Implementacija klienta in strežnika v Javi

Komunikacije v avtomatiki – Laboratorijske Vaje

9. december 2013

Povzetek

Ta vaja je pripravljalna vaja na nalogo, ki jo boste izvajali v naslednjih vajah. Namen te vaje je spoznati se s primerom (skope) implementacije TCP strežnika in TCP klienta. Zaradi preprostosti, boste vajo programirali v Javi. Priporočamo, da se čim več poslužujete spletne pomoči – na spletu namreč obstaja mnogo navodil in primerov uporabe različnih funkcionalnosti Jave. Kodo lahko urejate v poljubnem urejevalniku, predlagamo pa uporabo *Notepad++* v okolju Windows (če boste nalogo reševali tudi doma) oziroma urejevalnika *kwrite* v okolju Linux. Implementacije programov in teorija sistema TCP strežnik/klient je zelo dobro razložena v knjigi *Computer Networks – A top down approach* [1], poglavje 2.7, ki je na voljo tudi v fakultetni knjižnici. V vaji boste najprej prevedli ter zagnali programa za klient in strežnik, napisane v Javi. Predlagamo, da si zelo natančno ogledate programa in si razložite, kaj vsaka vrstica počne. V nadaljevanju boste opisali delovanje programov preko analize prometa v Wiresharku. Priporočamo, da s kodo tudi eksperimentirate in poskusite dodati lastne izboljšave/razširitve. Poročilo, ki ga boste izdelali danes, boste dopolnili na naslednjih vajah, in ga tudi oddali ob koncu naslednjih vaj na asistentov elektronski naslov. Na podlagi tega oddanega poročila boste ocenjeni. Pozor: če bo ura sporočila v elektronski pošti presegla konec vašega bloka laboratorijskih vaj, se bo poročilo tretiralo kot neopravljeno, oziroma, oddano z enotedensko zamudo.

1 Začetek z Javo

Če se boste želeli ukvarjati s to nalogo tudi doma, morate imeti nameščeno razvojno okolje za Javo (Java jdk). V tem primeru sledite spodnjim točkam. V Laboratoriju je Java že nameščena, zato spodnje tri točke preskočite. Med programiranjem boste verjetno naleteli na težave kako kakšen del programa implementirati. Zato se poslužujte literature o programskem jeziku Java. Priporočamo knjige iz knjižnice, predvsem pa spletno pomoč, na primer `http://docstore.mik.ua/orelly/java/index.htm` in `http://docs.oracle.com/javase/1.5.0/docs/api/`.

- Pojdite na spletno stran `http://www.oracle.com/technetwork/java/javase/downloads/index.html` in si prenesite ustrezno različico Jave za vaš operacijski sistem (če imate že nameščen Java jre, pazite, da si ne namestite drugačne verzije Jave jdk). Verzijo jave preverite v Windows konzoli tako, da vtipkate `java -version`. Verzija `1.6.xx` pomeni verzijo 6, `1.7.xx` pa pomeni verzijo 7.
- Ko si namestite Javo, morate dodati pot do `java` in `javac` v vašo sistemsko pot, da bodo ukazi vidni v katerikoli mapi. V okolju Windows pojdite na *MyComputer→Properties→Advanced System Settings→Environment Variables*. V oknu *System variables* najdite in se postavite na značko *Path*, pritisnite *Edit*, pod *Variable value* dodajte razločni znak `;` nato vpišite pot do vaše verzije Jave, na primer, `c:\Program Files\Java\jdk1.7.0\bin\`. Bodite pazljivi, da dodajate mapo, kjer piše `jdk`, in ne `jre`. Shranite spremembe.
- Preizkusite, če ste nastavili poti pravilno. Odprite Windows komandno konzolo (*Start→Run:cmd*). Ko se konzola odpre, vpišite `java -version` in `javac -version`. Če ne najde katerega od obeh ukazov, potem niste poti dodali prav. Če verzije Jave nista enaki pri obeh ukazih, imate nameščeni vsaj dve verziji Jave. V tem primeru lahko pride do problemov, ko prevajate z eno verzijo, poganjate pa z drugo, kar bo javljalo napako (verjetno, da ne najde metode `main()`) – odinstalirajte staro verzijo Jave.

