INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No** | **Date** | **Practical** | **Signature** |
| 1 |  | Port 17 is known as the ‘Quote of the day service’. When a client connects to port 17 on a server, the server responds with a quote for that day. Write a server program so that it delivers a quote of the day. The quotes should be printable ASCII characters and should contain fewer than 512 characters, although multiple lines are allowed. Since port 17 is considered well known and therefore unavailable, have your server listen to port 6017.  Write the client code used to read the quotes returned by the server. |  |
| 2 |  | Write a client–server application using Java sockets that allows a client to write a message (as a String) to a socket.  A server will read this message, count the number of characters and digits in the message, and send these two counts back to the client.  The server will listen to port 6100.  The client can obtain the String message that it is to pass to the server either from the command line or by using a prompt to the user.  One strategy for sending the two counts back to the client is for the server to construct an object containing:  a. The message it receives from the client.  b. A count of the number of characters in the message.  c. A count of the number of digits in the message |  |
| 3 |  | Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections open at any point in time.  After N connections have been made, the server will not accept another incoming connection until an existing connection is released.  Write Java programs to demonstrate the scenario. |  |
| 4 |  | Assuming that a system has a 32-bit virtual address. Write a Java program i.e. passed. (1)The size of a page and (2) The virtual address. Your program will report the page number and offset of the given virtual address with the specified page size.  Page sizes must be specified as a power of 2 and within the range 1024-16384 (inclusive).  Assuming such a program is named address, it would run as follows:-  Java Address 4096 19986 and the correct output would appear as:-  The address 19986 contains:- Page Number = 4 offset =  3602 |  |
| 5 |  | Write a program that implements the FIFO and LRU page-replacement algorithms presented in this chapter. First, generate a random page reference string where page numbers range from 0 to 9. Apply the random page reference string to each algorithm, and record the number of page faults incurred by each algorithm.  Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used. Design and implement two classes—LRU and FIFO—that extend ReplacementAlgorithm. Each of these classes will implement the insert() method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm. Test your algorithm with suitable Java programs |  |
|  |  |  |  |

**PRACTICAL NO: 1**

**Aim:** Port 17 is known as the ‘Quote of the day service’. When a client connects to port 17 on a server, the server responds with a quote for that day. Write a server program so that it delivers a quote of the day. The quotes should be printable ASCII characters and should contain fewer than 512 characters, although multiple lines are allowed. Since port 17 is considered well known and therefore unavailable, have your server listen to port 6017.

Write the client code used to read the quotes returned by the server.

**INPUT:-**

Server:

import java.io.\*;

import java.net.\*;

import java.util.Random;

public class server {

private static final int PORT = 12068;

private static final String[] QUOTES = {

"The only limit to our realization of tomorrow is our doubts of today. - Franklin D. Roosevelt",

"Do not wait to strike till the iron is hot; but make it hot by striking. - William Butler Yeats",

"Whether you think you can or think you can't, you're right. - Henry Ford",

"The future belongs to those who believe in the beauty of their dreams. - Eleanor Roosevelt",

"It does not matter how slowly you go as long as you do not stop. - Confucius",

"Life is what happens when you're busy making other plans. - John Lennon"

};

public static void main(String[] args) {

System.out.println("Quote of the Day Server is running.");

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

while (true) {

try (Socket clientSocket = serverSocket.accept();

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

String quote = getRandomQuote();

out.println(quote);

} catch (IOException e) {

System.err.println("Error handling client connection: " + e.getMessage());

}

}

} catch (IOException e) {

System.err.println("Could not start server on port " + PORT);

e.printStackTrace();

}

}

private static String getRandomQuote() {

Random random = new Random();

int index = random.nextInt(QUOTES.length);

return QUOTES[index];

}

}

Client:

import java.io.\*;

import java.net.\*;

public class client {

private static final String SERVER\_ADDRESS = "localhost";

private static final int PORT = 12068;

public static void main(String[] args) {

try (Socket socket = new Socket(SERVER\_ADDRESS, PORT);

