



Praktikum Kommunikationssysteme

Aufgaben 2: IBR-Chat (IBRC)

Letzter Abgabetermin: in der Woche vor dem 28. Mai

1 Aufgabenstellung

Die folgende Aufgabe besteht aus zwei Teilen. Der erste dient dazu, ein Protokoll zu spezifizieren, das für alle Gruppen bindend ist. Im zweiten Teil dieser Aufgabe soll dieses Protokoll implementiert werden und zwar in einem Client-Server-basierten Programm.

Es ist ein einfaches Chat-Programm namens IBRC (*IBR Chat*) zu entwickeln, welches sich an das bekannte IRC (*Internet Relay Chat*) anlehnt. Das IBRC ist allerdings deutlich einfacher gestaltet als das originale IRC und weist ihm gegenüber einige Modifikationen auf.

IRC stellt ein System zur Durchführung von Textkonferenzen dar und bietet dem Benutzer die Möglichkeit, sich mit anderen Personen via Rechner zu „unterhalten“, d. h. geschriebenen Text auszutauschen. Hierzu existieren verschiedene „Diskussionsrunden“ (Kanäle genannt), in denen Teilnehmer über bestimmte Themen ihrer Meinung freien Lauf lassen können. Die Anzahl der Teilnehmer pro Diskussionsrunde wird nur durch die verfügbaren Ressourcen begrenzt.

IRC ist Client-Server-basiert, d. h. das Programm besteht aus zwei Teilen, einem Client und einem Server. Es wird im Detail in [2] beschrieben, wer Spaß daran hat, kann mal einen Blick in diesen RFC riskieren (RFC-Dokumente können am Institut mit dem Kommando `rfc <nummer>` gelesen werden).

Im folgenden werden kurz die grundlegenden Konzepte des IBRC erläutert, bevor die zu bearbeitenden Aufgaben im Detail vorgestellt werden.

2 Clients

Als Clients werden diejenigen Rechner bezeichnet, an denen ein Teilnehmer der Diskussionsrunde sitzt, d. h. sie bilden die eigentliche Talkrunde. Jeder Client unterhält eine Verbindung zu genau einem Server. Zu diesem werden alle vom Client eingegebenen Daten gesendet.

Der Server muß dann über deren weitere Verwendung bzw. deren Verarbeitung entscheiden. Identifiziert werden Clients durch den Namen des entsprechenden Rechners (Hostname). Die Teilnehmer werden normalerweise nicht durch ihre wahren Namen identifiziert, sondern durch sogenannte *nicknames* (Spitznamen), welche sie sich frei aussuchen können. Diese Spitznamen dürfen max. neun Zeichen lang sein und müssen aus Buchstaben oder Ziffern bestehen.

3 Server

Server erfüllen im wesentlichen zwei Aufgaben. Zum einen nehmen sie alle Daten von den angeschlossenen Clients entgegen, um diese weiterzuleiten oder zu bearbeiten. Die Teilnehmer (in Form von Clients) kommunizieren also niemals direkt miteinander, sondern immer über einen oder mehrere Server.

Zum anderen muß jeder Server Informationen über die Clients im Netzwerk verwalten. Dies ist u. a. notwendig, um die Kontrolle über alle existierenden Teilnehmer bzw. Kanäle zu behalten (zur Beschreibung der Kanäle s. nächster Abschnitt). Es muß beispielsweise verhindert werden, daß ein Teilnehmer mehr als einem Kanal gleichzeitig angehört oder daß zwei Kanäle dieselbe Bezeichnung erhalten.

Weitere Einzelheiten zum Ablauf der Kommunikation werden im Abschnitt 6 gegeben.

4 Kanäle

Unter einem Kanal kann man sich eine Gruppe von Teilnehmern vorstellen, die untereinander Nachrichten austauschen. Die Nachrichten eines Teilnehmers werden an alle Teilnehmer, die diesem Kanal angehören, verschickt. Ein Teilnehmer kann nicht mehreren Kanälen gleichzeitig angehören. Wenn sich der allererste Teilnehmer einem Kanal anschließt, d.h. wenn der Kanal noch nicht existiert, so wird dieser erzeugt. Wichtig ist, daß sich der Zustand eines Kanals dynamisch ändern kann, etwa wenn neue Teilnehmer hinzukommen. Diese Änderung muß vom entsprechenden Server bemerkt und die notwendigen Informationen an alle weiteren Server im Netz gesendet werden.

