Universitatea Tehnică din Cluj-Napoca
Facultatea de Automatică și Calculatoare

Documentație

Comunicare în Inel (Ring Comunication)

Disciplina: Rețele de Calculatoare

Studenți: Maria-Magdalena Creț & Rad Daniel-Cristian

Grupa 30233

Anul Universitar 2024-2025

Implementarea proiectului poate fi găsită în repository-ul GitHub:
Accesează repository-ul

Cuprins

1 Introducere

Se dorește implementarea unei comunicari în Inel (Ring Comunication) bazată pe imaginea de mai jos:

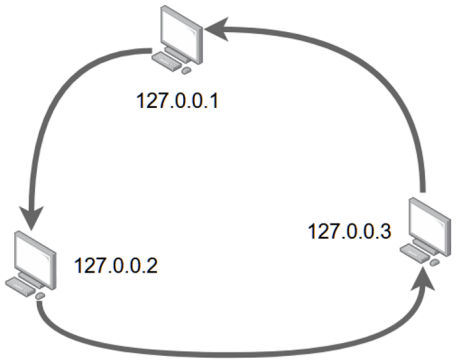

Figura 1: Topologie Ring Comunication

Se menționează că:

- Trei calculatoare comunică într-o singură direcție creând o buclă
- Unul dintre calculatoare inițiază comunicarea trimițând valoarea '1'
- La primire, fiecare dispozitiv de rețea incrementează valoarea primită și o trimite către dispozitivul următor
- Comunicarea se încheie când încărcătura livrată atinge valoarea '100'

2 Implementare

S-au respectat în implementare următoarele reguli:

1. **Reutilizarea codului**: S-a implementat o singură clasă `RingNode` care este instanțiată de 3 ori cu diferiți parametri de comunicare
2. **Comunicare:** Se folosesc socket-uri TCP pentru comunicare
3. **Adresare:** Se folosește intervalul de adrese pentru buclă locală (loopback): 127.0.0.1 - 127.0.0.3

2.1 Structura nodurilor

Nod 1 (inițiator):

Ascultă pe: 127.0.0.1:5001

Trimite către: 127.0.0.2:5002

Inițiază comunicarea cu valoarea 1

Nod 2:

Ascultă pe: 127.0.0.2:5002

Trimite către: 127.0.0.3:5003

Nod 3:

Ascultă pe: 127.0.0.3:5003

Trimite către: 127.0.0.1:5001

Diagrama conceptuală a topologiei

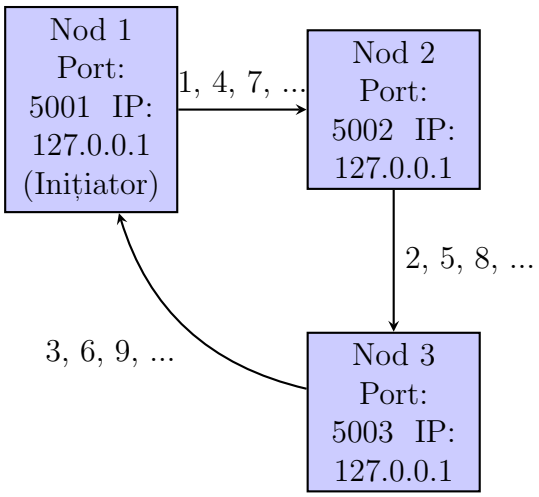

Figura 2: Comunicarea nodurilor exemplificată

2.2 Analiza clasei RingNode

Atribute principale

Clasa `RingNode` are următoarele atribute esențiale:

```
private int nodeId;
private int listenPort;
private String nextNodeIP;
private int nextNodePort;
private boolean isInitiator;
private static final int FINAL_VALUE = 100;
```

Aceste atribute definesc comportamentul și identitatea fiecărui nod în rețea:

- `nodeId` identifică în mod unic fiecare nod
- `listenPort` specifică portul pe care nodul ascultă mesaje de intrare
- `nextNodeIP` și `nextNodePort` specifică adresa următorului nod în circuit
- `isInitiator` indică dacă nodul inițiază comunicarea
- `FINAL_VALUE` definește valoarea la care comunicarea se oprește

Acest design permite instanțierea aceleiași clase pentru a crea noduri cu comportamente diferite, respectând principiul reutilizării codului.

