Lecture Rechnernetze I
Winter 2006/2007

Peter Martini, Matthias Frank, Nils Aschenbruck,
Christoph Fuchs, Sascha Lettgen, Patrick Peschlow, Lukas Pustina

Practical Assignment Sheet #1

Release date: October 24, 2006. Presentation planned between December 4 – 8, 2006.

Information about practical exercises:

- Successful participation in both practical exercises is mandatory for the admission to the oral exam.
- The practical exercises are to be solved and presented in groups of two students.
- The presentation will take place between December 4 – 8, 2006.
- You will have to register for the presentation between November 20 - 24 using the online-form on:
  http://web.informatik.uni-bonn.de/IV/martini/Lehre/Anmeldung/Prakt-Aufgabe.html
- During the presentation, your program has to run error-free. Furthermore, you must be able to answer questions regarding concepts, design decisions, and implementation issues. Detailed information concerning the presentation will be issued on a separate sheet.

Practical exercise 1: (Peer-to-peer file sharing)
The goal of this exercise is to implement a peer-to-peer system (consisting of one server and several clients) for the transmission of arbitrary (binary) files. The server is supposed to be developed in C for Linux whereas the client shall be implemented in Java. The transmission of data as well as the exchange of control information is conducted using TCP. Additionally, UDP is used for server discovery on the network.

Information concerning UNIX network programming can be found in [1] and in the slides to our lecture *Programmierung II* [2]. For Java programming see the comprehensive collection of tutorials and examples on [3] and the lecture slides respectively. Copied extracts of the book of Stevens are available in our secretary office.

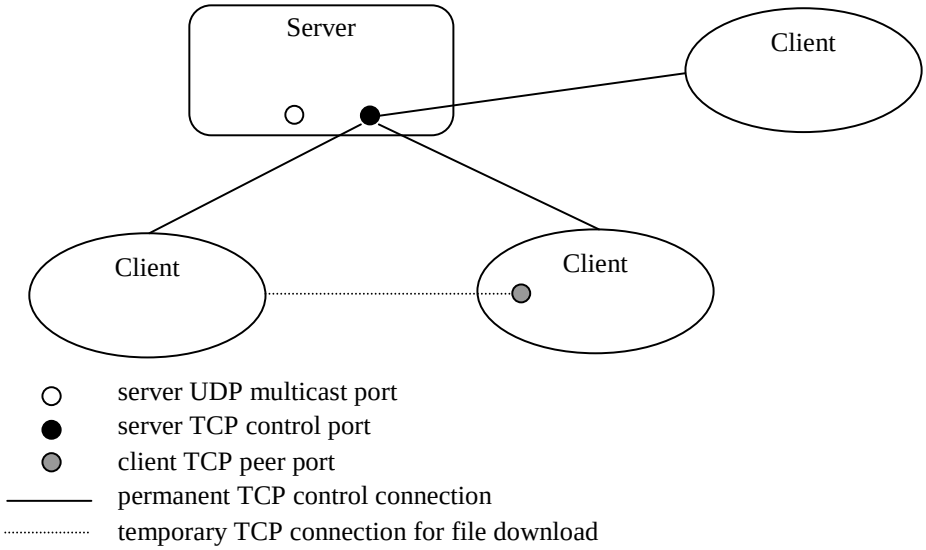
Functional overview:
Each client provides a number of files for download by other clients. The server acts as a broker for the clients and maintains a TCP connection to each of them in order to exchange status information (e.g. file lists). Each client offers an UI that displays a list of the provided files of all clients registered with the server. The UI allows to select a file to download. The download is then performed directly between the requesting and the providing client.

Feature requests:

1) The server has to support an arbitrary number of clients.
2) Clients discover a server by sending a UDP multicast message.

3) In order to download a file, the client directly connects to the providing client using TCP's "active open". The client specifies the filename whereupon the peer transmits the file. At the client side, the progress is displayed and the transmitted file is saved to disk.

4) The client's UI shall display the list of available files including the providing client and the filesize for each file. The UI must offer the ability to select a file for download.



Implementation requests:

1) The server is permanently listening for incoming control connections on a TCP port assigned by the operating system. The clients connect to that port permanently and use the connection only for exchanging status information.

2) Additionally, the server is permanently listening on multicast address 224.0.0.42 and a (UDP-) port of your choice. It answers each request with a message containing the abovementioned TCP port the client should use to establish the control connection.

3) The control connection is used by the client to submit a list of files it offers for download as well as the TCP port other peer clients have to use for downloading files. In turn, the server uses the control connection to promote the cumulative list of offered files to each client.

4) In order to provide its files to other peers, the client has to listen on a TCP port assigned by the operating system that is only used for the file transfer.

5) As only a very limited amount of data is exchanged between the clients and the server, the latter shall be implemented as an iterative server (do not use "fork()" or threads here). To multiplex multiple connections you should use "select()".

6) The client, on the other hand, shall be able to send a file concurrently, so that the user is not affected (here, "threading" could be very useful!). In contrast, receiving a file shall not be concurrent (cf. feature request 3: a progress bar shall be displayed).

7) Whenever a client sends its file list to the server, the server has to promote the updated file list to all connected clients.

8) The messages exchanged between the clients and server are composed of readable ASCII text strings and are specified in the following (each message must be terminated with a line break):

Messages from client to server:

`server_discovery\n`	Sent to multicast address to discover the server
`register <client tcp peer port>\n`	Registration at server. Client provides its listening TCP peer port for download connections from other clients. Example: `register 6556`
`send_filelist <number of files>\n`	Initializes the transmission of the client's file list
`unregister\n`	Client logs off from server

Messages from server to client:

`discovery_reply <server tcp port>\n`	Sent in response to server_discovery, providing the server's TCP port for control connections
`update_filelist <number of files>\n`	Initializes the server driven update of the file list

Messages from client to client:

`get <filename>\n`	requests the specified file from another client

The list of files shall be transferred between client and server as a multi-line ASCII stream where each file corresponds to one row and is terminated with a line break using the format specified below:

Server to client:

```
<filesize> <ip address of hosting client> <tcp peer port of hosting client>
<filename>\n
```

Client to server: (ip address and port have already been sent within the register message)

```
<filesize> <filename>\n
```

Example (server to client):

```
2145 131.220.242.115 6568 myfile1.txt\n
127 131.220.242.115 6568 myfile2.txt\n
                    ...
```

References:

[1] W. Richard Stevens, UNIX Network Programming, Volume 1, 2nd Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998.

[2] Lecture slides to “Programmierung II”, SS 05. Available in the assignment sheet section of this year’s Rechnernetze I in our media library. http://web.informatik.uni-bonn.de/IV/martini/Lehre/index.html

[3] The Java Tutorials. http://java.sun.com/docs/books/tutorial/index.html