Kanäle werden (etwas abweichend vom originalen IRC) durch Kurzbezeichnungen identifiziert, z.B. „Ozonloch“. Diese Bezeichner sind max. 12 Zeichen lang und müssen eindeutig sein, d.h. es darf keine zwei Kanäle mit derselben Bezeichnung geben. Die Bezeichner werden bei allen Befehlen verwendet, um einen Kanal eindeutig zu identifizieren. Um beispielsweise dem Kanal „Ozonloch“ beizutreten, muß ein Teilnehmer den Befehl `JOIN Ozonloch` eingeben. Darüber hinaus existiert für jeden Kanal eine ausführlichere Beschreibung (Thema) des diskutierten Inhalts. Das Thema des Kanals „Ozonloch“ könnte z.B. lauten: „Meteorologische Auswirkungen der schwindenden Ozonschicht oder wir werden langsam geröstet“.

Das Thema kann ausschließlich von demjenigen Teilnehmer, der den Kanal erzeugt hat, bestimmt werden, wobei auch nachträgliche Änderungen möglich sind. Dieses Recht auf Themenänderung ist nicht auf andere Teilnehmer übertragbar.

5 Nachrichten und Befehle

Bei den Eingaben, die der Benutzer des IBRC vornimmt, ist prinzipiell zwischen Nachrichten und Befehlen zu unterscheiden. Nachrichten entsprechen geschriebenen Worten, die an alle Teilnehmer des zugehörigen Kanals gesendet werden. Sie sind zeilenbasiert, d.h. nach Betätigen der Return-Taste wird die aktuelle Zeile an den angeschlossenen Server gesendet, der sie an die entsprechenden Clients oder ggf. an weitere Server weiterleitet.

Befehle dienen zur Steuerung der Kommunikation für die Teilnehmer, z.B. um einem Kanal beizutreten, Informationen über andere Teilnehmer einzuholen etc. Sie werden durch einen vorangestellten Schrägstrich (/) gekennzeichnet, wobei die Groß-/Kleinschreibung keine Rolle spielt.

Alle Befehle haben die Form `<Befehl> [<Parameter1> [<Parameter2>]]`.

Einen Überblick über alle Befehle des IBRC gibt Tabelle 1. Die exakte Beschreibung befindet sich in der Aufgabenstellung zu Teil 2, da erst dort die Befehle zu implementieren sind.

Befehl	Beschreibung
JOIN	Einem Kanal beitreten
LEAVE	Aktuellen Kanal verlassen
NICK	Eigenen Nickname ändern
LIST	Alle existierenden Kanäle auflisten
GETTOPIC	Beschreibung eines Kanals ausgeben
SETTOPIC	Beschreibung des aktuellen Kanals setzen
PRIVMSG	(Private) Nachricht an einzelnen Teilnehmer senden
QUIT	IBRC-Tool verlassen

Tabelle 1: IBRC – Befehle

6 Kommunikation

Im IBRC sind zwei Teilnehmer (Clients) niemals direkt miteinander verbunden, sondern immer über einen oder mehrere Server. Ein Client ist immer mit genau einem Server verbunden. Die Server haben dafür Sorge zu tragen, daß eine Nachricht, welche in einem Kanal versendet wird, an alle Teilnehmer dieses Kanals weitergeleitet wird. Wichtig ist hierbei, daß vom Client die Nachricht nur *einmal* an den angeschlossenen Server geschickt wird (auch wenn an diesem noch weitere, dem Kanal angehörende, Clients angeschlossen sind). Der Server entscheidet, an welche Clients und ggf. an welche weiteren Server die Nachricht weiterzuleiten ist. Es ist selbstverständlich, daß die Nachricht nur an diejenigen Server gesendet wird, die die Nachricht benötigen, um sie ihrerseits an weitere, dem entsprechenden Kanal angehörende, Clients zu senden.

