Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică și Microelectronică

Catedra Tehnologia Informației

Laboratorul 4.

Programarea în rețea

A efectuat: studenta gr. FI-171 Ciocanu Carolina

A examinat: Buldumac Oleg

Chisinau, 2020

**Condiții:**

Aplicație Client-Server TCP

**Pentru nota 8, 9 si 10:**

• Să proiectați și elaborați propriul protocol

**Atenție:**

• Fiecare client trebuie procesat de către server într-un fir de execuție aparte

• Nu se admite aplicații simple de genul Echo Client-Server

• Aplicația poate fi GUI sau consolă

• Pentru cei ce doresc 8, 9 și 10, ca exemplu luați protocoalele HTTP, SMTP unde clientul și

serverul discută printr-un set de reguli bine definite. De exemplu, pentru o aplicație de tip chat

pentru ca clientul să fie identificat de către server acesta trebuie să transmită un mesaj de tipul:

HELLO-REQUEST după care serverul va răspunde cu HELLO-APPROVE. Mai detailat

accesați ultimele 3 referințe de la sfîrșitul acestui fișier.

**Întrebări:**

• Ce este un protocol orientat pe conexiune ?

• Ce tipuri de aplicații beneficiază în general de utilizarea protocolului TCP ?

• Cum TCP garantează că datele vor fi transmise cu succes ?

• Diferența dintre blocking si non-blocking sockets

• Diferența dintre blocking multithreaded și non-blocking single thread socket

• Cum are loc procesul TCP Three Way Handshake ?

• Numiti cele 4 apeluri de sistem necesare pentru a crea un server TCP

• Care este rolul metodei bind() ?

• Care este rolul metodei accept() ?

• Ce se întîmplă cînd apelați mai întîi connect() apoi bind() ?

• Ați avea vreodată nevoie să implementați un timeout într-un client sau server care utilizează

TCP?

• Într-o conexiune TCP, clientul sau serverul trimite mai întâi datele ?

• Care este adresa de loopback IPv4 și care este rolul ei ?

• De unde știe un sistem de operare ce aplicație este responsabilă pentru un pachet primit din

rețea ?

• Datele primite prin recv() au întotdeauna aceeași dimensiune cu datele trimise cu send() ?

• Este acceptabil să închei executia programului dacă este detectată o eroare de rețea ?

• Puteți îmbunătăți performanța aplicației prin dezactivarea algoritmului Nagle ?

• Ce instrumente listează socket-urile TCP deschise în sistemele de operare Windows și Linux ?

• Tehnici de sincronizare a firelor de execuții

**Ce este un protocol orientat pe conexiune ?**

Se utilizeaza in special pentru a asigura integritatea informatiei transportate. Un protocol orientat către conexiune (COP) este un protocol de rețea utilizat pentru a stabili o sesiune de comunicare a datelor în care dispozitivele finale utilizează protocoale preliminare pentru a stabili conexiuni end-to-end și apoi fluxul de date ulterioare este livrat în modul de transfer secvențial.

**Ce tipuri de aplicații beneficiază în general de utilizarea protocolului TCP ?**

TCP este utilizat pe scară largă de multe aplicații de internet, inclusiv World Wide Web (WWW), e-mail, Protocol de transfer de fișiere, Secure Shell, partajare de fișiere peer-to-peer și media de streaming.

**Cum TCP garantează că datele vor fi transmise cu succes ?**

Din punctul de vedere al aplicaţiei, TCP transferă un şir (stream) continuu de octeţi prin reţea. Aplicaţia nu trebuie să-şi facă griji cu segmentarea datelor în blocuri de bază sau datagrame. O face TCP prin gruparea octeţilor în fragmente TCP, care sunt transferate nivelui reţea (IP) pentru a fi transmise la destinaţie. De asemenea, însuşi TCP decide cum să segmenteze datele şi poate trimite datele mai departe într-un mod convenabil.

Aplicaţia TCP destinaţie, când transmite o confirmare (ACK) către entitatea emitentă, indică de asemenea numărul de octeţi pe care îi poate recepţiona pe langă ultimul segment TCP primit, fără să se suprîncarce sau să se confrunte cu depăşirea memoriilor tampon.

