2. Programiranje sa soketima

Na ovim vježbama ćemo se baviti soketima, TCP protokolom, klijent-server arhitekturom i klasama u C#-u koje se koriste za realizaciju komunikacije preko soketa. Kako bi kod bio jednostavniji, uvest ćemo async i await elemente TPL-a.

2.1 OSI model

Internationl Standards Organization/Open System Interconnection (ISO/OSI) model je referentni model za opis komunikacije između dviju točaka na mreži. ISO model definira mrežni *framework* za implementaciju protokola u skladu sa 7 definiranih slojeva. Svaki sloj je funkcionalno neovisan o ostalima, ali pruža servise sloju iznad sebe te koristi servise sloja ispod sebe. Ovdje su slojevi navedeni od niže prema višoj razini.

1. Fizički sloj - definira električka i fizička svojstva mrežnih uređaja. Ovaj sloj prenosi bitove od jednog do drugog računala i regulira prijenos *stream*-a bitova preko fizičkog medija;
2. Podatkovni sloj - odgovoran za pakiranje bitova u frame-ove - logičke, strukturirane pakete podataka. Odgovoran je za prijenos podataka od jednog do drugog uređaja bez pogrešaka. U WAN (*Wide Area Network*) mreži ovo mogu biti dva rutera, ili ruter i host. U LAN (*Local Area Network*) mreži ovo je između dva hosta;
3. Mrežni sloj - osigurava funkcionalne i proceduralne zahtjeve potrebne za prijenos podataka proizvoljne veličine među računalima unutar iste mreže. Ovaj sloj brine o usmjeravanju podataka kroz mrežu te prevodi logičke adrese u fizičke adrese. Ujedno generira rutu kojom će podatci putovati kroz mrežu. Ovo nije potrebno raditi kada je u pitanju LAN mreža. Ovaj sloj rješava i problem zagušenosti mreže. Na ovoj razini djeluje IP protokol;
4. Transportni sloj - najniža razina na kojoj se upravlja porukama (a ne paketima). Poruke se šalju komunikacijskim portovima koji su vezani uz procese. Protokoli na ovome sloju mogu biti bazirani na stvaranju konekcije ili bez konekcije. Primjeri su TCP i UDP;
5. Sloj sesije - koji je zadužen za stvaranje, održavanje i prekid konekcija između lokalne i udaljene aplikacije. na ovom sloju se detektiraju pogreške i provodi automatski oporavak;
6. Prezentacijski sloj - protokoli koji se nalaze na ovoj razini prenose podatke u obliku koji je neovisan o prikazu unutar pojedinih računala, a koji se mogu razlikovati. Na ovome sloju se ujedno vrši i enkripcija, ako je potrebna. Npr. TLS;
7. Aplikacijski sloj - ovo je sloj koji je najbliži krajnjem korisniku. U prijevodu to znači da ovaj sloj OSI modela i korisnik direktno komuniciraju s aplikacijom. Važno je napomenuti da na ovome sloju ne spadaju aplikacije tipa web browseri, već su oni samo aplikacije koje koriste komunikacijske protokole ovoga sloja. Funkcije ovog sloja uključuju identifikaciju komunikacijskih partnera, određivanje dostupnosti resursa i sinkroniziranje komunikacije. Primjeri protokola: HTTP, FTP, SMTP;

2.2 Soketi

Soketi predstavljaju krajnje točke u komunikaciji računalnih procesa preko računalne mreže, a mogu se koristiti i za komunikaciju između procesa koji se nalaze na istome računalu. U osnovi se koriste za sve internet aplikacije.

Da bi mogli izraditi program koji će komunicirati preko mreže, potreban nam je konkretan API koji će omogučiti aplikacijama koje se vrte na *host* računalu da komuniciraju s mrežom. **Socket API** je jednostavna abstrakcija koja nam ovo omogućuje. Soketi su se pojavili 1983. godine i postali su sastavni dio svakog operacijskog sustava. Skoro svi programski jezici danas imaju biblioteke koje nam omogućuju komunikaciju sa soket API-om operacijskog sustava. Soket API predstavlja abstrakciju za upravljanje krajnjom točkom konekcije.