Darüber hinaus obliegt den Servern die Bearbeitung der Befehle, die die Teilnehmer eingeben können. Hierzu ist eine Speicherung der Daten über alle existierenden Teilnehmer, Kanäle etc. notwendig. Wichtig ist, daß *alle* Server im Netz ihre Daten synchronisieren, d.h. daß alle

Server stets über die gleichen Daten verfügen. Es ist klar, daß hierzu eine Kommunikation der Server untereinander notwendig ist. Im originalen IRC werden hierfür eigene Server-Befehle definiert. Für dieses Praktikum ist nichts derartiges vorgeschrieben, es bleibt den Teilnehmern überlassen, wie sie die Serverkommunikation realisieren.

6.1 Topologie des Chat-Netzwerks

Das aus Clients und Servern bestehende IBRC-Netzwerk soll eine Baumstruktur aufweisen und ist somit zyklensfrei. Die genaue Topologie kann von den Teilnehmern frei gewählt werden. Es muß allerdings möglich sein, in das Netzwerk mindestens fünf Server und 10 Clients integrieren zu können. Die geforderte Zyklensfreiheit kann dadurch sichergestellt werden, daß sich neue Server nur an *einen* bereits bestehenden Server anschließen dürfen.

Wenn die Servertopologie feststeht, dürfen sich Clients an beliebige Server anschließen, wobei ein Client natürlich nur an *einen* Server angeschlossen sein darf. Nachdem der erste Client in das Netzwerk integriert wurde, ist die Topologie für die Server fest, d.h. es dürfen keine Verbindungen zwischen Servern neu gebildet oder entfernt werden. Das Einfügen neuer Server ist ebenfalls nicht mehr erlaubt. Es kann davon ausgegangen werden, daß niemals mehr als 10 Server und 20 Clients vorhanden sind (d.h. es können Arrays für die Speicherung verwendet werden).

6.2 Kommunikationsformen

Im IBRC (und ebenfalls im IRC) existieren zwei verschiedene Kommunikationsarten: *one-to-many* und *one-to-one*.

one-to-many: Diese Kommunikationsform entspricht der normalen kanalbasierten Form, d.h. die Nachrichten werden an alle Teilnehmer eines Kanals gesendet.

one-to-one: Es findet eine Kommunikation innerhalb eines Kanals zwischen genau zwei Teilnehmern statt. Alle anderen bemerken von dieser „verdeckten“ Kommunikation nichts. Sie kann benutzt werden, um Rückfragen zu halten, über andere Teilnehmer zu lästern etc.

Für diese Art der Kommunikation existiert der PRIVMSG-Befehl.

7 Grafische Oberfläche

Um den Programmieraufwand für die Praktikums Teilnehmer zu verringern, wird ein Programmgerüst zur Verfügung gestellt, das eine grafische Oberfläche für den Client implementiert. Diese grafische Oberfläche ist in Abbildung 1 dargestellt.

Nach erfolgreichem Kompilieren und Linken läßt sich der Client dann von der Kommandozeile mit dem Kommando `client <server> <port>` starten.

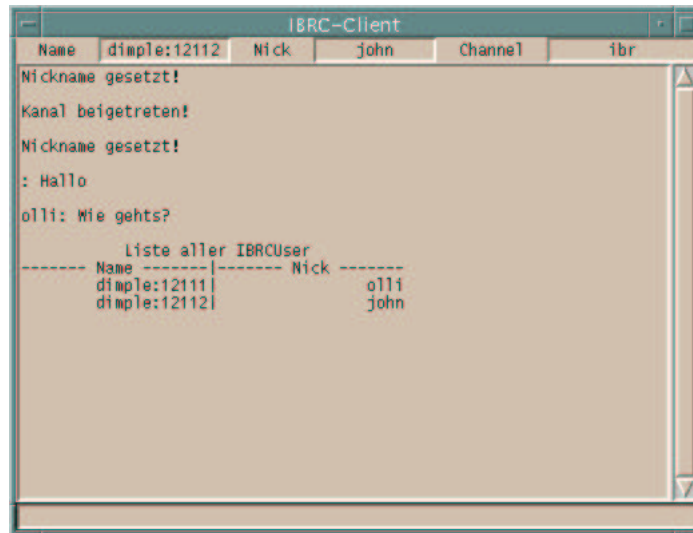


Abbildung 1: Die grafische Oberfläche des IBRC-Clients

7.1 Funktionen der grafischen Oberfläche

Das Programmgerüst stellt dem Praktikumsteilnehmern folgende Funktionen zur Verfügung:

`void bye()`

Beendet den Client. Diese Funktion ruft die in Abschnitt 7.2 beschriebene Funktion **quit()** auf, um nicht mehr benötigte Ressourcen wieder freizugeben.