Implementare metode

Metoda start()

Metoda `start()` inițiază operațiunile nodului:

```
public void start() {
System.out.println("Nod " + nodeId + ": Pornire...");
System.out.println("Nod " + nodeId + ": Ascult pe portul " + listenPort)
   ;
System.out.println("Nod " + nodeId + ": Conectez la " + nextNodeIP + ":"
    + nextNodePort);
if (isInitiator) {
    try {
        Thread.sleep(2000);
        System.out.println("Nod " + nodeId + ": Initez prima conexiune
           catre Nod 2");
        sendMessage(1);
        System.out.println("Nod " + nodeId + ": Am initiat comunicarea
           cu valoarea 1");
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

new Thread(() -> listenForMessages()).start();
}
```

Această metodă:

1. Afișează informații despre configurația nodului
2. Dacă nodul este inițiator, trimite primul mesaj cu valoarea 1
3. Pornește un thread separat pentru a asculta mesajele primite

Metoda listenForMessages()

Acest metoda implementează comportamentul de ”server” al nodului:

```
private void listenForMessages() {
try (ServerSocket serverSocket = new ServerSocket(listenPort)) {
System.out.println("Nod " + nodeId + ": Server pornit pe portul " +
   listenPort);
    while (true) {
        try (Socket clientSocket = serverSocket.accept();
             BufferedReader in = new BufferedReader(new
                InputStreamReader(clientSocket.getInputStream()))) {

            System.out.println("\nNod " + nodeId + ": Am primit o noua
               conexiune:");
        ...
}
```

Această metodă:

1. Creează un `ServerSocket` care ascultă pe portul specificat

2. Intră într-o buclă infinită, așteptând conexiuni
3. Când o conexiune este stabilită:
    - Primește și citește mesajul
    - Extrage valoarea din mesaj
    - Verifică dacă valoarea a atins pragul final (100)
    - Incrementează valoarea
    - Trimite noua valoare la următorul nod
4. Introduce o pauză de 1 secundă pentru a încetini comunicarea

Metoda sendMessage()

Metoda `sendMessage()` implementează comportamentul de "client" al nodului:

```
private void sendMessage(int value) {
try (Socket socket = new Socket(nextNodeIP, nextNodePort);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {
    System.out.println("\nNod " + nodeId + ": Initez conexiune catre Nod
        " + ((nodeId % 3) + 1) + ":");
    System.out.println("  - La adresa: " + socket.getInetAddress().
       getHostAddress());
    System.out.println("  - La portul: " + socket.getPort());
    System.out.println("  - Din portul local: " + socket.getLocalPort())
       ;
    ...
}
```

Această metodă:

1. Creează un `Socket` pentru a se conecta la următorul nod
2. Construiește un mesaj formatat ce conține valoarea
3. Trimite mesajul prin conexiunea TCP
4. Afișează informații despre conexiunea stabilită

Metoda extractValueFromMessage()

Această metodă extrage valoarea numerică din mesajul primit:

```
private int extractValueFromMessage(String message) {
String[] parts = message.split("_");
return Integer.parseInt(parts[parts.length - 1]);
}
```

Mesajele au formatul `NOD_X_TO_NOD_Y_VALUE_Z`, iar metoda extrage și convertește la `int` valoarea Z.

2.3 Analiza clasei RingLauncher

Clasa `RingLauncher` oferă o modalitate de a porni individual fiecare nod:

```
public class RingLauncher {
public static void main(String[] args) {
if (args.length < 1) {
System.out.println("Utilizare: java RingLauncher <nodeNumber>");
System.out.println("nodeNumber: 1, 2 sau 3 pentru a porni nodul
   corespunzator");
return;
}
    int nodeNumber = Integer.parseInt(args[0]);

    switch (nodeNumber) {
        case 1:
            // Pornire Nod 1 (initiator)
            RingNode node1 = new RingNode(1, 5001, "127.0.0.1", 5002,
               true);
            node1.start();
            System.out.println("Node 1 started as initiator.");
            break;
     ...
    }
}
}
```

Această clasă:

1. Primește un argument de la linia de comandă pentru a specifica numărul nodului
2. Creează și pornește doar nodul specificat
3. Permite rularea nodurilor în procese separate

2.4 Analiza conceptului de thread-uri în implementare

2.4.1 Necesitatea utilizării thread-urilor

Thread-urile sunt esențiale în această implementare din mai multe motive:

1. **Evitarea blocării (deadlock):** Fără thread-uri, un nod ar aștepta să primească un mesaj și nu ar putea niciodată să trimită unul, creând un deadlock în sistem.
2. **Procesare paralelă:** Fiecare nod trebuie să realizeze simultan două operațiuni:
   - Să asculte și să proceseze mesajele primite
   - Să trimită mesaje către următorul nod
3. **Modelul Server-Client:** Fiecare nod funcționează atât ca server (ascultând conexiuni) cât și ca client (inițiind conexiuni).
4. **Asincronitate:** Comunicarea în rețea este inerent asincronă, iar thread-urile permit gestionarea eficientă a acestei asincronii.