U svrhu boljeg objašnjenja pogledajmo upravo rečeno iz druge perspektive. Aplikacija se vrti na lokalnome računalu i želi pristup mreži. Socket API je abstrakcija između aplikacije i mreže koji će aplikaciji omogućiti korištenje mreže. Soketi koriste strukturu podataka koja se također zove Soket kako bi omogućili aplikaciji da se poveže sa mrežom. U slučaju da imamo više aplikacija koje koriste mrežu, svaka aplikacija mora imati instancu ove Soket strukture podataka, ali soket API je isti za sve aplikacije.

Unix sustav koristi *Berkeley sockets* te ovaj API predstavlja sokete kao file deskriptore u UNIX sustavu zbog UNIX filozofije - "Sve je datoteka" i sličnosti između konekcije i datoteke. File deskriptor je jednostavno broj koji predstavlja otvorenu datoteku. Sličnost između konekcije i datoteke proizlazi iz toga što se iz oboje može čitati, u oboje se može upisivati, mogu se otvoriti i zatvoriti.

Adresa soketa je kombinacija IP adrese računala i broja porta. Proces ili nit koja se poveže na soket može slati i primati podatke sa tog soketa.

U tablici je navedena podjela portova.

Broj porta	**Objašnjenje**
0 - 1023	portovi koji su rezervirani i koriste ih postojeće aplikacije
1024 - 49151	registrirani portovi - vendori ih koriste za svoje aplikacije; neke portove možete koristiti
49152 - 65535	slobodni portovi

Soketi se obično nalaze na sloju 4 OSI modela.

Neki česti portovi:

- 21 - *File Transfer Protocol* (FTP)
- 22 - *Secure Shell* (SSH)
- 23 - *Telnet* servis za udaljeni login
- 25 - *Simple Mail Transfer Protocol* (SMTP)
- 53 - *Domain Name System* (DNS) servis
- 80 - *Hypertext Transfer Protocol* (HTTP)
- 110 - *Post Office Protocol* (POP3)
- 119 - *Network News Transfer Protocol* (NNTP)
- 123 - *Network Time Protocol* (NTP)
- 143 - *Internet Message Access Protocol* (IMAP)
- 161 - *Simple Network Managment Protocol* (SNMP)
- 194 - *Internet Relay Chat* (IRC)
- 443 - *HTTP Secure* (HTTPS)

2.3 TCP/IP

TCP protokol, *Transmission Control Protocol*, u kombinaciji s IP protokolom definira način na koji uređaji moraju biti povezani da bi uspješno komunicirali preko interneta.

IP protokol, *Internet Protocol*, čiji je zadatak da na osnovu adrese paket s jednog računala dođe na drugo. IP paket ima veličinu od 64 KB, a sastoji se od nekoliko polja, ali osnovna podjela je na zagalavlje i podatke. IP pruža servis dostave paketa opisan *best effort* ili *unreliable* semantikom zbog toga što ne postoji garancija o dostavi paketa. Jedino što se provjerava u ovome protokolu jest zaglavlje paketa kako bi se osiguralo da paket stigne na zadano odredište. U IP protokolu ne postoji provjera o integritetu poslanih podataka. Time se smanjuje potrebno vrijeme za prolazak kroz brojne rutere kojima paket putuje. Protokoli više razine TCP i UDP mogu implementirati svoju provjeru integriteta podataka. IP sloj također mora u mrežni sloj ubaciti fizičku adresu računala kojemu je poruka namijenjena. U detalje ovoga nećemo ulaziti.

UDP protokol je skoro pa replika IP protokola, samo na drugoj razini. Ima kratko zaglavlje u kojemu su navedene adrese pošiljatelja i primatelja, polje za duljinu poruke i kontrolni zbroj (engl. *checksum*). UDP ne garantira da će se poruka dostaviti primatelju, a sam *checksum* je opcionalan. Kontrolni zbroj služi kako bi se prepoznala prisutnost pogreške u primljenoj poruci, ukoliko se ista dogodi tijekom slanja. UDP-om se može poslati poruka veličine do 64 KB, što je maksimalna dopuštena poruka IP-a. Kod korištenja UDP-a nema stvaranja konekcije među uređajima, što znači

da se cijela komunikacija između dva uređaja svodi na to da jedan šalje podatke drugome. UDP ne zahtjeva nikakve potvrde o primitku. Pogodan je za aplikacije kojima je potreban brz i efikasan prijenos podataka, a nije potrebno pouzdano slanje poruke. S obzirom da su svi paketi UDP protokola zasebni, ne postoji nikakvo uređenje među istima. U slučaju da je uređenje potrebno, onda je to zadatak aplikacijskog sloja. UDP protokol koriste DNS i DHCP protokoli.

