Mrežno programiranje

U mrežno programiranje spada razvoj programa koji će da komuniciraju sa drugim programima. Za početak radićemo primer mrežnog programiranja gde su Client i Server konzolne aplikacije.

Protokol komunikacije

Protokol je pravilo koje se definiše za komunikaciju između više komponenti. U našem primeru to je TCP protokol komunikacije između klijentske i serverske aplikacije.
TCP (eng. Transmission Control Protocol) protokol omogućava pouzdano, organizovano, slanje, proveru bajtova između pokrenutih aplikacija koji komuniciraju preko IP mreže. Mnoge internet aplikacije oslanjaju se na TCP protokol (npr. WWW, email, prenos podataka, ...).
UDP (eng. User Datagram Protocol) Ne zahteva konekciju prilikom komunikacije između aplikacija. UDP protokol se koristi za razmenu podataka u realnom vremenu, npr. u emitovanju video klipova, igricama, .itd.

Socket

Socket je End-point za dvosmernu komunikaciju između dva programa (klijenta i servera).
Socket je predstavljen kao par (IP adresa, port)
Definisanje ip-adrese je putem IPAddress klase, tj. definisanje lokalne ip adrese je moguće sa IPAddress.Loopback.
Neophodno je definisati i ip end point: IPEndPoint(ipAddress, port). Parametar port treba da bude slobodan port na računaru.
Prilikom definisanja socketa (na serveru ili klijentu) treba da se definiše Socket(addressFamily, socketType, protocolType) gde su parametri sledeći:

- addressFamily - ipAddress.AddressFamily ili AddressFamily.InterNetwork - IPv4 adresa je 32-bitni broj koji predstavlja jedinstveni identifikator za mrežu na mašini. U našem slučaju to je localhost (127.0.0.1)
- socketType - SocketType.Stream podržava dvosmernu konekciju koja je bazirana na protok bajtova, i sprečava dupliranje podataka i poštuje ograničenja. SocketType.Stream se koristi u kombinaciji sa AddressFamily.InterNetwork i ProtocolType.Tcp
- protocolType - ProtocolType.Tcp

Otvaranje/Kreiranje konekcije se razlikuje na serverskoj i klijentskoj strani:

1. Serverska strana:

    - serverSocket.Bind(EndPoint): server socket mora da se poveže na definisan localEndPoint kako bi mogao da osluškuje klijente koji će se javiti na isti endpoint
    - serverSocket.Listen(backlog): definišemo backlog:int maksimalni broj konekcije koje mogu da budu u redu za čekanje
    - serverSocket.Accept(): nakon uspešnog povezivanja sa endpoint-om, neophodno je program smestiti u beskonačnu petlju, gde server prihvata nove konekcije od strane klijenata sa metodom serverSocket.Accept()
    - Task t = Task.Factory.StartNew(() => Run(socket)); - TaskFactory je metoda koja kreira i pokreće task. Task je asinhrona operacija (u našem slučaju poziv metode Run(socket) je asinhrono)
    - stream: NetworkStream - stream je tip razmene podataka između servera i klijenta

- streamWriter: StreamWriter - server šalje podatke klijentu
- streamReader: StreamReader - server čita podatke koje je klijent poslao

2. Klijentska strana:

- klijentSocket.Connect(remoteEP): klijent se javlja serveru koji sluša na remoteEP (remote Endpointu)
- stream: NetworkStream - stream je tip razmene podataka između klijenta i servera
- streamWriter: StreamWriter - korisnik šalje podatke serveru
- streamReader: StreamReader - korisnik čita podatke koje je server poslao
- kako bi korisnik znao dokle treba podatke da čita od strane servera treba da postoji indikator KRAJ PORUKE, u ovim primerima to je "END"

Napomena za zadatak: Dopunite primer sa časa RegistrationClient i RegistrationServer (nalaze se u Files/Predavanje/2-konkuretno i mrežno programiranje/MreznoProgramiranje.zip). Primer radi tako što klijent prosledi novog korisnika serveru, i server odgovara sa listom svih korisnika i odmah se konekcija zatvara između klijenta i servera. Treba da omogućite rad klijentske aplikacije u beskonačnoj petlji (dok korisnik ne odabere opciju da se ugasi program, ali samo klijentski deo aplikacije). Funkcionalnosti su implementirane sa serverske strane aplikacije. Neophodno je prvo pokrenuti server, potom klijenta. Aplikacija treba da podrži pokretanja više klijenata istovremeno.