2.4.2 Implementarea thread-urilor

Thread-urile sunt implementate prin crearea unei noi instanțe de thread și pornirea acesteia cu o expresie lambda:

```
new Thread(() -> listenForMessages()).start();
```

Această linie din metoda `start()` creează un thread separat care rulează metoda `listenForMessages()`. În acest fel:

- Thread-ul principal poate continua execuția și poate returna controlul apelantului
- Thread-ul secundar rulează continuu, ascultând pentru mesaje
- Ambele thread-uri pot opera simultan, permițând nodului să funcționeze atât ca server, cât și ca client

2.4.3 Diagrama conceptuală a thread-urilor

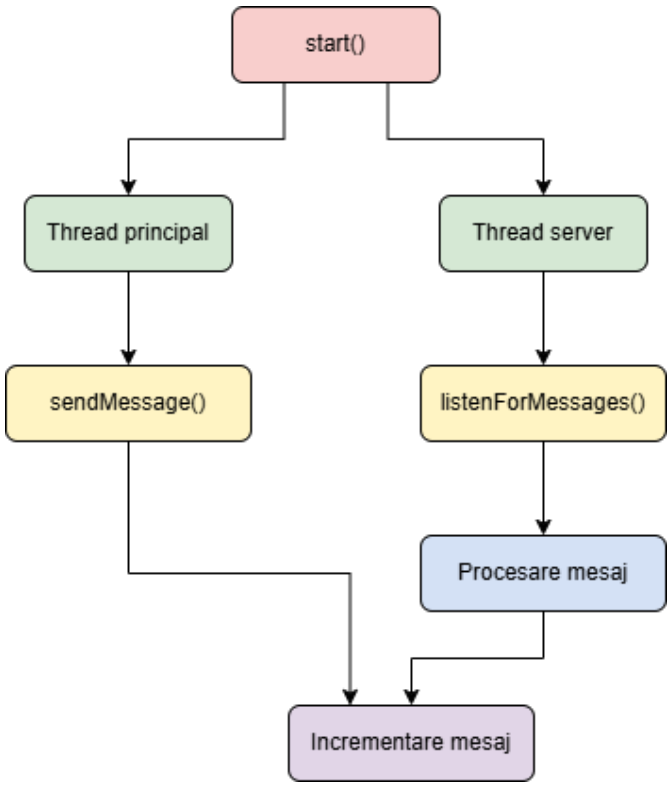

Figura 3: Topologie fluxului de execuție cu thread-uri

2.5 Fluxul comunicării

2.5.1 Inițierea comunicării

Comunicarea începe cu Nodul 1 (inițiatorul):

1. Nodul 1 trimite valoarea 1 către Nodul 2
2. Mesajul are formatul `NOD_1_TO_NOD_2_VALUE_1`

2.5.2 Procesarea mesajelor

Atunci când un nod primește un mesaj:

1. Extrage valoarea din mesaj
2. Verifică dacă valoarea este mai mare sau egală cu 100
3. Dacă da, oprește comunicarea
4. Dacă nu, incrementează valoarea cu 1
5. Construiește un nou mesaj cu valoarea incrementată
6. Trimite mesajul către următorul nod din inel

2.5.3 Desfășurarea comunicării

Fluxul complet de comunicare urmează acest model ciclic:

1. Nodul 1 trimite valoarea 1 → Nodul 2
2. Nodul 2 primește valoarea 1, o incrementează la 2, și trimite 2 → Nodul 3
3. Nodul 3 primește valoarea 2, o incrementează la 3, și trimite 3 → Nodul 1
4. Nodul 1 primește valoarea 3, o incrementează la 4, și trimite 4 → Nodul 2
5. ...și așa mai departe
6. Când un nod primește valoarea 99, o incrementează la 100 și comunică terminarea procesului

2.5.4 Terminarea comunicării

Comunicarea se încheie atunci când un nod primește o valoare mai mare sau egală cu 100:

```
if (receivedValue >= FINAL_VALUE) {
System.out.println("Nod " + nodeId + ": Valoarea finala " + FINAL_VALUE
   +
" a fost atinsa. Oprirea comunicarii.");
break;
}
```

La acest punct, nodul respectiv întrerupe bucla de ascultare și își încheie execuția.