TCP prtokol pruža mnogo sofisticiraniji način prijenosa podataka. Ovaj protokol pruža pouzdan prijenos proizvoljno dugog niza bajtova putem programske abstrakcija bazirane na *stream*-u. Pouzdanost TCP protukala osigurava da će primatelj dobiti sve podatke koji su mu poslani. TCP paket u sebi sadrži informaciju o tome koji je redosljed poslanih podataka. TCP je konekcijski orijentiran protokol, što znači da pošiljatelj i primatelj poruke moraju uspostaviti dvosmjerni komunikacijski kanal. Konekcija između dva računala je jednostavno sporazum da se započme s pouzdanom komunikacijom. Među čvorovi kojima paketi moraju proći od jednog do drugog čvora nemaju nikakvog znanja o TCP konekciji, a IP paketi kojima se prenose podatci ne moraju svaki put ići istim putem od jednog do drugog čvora.

Kako bi pouzdano slao podatke, TCP sloj uključuje dodatne mehanicme:

- **Sekvencioniranje** - proces koji šalje podatke preko TCP-a podjeli niz bajtova u sekvencu segmenata i šalje ih preko IP-a. Broj sekvence je povezan uz svaki TCP segment. Primatelj koristi ovaj broj da bi poredao primljene segmente prije nego ih ubaci prosljedi dalje kao ulazni niz bajtova. Nijedan segment se nemože ubaciti ulazni niz bajtova (engl. *input stream*) sve dok svi segmenti koji imaju manji broj sekvence nisu primljeni i postavljeni u stream. Svi segmenti koji nisu stigli odgovarajućim redosljedom se moraju čuvati u *buffer*-u.
- **Kontrola protoka** - pošiljatelj podataka brine o tome da se primatelju i međučvorovima ne pošalje previse podataka. Ovo se rješava sustavom slanja potvrda o primljenim segmentima. Kadgod primatelj primi segment, zabilježi broj sekvence. Primatelj povremeno pošalje pošiljatelju potvrdu o primitku koja sadrži najveći broj sekvence koji se nalazi u ulaznom streamu skupa s brojem koji određuje koju količinu podataka pošiljatelj smije poslati prije sljedeće potvrde o primitku. Ako postoji protok podataka u suprotnom smjeru, potvrde se šalju s običnim segmentima podataka; u suprotnom se šalju s segmentima potvrde *acknowledgment segments*.
- **Ponovno slanje** - pošiljatelj bilježi broj sekvence segmenata koje šalje, kada dobije potvrdu o primitku zabilježi da je segment uspješno poslan te ga može izbrisati iz svog izlaznog *buffer*-a. U slučaju da bilo koji od segmenata nije potvrđen unutar zadanog vremenskog intervala, pošiljatelj ponovno šalje segment.
- **Buffering** - ulazni buffer ima zadatak da balansira prijenos podataka između pošiljatelja i primatelja. Ako podatci pristižu brže nego što ih primatelj može primiti, količina podataka u *buffer*-u će narasti. Obično se podatci stignu izvaditi iz *buffer*-a prije nego se napuni, ali zna se nekada dogoditi da *buffer* primi previše više podataka nego što može sadržavati. U tom slučaju se ulazni segmenti ignoriraju i ne šalje se potvrda o njihovom dolasku. Stoga pošiljatelj te segmente mora ponovno poslati.
- **Kontrolni zbroj** - svaki segment sa sobom sadrži kontrolni zbroj kojim se može provjeriti jesu li podatci ispravno primljeni.

Stoga je TCP u odnosu na UDP sporiji, ali pouzdaniji. Sigurni smo da će svi podatci stići primatelju i da će ih primatelj moći posložiti. Podatci se šalju kao stream bajtova (niz bajtova) te nema nikakvih očitih indikacija kojima se razlikuju segmenti poruke. Kod UDP-a je stvar nešto drugačija te paketi imaju točno određene granice koje se poštuju. Za razliku od UDP-a, da bi se uspostavila konekcija TCP-om potrebno je slanje tri paketa prije nego se bilo kakvih podatci počmu razmijenjivati.