2 Hello world

V tem poglavju boste prevedli in zagnali najosnovnejši programček v Javi. Vse kar program naredi po zagonu, je izpis `Hello world!` na ekran in zaključi izvajanje. Program lahko skopirate v datoteko z imenom `helloworld.java` ali pa si ga prenesete z naslova `http://vision.fe.uni-lj.si/classes/KA-vaje/vaje/2012/helloworld.java`. Vidite, da je glavni del programa definicija razreda `helloworld` (opazite, da ima enako ime kot je ime datoteke). Kot vsak razred, mora tudi ta razred vsebovati metode. V našem primeru vsebuje samo eno metodo, in sicer metodo `main`. Ta metoda se avtomatično kliče ob zagonu programa. Če pogledamo vsebino metode `main` vidimo, da vsebuje eno samo vrstico in sicer ukaz za izpis stavka `Hello world!`. Opazite, da sta v izpisu vključena tudi znaka `\n`. Ta znaka pomenita "*nova vrstica*".

```
import java.io.*;

public class helloworld {
 public static void main (String args[]) throws Exception {
   System.out.print("\nHello world!\n") ;
 }
}
```

Prevajanje in zagon:

- Zaženite Windows Command prompt in se postavite na mapo, v kateri se nahaja program `helloworld.java`.
- Program prevedemo z ukazom "`javac helloworld.java`" . Če se je program prevedel brez napake, potem se na disku pojavi `helloworld.class` in nadaljujemo z naslednjo točko.
- Prevedeni program zaženemo z ukazom "`java helloworld`" .
- Če vam program izpiše na ekran `Hello world!`, potem nadaljujte z vajo.

3 Osnovna programa TCP klient/strežnik

Na disku si ustvarite mapo `vaja5` in vanjo prenesite s spletnega naslova program za klient `http://vision.fe.uni-lj.si/classes/KA-vaje/vaje/2012/TCPClient.java` in program za strežnik `http://vision.fe.uni-lj.si/classes/KA-vaje/vaje/2012/TCPServer.java` (za-enkrat še ne zaganjajte programov). Zaradi preprostosti, bomo tako klient kot tudi strežnik zaganjali na istem računalniku, vendar sta oba programa dovolj splošna, da ju lahko zaganjate na različnih računalnikih. Če bi najprej zagnali program za strežnik, ga pustili prižganega in nato zagnali še program za klienta, bi se izvedel naslednji scenarij:

- Ob zagonu programa `strežnik.class` se strežnik vzpostavi in odpre *Wellcoming socket* (sprejemno vtičnico) na portu 9000.
- Ob zagonu programa `klient.class` se klient vzpostavi in kontaktira strežnik na port 9000.
- Ko klient kontaktira strežnik, ta odpre novo *Connection socket* vtičnico, kamor "priklopi" povezavo s klientom. Sedaj komunikacija lahko steče. Bodite pozorni, da je TCP vtičnica določena s štirimi parametri: *(i) IP oddaljene naprave, (ii) Port aplikacije na oddaljeni naprave, (iii) IP lokalne naprave, (iv) Port aplikacije na lokalni napravi.* V našem primeru je lokalna naprava računalnik, na katerem teče program strežnik, medtem, ko je oddaljena naprava računalnik, na katerem teče program klient.
- Klient strežniku pošlje tekstovno sporočilo z malimi črkami.
- Strežnik spremeni sporočilo v velike črke, spremenjeno sporočilo pošlje nazaj in prekine povezavo.
- Klient prejme spremenjeno sporočilo s strežnika in prekine povezavo (načeloma mu je ne bi bilo treba, saj to naredi že strežnik).

Zelo dobro skico poteka komunikacije najdete v knjigi *Computer Networks – A top down approach* [1], poglavje 2.7. To poglavje si preberite, da boste razumeli teorijo od TCP klientu/strežniku. Nato nadaljujte s spodnjimi navodili za vajo.

3.1 Razlaga programa TCPClient.java

Poglejmo si najprej program `TCPClient.java` (odprite si ga v tekstovnem editorju, npr., Notepad++ pod Windows ali kwrite pod Linuxom). Zaradi preglednosti smo v nadaljevanju pri opisu programa nekatere vrstice združili, vendar je povezava s kodo v `TCPClient.java` jasna:

```
import java.io.*;
import java.net.*;
public class TCPClient {
 public static void main (String args[]) throws Exception {
  String messageToServer = "komunikacije";
  Socket clientSocket = new Socket("localhost", 9000);
  OutputStream os = clientSocket.getOutputStream();
  DataOutputStream outToServer = new DataOutputStream(os);
  InputStreamReader isrServer = new InputStreamReader(clientSocket.getInputStream());
  BufferedReader inFromServer = new BufferedReader(isrServer);
  outToServer.writeBytes(messageToServer + '\n');
  String messageFromServer = inFromServer.readLine();
  System.out.println("Sporocilo iz streznika: " + messageFromServer);
  clientSocket.close();
 }
}
```