`void setName(char *name)`

Trägt den angegebenen Name an der dafür vorgesehenen Stelle in die GUI ein.

`void setNick(char *nick)`

Trägt den angegebenen Nicknamen an der dafür vorgesehenen Stelle in die GUI ein.

`void setChannel(char *channel)`

Trägt den angegebenen Kanalnamen an der dafür vorgesehenen Stelle in die GUI ein.

`void printSystemMessage(char *message)`

Gibt eine Systemnachricht im Textbereich der GUI aus. Eine typische Systemnachricht ist z.B. „User XY hat den Kanal verlassen!“.

`void printOwnMessage(char *message)`

Gibt eine eigene Nachricht im Textbereich der GUI aus.

`void printMessage(char *from, char *message)`

Gibt eine Nachricht eines anderen Users im Textbereich der GUI aus.

`void printPrivMessage(char *from, char *message)`

Gibt eine private Nachricht eines anderen Users im Textbereich der GUI aus.

7.2 Zu implementierende Funktionen

Das zur Verfügung gestellte Programmgerüst ist von den Teilnehmer um die nachfolgend aufgeführten Funktionen zu erweitern (in der Datei `client.c`).

`void init(char *server, int port)`

Wird als erste Funktion nach Programmstart aufgerufen, d.h. in dieser sind notwendige Initialisierungen von Daten, das Öffnen von Sockets etc. durchzuführen. Dabei ist der erste Parameter der Name des Rechners, auf dem der Server gestartet wurde und der zweite Parameter der Port, auf dem der Server Anfragen entgegennimmt. Da diese Funktion aufgerufen wird, bevor das Fenster der GUI erzeugt wird, können die Funktionen `setName()`, `setNick()`, `setChannel()`, `printMessage()`, `printSystemMessage()`, `printOwnMessage()` und `printPrivMessage()` in dieser Funktion noch nicht aufgerufen werden.

`void quit()`

Diese Funktion wird, wie der Name verrät, unmittelbar vor Beendigung des Clientprogramms aufgerufen. Sie kann benutzt werden, um nicht mehr benötigte Ressourcen, wie z.B. Socket-Deskriptoren, dynamisch allozierten Speicher etc., freizugeben.

`void userInput(char *input)`

Wird aufgerufen, sobald der Benutzer eine Eingabe in der Eingabezeile mit **Return** abschliesst. In dieser Funktion kann die Eingabe, die in der Variable `input` übergeben wird, ausgewertet und bearbeitet werden.

`void mainloop()`

Wird periodisch aus einer Schleife des Hauptprogramms aufgerufen. Innerhalb dieser Funktion soll überprüft werden, ob Nachrichten vom Server empfangen wurden. Es ist wichtig, dass innerhalb dieser Funktion **keine** blockierenden Systemaufrufe getätigt werden oder aktiv gewartet wird, da ansonsten nicht mehr auf Eingaben des Benutzers reagiert werden kann (Tip: `select()` verwenden).

Aufgabe

1. Teil: Protokollspezifikation

Zu Beginn dieser Aufgabe sollen sich alle Gruppen ein Protokoll überlegen, das für die gesamte Kommunikation zwischen Clients und Servern eingesetzt werden kann. Diese Spezifikation soll dann von allen Gruppen bei einem gemeinsamen Treffen vorgestellt und diskutiert werden. Das Ziel dieses Treffens ist es, eine gemeinsame Spezifikation zu erstellen, die für alle Gruppen bindend ist, so dass jeder Client mit jedem Server einer anderen Gruppe kommunizieren kann.

2. Teil: Implementation

In dieser Aufgabe soll nun ein kleines IBRC-Netzwerk mit mehreren Servern implementiert werden. Das Netz muß mindestens fünf Server und 10 Clients verwalten können. Die geforderte Topologie wurde bereits in Abschnitt 6.1 beschrieben.