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

String quote;

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

System.out.println("Quote of the Day: " + quote);

}

} catch (IOException e) {

System.err.println("Error connecting to server: " + e.getMessage());

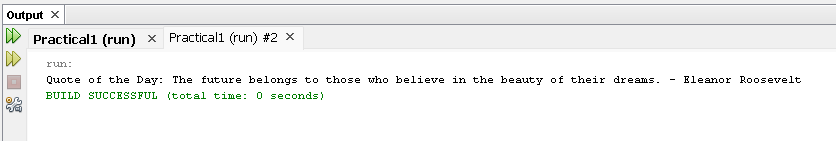
e.printStackTrace();

}

}

}

**OUTPUT:-**

****

**PRACTICAL NO: 2**

**Aim:** Write a client–server application using Java sockets that allows a client to write a message (as a String) to a socket.

A server will read this message, count the number of characters and digits in the message, and send these two counts back to the client.

The server will listen to port 6100.

The client can obtain the String message that it is to pass to the server either from the command line or by using a prompt to the user.

One strategy for sending the two counts back to the client is for the server to construct an object containing:

a. The message it receives from the client.

b. A count of the number of characters in the message.

c. A count of the number of digits in the message.

**INPUT:-**

Server:

import java.io.\*;

import java.net.\*;

class MessageInfo implements Serializable {

private String message;

private int characterCount;

private int digitCount;

public MessageInfo(String message, int characterCount, int digitCount) {

this.message = message;

this.characterCount = characterCount;

this.digitCount = digitCount;

}

public String getMessage() {

return message;

}

public int getCharacterCount() {

return characterCount;

}

public int getDigitCount() {

return digitCount;

}

}

public class server {

public static void main(String[] args) {

int port = 12068;

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

System.out.println("Server listening on port " + port);

while (true) {

try (Socket clientSocket = serverSocket.accept();

ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());

ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream())) {

String message = (String) in.readObject();

int characterCount = 0;

int digitCount = 0;

for (char c : message.toCharArray()) {

if (Character.isDigit(c)) {

digitCount++;

} else if (Character.isLetter(c)) {

characterCount++;

}

}

MessageInfo messageInfo = new MessageInfo(message, characterCount, digitCount);

out.writeObject(messageInfo);

} catch (IOException | ClassNotFoundException e) {

System.err.println("Error in communication with client: " + e.getMessage());

}

}

} catch (IOException e) {

System.err.println("Error starting server: " + e.getMessage());

}

}

}

Client:

import java.io.\*;

import java.net.\*;

public class client {

public static void main(String[] args) {

String host = "localhost";

int port = 12068;

String message;

if (args.length > 0) {

message = args[0];

} else {

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

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

try {

message = reader.readLine();

} catch (IOException e) {

System.err.println("Error reading input: " + e.getMessage());

return;

}

}

try (Socket socket = new Socket(host, port);

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

out.writeObject(message);

MessageInfo messageInfo = (MessageInfo) in.readObject();

System.out.println("Message: " + messageInfo.getMessage());

System.out.println("Character Count: " + messageInfo.getCharacterCount());

System.out.println("Digit Count: " + messageInfo.getDigitCount());

} catch (IOException | ClassNotFoundException e) {

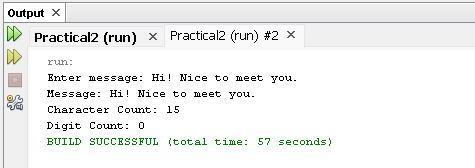
System.err.println("Error in communication with server: " + e.getMessage());

}

}

}

**OUTPUT:-**



**PRACTICAL NO: 3**

**Aim:** Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections open at any point in time.

After N connections have been made, the server will not accept another incoming connection until an existing connection is released.

Write Java programs to demonstrate the scenario.

