Rechnernetze
SoSe 2012

Nils Aschenbruck;
Jan Bauer, Alexander Bothe, Matthias Schwamborn

Praktisches Übungsblatt Nr.1

Veröffentlichung: 07.05.2012
Abgabe: 31.05.2012

Allgemeine Informationen zu den praktischen Aufgaben:

- Die erfolgreiche Teilnahme an beiden praktischen Aufgaben ist für die Zulassung zur Prüfung erforderlich.
- Es wird davon abgeraten Code zu kopieren. Die Abgabe von identischem Code durch zwei verschiedene Gruppen resultiert darin, dass beide Gruppen die praktische Aufgabe nicht bestehen und somit nicht zur Prüfung zugelassen werden.
- Die Abgabe der Aufgabe muss bis zum 31.05.2012, 23:59 per E-Mail an folgende Adresse

  bothe@informatik.uos.de

  erfolgen, wobei der Code in einer Zip-Datei zu packen und als Anhang beizufügen ist. Der Betreff der E-Mail und der Name der angehängten Datei muss nach dem folgenden Schema gewählt sein:

  ReNe_SoSe_2012_PA1_Matrikelnummer1_Matrikelnummer2

  wobei Matrikelnummer1/2 die jeweiligen Matrikelnummern der beiden Gruppenmitglieder sind.
- Das zweite praktische Übungsblatt wird kurz nach Abgabe des ersten Übungsblattes erscheinen. Bitte beachten Sie, dass Sie für beide Abgaben in der gleichen Gruppe bleiben sollen. Wechseln Sie also nach der ersten Abgabe nicht von sich aus die Gruppen.
- Bei Fragen verwenden Sie bitte die Mailingliste zur Vorlesung, oder die oben angegebene E-Mail Abgabeadresse. Alternativ können Sie auch in die wöchentliche Fragestunde kommen, wobei eine kurze Problembeschreibung vorab per E-Mail wünschenswert ist.

Aufgabe 1: Chat-System

Ein normales Gespräch folgt gewissen Regeln und Abläufen. Man hat sich, möglichst im Vorfeld, auf eine Sprache geeinigt, die alle Gesprächsteilnehmer verstehen, beginnt das Gespräch mit einer Begrüßung und beendet es mit einer Verabschiedung. Auch bei der Kommunikation in verteilten Systemen ist solch ein abgesteckter Rahmen möglich und wird in einem Netzwerkprotokoll festgelegt.

Ihre erste praktische Aufgabe ist es, in einem fest vorgegebenem Rahmen, ein Chat-System, bestehend aus Server und Client, mit der Unterstützung für separate Chat-Räume, zu implementieren. Der Server muss die gleichzeitige Kommunikation mehrerer Clients ermöglichen. Dabei ist die in der Vorlesung (vgl. Kap. 0 - C-Crash-Kurs) vorgestellte Programmiersprache C zu verwenden, wobei nur Standardbibliotheken verwendet werden dürfen. Der Code ist ausführlich zu kommentieren und muss auf einem Linux-Rechner (z.B. im Raum 31/145) kompilierbar und lauffähig sein. Weiterhin soll ein „Makefile“ beigefügt werden, um sowohl Server als auch Client zu kompilieren. Nähere Informationen zur Funktion und Erstellung einer solchen Datei findet sich auf dem ersten theoretischen Übungsblatt in Aufgabe 2. Für die bidirektionale Kommunikation zwischen Server und Client(s) ist das Transportprotokoll UDP zu verwenden (vgl. Kap. 2 – Netzwerkprogrammierung).

Hinweis 1: Um die Kompatibilität der Systeme untereinander und zu unserem Testsystem zu gewährleisten, halten Sie sich bitte **genau an die Spezifikationen** für:

- das Netzwerkprotokoll
- Ordnerstruktur und Ordner-/Dateinamen
- Textein-/ausgabe
- Kommandozeilenparameter

Hinweis 2: Sie können Ihre Programme auch auf einem einzelnen Rechner testen, indem Sie als Hostnamen *localhost* (IP: 127.0.0.1) verwenden.

Hinweis 3: Für ein leichtes zeilenweise Einlesen von der Standardeingabe können Sie die Funktion „readline(...)“ von unserer Homepage verwenden, die ähnlich wie „read(...)“ funktioniert, dabei aber immer ganze Zeilen (d.h. bis zum nächsten „\n“) liest.

Allgemeines:

Bitte entnehmen Sie das genaue Format der Nachrichten und die erwartete Textausgabe aus den Tabellen am Ende des jeweiligen Abschnittes, indem Sie die `hervorgehobenen Bezeichner` nachschlagen. Alle Nachrichten beginnen zunächst mit einem Byte für die entsprechende Befehls-Id. Sofern nicht anders angegeben, sind alle Felder als „unsigned“ zu wählen. Der allgemeine Ablauf soll wie folgt aussehen:

Start des Servers:

Zunächst wird der Server im entsprechendem Ordner mittels „make“ kompiliert und dann mit dem Befehl

./rene_pa1/udp_chat_server/udp_cs <serv_port>

gestartet. Der Kommandozeilenparameter hat folgende Bedeutung:

<serv_port> Port auf dem der Server Verbindungsanfragen erwartet und bearbeitet.