3 Justificarea alegerii limbajului de programare

Pentru implementarea topologiei de comunicare în inel, s-a ales limbajul de programare Java.

Avantajele Java pentru implementarea rețelelor

1. **Independența de platformă**
   - Java urmează principiul "Write Once, Run Anywhere"
   - Codul poate rula pe orice sistem care are instalată Java Virtual Machine (JVM)
   - Facilitează testarea în medii heterogene fără modificări în cod

- Ideală pentru implementări distribuite care pot rula pe diverse sisteme de operare

2. **Suport nativ pentru rețelistică**
   - API-ul standard Java oferă pachete comprehensive pentru programarea în rețea:
     - `java.net`: Socket-uri, ServerSocket, URL, URLConnection
     - `java.io`: Stream-uri pentru transferul eficient de date
   - Implementarea socket-urilor TCP și UDP este simplă și directă
3. **Suport robust pentru concurență și thread-uri**
   - Java oferă suport nativ pentru thread-uri și sincronizare:
     - Clasa `Thread` și interfața `Runnable`
     - Cuvinte cheie `synchronized` pentru controlul accesului
     - API `java.util.concurrent` pentru structuri de date și primitive de sincronizare avansate
   - Facilitează implementarea comunicării asincrone în aplicații de rețea
4. **Gestionarea automată a memoriei**
   - Garbage collector-ul Java elimină necesitatea gestionării manuale a memoriei
   - Reduce riscul de memory leaks în aplicații de rețea cu rulare îndelungată
   - Utilizarea blocurilor `try-with-resources` asigură eliberarea resurselor (socket-uri, stream-uri)
   - Simplificarea codului și reducerea erorilor în gestionarea resurselor
5. **Orientarea pe obiecte**
   - Paradigma OOP a Java facilitează modelarea clară a nodurilor din topologie
   - Permite reutilizarea codului - aceeași clasă `RingNode` este instanțiată pentru toate nodurile
   - Încapsularea comportamentului și stării în obiecte face codul mai ușor de întreținut
   - Ușurează implementarea unei arhitecturi multi-thread

4 Analiza traficului în Wireshark

Wireshark este un analizor de protocol de rețea care permite inspectarea detaliată a pachetelor care circulă în rețea. În cazul implementării noastre, Wireshark ne oferă posibilitatea de a observa și analiza comunicarea TCP dintre nodurile din inel.