Program ustvari dva *podatkovna toka* (angl., data streams), `inFromServer` ter `outToServer` ,in eno *vtičnico* (angl., socket), `clientSocket`. Podatkovni tok `inFromServer` je priklopljen na vtičnico `clientSocket` iz katere klient lahko *prejema* znake, enega za drugim, iz mrežnega toka. Drugi podatkovni tok se imenuje `outToServer` in je prav tako priklopljen na vtičnico `clientSocket`. Preko tega toka klient lahko *pošilja* znake ven v mrežni tok. Poglejmo si sedaj natančneje posamezne dele zgornje kode:

```
import java.io.*;
import java.net.*;
```

Zgornji dve vrstici dodata v program pakete, ki so pomembni za delo z vhodno-izhodnimi tokovi (`java.io`) in paketi za delo z mrežnimi povezavami (`java.net`).

```
public class TCPClient {
 public static void main (String args[]) throws Exception
    { ..... }
}
```

Zgornji blok je standardna oblika, s katero začnemo vsak programček v Javi. Beseda `class` začne definicijo razreda z imenom `TCPClient`. Ta razred vsebuje definicije spremenljivk in metod. Naš razred ima samo eno metodo, in sicer, `main()`, ki je metoda, ki se kliče ob zagonu programa. Za te vaje ne bo pomemben pomen izrazov `public`, `static`, `void` in `throws Exception`.

```
String messageToServer = "komunikacije";
Socket clientSocket = new Socket("localhost", 9000);
```

V zgornjem bloku `String messageToServer` kreira spremenljivko s sporočilom "`komunikacije`", ki ga bomo poslali strežniku. Naslednja vrstica odpre vtičnico. To stori tako, da z ukazom `new` kreira objekt `clientSocket`, ki je tipa `Socket`. Beseda `localhost` pomeni računalnik, na katerem trenutno teče strežnik `TCPClient`, `9000` pa je port, na katerem strežnik posluša na svoji napravi. **Pozor, v kreiranju vtičnice `Socket(IP, Port);` bi v splošnem morali podati številko `IP` in `Port` naprave na kateri teče strežnik. V vašem primeru strežnik teče na isti napravi kot klient in samo zato smo za vrednost `IP`-ja napisali `localhost`.**

```
DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
```

Vrstica odpre podatkovni tok `outToServer`, s katerim bomo brali sporočila s strežnika, in ta tok priklopi na vtičnico `clientSocket`, saj bomo vso komunikacijo izvajali preko te vtičnice.

```
InputStreamReader isrServer = new InputStreamReader(clientSocket.getInputStream());
BufferedReader inFromServer = new BufferedReader(isrServer);
```

Prva vrstica zgoraj kreira objekt `isrServer` tipa `InputStreamReader` in ga priklopi na vtičnico `clientSocket`. Druga vrstica zgoraj odpre tok `inFromServer`, s katerim bomo pošiljali sporočila na strežnik, in ga priklopi na objekt `isrServer`, ki pa je priklopljen na vtičnico `clientSocket`.

```
outToServer.writeBytes(messageToServer + '\n');
```

Vrstica doda sporočilu znak za novo vrstico[1] `\n` in ga pošlje v podatkovni tok `outToServer` z metodo `writeBytes(...)`.

```
String messageFromServer = inFromServer.readLine();
System.out.println("Sporočilo iz strežnika: " + messageFromServer);
clientSocket.close();
```

V zgornjih treh vrsticah klient prebere *vrstico* sporočila iz toka `inFromServer` z metodo `readLine()`, izpiše njegovo vsebino z ukazom `System.out.println(...)` in zapre vtičnico (prekine povezavo) z ukazom `clientSocket.close();`.

3.2 Razlaga programa TCPServer.java

Poglejmo si sedaj program `TCPServer.java` (odprite si ga v tekstovnem editorju). Zaradi preglednosti smo v nadaljevanju pri opisu programa ponovno nekatere vrstice združili:

```
import java.io.*;
import java.net.*;
class TCPServer {
 public static void main (String args[]) throws Exception {
  ServerSocket welcomeSocket = new ServerSocket(9000);
   while (true) {
    System.out.println("waiting for connection at 9000");
    Socket connectionSocket = welcomeSocket.accept();
    System.out.println("connection established from "+connectionSocket.getInetAddress());
    InputStreamReader isrServer = new InputStreamReader(clientSocket.getInputStream());
    OutputStream os = clientSocket.getOutputStream();
    BufferedReader inFromClient = new BufferedReader( isrServer );
    DataOutputStream outToClient = new DataOutputStream( os );
    String messageFromClient = inFromClient.readLine();
    outToClient.writeBytes(messageFromClient.toUpperCase() + "\n");
    connectionSocket.close();
   }
 }
}
```

