full = 0;

int empty = 5; // Assuming buffer size is 5

int x = 0;

void producer() {

empty = wait(empty);

mutex = wait(mutex);

x++;

printf("Producer produces item %d\n", x);

mutex = signal(mutex);

full = signal(full);

}

void consumer() {

full = wait(full);

mutex = wait(mutex);

printf("Consumer consumes item %d\n", x);

x--;

mutex = signal(mutex);

empty = signal(empty);

}

int main() {

int n;

printf("1. Press 1 for Producer\n");

printf("2. Press 2 for Consumer\n");

printf("3. Press 3 for Exit\n");

while (1) {

printf("\nEnter your choice: ");

scanf("%d", &n);

switch (n) {

case 1:

if ((mutex == 1) && (empty != 0)) {

producer();

} else {

printf("Buffer is full!\n");

}

break;

case 2:

if ((mutex == 1) && (full != 0)) {

consumer();

} else {

printf("Buffer is empty!\n");

}

break;

case 3:

exit(0);

default:

printf("Invalid choice!\n");

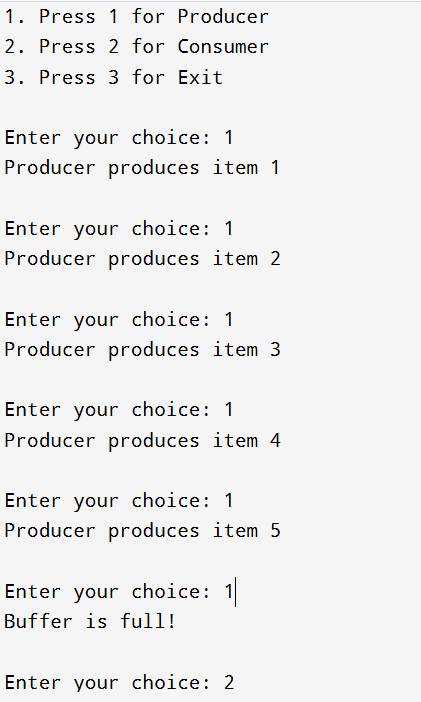
}

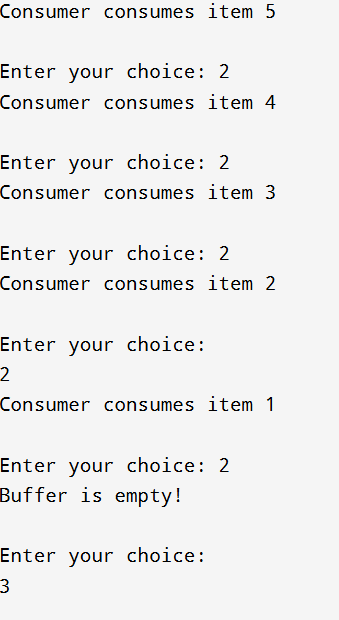
}

return 0;

}

**OUTPUT:**

****

****

**FIRST FIT:**

#include<stdio.h>

#include<stdlib.h>

void firstFit(int blockSize[],int m,int processSize[],int n){

int allocation[n];

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

allocation[i]=-1;

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

{

int firstIdx=-1;

for(int j=0;j<m;j++)

{

if(blockSize[j]>=processSize[i]){

firstIdx=j;

break;

}

}

if(firstIdx!=-1){

allocation[i]=firstIdx;

blockSize[firstIdx]=blockSize[firstIdx]-processSize[i];

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

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

{

printf("%d\t\t%d\t\t",i,processSize[i]);

if(allocation[i]!=-1){

printf("%d\n",allocation[i]);

}else{

printf("Not Allocation\n");

}

}

}void main(){

int i,bs,p,blockSize[10],processSize[10];

printf("Enter no.of blocks:");

scanf("%d",&bs);

for(i=0;i<bs;i++){

printf("Enter %d block size:",i);

scanf("%d",&blockSize[i]);

}

printf("Enter no.of process:");

scanf("%d",&p);

for(i=0;i<p;i++){

printf("Enter %d process size:",i);

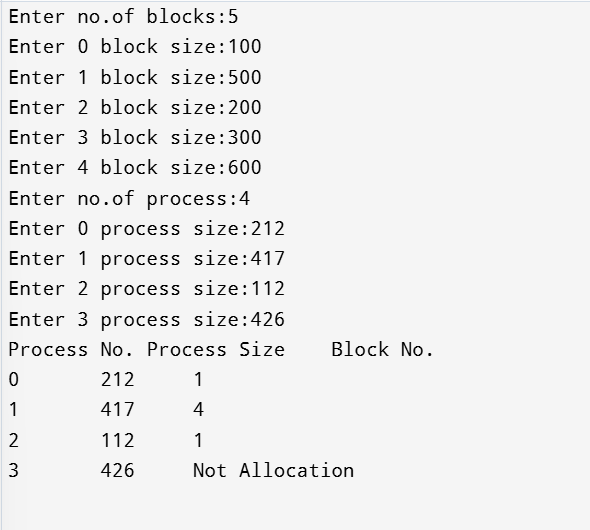
scanf("%d",&processSize[i]);