4.1 Captură și analiză de pachete TCP în Wireshark

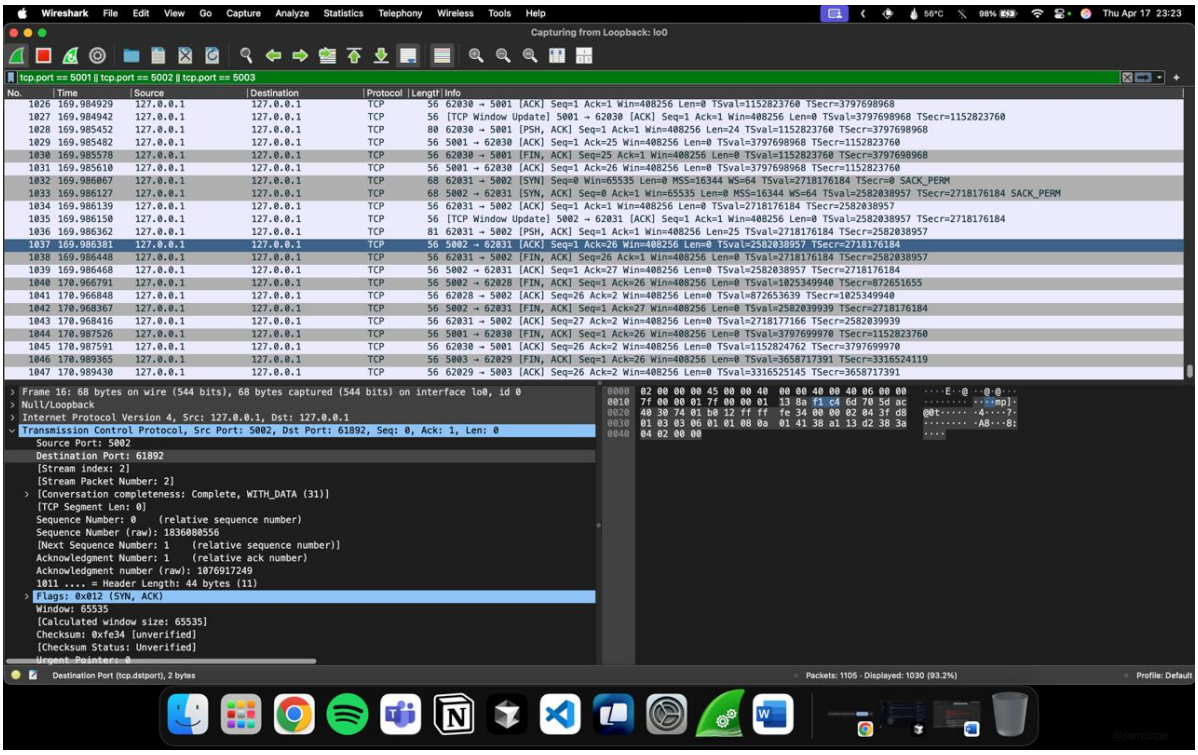

Figura 4: Captură și analiză de pachete TCP în Wireshark

Explicații:

- Filtrul `tcp.port == 5001 || tcp.port == 5002 || tcp.port == 5003` este utilizat pentru a izola traficul pe porturile folosite de cele trei noduri
- Mai multe pachete TCP sunt afișate în lista de pachete, demonstrând comunicarea circulară
- Panoul inferior arată detaliile unui pachet TCP selectat (transmisie de la portul 5002 la portul 61892)
- Flag-urile TCP (SYN, ACK) sunt evidențiate, arătând mecanismul de handshake al TCP
- Numerele de secvență și confirmare sunt vizibile, demonstrând mecanismul de fiabilitate al TCP
- Această captură confirmă că modelul de comunicare în inel funcționează conform proiectării, fiecare nod incrementând și transmițând valorile

