**Kreiranje chat aplikacije pomocu socket programiranja**

**Ključni koraci u ovom primeru:**

1. **Inicijalizacija Winsock-a** pomoću WSAStartup.
2. **Kreiranje socket-a** pomoću socket.
3. **Konfiguracija serverove adrese** (IP adresa i port).
4. **Bindovanje socket-a** na određenu adresu i port pomoću bind.
5. **Postavljanje socket-a u pasivni mod** pomoću listen.
6. **Prihvatanje dolazne konekcije** pomoću accept.
7. **Komunikacija sa klijentom** putem novog socket-a (clientSocket).

**Kreiranje servera**

* WinSock2.h:

Ova biblioteka je osnovna za Windows Sockets (WinSock) API, koji omogućava programerima da razvijaju mrežne aplikacije u Windows operativnom sistemu.

WinSock2.h sadrži deklaracije funkcija, struktura i definicija konstanti potrebnih za mrežno programiranje kao što su kreiranje socket-a, upravljanje konekcijama, slanje i primanje podataka itd.

* WS2tcpip.h:

Ova biblioteka proširuje funkcionalnost WinSock-a i uključuje dodatne definicije za napredne mrežne funkcije, posebno vezane za TCP/IP protokole.

WS2tcpip.h uključuje definicije struktura poput sockaddr\_in (za IPv4 adrese), sockaddr\_in6 (za IPv6 adrese), kao i funkcije za konverziju IP adresa i portova.

* tchar.h:

Ova biblioteka pruža podršku za Unicode i karaktere sa širokim opsegom (wide characters) u Windows okruženju.

Definiše makroe i funkcije koji omogućavaju programerima da kreiraju višepločne (multibyte) ili Unicode kompatibilne aplikacije, uključujući makroe kao što su \_TCHAR, \_TEXT, TEXT koji omogućavaju prelazak između višepločnog i Unicode moda.

* WSADATA data; - Deklaracija promenljive data tipa WSADATA koja će sadržati informacije o WinSock implementaciji.
* WSAStartup(MAKEWORD(2, 2), &data) == 0

-Poziv funkcije WSAStartup koja inicijalizuje WinSock biblioteku.

-MAKEWORD(2, 2) specifikuje verziju WinSock-a koju želimo da koristimo (u ovom slučaju verziju 2.2).

-&data je pokazivač na WSADATA strukturu gde će biti smeštene informacije o WinSock implementaciji

* WSACleanup();

- WSACleanup zatvara WinSock biblioteku i oslobađa sve resurse koje je alocirala tokom rada aplikacije.

-Zatvara sve otvorene sockete koji su kreirani tokom rada aplikacije.

* SOCKET listenSocket = socket(AF\_INET, SOCK\_STREAM, 0);

-domain:Za komunikaciju procesa na razlicitim hostovima povezani sa IPV4, koristi se AF\_INET i AF\_I NET 6 za procese povezani preko IPV6.

-type: tip komunikacije

SOCK\_STREAM: TCP(reliable, connection-oriented)

SOCK\_DGRAM: UDP(unreliable, connectionless)

-protocol: Vrednost protokola za Internet Protocol(IP), sto je 0.

This is the same number that appears on the protocol field in the IP header of a packet.(man protocols for more details)

* serveraddr.sin\_port = htons(12345);

-Funkcija htons služi za konverziju broja porta iz "host byte order" u "network byte order"

 **Host byte order**: Redosled byte-ova koji je specifičan za arhitekturu računara na kojem se program izvršava.

 **Network byte order**: Standardizovani redosled byte-ova koji se koristi u mrežnim komunikacijama, obično je to Big-endian redosled (najznačajniji byte prvi).

-Kada se kreira socket i binduje na određeni port, bitno je da se port pravilno predstavi u network byte order-u kako bi mrežni protokoli mogli pravilno tumačiti tu informaciju. Korišćenjem htons funkcije, obezbeđujemo da port bude pravilno reprezentovan u formatu koji je očekivan za mrežnu komunikaciju.

* if(InetPton(AF\_INET, \_T("0.0.0.0"), &serveraddr.sin\_addr) != 1)

int InetPton(

int Family,

PCTSTR pszAddrString,

PVOID pAddrBuf

);

-Family: Specifikacija IP adrese, može biti AF\_INET za IPv4 ili AF\_INET6 za IPv6.

-pszAddrString: Tekstualni format IP adrese koju želimo da konvertujemo (npr. "0.0.0.0" za IPv4).

-pAddrBuf: Pokazivač na strukturu (struct in\_addr za IPv4 ili struct in6\_addr za IPv6) u koju će se smestiti rezultujući binarni format adrese.

