TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Intelligente Netze und Management verteilter Systeme
Prof. Anja Feldmann, Ph. D.
Harald Schiöberg, Bernhard Ager, Vlad Manilici

2. Blatt Protokolldesign WS 08/09

Aufgabe 1: (60 points) Einfacher Client und Server

Programmiere einen Dateitransferklienten, und einen Dateitransferserver. Als Programmiersprache kannst Du eine beliebige Sprache **außer Java** einsetzen. In Perl ist die Verwendung des Moduls IO::Socket zur Maximierung des Trainingseffektes nicht erlaubt. Der Client und der Server sollten ein gemeinsames Programm sein, das an der Kommandozeile unterscheidet, was es tun soll.

Dabei sollen für Client-, und Server-Funktionalität jeweils eigene Funktionen erstellt werden!!

(a) Der Client soll folgende Eigenschaften haben:

- Man soll ihn mittels folgender Kommandozeile bedienen können:
  `transfer <rechner> <port> <datei>`.
- Er soll sich dann mit dem genannten Rechner und Port verbinden und mittels
  `get <datei>`
  die Datei anfordern. Alle Zeilen werden mit `\n` beendet.
- Der Server wird `200 <Anzahl> Bytes` antworten, dann mit einer leeren Zeile und danach die Datei schicken, sofern sie da ist, oder `5<xx> <Fehlermeldung>`, sofern nicht.
- Der Client soll die Datei auf die Standardausgabe ausgeben, sofern kein Fehler auftrat, oder eine passende Fehlermeldung auf die Fehlerausgabe schreiben. Anschließend soll er sich mit einem passenden Exitcode beenden.

Die Client-Funktionalität soll dabei in eine eigenen Funktion gepackt werden, die Ziel-IP und Ziel-Port als Parameter erwartet.

(b) Der Server soll folgende Eigenschaften haben:

- Man soll ihn mittels folgender Kommandozeile bedienen können: `transfer -listen <port>`.
- Er soll dann auf dem genannten Port auf Verbindungen warten.
- Wenn ein Client eine Anfrage, wie oben beschrieben schickt, soll er sie entsprechend beantworten, sofern eine entsprechende Datei im aktuellen Directory liegt.
- Wenn der Dateiname ein `/` enthält, soll er `551 Current directory only` ausgeben.
- Wenn er keine passende Datei findet soll er `550 No such file or directory` ausgeben.

Die Server-Funktionalität soll dabei in eine eigenen Funktion gepackt werden, die den Port als Parameter erwartet.

Abzugeben sind: das Programm als gut dokumentierter Quellcode und gegebenenfalls als compilierte Version.

Aufgabe 2: (40 points) Einfacher Proxy für das Dateitransferprogramm

Jetzt soll das Dateitransferprogramm so *erweitert* werden, daß es auch als Proxy fungiert, mit dem sich ein Client verbinden kann, um den Dateitransfer über ihn abzuwickeln.

Zu diesem Zwecke führen wir folgende Modifikationen ein:

- Der Client kann mit `transfer <proxy> <port> <server> <port> <datei>` gestartet werden, und verhält sich wie ein Client aus Aufgabe 1. Allerdings wird eine Verbindung zum Proxy statt zum Server aufgebaut und die Anfrage muß lauten
  `get <hostname> <port> <weiterer String>`

- Wenn ein Proxy ein Kommando dieser Form erhält, öffnet er eine Client-Verbindung mit `hostname` und gibt ein `get <weiterer String>` an `hostname` weiter. Die vom Server eintreffenden Daten oder Fehlermeldungen werden dann vom Proxy an den Client durch gereicht.
- Wenn der Proxy keinen Server des angegebenen Namens kennt, antwortet er mit `553 <hostname1>: <hostname> not known`, wobei `<hostname1>` sein eigener Hostname ist.

Die Proxy-Funktionalität soll dabei als Erweiterung der Server-Funktion implementiert werden und von der Client-Funktion Gebrauch machen. Dazu muß die Server-Funktion mit zusätzlichen Parametern (mindestens 2, für Proxy-IP und -Port!) umgehen können.

Abzugeben sind: das Programm als gut dokumentierter Quellcode und gegebenenfalls als compilierte Version.

Details zur Abgabe der Aufgaben:
siehe FAQ: `http://www.net.t-labs.tu-berlin.de/teaching/ws0809/PD_labcourse/faq.shtml`
Abgabedatum: Mittwoch, 5.11.2008, 23:59 s.t.