**INPUT:-**

Server:

import java.io.\*;

import java.net.\*;

import java.util.concurrent.Semaphore;

public class server {

private static final int PORT = 6017;

private static final int MAX\_CONNECTIONS = 3; // Limit to 3 simultaneous connections

private static final Semaphore semaphore = new Semaphore(MAX\_CONNECTIONS);

public static void main(String[] args) {

System.out.println("Server is running on port " + PORT);

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

while (true) {

try {

semaphore.acquire(); // Acquire a permit

Socket clientSocket = serverSocket.accept();

new Thread(new ClientHandler(clientSocket)).start();

} catch (InterruptedException e) {

System.err.println("Semaphore acquisition interrupted: " + e.getMessage());

}

}

} catch (IOException e) {

System.err.println("Could not start server on port " + PORT);

e.printStackTrace();

}

}

private static class ClientHandler implements Runnable {

private final Socket clientSocket;

public ClientHandler(Socket clientSocket) {

this.clientSocket = clientSocket;

}

@Override

public void run() {

try (Socket socket = clientSocket; // Explicitly initialize within try-with-resources

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

out.println("You are connected to the server!");

// Simulate some server processing

Thread.sleep(5000);

} catch (IOException | InterruptedException e) {

System.err.println("Error handling client connection: " + e.getMessage());

} finally {

semaphore.release(); // Release the permit

} } } }

Client:

import java.io.\*;

import java.net.\*;

public class client {

private static final String SERVER\_ADDRESS = "localhost";

private static final int PORT = 6017;

public static void main(String[] args) {

try (Socket socket = new Socket(SERVER\_ADDRESS, PORT);

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

String message;

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

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

}

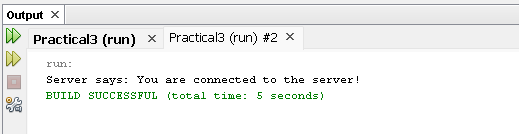
} catch (IOException e) {

System.err.println("Error connecting to server: " + e.getMessage());

e.printStackTrace();

} } }

**OUTPUT:-**



**PRACTICAL NO: 4**

**Aim:** Assuming that a system has a 32-bit virtual address. Write a Java program i.e. passed.

(1)The size of a page and

(2) The virtual address. Your program will report the page number and offset of the given virtual address with the specified page size. Page sizes must be specified as a power of 2 and within the range 1024-16384 (inclusive).

Assuming such a program is named address, it would run as follows:-

Java Address 4096 19986 and the correct output would appear as:-

The address 19986 contains:- Page Number = 4offset=3602

**INPUT:-**

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Address

{

public static final int ADDRESS\_SIZE=32; public static void main(String[]args)

{

try

{

if(args.length!=2)

{

System.out.println("Usage:java Address<page size><address>");

}

System.out.println("Please enter the parameters <pagesize><address>"); BufferedReader bt=new BufferedReader(new InputStreamReader(System.in)); int pageSize=Integer.parseInt(bt.readLine().trim());

int address=Integer.parseInt(bt.readLine().trim()); int pageBits=0;

int pageMask=0; int offsetMask=0; switch(pageSize)

{

case 1024: pageBits=10;

offsetMask=0x000003ff; pageMask=0xfffffc00; break; case 2048:

pageBits=11;

offsetMask=0x000007ff; pageMask=0xfffff800; break; case 4096:

pageBits=12; offsetMask=0x00000fff;

pageMask=0xfffff000; break;

case 8192: pageBits=13; offsetMask=0x00001fff; pageMask=0xffffe000; break;

case 16384: pageBits=14; offsetMask=0x000003ff; pageMask=0xfffffc00; break;

}

int pageNumber=(address & pageMask)>>pageBits; int offset=(address & offsetMask);

System.out.println("For address"+address+":pageNumber="+pageNumber+"offset="+offset);

