Szerver-kliens programok

Hogyan is kezdjünk hozzá egy kliens-szerveres program elkészítéséhez? Nos, a dolog nem túl bonyolult. Kell egy protokoll, amivel a kommunikáció végbemegy, erre rengetegféle lehetség van. Ha ez megvan, akkor a protokoll használatával ki kell alakítani a parancsok szintaktikáját, miután persze átgondoltuk, hogy milyen parancsokra is lesz szükségünk. Na de kezdjük az elején. Internethálózaton TCP vagy UDP kapcsolatot létesíthetünk. TCP kapcsolat esetén, adott porton egyszerre csak egy kapcsolat lehetséges. UDP esetén nincs kötött kapcsolat, és a csomagok megérkezésének egymásutáni sorrendje sem garantált, és az adatvesztés is lehetséges. Íme Java-ban néhány sor, amivel létre is hoztunk egy TCP kapcsolatot.

Szerver oldal

```
ServerSocket server = new ServerSocket((int) port);
```

Kliens oldal

```
Socket connection = new Socket((String) IP_address, (int) port);
```

A megadott int típusú port-on, ServerSocket típusú server objektum létrehozza a kapcsolódási lehetséget a szerver oldalon. Ha ez megvan, akkor a kliens az adott porton kapcsolatba tud lépni a szerverrel, ismerve a szerver IP címét. Ha nem tud kapcsolatot létesíteni, akkor Exception-t (kivételt) generál, vagyis a program hibával leáll. Ezt elkerülend, try-catch szerkezetbe kell ágyazni a kapcsolatfelépítést:

```
try
{
  connection = new Socket("127.0.0.1", 5555);
}
catch (Exception e)
{
  System.err.println("Connection error: " + e + "\n");
}
```

Így a program lekezeli a hibát. Sikertelen kapcsolódás esetén célszer lehet egymás után többször is megkísérelni a kapcsolatfelvételt, kisebb szünetek (pl.: 0,1 másodperc) beiktatásával. UDP kapcsolat létesítésére is hasonló a forráskód. Az UDP kapcsolat abból a szempontból elnyösebb, hogy egyszerre több kliens is kommunikálhat ugyanazzal a szerverrel, ugyanazon a porton. Ugyanakkor TCP esetén sem jelent gondot több kliens kiszolgálása, csak más módszerre van szükség:

1. A szervert szálakra bontjuk, mindegyik szálhoz más-más porton tudnak kapcsolódni a kliensek. Kérdés, hogy hogyan történjen a portok kiosztása a kliensek számára.
2. Csak egy portot használunk a kapcsolathoz, de a kapcsolatot gyakran megszakítjuk, hogy más kliensek is csatlakozni tudjanak, vagyis a kliensek felváltva kapcsolódnak a szerverhez.

Ez a módszer sok kliens esetén nagy mértékben csökkenti az alkalmazható kommunikációs sebességet. Localhost-on tesztelve a programunkat, egy portot használva, csak TCP használatával futhat több kliens, így ebben a helyzetben gondot okozhat UDP-s megoldásnál a program tesztelése. Ez a probléma nem áll el, ha a szerver és a kliensek más-más gépen futnak. A gond abból ered, hogy ha egy gépen több program is fut, akkor közülük csak az egyik kap meg egy adott csomagot egy adott porton.

A felsorolt módszerek közül bármelyik használható, annak függvényében, hogy mi a megoldandó probléma. UDP esetén viszont mindig szem eltt kell tartani, hogy nem garantált a csomagok megérkezése, míg TCP esetén az elveszett vagy hibás csomagot újraküldi az adatkapcsolat. Tehát felépítettünk egy kapcsolatot, ezt felhasználva csomagokat kell küldenünk:

Kliens oldali üzenet fogadás

```
InputStreamReader readConnection = new InputStreamReader(connection.getInputStream());
BufferedReader fromServerReader = new BufferedReader(readConnection);
String strMsg = fromServerReader.readLine();
```

