Fortgeschrittene Techniken der Funktionalen Programmierung WS 04/05

Christoph Lüth <cxl>

2. und 3. Übungsblatt

Ausgabe: 06.12.04

Abgabe: 05.01.05/19.01.05

Dieses ist ein Doppelübungsblatt: die erste Aufgabe ist das zweite, die zweite Aufgabe das dritte Übungsblatt. Beide stehen allerdings in engen inhaltlichen Zusammenhang.

In diesen Aufgaben geht es darum, das in der Vorlesung vorgestellte `chat`-System zur Marktreife zu entwickeln, um mit den erhofften Einnahmen die Weihnachtseinkäufe des Veranstalters auszulegen. Insbesondere soll folgende Funktionalität hinzugefügt werden:

- Eine Benutzerverwaltung, bei der sich Benutzer registrieren und anmelden (wir wollen schon wissen mit wem wir reden);
- Ein echter Benutzerclient (mit `telnet` hat noch niemand den internationalen Unterhaltungsmarkt aufgerollt);
- Ein Nachrichten-Service (kann auch benutzt werden, um Werbung einzuspielen);
- Verschlüsselung und Authentifizierung (damit das ganze auch für Führungskräfte interessant wird, die über streng vertrauliche Geschäftsgeheimnisse plaudern möchten, und denen wir viel Geld eine *executive platinum version* andrehen können).

Im einzelnen sieht das Einsatzszenario wie folgt aus:

1. Bevor man das System zum ersten Mal benutzt, muss man sich mit vollständigen Benutzerdaten (Name, Vorname, Adresse (Stadt oder Stadtteil), ggf. Rufname, öffentlicher Schlüssel) bei der Benutzerverwaltung registrieren. Hierbei dienen der vollständige Name und Vorname zusammen mit der Stadt und ggf. dem Rufnamen der eindeutigen Identifizierung, d.h. es darf nicht zwei registrierte Benutzer mit denselben Daten geben. Dann erhält man von dem System einen eindeutige (aber nur intern verwendete) Benutzerkennung.
2. Mit dieser Benutzerkennung kann man sich anmelden. Nach der Authentifizierung kann man:
   - Sich mit allen anderen Benutzern unterhalten, wie in dem `talk`-Beispiel;
   - Nach anderen registrierten Besuchern suchen;
   - Anderen Benutzern Nachrichten schicken, die dann direkt weitergeleiten werden, wenn der Benutzer gerade angemeldet ist, oder ansonsten gespeichter und weitergeleitet werden, sobald der Benutzer sich anmeldet. Wenn ein Benutzer eine Nachricht liest, wird dem Sender (ggf. nur optional) eine Bestätigung geschickt (auch wieder als Nachricht).

Benutzer werden grundsätzlich nicht an ihrer Benutzerkennung, sondern durch ihren vollen Namen referenziert, ggf. mit Stadt, Adresse, Rufname eindeutig qualifiziert, also *Thomas Schaaf hat sich angemeldet*[1], oder *Nachricht von Hannes Müller (Bremen).*

2 *Der Server* — *20 Punkte*

In der ersten Aufgabe implementieren wir den Server. Der Server sollte alle die oben beschrieben Aufgaben erfüllen. Daten sollten persistent gespeichert werden, d.h. wenn der Server abstürzt (oder regulär beendet wird) sollten Nachrichen, Benutzerdaten etc. nicht verloren gehen.

Zur Authentifizierung und Verschlüsselung existieren Büchereien für RSA in Haskell: die Verschlüsselung der Verbindung ist optional, aber jeder Benutzer muss sich authentifizieren.

Wenn ein registrierter Benutzer sich an- oder abmeldet, soll eine Rundnachricht an alle Benutzer verschickt werden; bei der Anmeldung eines Client sollte der Server diesem die Liste aller angemeldeten Benutzer schicken.