Diferența dintre blocking si non-blocking sockets?

În modul de blocare, apelurile API socket de recv, send, connect (doar TCP) și accept (numai TCP) vor bloca la nesfârșit până când acțiunea solicitată a fost efectuată. În modul care nu blochează, aceste funcții revin imediat. select se va bloca până când priza este gata. selectează un parametru de timp care controlează timpul de așteptare pentru finalizarea acțiunii sau o eroare returnată.

**Diferența dintre blocking multithreaded și non-blocking single thread socket**

Am lucrat la o aplicație de Client Server unde Server va merge la service (sendto + recepție) „x” număr de Clienți la un moment dat. În acest scop, am creat numărul „x” de thread-uri pe server, astfel încât fiecare thread să fie dedicat unui singur client. În fiecare fir există o priză specifică doar pentru clientul său. Mă gândeam să fac ca aceste socketuri să nu se blocheze, dar acum cred că utilizarea unei prize pentru blocare în fiecare fir este o idee mai bună. Blocking Socket așteaptă continuu să primească date și ori de câte ori este nevoie să trimiteți ceva, este apelat sendto (). Utilizarea blocării Socket într-o astfel de situație este o abordare bună sau ar trebui să folosesc socluri care nu blochează? :(

**Cum are loc procesul TCP Three Way Handshake ?**

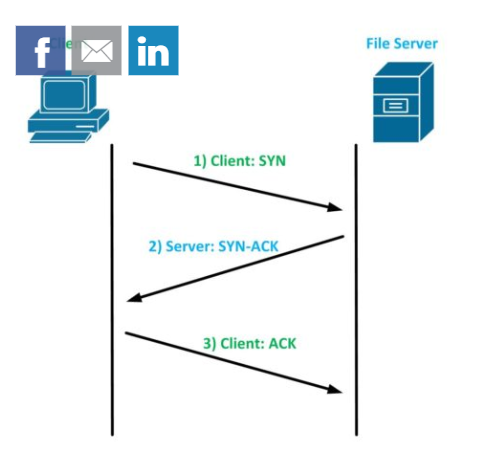
In momentul in care un server trebuie sa comunice cu un client, cei 2 vor forma o conexiune intre ei cu ajutorul 3-Way Handshake-ului, care functioneaza in felul urmator:

In prima faza, Clientul (cel care incepe conexiunea) ii va trimite serverului:

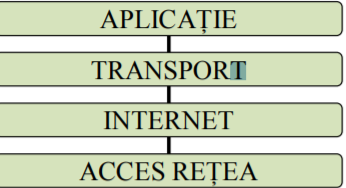
⦁ Un mesaj de sincronizare (SYN) sau de incepere a conexiunii

⦁ Serverul va raspunde cu o confirmare (SYN-ACK)

⦁ Clientul va raspunde si el cu un mesaj de confirmare (ACK)



**Numiti cele 4 apeluri de sistem necesare pentru a crea un server TCP**



**Care este rolul metodei bind() ?**

bind() pentru stabilirea adresei

Functia are ca efect atribuirea adresei specificate ˆın parametrul addr socket-ului identificat prin identificatorul sd. Aceast˘a funct¸ie se apeleaz˘a ˆın mod normal dintr-un proces server, pentru a preg˘ati un socket stream de asteptare sau un socket dgram pe care se a¸steapt˘a cereri de la clienti.

**Care este rolul metodei accept() ?**

Preluarea efectiv˘a a unui client conectat se face prin apelarea unei funct¸ii sistem numit˘a accept(). La apelul funct¸iei accept(), sistemul de operare execut˘a urm˘atoarele: • a¸steapt˘a cererea de conectare a unui client ¸si deschide conexiunea c˘atre acesta; • creaz˘a un nou socket, numit socket de conexiune, care reprezint˘a cap˘atul dinspre server al conexiunii proasp˘at deschise; • returneaz˘a apelantului (procesului server) identificatorul socket-ului de conexiune creat. Dup˘a un apel accept(), socket-ul de a¸steptare poate fi utilizat pentru a a¸stepta noi client¸i, iar socket-ul de conexiune nou creat se utilizeaz˘a pentru a comunica efectiv cu acel client.