Osvetlili bomo samo tiste dele, ki se razlikujejo od `TCPClient.java`.

```
ServerSocket welcomeSocket = new ServerSocket(9000);
Socket connectionSocket = welcomeSocket.accept();
```

V prvi vrstici strežnik odpre vtičnico `welcomeSocket` tipa `ServerSocket` na portu `9000`. Ta vtičnica je *sprejemna* vtičnica, ki bo sprejemala vse prošnje za povezavo. Ko klient "potrka" na vtičnico `welcomeSocket`, mu strežnik dodeli novo vtičnico za komunikacijo in jo zapiše v

[1] Bodite pozorni na tole dejstvo, ker je zelo pomembno za reševanje naloge.

spremenljivko `connectionSocket`, ki je tipa `Socket` (druga vrstica). Vsa nadaljna komunikacija s klientom bo tekla preko vtičnice `connectionSocket`, zato moramo (podobno kot pri klientu) tudi nanjo priklopiti vhodno/izhodne komunikacijske tokove. To se izvede s spodnjimi štirimi vrsticami.

```
InputStreamReader isrServer = new InputStreamReader(clientSocket.getInputStream());
OutputStream os = clientSocket.getOutputStream();
BufferedReader inFromClient = new BufferedReader( isrServer );
DataOutputStream outToClient = new DataOutputStream( os );
```

Strežnik nato preko vtičnice prebere *eno vrstico znakov* (t.j., do znaka `\n`), sporočilo pretvori v velike črke z metodo `toUpperCase()` in spremenjeno sporočilo *(ter znak za novo vrstico)* odda klientu preko toka `outToClient` z metodo `writeBytes(...)`.

```
String messageFromClient = inFromClient.readLine() ;
outToClient.writeBytes(messageFromClient.toUpperCase() + "\n") ;
```

3.3 Vaš prvi zagon programov

Sedaj, ko natančno razumete programa `TCPClient.java` in `TCPServer.java`, lahko poskusite zadevo zagnati. Sledite spodnjim točkam.

- Odprite dve Windows konzoli. V prvi konzoli boste poganjali klienta, v drugi pa strežnik. V obeh konzolah se postavite na mapo, kamor ste prenesli datoteki `TCPClient.java` in `TCPServer.java`.
- V prvi konzoli prevedite program za strežnik z ukazom "`javac TCPServer.java`" (brez narekovajev), nato pa ga zaženite z ukazom "`java TCPServer`" (brez narekovajev).
- V drugi konzoli prevedite in zaženite klienta. Če klient/strežnik deluje kot pričakovano, potem nadaljujte z nalogo.
- Spremenite program klienta tako, da bo strežniku poslal z malimi črkami napisano ime vaše skupine (npr., kac1s1), in preverite delovanje programa. V primeru pravilnega delovanja programa, naredite izpis konzole in ga dodajte v poročilo.

3.4 Opazovanje komunikacije v Wiresharku pod Windows

Do te točke ste uspeli zagnati tako klienta, kakor strežnik na istem računalniku. Ker sta oba zagnana na istem računalniku, klient pošilja pakete na privzeto številko IP `127.0.0.0` (imenovano tudi `localhost`). Pod Linuxom lahko vajo kar nadaljujete z opazovanjem prometa v Wiresharku, pod Windows (če delate doma) pa to ne bo šlo. Problem je v tem, da zaradi posebne implementacije protokola v okolju Windows ne moremo brez dodatnih posegov opazovati pretoka prometa skozi to številko. Zato lahko nalogo pod Windows izvedete le na naslednji način:

1. Spremenite kodo v klientu tako, da se bo priklapljal na oddaljeni strežnik. Spremeniti morate številko IP in port. Zamenjava porta bi morala biti trivialna, IP številko pa zamenjate tako, da namesto `localhost` v narekovajih napišete kar željeno številko.
2. Prevedite vašo kodo, zaženite strežnik na drugem računalniku in izvedite dostop na ta računalnik.

Če delate pod Linuxom (torej na vajah v učilnici) ali pa ste ste uspeli modificirati program po zgornjih navodilih tako, da teče na drugem računalniku, potem imate vse pripravljeno za spremljanje povezave v Wiresharku. Sledite spodnjim točkam in odgovorite na vprašanja (kjer je mogoče, naredite tudi izpis iz Wiresharka).

1. Zaženite Wireshark in pričnite z zajemom paketov, ki tečejo preko aktivnega mrežnega vmesnika.
2. Zaženite program `TCPClient.java`. Ko program konča, ustavite zajem paketov v Wiresharku.