}

firstFit(blockSize,bs,processSize,p);

}

**OUTPUT:**

****

**BEST FIT:**

#include<stdio.h>

#include<stdlib.h>

void BestFit(int blockSize[],int m,int processSize[],int n){

int allocation[n];

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

allocation[i]=-1;

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

{

int bestIdx=-1;

for(int j=0;j<m;j++)

{

if(blockSize[j]>=processSize[i]){

if(bestIdx==-1||blockSize[j]<blockSize[bestIdx])

bestIdx=j;

}

}

if(bestIdx!=-1){

allocation[i]=bestIdx;

blockSize[bestIdx]=blockSize[bestIdx]-processSize[i];

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

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

{

printf("%d\t\t%d\t\t",i,processSize[i]);

if(allocation[i]!=-1){

printf("%d\n",allocation[i]);

}else{

printf("Not Allocation\n");

}

}

}void main(){

int i,bs,p,blockSize[10],processSize[10];

printf("Enter no.of blocks:");

scanf("%d",&bs);

for(i=0;i<bs;i++){

printf("Enter %d block size:",i);

scanf("%d",&blockSize[i]);

}

printf("Enter no.of process:");

scanf("%d",&p);

for(i=0;i<p;i++){

printf("Enter %d process size:",i);

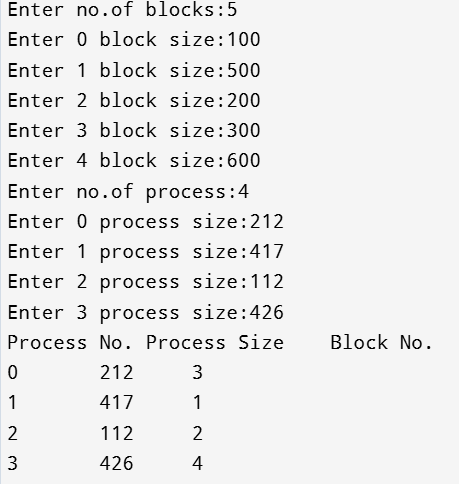
scanf("%d",&processSize[i]);

}

BestFit(blockSize,bs,processSize,p);

}

**OUTPUT:**

****

**WORST FIT:**

#include<stdio.h>

#include<stdlib.h>

void worstFit(int blockSize[],int m,int processSize[],int n){

int allocation[n];

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

allocation[i]=-1;

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

{

int worstIdx=-1;

for(int j=0;j<m;j++)

{

if(blockSize[j]>=processSize[i]){

if(worstIdx==-1||blockSize[j]>blockSize[worstIdx])

worstIdx=j;

}

}

if(worstIdx!=-1){

allocation[i]=worstIdx;

blockSize[worstIdx]=blockSize[worstIdx]-processSize[i];

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

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

{

printf("%d\t\t%d\t\t",i,processSize[i]);

if(allocation[i]!=-1){

printf("%d\n",allocation[i]);

}else{

printf("Not Allocation\n");

}

}

}void main(){

int i,bs,p,blockSize[10],processSize[10];

printf("Enter no.of blocks:");

scanf("%d",&bs);

for(i=0;i<bs;i++){

printf("Enter %d block size:",i);

scanf("%d",&blockSize[i]);

}

printf("Enter no.of process:");

scanf("%d",&p);

for(i=0;i<p;i++){

printf("Enter %d process size:",i);

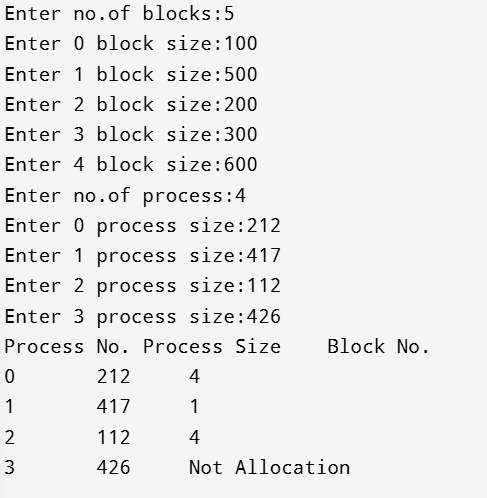
scanf("%d",&processSize[i]);

}

worstFit(blockSize,bs,processSize,p);

}

**OUTPUT:**

****

**CN**

**CALCULATOR:**

**Remote interface:**

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface Calculator extends Remote {

public double add(double a, double b) throws RemoteException;

public double subtract(double a, double b) throws RemoteException;

public double multiply(double a, double b) throws RemoteException;

public double divide(double a, double b) throws RemoteException;

}

**Remote implementation:**

import java.rmi.server.UnicastRemoteObject;

import java.rmi.RemoteException;

public class CalculatorImpl extends UnicastRemoteObject implements Calculator {

protected CalculatorImpl() throws RemoteException {

super();