Kliens oldali üzenet küldés

```
PrintWriter toServerWriter = new PrintWriter(connection.getOutputStream(), true);
toServerWriter.println(strMsg);
```

Szerver oldali üzenet fogadás

```
InputStreamReader readConnection = new InputStreamReader(connection.getInputStream());
BufferedReader fromClientReader = new BufferedReader(readConnection);
String strMsg = fromClientReader.readLine();
```

Szerver oldali üzenet küldés

```
PrintWriter toClientWriter = new PrintWriter(connection.getOutputStream(), true);
toClientWriter.println(strMsg);
```

Mint látható, a kliens és a szerver oldalon ugyanazt a kódot kell alkalmazni. Ezzel a módszerrel szöveges üzenetek küldhetek, de más módszer is használható. A kapcsolat lezárható a Socket, illetve SeverSocket close() metódusával.

Van egy kliens és egy szerver programunk, amelyek képesek egymással kommunikálni, de hogyan fogalmazzák meg, hogy mit akarnak egymástól? Küldhetek parancsszavak, adatok, melyeket a fogadó fél kiértékel, a meghatározott szabályrendszer szerint. A programok hálózati kommunikációja többféle lehet, bizonyos esetekben egy szerver kiszolgál több klienst, máskor több szerver szolgál ki több klienst, és a szerverek és a kliensek is kapcsolatban állhatnak egymással. A kliensek kapcsolatban lehetnek egymással a szerver közvetítésével, vagy direkt módon is. Egyes esetekben nincs szükség szerverre.

Az fent leírt kódokkal persze szükség van szerverre, mert anélkül nem hozható létre a kapcsolat, ellenben egy program mködhet szerver és kliens üzemmódban is. Például kett futó alkalmazás dinamikusan eldöntheti, hogy melyikük legyen a szerver. A program elször kapcsolódni próbál a szerverre, ha ez nem sikerül, akkor megpróbál szerverként mködni, ha nem kapcsolódnak rá, akkor újból próbál kapcsolódni. Ha mindkét program indításra került, akkor egy olyan állapotban, mikor az egyik program szerver, a másik kliens módban próbálkozik, létrejön a kapcsolat.

Szinkronizációs problémák: a példánál maradva, ha mindkét program egyszerre próbál szerver lenni, majd kliens, és a váltáskor nincs olyan idintervallum, melykor már egyikük másik módra váltott, de a másik még nem, akkor nem tudnak kapcsolódni. Persze nem egyszerre történik meg a váltás a két programon, de a hálózati átvitelre, a bejöv üzenet észlelésére és a visszacsatoló válaszküldésre is kell az id. Ha túl gyorsan vált szerver/kliens üzemmódot a program, akkor nincs id a kapcsolatépítésre, ha túl lassan, akkor sokáig tarthat a kapcsolódás. Nyilván kett futó alkalmazás esetén elég egyszer kapcsolódni, aztán már ezzel nincs gond. Adott több futó alkalmazás, mondjuk 100 darab kliens és 1 szerver (mindegyikük egyértelmen kliens ill. szerver). Ha minden kapcsolat ugyanazt a portot kívánja használni, éspedig TCP-vel, akkor szükségszer a folyamatos fel-le csatlakozás, itt már gond lehet a lassúság. Egy alkalmazás nem kötheti le túl sokáig a szervert. Túl rövid kapcsolatok esetén viszont egyszerre túl sok kliens próbálhat kapcsolódni sikertelenül, ami szintén lassulást eredményez. Az eddigiek még nem tényleges szinkronizációs gondok. De ha az alkalmazások számára elérhet hálózati sebesség más, akkor elfordulhat torlódás, az egyik gép már régóta válaszra vár, míg a másik gép meg még csak a csomagok felét kapta meg. Több alkalmazás esetén még inkább bonyolódhat a helyzet. Ha sok adat továbbítására van szükség, akkor egy lassú kliens hosszú idkre foghatja a szervert.

UDP esetén gondoskodni kell a sorrendiségre utaló információ elküldésérl.