2.4 Klijent-server model

Klijent-server model je struktura distribuiranog sustava kod kojeg su zadatci podijeljeni među poslužiteljima nekog resursa ili servisa, koji se zovu serveri, i korisnika resursa ili servisa, koji se zovu klijenti. Server dijeli svoje resurse s klijentima koji ih zatraže. Klijenti svoje resurse ne dijele, već po potrebi zatraže zahtjev od servera. Stoga server stalno čeka na zahtjeve koje iniciraju klijenti. U ovakvom modelu raspodijeljenog sustava, server je centralna komponenta koja kontrolira komunikaciju i pristup resursima na mreži. U slučaju jednog centralnog servera, on postaje slaba točka sustava jer bilo kakva pogreška na serveru će se odraziti na cijeli sustav.

2.5 TcpListener i TcpClient

`TcpListener` je klasa kojom instancirom objekt koji će slušati na dolazeće TCP konekcije.

`TcpClient` je klasa koju instanciramo kada želimo stvoriti konekciju na TcpServer. Ova klasa je na većoj razini abstrakcije od klase Socket (koja također postoji u C#-u) te sama referencira istoimenu klasu.

Metode koje su definirane na TcpListener-u te ćemo ih koristiti:

- Start() - metoda koja služi za pokretanje samog *listener*-a;
- AcceptSocket() - metoda kojom *listener* čeka na konekciju te uspostavom konekcije vraća Socket objekt;
- AcceptTcpClient() - kao i AcceptSocket() samo što vraća TcpClient kao povratnu vrijednost;
- GetStream() - služi za dobavljanje *stream*-a konekcije. Povratna vrijednost je instanca klase `NetworkStream`. NetworkStream objekt pruža metode za slanje i primanje podataka preko soketa;

Metode koje su definirane na TcpClient-u te ćemo ih koristiti:

- Connect(ipAddress, port) - inicijaliziramo konekciju na zadanu IP adresu i port;
- GetStream() - kao kod TcpListenera;

Stream-ove shvaćamo kao prikaz bajtova. Već smo gore naveli NetworkStream koji služi za slanje podataka preko mreže. Na prethodnim programiranjima ste se vjerojatno susreli sa `FileStream`-om koji zapisuje podatke u datoteku. U ovom kolegiju ćemo koristiti još i *MemoryStream* koji podatke zapisuje u memoriju računala. Za čitanje bajtova iz *stream*-a koristit ćemo *StreamReader* kojim pročitane bajtove možemo odmah pretvoriti u string.

`MemoryStream` ima metodu `Seek` koja nam omogućuje pozicioniranje unutar streama. Svi *stream*-ovi koje smo spomenuli do sada imaju metode `Read` i `Write` za čitanje i pisanje iz odgovarajućeg *stream*-a.

Primjer 2.1: Jednostavan primjer servera

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.IO;
5  using System.Linq;
6  using System.Net;
7  using System.Net.Sockets;
8  using System.Text;
9  using System.Threading.Tasks;
10
11 namespace HelloServer
12 {
13     class Program
14     {
```

```
15          static void Main(string[] args)
16          {
17              string ipString = "127.0.0.1";
18              int port = 8080;
19
20              // Stvaramo instancu ip adrese
21              IPAddress ipAddress = IPAddress.Parse(ipString);
22
23              Console.WriteLine("Starting the listener....");
24
25              // Prilikom instanciranja listenera u konstruktor
                    saljemo ip adresu i port na kojem ce slusati
26              // za nadolazece konekcije.
27              TcpListener listener = new TcpListener(ipAddress,
                    port);
28
29              listener.Start();
30
31              TcpClient socket = listener.AcceptTcpClient();
32
33              NetworkStream stream = socket.GetStream();
34              // Stream koji zapisuje u memoriju racunala
35              MemoryStream memoryStream = new MemoryStream();
36              StreamReader reader = new StreamReader(memoryStream);
37
38              // buffer - spremnik za podatke koje citamo
39              byte[] inBuffer = new byte[4096];
40
41              // procitani podatci se spremaju u buffer
42              int bytesRead = stream.Read(inBuffer, 0,
                    inBuffer.Length);
43
44              memoryStream.Write(inBuffer, 0, bytesRead);
45
46              // Uz pomoc Seek metode se postavimo na neku
                    poziciju unutar memory streama
47              // Pokusajte promijeniti SeekOrigin na End.
48              memoryStream.Seek(0, SeekOrigin.Begin);
49
50              // Procitajmo poruku.
51              string line = reader.ReadLine();
52
53              Console.WriteLine("Got msg: " + line);
54
55              string response = "200 OK";
56
57              byte[] buffer = Encoding.UTF8.GetBytes(response);
58
59              // zapisimo podatke u network stream
60              // parametri objasnjeni redom:
61              // - buffer iz kojeg citamo podatke - u ovom slucaju
                    niz byteova "data"
62              // - offset - od koje pozicije pocinjemo s citanjem
                    podataka - 0
63              // - size - koliko podataka zelimo zapisati -
                    data.length
```