Die Server nehmen Daten von den angeschlossenen Clients entgegen und zwar mit Hilfe des zuvor spezifizierten Protokolls. Handelt es sich um Nachrichten, so müssen diese an den oder die entsprechenden Clients gesendet werden. Ist ein Client nicht direkt vom Server erreichbar, muß die Nachricht „geroutet“ werden, d.h. an denjenigen Server geschickt werden, von dem aus der Client erreichbar ist. Da die verwendete Topologie zyklenfrei ist, sind diese Routingentscheidungen eindeutig. Es ist darauf zu achten, daß nur diejenigen Server, die die Nachricht wirklich benötigen, sie auch erhalten. Ein „Broadcast“ der Nachrichten von jedem Server an alle weiteren, um sich das Routing zu sparen, ist nicht zulässig. Als Transportprotokoll ist TCP zu benutzen. Zudem ist das Programm protokollfamilienunabhängig auf IPv4 und IPv6 auszulegen (s. [1]).

Als Server bzw. Clients können beliebige Rechner aus dem CIP-Pool (und **nur** aus diesem) benutzt werden. Zur Kontrolle, welche Nachricht ein Server erhält und an wen er sie weiterleitet, ist auf jedem Server eine einfache Ausgabe zu implementieren, die zu jeder empfangenen Nachricht die Hostnamen des Absenders sowie des Adressaten enthält.

Zu implementierende Befehle Alle in der nachfolgenden Beschreibung aufgeführten Befehle sind zu implementieren. Es sei noch einmal daran erinnert, daß Befehlen bei der Eingabe ein Schrägstrich voranzustellen ist. Leerzeichen an Anfang, in der Mitte (zwischen einzelnen Argumenten) oder am Ende eines Kommandos sollen ignoriert werden.

JOIN < Kanal >

Mit Hilfe dieses Befehls tritt ein Teilnehmer einem Kanal bei. *Kanal* ist die Kurzbezeichnung des Kanals (z.B. „Ozonloch“). Existiert der Kanal noch nicht, so wird er neu erzeugt. Ist der Teilnehmer bereits einem anderen Kanal angeschlossen, so wird der Befehl ignoriert und es erfolgt die Ausgabe einer Fehlermeldung. Der Teilnehmer kann nur chatten, wenn er einem Kanal angeschlossen ist.

LEAVE

Hiermit wird der aktuelle Kanal verlassen. Gehört der Teilnehmer keinem Kanal an, so erfolgt die Ausgabe einer Fehlermeldung. Falls der Teilnehmer das letzte Mitglied des Kanals ist, dann wird der Kanal gelöscht.

NICK < *Name* >

Setzt den Spitznamen (nickname) des Benutzers, unter dem dieser in Erscheinung tritt. Diese Namen sind frei wählbar, allerdings dürfen keine zwei Teilnehmer den gleichen Namen erhalten. Nicknames sind max. neun Zeichen lang und müssen aus Buchstaben oder Ziffern bestehen. Fehlerhafte Eingaben werden mit einer Fehlermeldung quittiert.

LIST

Listet alle im IBRC existierenden Kanäle mit ihren Kurzbezeichnungen auf.

GETTOPIC < *Kanal* >

Gibt die ausführliche Beschreibung (Thema) zum angegebenen Kanal aus. Existiert der Kanal nicht, wird eine Fehlermeldung ausgegeben.

SETTOPIC < *Thema* >

Setzt das Thema des aktuellen Kanals. Dieser Befehl kann nur von dem Teilnehmer, der diesen Kanal erzeugt hat, ausgeführt werden. Der Versuch anderer Teilnehmer, diesen Befehl auszuführen, wird mit einer entsprechenden Fehlermeldung quittiert.

PRIVMSG < *Name* > < *Nachricht* >

Sendet eine private Nachricht an den unter *Name* angegebenen Teilnehmer. Dieser Teilnehmer muß demselben Kanal angehören, dem auch der Absender angehört. Ist dies nicht der Fall oder existiert der angegebene Teilnehmer nicht, so erfolgt die Ausgabe einer Fehlermeldung.

QUIT

Falls ein Teilnehmer der Meinung ist, genügend gequatscht zu haben, kann er mit diesem Befehl den IBRC-Client verlassen. Gehört er noch einem Kanal an, wird seine Mitgliedschaft in diesem automatisch beendet.

Literatur

- [1] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens. Basic Socket Interface Extensions for IPv6. RFC 3493, February 2003.
- [2] J. Oikarinen and D. Reed. Internet Relay Chat Protocol. RFC 1459, Merit Network Inc., NASA, May 1993.