-Povratna vrednost

1 ako je konverzija uspešna.

0 ako pszAddrString nije validna IP adresa.

-1 ako se dogodila greška.

* if (bind(listenSocket, reinterpret\_cast<sockaddr\*>(&serveraddr), sizeof(serveraddr)) == SOCKET\_ERROR)

-**bind** funkcija se koristi za povezivanje (bindovanje) socket-a sa specifičnom mrežnom adresom i portom.

-listenSocket: Socket koji se binduje.

-reinterpret\_cast<sockaddr\*>(&serveraddr): Pointer na strukturu serveraddr, koja sadrži informacije o mrežnoj adresi i portu na koji se socket binduje.

-sizeof(serveraddr): Veličina strukture serveraddr.

-U slučaju da bind funkcija vrati SOCKET\_ERROR, obično to znači da povezivanje nije uspelo i da je potrebno rukovati greškom.

* if (listen(listenSocket, SOMAXCONN) == SOCKET\_ERROR) {}

int listen(SOCKET s, int backlog);

 **s**: Socket deskriptor koji treba staviti u pasivni mod (socket mora biti kreiran i bindovan pre poziva listen funkcije).

 **backlog**: Maksimalni broj nepovezanih konekcija koje sistem može da drži u redu dok čeka da budu prihvaćene. SOMAXCONN je konstantna vrednost koja označava maksimalni broj konekcija koje sistem može podržati.

* SOCKET clientSocket = accept(listenSocket, nullptr, nullptr);

SOCKET accept(SOCKET s, struct sockaddr\* addr, int\* addrlen);

 **s**: Socket deskriptor koji sluša dolazne konekcije (u ovom slučaju listenSocket).

 **addr**: Pokazivač na sockaddr strukturu koja će biti popunjena sa informacijama o adresi klijenta koji se povezuje. Može biti nullptr ako vas ne zanimaju informacije o klijentu.

 **addrlen**: Pokazivač na veličinu addr strukture. Može biti nullptr ako addr nije korišćen.

 Vraća novokreirani socket koji se može koristiti za komunikaciju sa klijentom.

 Vraća INVALID\_SOCKET ako je došlo do greške.

Ako je accept uspešan, novi socket (clientSocket) se može koristiti za slanje i primanje podataka od klijenta. U slučaju greške, vraća se INVALID\_SOCKET i prikladno se rukuje greškom.

**Kreiranje klijenta**

* void InteractWithClient(SOCKET clientSocket, vector<SOCKET>& clients) {]
  + Funkcija InteractWithClient se koristi za upravljanje komunikacijom između servera i klijenta u kontekstu više klijenata koji su povezani na server. Ova funkcija prima poruke od klijenta i šalje ih svim ostalim povezanim klijentima, osim onome od kojeg je poruka stigla.

 **clientSocket**: Socket deskriptor povezan sa trenutnim klijentom.

 **clients**: Vektor koji sadrži sokete svih povezanih klijenata.

* int byterecvd = recv(clientSocket, buffer, sizeof(buffer), 0);

 recv funkcija prima podatke od klijenta. Ako recv vrati 0 ili negativnu vrednost (byterecvd <= 0), to znači da je klijent prekinuo vezu ili je došlo do greške, te petlja se prekida.

 Primljeni podaci se smeštaju u buffer.

* auto it = find(clients.begin(), clients.end(), clientSocket);

if (it != clients.end())

{ clients.erase(it); }

closesocket(clientSocket);

* + Uklanjanje klijenta iz vektora osigurava da lista sadrži samo trenutno povezane klijente. Kada klijent prekine vezu, njegov socket više nije validan i ne bi trebalo da bude deo liste aktivnih klijenata.
  + Nakon uklanjanja socket-a iz vektora, poziva se closesocket kako bi se oslobodili resursi povezani sa tim socket-om
* while (1) {

//accept

SOCKET clientSocket = accept(listenSocket, nullptr, nullptr);

if (clientSocket == INVALID\_SOCKET) {

cout << "Invalid client socket\n";

}

clients.push\_back(clientSocket);

thread t1(InteractWithClient, clientSocket, ref(clients));

}

* Ova petlja služi za kontinuirano prihvatanje novih klijenata i pokretanje zasebne niti za svakog povezanog klijenta kako bi se omogućila simultana komunikacija sa više klijenata.

 **accept**: Čeka na dolaznu konekciju na listenSocket. Kada se klijent poveže, kreira se novi socket za komunikaciju sa tim klijentom (clientSocket).

 Ako je vrednost clientSocket jednaka INVALID\_SOCKET, to znači da je došlo do greške prilikom prihvatanja konekcije, i ispisuje se odgovarajuća poruka.