```
64              stream.Write(buffer, 0, buffer.Length);
65              // Obicno kada radimo sa streamovima, potrebno je
                    pozvati Flush() metodu kako bi se podatci
                    zapisali.
66              // To nije slucaj sa NetworkStreamom kako pise na
                    stranici microsofta:
67              // "The Flush method implements the Stream.Flush
                    method; however, because NetworkStream is not
                    buffered,
68              // it has no affect on network streams. Calling the
                    Flush method does not throw an exception."
69              // Pozivamo ovu metodu cisto zbog konvencije
70              stream.Flush();
71              // ocisti stream-ove
72              stream.Dispose();
73              memoryStream.Dispose();
74
75              Console.ReadLine();
76
77              Console.WriteLine("Stopping server...");
78          }
79      }
80  }
```

Primjer 2.2: Jednostavan primjer klijenta

```
1
2   using System;
3   using System.Collections.Generic;
4   using System.IO;
5   using System.Linq;
6   using System.Net;
7   using System.Net.Sockets;
8   using System.Text;
9   using System.Threading.Tasks;
10
11  namespace HelloClient
12  {
13      class Program
14      {
15          static void Main(string[] args)
16          {
17              int port = 8080;
18              string stringIp = "127.0.0.1";
19
20              IPAddress ip = IPAddress.Parse(stringIp);
21
22              // instanciramo klijenta
23              TcpClient client = new TcpClient();
24
25              // inicijaliziramo konekciju
26              client.Connect(ip, port);
27
28              // dohvati stream
29              NetworkStream stream = client.GetStream();
```