}

@Override

public double add(double a, double b) throws RemoteException {

return a + b;

}

@Override

public double subtract(double a, double b) throws RemoteException {

return a - b;

}

@Override

public double multiply(double a, double b) throws RemoteException {

return a \* b;

}

@Override

public double divide(double a, double b) throws RemoteException {

if (b == 0) throw new ArithmeticException("Division by zero!");

return a / b;

}

}

**SERVER:**

import java.rmi.Naming;

import java.rmi.registry.LocateRegistry;

public class Server {

public static void main(String[] args) {

try {

LocateRegistry.createRegistry(1099); // Start RMI registry on port 1099

Calculator calculator = new CalculatorImpl();

Naming.rebind("CalculatorService", calculator);

System.out.println("Calculator Service is ready.");

} catch (Exception e) {

System.err.println("Server exception: " + e.toString());

e.printStackTrace();

}

}

}

**CLIENT:**

import java.rmi.Naming;

public class Client {

public static void main(String[] args) {

try {

Calculator calculator = (Calculator) Naming.lookup("rmi://localhost/CalculatorService");

System.out.println("Addition: " + calculator.add(5, 3));

System.out.println("Subtraction: " + calculator.subtract(10, 4));

System.out.println("Multiplication: " + calculator.multiply(7, 6));

System.out.println("Division: " + calculator.divide(20, 4));

} catch (Exception e) {

System.err.println("Client exception: " + e.toString());

e.printStackTrace();

}

}

}

**TCP:**

**CN TCP**

SERVER:

import java.io.\*;

import java.net.\*;

public class TCPServer {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(5000)) {

System.out.println("TCP Server is running...");

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected!");

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

String message;

while ((message = in.readLine()) != null) {

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

out.println("Echo: " + message); // Echo message back to client

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**CLIENT:**

import java.io.\*;

import java.net.\*;

public class TCPClient {

public static void main(String[] args) {

try (Socket socket = new Socket("localhost", 5000)) {

System.out.println("Connected to TCP Server!");

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

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

String message;

while (true) {

System.out.print("Enter message: ");

message = userInput.readLine();

out.println(message);

System.out.println("Server: " + in.readLine());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**CN UDP**

**SERVER:**

import java.net.\*;

public class UDPServer {

public static void main(String[] args) {

try (DatagramSocket socket = new DatagramSocket(6000)) {

System.out.println("UDP Server is running...");

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

while (true) {

socket.receive(packet);

String message = new String(packet.getData(), 0, packet.getLength());

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

// Respond to the client

String response = "Echo: " + message;

byte[] responseData = response.getBytes();

DatagramPacket responsePacket = new DatagramPacket(responseData, responseData.length, packet.getAddress(), packet.getPort());

socket.send(responsePacket);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**CLIENT:**

import java.io.\*;

import java.net.\*;

public class UDPClient {

public static void main(String[] args) {

try (DatagramSocket socket = new DatagramSocket()) {

InetAddress serverAddress = InetAddress.getByName("localhost");

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

String message;

while (true) {

System.out.print("Enter message: ");

message = userInput.readLine();

byte[] sendData = message.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, 6000);

socket.send(sendPacket);

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String response = new String(receivePacket.getData(), 0, receivePacket.getLength());

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

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**CN HelloEcho program**

**SERVER:**

import java.io.\*;

import java.net.\*;

public class HelloEchoServer {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(5000)) {

System.out.println("Server is running...");

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected!");

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

String command = in.readLine();

if (command.equalsIgnoreCase("Hello")) {

out.println("Hello - Server is active!");

} else if (command.equalsIgnoreCase("Echo")) {

long startTime = System.currentTimeMillis(); // Start RTT calculation

out.println("Echo - Response from server!");

long endTime = System.currentTimeMillis(); // End RTT calculation

long rtt = endTime - startTime;

System.out.println("RTT measured by server: " + rtt + " ms");

} else {

out.println("Unknown command. Please use 'Hello' or 'Echo'.");

}

clientSocket.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**CLIENT:**

import java.io.\*;

import java.net.\*;

public class HelloEchoClient {

public static void main(String[] args) {

try (Socket socket = new Socket("localhost", 5000)) {

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

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

System.out.println("Type 'Hello' to check server availability or 'Echo' to measure RTT:");

String command = userInput.readLine();

if (command.equalsIgnoreCase("Echo")) {

long startTime = System.currentTimeMillis(); // Start RTT measurement

out.println(command);

String response = in.readLine(); // Wait for response

long endTime = System.currentTimeMillis(); // End RTT measurement

long rtt = endTime - startTime; // Calculate RTT

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

System.out.println("Round-Trip Time (RTT): " + rtt + " ms");

} else if (command.equalsIgnoreCase("Hello")) {

out.println(command);

String response = in.readLine(); // Wait for response

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

} else {

System.out.println("Invalid command. Please use 'Hello' or 'Echo'.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

****

****