In der ersten Abgabeversion (zum 05.01.05) sollte der Server alleine laufen, mit nur einem `telnet` auf der anderen Seite als Client. Die Benutzerführung auf Seiten des Clients kann beliebig unintuitiv sein, das ist Gegenstand der nächsten Aufgabe.

3 *Der Client* — *20 Punkte*

In der zweiten Aufgabe implementieren wir den Client. Der Client kann entweder in einem einfachen Konsolenfenster laufen, oder mit einer graphischen Benutzerschnittstelle.

1. Für die Konsolenschnittstelle solltet ihr die wesentlichen Funktionen aus der `ncurses`-Bücherei mit dem *foreign function interface* in Haskell verkapseln. `ncurses` erlaubt die portable Programmierung von Ein- und Ausgabe in rein zeichenbasierten Umgebungen (zum Beispiel auf einer Konsole, oder innerhalb eines XTerm). Leider gibt es `ncurses` nur für Linux/Unix, und meines Wissens nicht (oder nur kommerziell) für Windows.

2. Für eine graphische Benutzerschnittstelle könnt ihr auf die `wxHaskell`-Bücherei zurückgreifen. `wxHaskell` ist portabel und läuft unter Linux, Unix, Windows und sogar Mac OS X, ist allerdings nicht völlig trivial zu installieren.

Die Gestaltung der Benutzerschnittstelle ist nicht fest vorgebenen, aber in etwa sollte es folgenden Richtlinien folgen:

- Wenn es in einem Terminal läuft, sollte es das ganze Terminalfenster (bzw. den ganzen Bildschirm der Konsole) nutzen, ansonsten ein eigenes Fenster öffnen.

- Im größeren, oberen Teil des Bildschirms sollten die Nachrichten der anderen Benutzer angezeigt werden (sowohl private Nachrichten als auch die an alle gerichteten). Die Nachrichten sollten durchlaufen (scrollen), wobei man auf die durchgelaufenen Nachrichten noch zugreifen können soll (z.B. indem man auf Tastendruck zurückblättern kann, oder graphisch).

[1] Das wäre eindeutig, denn es gibt, wie wir wissen, nur einen Thomas Schaaf.

- Im unteren, kleineren Teil (vielleicht nur eine Zeile) tippt der Benutzer Kommandos und Nachrichten. Wie genau Kommandos und Nachrichten eingegeben werden, ist nicht festgelegt (in einer graphischen Schnittstelle wahrscheinlich über ein Menue), aber es muss Möglichkeiten, folgendes einzugeben:
  - Nachrichten an alle Benutzer;
  - Privatnachrichten an einzelne Benutzer;
  - Suche nach anderen Benutzern;
  - Verbindung zum Server aufbauen und abbauen.
- Außerdem sollte dann auch noch eine Statuszeile existieren, die anzeigt, ob eine Verbindung zum Server besteht, Verbindungsdaten (Anzahl Nachrichten), Anzahl Nutzer im System und andere nützliche Informationen (hier kann man auch Werbung einblenden).

Der Client sollte auch die Schlüssel des Benutzers verwalten. Es muss nur eine Identität unterstützt werden, d.h. es kann davon ausgegangen werden, dass sich die Daten des Benutzers nicht verändern; sie können zum Beispiel aus einer Konfigurationsdatei ausgelesen werden.

Anhang: Referenzen

RSA und andere kryptographischen Algorithmen in Haskell:

- `http://www.electronconsulting.com/rsa-haskell/`
- `http://www.haskell.org/crypto/ReadMe.html`

Curses:

- `man ncurses`. Ihr benötigt im wesentlichen nur Funktionen für die Initialisierung, die Positionierung des Cursors, die Erzeugung (und Verwaltung) von scrollbaren Teilbereichen des Bildschirms (sog. „Windows“) und Eingabe.

wxHaskell:

- `http://wxhaskell.sourceforge.net/`

Dies ist Version 1.0 vom 2004/11/15 15:12:40.