```
30              // Stream koji zapisuje u memoriju racunala
31              MemoryStream memoryStream = new MemoryStream();
32              StreamReader reader = new StreamReader(memoryStream);
33
34              // buffer za odlazne poruke
35              byte[] inBuffer = new byte[4096];
36
37              // P2
38              string msg = Console.ReadLine();
39
40              // Pretvorimo string u niz byte-ova
41              byte[] data = Encoding.UTF8.GetBytes(msg);
42
43              // parametri objasnjeni redom:
44              // - buffer iz kojeg citamo podatke - u ovom slucaju
                    niz byteova "data"
45              // - offset - od koje pozicije pocinjemo s citanjem
                    podataka - 0
46              // - size - koliko podataka zelimo zapisati -
                    data.length
47              stream.Write(data, 0, data.Length);
48              // Obicno kada radimo sa streamovima, potrebno je
                    pozvati Flush() metodu kako bi se podatci
                    zapisali.
49              // To nije slucaj sa NetworkStreamom kako pise na
                    stranici microsofta:
50              // "The Flush method implements the Stream.Flush
                    method; however, because NetworkStream is not
                    buffered,
51              // it has no affect on network streams. Calling the
                    Flush method does not throw an exception."
52              // Pozivamo ovu metodu cisto zbog konvencije
53              stream.Flush();
54
55              int bytesRead = stream.Read(inBuffer, 0,
                    inBuffer.Length);
56
57              memoryStream.Write(inBuffer, 0, bytesRead);
58
59              memoryStream.Seek(0, SeekOrigin.Begin);
60
61              string recv = reader.ReadLine();
62
63              Console.WriteLine(recv);
64
65              // zatvorimo streamove
66              memoryStream.Dispose();
67              stream.Dispose();
68
69              Console.ReadLine();
70          }
71      }
72  }
```

Ubuduće ćemo koristiti OOP koncepte kako bismo aplikacije organizirali na manje komponente zbog bolje organizacije o jednostavnije održavanja. Primjer možete pronaći na moodle-u. Ovdje

nije uključen.

2.6 Async-await

Na prošlim vježbama smo spominjali Task-ove i objasnili što su i kako funkcioniraju. Kod asinkronog programiranja, programiranja s taskovima, čest je slučaj da se nakon završetka nekog taska pozove druga operacija kojoj se pošalju podatci iz završenog taska. U TPL-u ovo se postiže takozvanim *continuation task*. Bez da ulazimo previše u detalje, `async` i `await` ključne riječi nam služe za pojednostavniti kod prilikom korištenja ovakvih taskova. U .NET-u 4.5 je uvedena ključna riječ `await` koja služi da pauziramo metodu na nekoj liniji dok ćekamo rezultat taska, a pri tome ne blokiramo nit. Nit na kojoj je pozvan await se vraća u ThreadPool te ostatak metode može biti izvršen na istoj ili nekoj drugoj niti. Ključnu riječ `await` možete koristiti samo u metodama koje su označene s ključnom riječju `async`. Takve metode kao povratnu vrijednost mogu imati samo: `void`, `Task` i `Task<T>`.

Primjer "MultipleClientExample" demonstrira primjenu `async` i `await` ključnih riječi za posluživanje više klijenata istovremeno.

2.7 Zadatci

1. Napravite Ping-Pong aplikaciju u kojoj će klijent slati poruku "Ping", a server će odgovoriti porukom "Pong". Aplikacija se izvršava dok korisnik ne unese q i pritisne <Enter>. Uočite da jedna nit neće biti dovoljna.
2. Modificirajte primjer "MultipleClientExample" na način da na sučelje klijenta dodate `TextBox` elemente za unos ip adrese i porta te botun kojim započinjete konekciju na server. Ostali botuni na sučelju su isključeni dok se ne uspostavi koenkcija. U slučaju da je konekcjia uspješno započela potrebno je onemogućiti navedene elemente. Ostatak klijenta mora funkcionirati kao i prije. Uzmite u obzir i slučaj prekida konekcije - potrebno je ponovno unjeti ip adresu i port te pritisnuti botun za uspostavu konekcije, a ostali elementi su isključeni. Ujedno modificrajte primjer da ne puca ako se klijent pokrene prvi.
3. Modificirajte primjer "MultipleClientExample" na način da pruža osnovne funkcije kalkulatora: zbrajanje, oduzimanje, množenje, djeljenje, potenciranje, logaritam i eksponent (e^x). Razmislite kako ćete predstaviti poruku. Sve operacije osim eksponenta očekuju dva argumenta. U slučaju pogreške ili pogrešnih parametara server vraća poruku "500 internal server error", a u suprotnom vraća "200 OK <Rezultat>".
4. Modificirajte primjer "MultipleClientExample" (bez modifikacije iz zadatka 3) na način da server nakon primitka poruke istu šalje svim klijentima koji su spojeni (ovo se zove *Broadcast*). Ako ispravno implementirate trebali biste dobiti jednostavnu chat aplikaciju. Razmislite kako ovo implementirati.

2.8 Izvori

Distributed Systems: Concepts and Design, fifth edition, George Coulooouris, Jean Dollimore, Tim Kindberg, Gordon Blair.

`http://www.cisco1900router.com/what-is-ios-model-the-overall-explanation-of-ios-7-layers.html`

`https://en.wikipedia.org/wiki/OSI_model`

`https://msdn.microsoft.com/en-us/library/fx6588te(v=vs.110).aspx`

`https://msdn.microsoft.com/en-us/library/hh191443.aspx`

`https://en.wikipedia.org/wiki/Transmission_Control_Protocol`

`https://en.wikipedia.org/wiki/User_Datagram_Protocol`

http://www.diffen.com/difference/TCP_vs_UDP
https://en.wikipedia.org/wiki/Internet_Protocol
https://en.wikipedia.org/wiki/Network_socket
https://en.wikipedia.org/wiki/Berkeley_sockets
http://whatis.techtarget.com/definition/sockets
http://www.w3schools.com/website/web_tcpip.asp
https://en.wikipedia.org/wiki/Port_(computer_networking)