**Ce se întîmplă cînd apelați mai întîi connect() apoi bind() ?**

Functia bind() nu poate fi apelat˘a pentru un socket de conexiune creat prin funct¸ia accept() ¸si nici pentru un socket asupra c˘aruia s-a apelat ˆın prealabil vreuna dintre funct¸iile connect().

Daca te conectezi cu connect() la o masina aflata la distanta si nu te intereseaza ce port local ai (cum este in cazul telnet), poti sa apelezi simplu connect(), ea va verifica daca socket-ul este neatasat, si ii va atasa cu bind() un port local neutilizat.

**Ați avea vreodată nevoie să implementați un timeout într-un client sau server care utilizează**

**TCP?**

In caz de retineri de lunga durata, v-a trebui sa afisam un mesaj de asteptare sau de esuare.

**Într-o conexiune TCP, clientul sau serverul trimite mai întâi datele ?**

Clientul

**Care este adresa de loopback IPv4 și care este rolul ei ?**

Mecanismul local de loopback poate fi utilizat pentru a rula un serviciu de rețea pe o gazdă, fără a fi nevoie de o interfață de rețea fizică sau fără ca serviciul să fie accesibil din rețelele la care computerul poate fi conectat. De exemplu, un site web instalat local poate fi accesat dintr-un browser Web de URL-ul http: // localhost pentru a-și afișa pagina de pornire.

Adresa de loopback IPv4 127.0.0.1 și la adresa loopback IPv6

**De unde știe un sistem de operare ce aplicație este responsabilă pentru un pachet primit din**

**rețea ?**

Prin intermediul setarilor firewall, Prin folosirea unui firewall avem posibilitatea de a seta exceptii sau de a bloca traficul de date al anumitor aplicatii in functie de caz.

**Datele primite prin recv() au întotdeauna aceeași dimensiune cu datele trimise cu send() ?**

Sistemul garanteaza sosirea la destinatie a tuturor octetilor trimisi(sau instiintarea receptorului, printr-un cod de eroare, asupra caaderii cone-xiunii), in ordinea in care au fost trimisi. Nu se pastreaza insa demarcarea intre secventele de octeti trimise in apeluri send() distincte

**Puteți îmbunătăți performanța aplicației prin dezactivarea algoritmului Nagle ?**

Algoritmul lui Nagle procesează datele de pe partea de trimitere a aplicațiilor TCP printr-o metodă numită nagling . Detectează mesaje de dimensiuni mici și le acumulează în pachete TCP mai mari înainte de a trimite date prin fir, evitând astfel generarea unui număr inutil de mari de pachete mici. Specificația tehnică pentru algoritmul lui Nagle a fost publicată în 1984 ca RFC 896. Deciziile privind multitudinea de date care se acumulează și durata de așteptare între acestea sunt esențiale pentru performanța sa globală.

Nagling poate utiliza mai eficient lățimea de bandă a unei conexiuni de rețea în detrimentul adăugării întârzierilor.

Nagle algoritmul este utilizabil doar cu TCP. Alte protocoale, inclusiv UDP , nu o acceptă.

Aplicațiile TCP care au nevoie de răspuns rapid la rețea, cum ar fi apelurile telefonice prin Internet sau jocurile de tip shooter pentru prima persoană, ar putea să nu funcționeze bine când Nagle este activat. Întârzierile cauzate în timp ce algoritmul necesită un timp suplimentar pentru a asambla mai multe bucăți de date mai mici poate provoca o vizibilitate vizibilă pe ecran sau într-un flux audio digital. Aceste aplicații dezactivează de obicei Nagle.

**Ce instrumente listează socket-urile TCP deschise în sistemele de operare Windows și Linux ?**

Condition objects with producer and consumer

Producer and Consumer with Queue

Semaphore objects & thread pool

**Tehnici de sincronizare a firelor de execuții**

1. Deadlock
2. Starvation
3. Priority Inversion
4. Busy Waiting

Următoarele sunt cele patru categorii care gestionează mecanismul de sincronizare:

Metode de blocare

Blocare Construct

convertoare de semnalizare

Fără sincronizare de blocare