4.2 Analiza antetului TCP

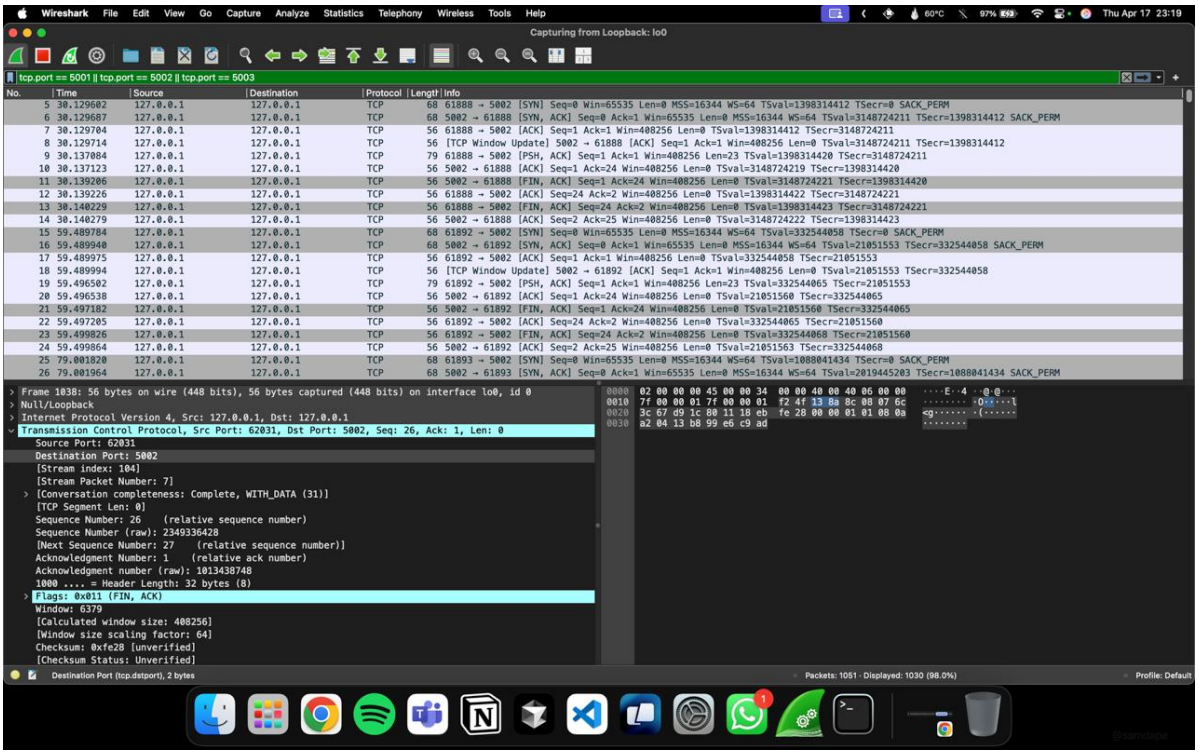

Figura 5: Wireshark - Detalii ale antetului TCP pentru un pachet de date

Explicații:

- Arată un pachet diferit cu transmisie de la portul 62031 la portul 5002
- Secțiunea “Transmission Control Protocol” detaliază componentele antetului TCP
- Flag-ul “FIN, ACK” (0x011) este evidențiat, indicând finalizarea unei conexiuni
- Detalii precum dimensiunea ferestrei (6379), numărul de secvență (26) și numărul de confirmare (1) sunt vizibile
- Această imagine demonstrează eficient utilizarea TCP pentru comunicarea fiabilă între noduri

4.3 Rularea Aplicației în cele 3 moduri

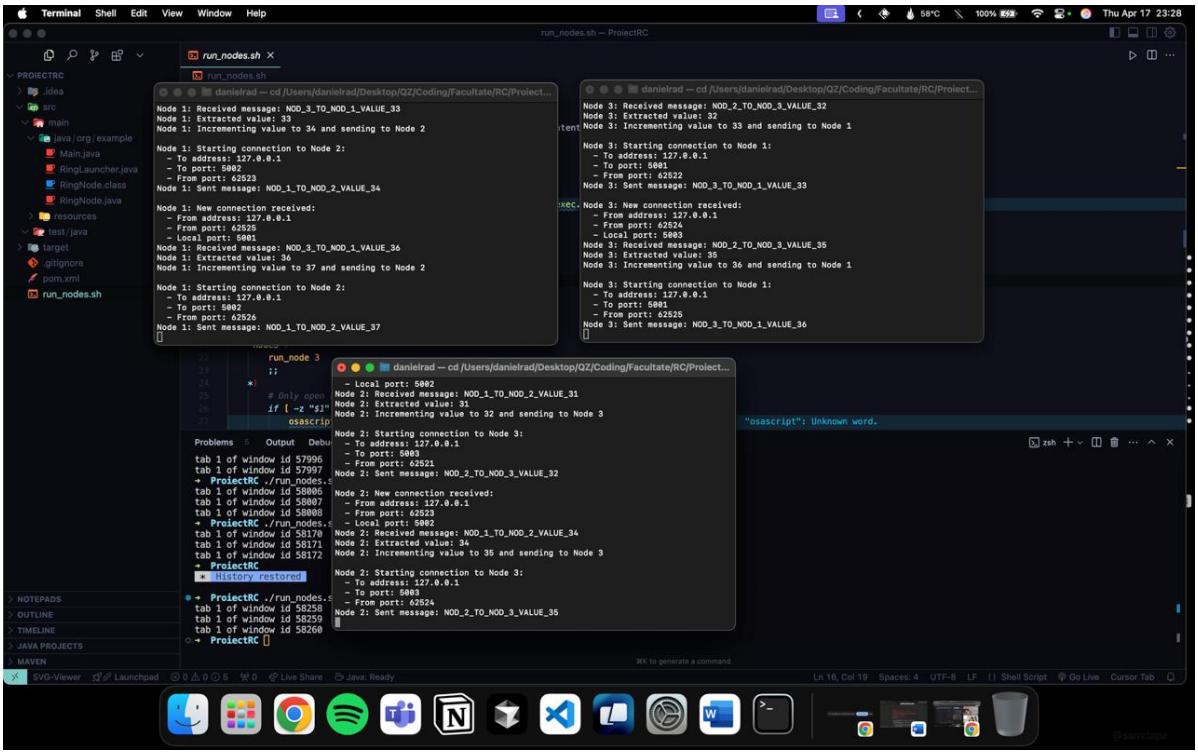

Figura 6: Rularea aplicației în cele 3 moduri

Imaginea de mai sus arată aplicația în funcțiune cu cele trei noduri rulând în terminale separate:

- Fiecare fereastră reprezintă un nod din inel (Nodul 1, Nodul 2, Nodul 3)
- Se poate observa clar fluxul de mesaje între noduri, cu formatul "NOD_X_TO_NOD_Y_VALUE_Z"
- Valorile incrementate (31, 32, 33, 34, 35, 36, 37) sunt vizibile, demonstrând procesul de incrementare
- Detalii despre conexiuni sunt afișate, inclusiv adresele IP, porturile sursă și destinație
- Această imagine este o dovadă excelentă a funcționării corecte a implementării, arătând comunicarea în inel și procesul de incrementare așa cum a fost specificat în cerințe

5 Bibliografie

Wireshark User's Guide. https://www.wireshark.org/docs/

Oracle Java Documentation - Java Networking. https://docs.oracle.com/javase/tutorial/networking/

GeeksforGeeks - Socket Programming in Java. https://www.geeksforgeeks.org/socket-programming-in-java/

Cloudflare - What is TCP/IP?. https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/

Cloudflare - TCP Handshake. https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

6 Anexă: Codul sursă complet

6.1 RingNode.java

```
package org.example;

import java.io.*;
import java.net.*;

/**
 * RingNode - Implementare pentru un nod in topologia de comunicare  n
    inel
 * Fiecare nod asculta pe un port si primeste un mesaj, apoi
    incrementeaz  valoarea
 * trimite valoarea incrementata la nodul urmator din inel
 */
public class RingNode {
    private int nodeId;           // ID-ul nodului
    private int listenPort;       // Portul pe care asculta nodul
    private String nextNodeIP;    // IP-ul nodului urmator
    private int nextNodePort;     // Portul nodului urmator
    private boolean isInitiator;  // verifica daca este nod initiator
    private static final int FINAL_VALUE = 100; // Valoarea finala
       pentru oprire

    public RingNode(int nodeId, int listenPort, String nextNodeIP, int
       nextNodePort, boolean isInitiator) {
        this.nodeId = nodeId;
        this.listenPort = listenPort;
        this.nextNodeIP = nextNodeIP;
        this.nextNodePort = nextNodePort;
        this.isInitiator = isInitiator;
    }

    /**
     * Metoda principala de pornire a nodului
     */
    public void start() {
        System.out.println("Nod " + nodeId + ": Pornire...");
        System.out.println("Nod " + nodeId + ": Ascult pe portul " +
           listenPort);
        System.out.println("Nod " + nodeId + ": Conectez la " +
           nextNodeIP + ":" + nextNodePort);

        if (isInitiator) {
            try {
                Thread.sleep(2000);
                System.out.println("Nod " + nodeId + ": Initez prima
                   conexiune catre Nod 2");
                sendMessage(1);
                System.out.println("Nod " + nodeId + ": Am initiat
                   comunicarea cu valoarea 1");
            } catch (InterruptedException e) {
                e.printStackTrace();
```

```
            }
        }

        new Thread(() -> listenForMessages()).start();
    }

    /**
     * Metoda de ascultare pentru mesaje primite de la nodul anterior
     */
    private void listenForMessages() {
        try (ServerSocket serverSocket = new ServerSocket(listenPort)) {
            System.out.println("Nod " + nodeId + ": Server pornit pe
               portul " + listenPort);

            while (true) {
                try (Socket clientSocket = serverSocket.accept();
                     BufferedReader in = new BufferedReader(new
                        InputStreamReader(clientSocket.getInputStream())
                        )) {

                    System.out.println("\nNod " + nodeId + ": Am primit
                       o noua conexiune:");
                    System.out.println("  - De la adresa: " +
                       clientSocket.getInetAddress().getHostAddress());
                    System.out.println("  - De la portul: " +
                       clientSocket.getPort());
                    System.out.println("  - Pe portul local: " +
                       clientSocket.getLocalPort());

                    String messageStr = in.readLine();
                    System.out.println("Nod " + nodeId + ": Am primit
                       mesajul: " + messageStr);

                    int receivedValue = extractValueFromMessage(
                       messageStr);
                    System.out.println("Nod " + nodeId + ": Valoarea
                       extrasa: " + receivedValue);

                    if (receivedValue >= FINAL_VALUE) {
                        System.out.println("Nod " + nodeId + ": Valoarea
                            finala " + FINAL_VALUE + " a fost atinsa.
                           Oprirea comunicarii.");
                        break;
                    }

                    int newValue = receivedValue + 1;
                    System.out.println("Nod " + nodeId + ": Incrementez
                       valoarea la " + newValue +
                                       " si o trimit catre Nod " + ((
                                          nodeId % 3) + 1));
                    sendMessage(newValue);

                    Thread.sleep(1000);
                } catch (Exception e) {
```

```
                    System.err.println("Nod " + nodeId + ": Eroare la
                       procesarea conexiunii: " + e.getMessage());
                    e.printStackTrace();
                }
            }
        } catch (IOException e) {
            System.err.println("Nod " + nodeId + ": Eroare la pornirea
               serverului: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private int extractValueFromMessage(String message) {
        // Format mesaj: NOD_X_TO_NOD_Y_VALUE_Z
        // Extrage Z din mesaj
        String[] parts = message.split("_");
        return Integer.parseInt(parts[parts.length - 1]);
    }

    /**
     * Trimite un mesaj cu o valoare spre nodul urmator din inel
     *
     * @param value Valoarea de trimis
     */
    private void sendMessage(int value) {
        try (Socket socket = new Socket(nextNodeIP, nextNodePort);
             PrintWriter out = new PrintWriter(socket.getOutputStream(),
                 true)) {

            System.out.println("\nNod " + nodeId + ": Initez conexiune
               catre Nod " + ((nodeId % 3) + 1) + ":");
            System.out.println("  - La adresa: " + socket.getInetAddress
               ().getHostAddress());
            System.out.println("  - La portul: " + socket.getPort());
            System.out.println("  - Din portul local: " + socket.
               getLocalPort());

            String message = "NOD_" + nodeId + "_TO_NOD_" + ((nodeId %
               3) + 1) + "_VALUE_" + value;
            out.println(message);
            System.out.println("Nod " + nodeId + ": Am trimis mesajul: "
                + message);
        } catch (IOException e) {
            System.err.println("Nod " + nodeId + ": Eroare la trimiterea
                mesajului: " + e.getMessage());
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {

        // Nod 1 : asculta pe portul 5001  i  trimite catre Nod 2 pe
           portul 5002
        RingNode node1 = new RingNode(1, 5001, "127.0.0.1", 5002, true);
```

```
        // Nod 2: asculta pe portul 5002  i  trimite catre Nod 3 pe
           portul 5003
        RingNode node2 = new RingNode(2, 5002, "127.0.0.1", 5003, false)
           ;

        // Nod 3: asculta pe portul 5003 si trimite inapoi catre Nod 1
           pe portul 5001, se completeaza inelul astfel
        RingNode node3 = new RingNode(3, 5003, "127.0.0.1", 5001, false)
           ;

        // Pornim toate nodurile
        node1.start();
        node2.start();
        node3.start();

        System.out.println("Toate nodurile au fost pornite. Comunicarea
            n  inel a  nceput .");
    }
}
```