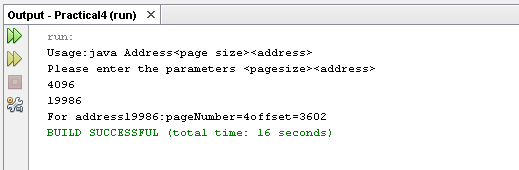
}

catch(Exception e){}

}

}

**OUTPUT:-**



**PRACTICAL NO: 5**

**Aim:** Write a program that implements the FIFO and LRU page-replacement algorithms presented in this chapter. First, generate a random page reference string where page numbers range from 0 to 9. Apply the random page reference string to each algorithm, and record the number of page faults incurred by each algorithm.

Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used. Design and implement two classes—LRU and FIFO—that extend ReplacementAlgorithm. Each of these classes will implement the insert() method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm. Test your algorithm with suitable Java programs

**INPUT:-**

FIFO:

import java.util.\*;

class FIFO extends ReplacementAlgorithm {

private final Queue<Integer> queue;

public FIFO(int capacity) {

super(capacity);

queue = new LinkedList<>();

}

@Override

public void insert(int page) {

if (!queue.contains(page)) {

if (queue.size() == capacity) {

queue.poll(); // Remove oldest page

}

queue.add(page);

pageFaults++;

}

updateFrames(queue);

}

private void updateFrames(Queue<Integer> queue) {

int i = 0;

for (int page : queue) {

frames[i++] = page;

}

while (i < capacity) {

frames[i++] = -1;

}

}

}

LRU:

import java.util.\*;

class LRU extends ReplacementAlgorithm {

private final LinkedHashMap<Integer, Boolean> map;

public LRU(int capacity) {

super(capacity);

map = new LinkedHashMap<>(capacity, 0.75f, true);

}

@Override

public void insert(int page) {

if (!map.containsKey(page)) {

if (map.size() == capacity) {

Iterator<Integer> iterator = map.keySet().iterator();

iterator.next();

iterator.remove();

}

map.put(page, true);

pageFaults++;

} else {

// Access the page to make it most recently used

map.get(page);

}

updateFrames();

}

private void updateFrames() {

int i = 0;

for (int page : map.keySet()) {

frames[i++] = page;

}

while (i < capacity) {

frames[i++] = -1;

}

}

}

ReplacementAlgorithm:

import java.util.\*;

abstract class ReplacementAlgorithm {

protected int pageFaults;

protected int[] frames;

protected int capacity;

public ReplacementAlgorithm(int capacity) {

this.capacity = capacity;

this.frames = new int[capacity];

Arrays.fill(this.frames, -1); // Initialize frame with -1, indicating empty frame

this.pageFaults = 0;

}

public abstract void insert(int page);

public int getPageFaults() {

return pageFaults;

}

}

PageReplacementTest:

import java.util.Random;

public class PageReplacementTest {

public static void main(String[] args) {

int numPages = 10; // Pages are numbered from 0 to 9

int numFrames = 3; // Number of page frames

int numReferences = 20; // Number of page references

int[] pageReferenceString = generateRandomPageReferenceString(numReferences, numPages);

System.out.println("Page Reference String: ");

for (int page : pageReferenceString) {

System.out.print(page + " ");

}

System.out.println();

// Test FIFO algorithm

FIFO fifo = new FIFO(numFrames);

for (int page : pageReferenceString) {

fifo.insert(page);

}

System.out.println("FIFO - Number of page faults: " + fifo.getPageFaults());

// Test LRU algorithm

LRU lru = new LRU(numFrames);

for (int page : pageReferenceString) {

lru.insert(page);

}

System.out.println("LRU - Number of page faults: " + lru.getPageFaults());

}

private static int[] generateRandomPageReferenceString(int length, int numPages) {

Random random = new Random();

int[] pageReferenceString = new int[length];

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

pageReferenceString[i] = random.nextInt(numPages);

}

return pageReferenceString;

}

}

**OUTPUT:-**

