**LUCRARE DE LABORATOR**

Nivelul Aplicație: Programare în rețea utilizând socket-uri

1. Obiective

*Precondiție: Utilizați un mediu de lucru funcțional pentru limbajul de programare preferat (Java, C#, Python, C/C++, etc.)*

La sfârșitul activității, studenții vor putea să scrie software pentru aplicații de socket-uri și să depaneze aplicații de rețea folosind Wireshark.

2. Consideraţii teoretice

Lucrările practice curente se concentrează pe nivelurile de Transport și de Aplicație ale stivei de protocoale ISO/OSI. (Figura 8.1).

**Diagram

Description automatically generated**

**Figura 8.1** *Modelele de stivă de rețea și denumirea PDU (Protocol Data Unit - Unitate de Date la nivelul Protocolului) în fiecare nivel. Săgețile indică nivelurile adresate în activitatea curentă*

Această activitate practică abordează partea de programare din ingineria software și a comunicațiilor prin utilizarea socket-urilor de rețea într-un mediu desktop. Programarea socket-urilor este disponibilă în orice limbaj de programare de nivel înalt, iar socket-urile transmit informații la Nivelul Aplicației. Socket-urile sunt utilizate în diferite tipuri de aplicații, precum: Client-Server, sisteme peer-2-peer, comunicare inter-proces (pe aceeași mașină).

Socket-urile de rețea pot fi construite pentru a folosi atât adrese IPv4, cât și IPv6. Un socket este combinația unei adrese IP și a unui număr de port și se utilizează într-o aplicație de rețea. O aplicație de rețea furnizează conectivitate între diferite dispozitive de rețea. Nu este posibil să legați un socket la un port care este deja folosit de către o altă aplicație, totuși același port poate fi folosit concurent de protocoalele de nivel transport TCP și UDP. Adresele IP identifică dispozitivul de rețea, dar numărul portului identifică în mod unic fiecare aplicație în executare pe dispozitivul de rețea curent.

Operațiile pe care o aplicație le poate efectua pe un socket sunt următoarele:

* **Create** - Crearea unui obiect de tip socket
* **Bind** - Configurarea obiectului de tip socket pentru a folosi o pereche locală de adresă IP și număr de port pentru a accepta conexiuni
* **Listen** - Programarea socket-ului pentru a aștepta conexiuni de intrare
* **Accept** - Acceptarea conexiunii de intrare
* **Connect** - Această operație este folosită de un client care dorește să se conecteze la un server
* **Send** - Folosit pentru a trimite date prin socket către destinația remote
* **Receive** - Folosit pentru a primi date trimise dintr-o locație remote
* **Close** - Închiderea conexiunii între cele două socket-uri

**2.1. Lucrul cu socket-uri pe mașina locală**

* Pentru a simula o rețea pe mașina locală, întregul interval disponibil de IP de tip loopback (127.0.0.0 - 127.255.255.255) poate fi folosit. Interfața de rețea loopback este disponibilă doar pe dispozitivul gazdă local și este folosită în principal pentru diagnosticare și aplicații de rețea independente. Prin urmare, o rețea locală simulată poate folosi aceste adrese IP pentru comunicare. Pentru a testa și confirma că acest interval poate fi folosit, poate fi rulată o comandă ping de la terminalul local pentru a verifica conectivitatea la aceste adrese IP (Figura 8.2):

A screenshot of a computer

Description automatically generated with medium confidence

**Figura 8.2** *Testarea adresării loopback*

* De asemenea, este posibil să atribuiți mai multe adrese IP valide pe interfața locală, dar acest lucru trebuie făcut manual prin alocarea statică a adreselor IP interfeței. În acest caz, rularea comenzii **ipconfig** arată toate adresele IP atribuite aceleiași interfețe. Vezi un exemplu mai jos (Figura 8.3):

A picture containing text

Description automatically generated

**Figura 8.3** *Vizualizarea configurației IP - CLI*

* După ce au fost atribuite adresele IP, acum pot fi create socket-uri pentru a utiliza aceste adrese IP.