3. Najprej moramo prepovedati Wiresharku sestavljanje razdeljenih TCP paketkov tako, kot v eni od predhodnjih nalog. Pojdite na *Edit→Preferences→Protocols→TCP*, in spremenite kljukice tako, da bo odkljukano samo *Analyze TCP sequence numbers* in *Relative sequence numbers*.
4. Ker nas bodo zanimali samo TCP segmenti poslani med strežnikom in klientom, v filter vpišite izraz, ki bo prikazoval samo paketke poslane ali prejete z IP naslova računalnika, na katerem teče strežnik. V polje *filter* vpišite izraz s sintakso `ip.addr==<IP naslov ponora>`.
5. Če ste vse naredili prav, potem Wireshark prikazuje približno enajst TCP paketkov.
6. Z analizo paketkov izpišite IP ter port na katerem je priklopljen program strežnik. Izpišite tudi IP ter port na katerem je priklopljen klient.
7. Kaj pomeni angleški izraz *piggy-back* v žargonu mrežnih protokolov, posebej pri protokolu TCP? Če ne veste, poglejte na splet, oziroma v knjigo [1].
8. Nekateri paketki, ki so prikazani v Wiresharku, so del vzpostavitve povezave (tristransko rokovanje), nekateri paketki vsebujejo podatke poslane med strežikom in klientom, nekateri paketki so potrditveni paketki, nekateri paketki pa so del porušitve povezave. Vse paketke si izpišite po vrsti in za vsakega opišite njegovo nalogo z enim stavkom (če se slabo spomnite teorije o vzpostavitvi in porušitvi povezave pri TCPju, glejte spletno literaturo in [1]). Če paket nosi sporočilo, potem to sporočilo tudi izpišite pri njegovem opisu. Za vsak paketek navedite njegovo izvorno številko IP in port, kakor tudi ponorno številko IP in port.
9. V razmislek: razmislite kako bi kar najpregledneje opazovali poslane podatke med strežnikom in klientom (namig: prikaz smo uporabili v eni od prvih vaj).
10. Izpostavite TCP segment, ali več segmentov, ki so bili poslani s klienta na strežnik in prenašajo ime vaše skupine (ozirom, sporočilo, ki ste ga poslali). Spomnite se, da klient ne pošlje samo vašega imena, pač pa mu doda znak `\n`. Najdite ta znak v ustreznem segmentu in naredite ustrezno sliko iz Wiresharka, ki jo vključite v poročilo.

3.5 Do naslednjih vaj...

Ne pozabite, da boste na naslednjih vajah reševali naloge, ki zahtevajo nekaj predzananja. To predzanaje črpate iz dosedanjega dela na vajah in teorije, ki jo morate predelati do naslednjič. Do naslednjič si preberite o delovanju zelo preprostega šifriranja, ki mu pravimo *Cezarjevo šifriranje* (angl., Ceaser cipher). Kratek opis najdete v [1] ali na strani `http://en.wikipedia.org/wiki/Caesar_cipher`.

Pri naslednjih vajah bo zelo pomembno, da vam je jasno, kako v Javi poslati neko sporočilo na strežnik in kako s strežnika prebrati vrstico poslanega sporočila. Če ti postopki (oziroma, ukazne vrstice) niso jasni, potem si ponovno poglejte programe, ki ste jih uporabljali danes. Priporočamo, da jih poskusite spremeniti po svoje.

Za vajo poskusite spremeniti program klient in program strežnik tako, da si znotraj ene seje komunikacije strežnik in klient izmenjata več sporočil (v trenutnem programu vsak pošlje in prejme samo eno sporočilo). Poskusite implementirati naslednji primer:

- Strežnik čaka na klienta.
- Klient kontaktira in se priklopi na strežnik. Ob priklopu izpiše na ekran "Vzpostavil sem sejo s strežnikom".
- Klient pošlje strežniku sporočilo A_1.
- Strežnik prejme sporočilo A_1, ga izpiše na ekran in odgovori s sporočilom B_1.
- Klient prejme sporočilo B_1, ga izpiše na ekran, in pošlje strežniku sporočilo A_2.
- Strežnik prejme sporočilo A_2, ga izpiše na ekran in poruši sejo.
- Klient zazna porušitev seje, izpiše na ekran "seja končana"in konča izvajanje.

Vsebino sporočil A_1, A_2, B_1 in B_2 si sami izmislite.

Literatura

[1] J.F. Kurose and K.W Ross, *Computer networking – a top-down approach*, Addison Wesley, 2009.