Listing 1: Implementarea clasei RingNode

6.2 RingLauncher.java

```
package org.example;

/**
 * Clasa RingLauncher permite pornirea individuala a fiecarui nod
 * Se utilizeaza aceasta clasa daca se doreste o pornire individuala a
    nodurilor, pentru a vedea mai clar comunicarea dintre noduri
 */
public class RingLauncher {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Utilizare: java RingLauncher <nodeNumber
               >");
            System.out.println("nodeNumber: 1, 2 sau 3 pentru a porni
               nodul corespunzator");
            return;
        }

        int nodeNumber = Integer.parseInt(args[0]);

        switch (nodeNumber) {
            case 1:
                // Pornire Nod 1
                RingNode node1 = new RingNode(1, 5001, "127.0.0.1",
                   5002, true);
                node1.start();
                System.out.println("Node 1 started as initiator.");
                break;
            case 2:
```

```
                // Pornire Nod 2
                RingNode node2 = new RingNode(2, 5002, "127.0.0.1",
                    5003, false);
                node2.start();
                System.out.println("Node 2 started.");
                break;
            case 3:
                // Pornire Nod 3
                RingNode node3 = new RingNode(3, 5003, "127.0.0.1",
                    5001, false);
                node3.start();
                System.out.println("Node 3 started.");
                break;
            default:
                System.out.println("Numar de nod invalid. Folositi 1, 2
                    sau 3.");
        }
    }
}
```

Listing 2: Implementarea clasei RingLauncher