**2.2. Socket-uri TCP**

* Socket-urile TCP (Transmission Control Protocol) sunt orientate pe conexiune și reprezintă un mecanism de transmitere a datelor fiabil care permite ca datele să fie primite și procesate în aceeași ordine în care au fost transmise.
* Figura 8.4 arată o captură de trafic Wireshark pe interfața pentru capturarea traficului pe interfața de rețea loopback. Captura de ecran arată o comunicare client-server prin socket-uri folosind adresele loopback. Filtrul aplicat este **tcp.port == 1234**. Serverul este legat de adresa 127.0.0.1 și așteaptă conexiuni pe portul 1234, în timp ce clientul este legat de adresa 127.0.0.2, trimitând 14 octeți de date către server.

Table

Description automatically generated with low confidence

**Figura 8.4** *Captura Wireshark pentru comunicație prin socket-uri TCP*

* Captura de ecran evidențiază mecanismul TCP reprezentat de pachetele de confirmare (ACK). Primele trei schimburi de pachete (Figura 8.4) reprezintă strângerea de mână în trei pași (3-way handshake), necesară pentru stabilirea conexiunii pentru orice conexiune TCP (Figura 8.5), iar mai apoi este vizibil pachetul care trimite datele - încărcătura utilă. Această strângere de mână asigură că ambele gazde doresc să comunice și recunosc intenția celeilalte gazde de a comunica.

Diagram

Description automatically generated

**Figura 8.5** *TCP 3-way handshake*

* Imaginea Wireshark arată o comunicare socket care rămâne deschisă.
  + Răspundeți la următoarea întrebare în timp ce lucrați la activitatea practică: Dacă conexiunea socket-ului este închisă, care sunt indicatoarele TCP (TCP flags) care sunt setate pentru a închide conexiunea?

**2.3. Socket-uri UDP**

* În contrast cu socket-urile TCP, socket-urile UDP (User Datagram Protocol) nu sunt orientate spre conexiune și nu furnizează comunicare fiabilă. Acest lucru înseamnă că nu garantează că pachetele de rețea sunt livrate la destinație. Figura 8.6 reprezintă o captură Wireshark (din nou, capturarea traficului pe interfața de rețea loopback) a unei comunicări UDP între două gazde. Filtrul aplicat este **udp.port == 1234**. După cum se poate observa, nu există o metodă de stabilire a conexiunii și nu sunt transmise niciun fel de pachete ACK.

Text

Description automatically generated

**Figura 8.6** *Captura Wireshark pentru comunicație prin socket-uri UDP*

Comunicația prin socket-uri TCP și UDP sunt prezentate în Figura 8.7 a și b.

|  |  |
| --- | --- |
| **Diagram, schematic  Description automatically generated** |  |
| 1. Comunicație prin socket TCP | 1. Comunicație prin socket UDP |

**Figura 8.7** *Comunicare prin socket-uri*

1. **Șablon de implementare**

* Un dispozitiv de rețea poate funcționa în 3 moduri:
  + **Server:** Dispozitiv care recepționează
  + **Client:** Dispozitiv care trimite
  + **Releu (Relay):** Acționând ca un nod intermediar în comunicare și acționează atât ca dispozitiv de trimitere, cât și de recepție. Acest tip de nod de rețea poate fi întâlnit în Rețelele de Senzori Fără Fir (WSN) unde nu toate nodurile senzor sunt în raza wireless a dispozitivului colector, prin urmare unele noduri trebuie să transmită informațiile către nodul destinatar (nodul colector).
* Acest subcapitol oferă un șablon pentru implementarea nodului de comunicare releu folosind concepte OOP (șablonul este scris în pseudocod, nu într-un limbaj de programare anume). Aceasta nu este singura posibilitate de organizare a codului, studenții pot alege orice metodologie de proiectare a software-ului cu care se simt confortabili.

|  |
| --- |
| **Relay node implementation template** |
| **class** RelayNode {  **public:**  RelayNode(IPAddress, serverPortNr) {  m\_server.listen(IPAddress, serverPortNr);  m\_client.bind(IPAddress);  m\_server.onReceive() => {  **ByteArray** receivedData = m\_server.readData();  m\_client.connectToHost(m\_nextHopIpAddress, m\_nextHopPortNr);  m\_client.sendData(receivedData);  m\_client.close();  }  }  **void** setNextHopInformation(nextHopIPAddress, nextHopServerPortNr) {  m\_nextHopIpAddress = nextHopIPAddress;  m\_nextHopPortNr= nextHopServerPortNr;  }  **private:**  Server m\_server; *// server instance accepting connections*  Client m\_client; *// client instance sending data to the next hop*  IPAddress m\_nextHopIpAddress; *// next hop address used by the client instance*  **int** m\_nextHopPortNr; *// next hop port nr used by the client instance*  }  **void** main() {  RelayNode relay(127.0.0.1, 1234);  relay.setNextHopInformation(127.0.0.2, 2345)  …  *// run application event loop*  } |

1. Desfăşurarea lucrării

* Fiecare student îi va fi asignată una dintre topologiile de mai jos și scenariul de simulare trebuie implementat în software.
* În afară de constrângerile impuse de fiecare scenariu de simulare, sarcinile comune pentru fiecare implementare sunt următoarele:
  + Folosiți un limbaj de programare la alegere pentru a implementa simularea rețelei
  + Folosiți intervalul de adrese pentru buclă locală (loopback) pentru adresare: 127.0.0.0 – 127.255.255.255
  + Testați implementarea folosind Wireshark
  + Livrați implementarea (codul sursă sau link către un depozit de cod versiune online - code versioning repository)
  + Prezentați o captură Wireshark pentru a dovedi comunicarea între diferitele adrese IP
  + Inspectați raportul dintre totalul încărcăturii livrate comparativ cu traficul de nivel de aplicație relevant / raportul dintre lungimea totală a pachetului (antete și date) în comparație cu lungimea datelor trimise (folosiți statistici Wireshark sau inspecție manuală a pachetelor)
  + În funcție de simularea implementată, cercetați antetele pentru protocoalele TCP și/sau UDP. Utilizând Wireshark, identificați elementele antetului în traficul capturat.

3.1 Comunicare în Inel (Ring Comunication)

* Trei calculatoare comunică într-o singură direcție creând o buclă (Figura 8.8)
* 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'
* Sugestii de implementare:
  + Implementați o singură clasă care este instanțiată de 3 ori cu diferiți parametri de comunicare (reutilizați codul și nu-l duplicați pentru fiecare instanță)
  + Toată comunicarea folosește socket-uri TCP (opțional)

Diagram

Description automatically generated

**Figura 8.8** *Topologia rețelei pentru comunicație în inel*

3.2 Selector de noduri (Node selector)

* Există trei noduri în topologie: N1, N2, N3 (Figura 8.9)
* N1 crește o valoare de 100 de ori și după fiecare incrementare trimite valoarea fie către N2 sau către N3, noduri care sunt selectate aleatoriu pentru transmitere
* Când N2 primește o valoare întreagă care este un multiplu de 3, va trimite înapoi un pachet ACK către N1
* Când N3 primește o valoare întreagă care este un multiplu de 5, va trimite înapoi un pachet ACK către N1
* Sugestii de implementare:
  + Implementați o singură clasă pentru N2 și N3 care este instanțiată cu diferiți parametri de comunicare (reutilizați codul și nu-l duplicați pentru fiecare instanță)
  + Toată comunicarea folosește socket-uri UDP (opțional)

Diagram

Description automatically generated

**Figura 8.9** *Topologia rețelei pentru selector de noduri*

3.3 Noduri Releu (Relay Nodes)

* În topologie există patru noduri (Figura 8.10): Expeditorul și trei destinații posibile (D1, D2 și D3).
* Nodul Expeditor transmite 100 de pachete, conținând un număr întreg aleator, către una dintre cele 3 destinații posibile (D1, D2 sau D3).
* După fiecare transmitere a pachetului, numărul întreg este incrementat.

Fiecare nod poate trimite date doar către următorul hop la care este conectat, prin urmare, un pachet de la Expeditor către D3 trebuie să treacă prin D1 și D2.

Diagram

Description automatically generated

**Figura 8.10** *Topologia rețelei pentru noduri releu*

* Sugestii de implementare:
  + Datele utile care sunt transmise prin socket trebuie să conțină adresa IP destinație, astfel că datele au următorul format (Figura 8.11):

|  |  |
| --- | --- |
| **Adresa IP țintă** | **Valoare** |

**Figura 8.11** *Formatul datelor utile transmise*

* + De fiecare dată când un nod primește un pachet, acesta verifică dacă adresa IP destinație a datelor utile primite este aceeași cu adresa IP curentă a nodului. Dacă este identică, comunicarea se oprește aici, în caz contrar datele sunt trimise mai departe la următorul salt.
  + Implementați o singură clasă pentru D1, D2 și D3, care este instanțiată cu diferiți parametri de comunicare (reutilizați codul și nu-l duplicați pentru fiecare instanță).