Start des Clients:

Läuft der Server, so wird auch der Client, mittels folgender Eingabe auf der Kommandozeile

./rene_pa1/udp_chat_client/udp_cc <serv_addr> <serv_port> <user>

(mehrfach) gestartet. Die Parameter stehen für:

<serv_addr>	Hostname/IPv4 Adresse des Chat-Servers zu dem die Verbindung aufgebaut werden soll.
<serv_port>	Port auf dem der Chat-Server auf eingehende Verbindungen wartet (siehe oben).
<user>	Benutzername, der im Chat sichtbar sein soll.

Verbindungsaufbau:

Der Start des Clients initiiert den Verbindungsaufbau zum Server, indem eine `CL_CON_REQ` Nachricht an den Server geschickt wird und auf eine Antwort von diesem gewartet wird. Es können nun drei Fälle auftreten:

1. Hat der Client nach fünf Sekunden keine Antwort vom Server erhalten, so soll er es dreimal erneut versuchen und sich danach automatisch beenden.
2. Der Server lehnt den Verbindungswunsch des Clients ab, da bereits ein Benutzer mit dem gewünschten Namen im Chat-System vorhanden ist. Er sendet eine entsprechende `SV_CON_REP` Antwort an den Client.
3. Der Server nimmt die Verbindung an. Auch in diesem Fall wird die entsprechende `SV_CON_REP` Antwort verschickt. Diese enthält den Port, auf dem der Server zukünftige Nachrichten vom Client erwartet.

Nachrichtenformat:

Nachricht	**Id**	**Feld (Länge in Byte)**
CL_CON_REQ	1	Länge des Benutzernamens (4) Benutzername (variabel)
SV_CON_REP	2	Antwort auf Verbindungsaufbauwunsch (1): • 0 (akzeptiert) • 1 (Benutzername belegt) Port für weitere Kommunikation (4)

Textausgabe:

Nachricht	**Textausgabe**
CL_CON_REQ	Verbinde als %u zu Server %h (%s) auf Port %p. • %u: Benutzername • %h: Hostname des Servers • %s: IPv4 Adresse des Servers • %p: Port für Verbindungsanfragen des Servers
SV_CON_REP	Verbindung akzeptiert. Der Port für die weitere Kommunikation lautet %p. • %p: Port für die weitere Kommunikation mit dem Server Verbindung fehlgeschlagen. Benutzername %u bereits vergeben. • %u: Benutzername Verbindung fehlgeschlagen. Wartezeit verstrichen.

Betreten/Verlassen eines Kanals:

Ist der Client erfolgreich mit dem Server verbunden, so muss der Benutzer zunächst einen Chat-Raum betreten. Dies geschieht über die Eingabe von

/join <Chat-Raumname>

auf der Kommandozeile des Clients, was dazu führt, dass eine *CL_ROOM_MSG* an den Server geschickt wird. Existiert dieser Chat-Raum bisher noch nicht, so wird er zunächst auf dem Server erstellt. Der Benutzer wird ihm zugeordnet und eine *SV_ROOM_MSG* an alle Benutzer im Raum verschickt., die mitteilt, dass der Benutzer den Chat-Raum betreten hat.

Das Verlassen eines Chat-Raumes funktioniert analog, mittels der Eingabe des Befehls

/leave <Chat-Raumname>

und den entsprechenden Nachrichten. Beachten Sie bitte, dass nur Befehle mit einem „Slash (/)“ beginnen dürfen und ungültige Befehle ignoriert werden sollen. Weiterhin dürfen die Namen der Chat-Räume keine Leerzeichen enthalten**.**

Nachrichtenformat:

Nachricht	**Id**	**Feld (Länge in Byte)**
CL_ROOM_MSG	4	Länge des Raumnamens (4) Raumname (variabel) Aktion (1): • 0 (betreten) • 1 (verlassen)
SV_ROOM_MSG	5	Länge des Raumnamens (4) Raumname (variabel) Länge des Benutzernamens (4) Benutzername (variabel) Aktion (1): • 0 (betreten) • 1 (verlassen)

Textausgabe:

Nachricht	**Textausgabe**
CL_ROOM_JOIN	Betrete den Raum: %r. • %r: Chat-Raumname
CL_ROOM_LEAVE	Verlasse den Raum: %r. • %r: Chat-Raumname
SV_ROOM_MSG	%u hat den Raum %r betreten. • %u: Benutzername • %r: Chat-Raumname %u hat den Raum %r verlassen. • %u: Benutzername • %r: Chat-Raumname

Senden einer Nachricht:

Die eigentliche Aufgabe des Chat-Systems ist es, dass einzeilige Nachrichten zwischen den Benutzern ausgetauscht werden können. Der Benutzer gibt den Raumnamen und die Nachricht, getrennt mit einem einfachen Leerzeichen, auf der Kommandozeile seines Clients ein. Aus der Eingabe werden dann sowohl der Raumname, als auch der Inhalt der Nachricht extrahiert und in einer *CL_MSG* an den Server geschickt. Dieser wertet die empfangenen Informationen aus, erstellt eine entsprechende *SV_AMSG* und verschickt sie an alle Benutzer im betroffenen Chat-Raum. Erhält ein Client eine solche Nachricht, so gibt er sie im entsprechenden Format auf der Kommandozeile aus.

Nachrichtenformat:

Nachricht	**Id**	**Feld (Länge in Byte)**
CL_MSG	6	Länge des Raumnamens (4) Raumname (variabel) Länge der Nachricht (4) Nachricht (variabel)
SV_AMSG	7	Länge des Raumnamens (4) Raumname (variabel) Länge des Benutzernamens (4) Benutzername (variabel) Länge der Nachricht (4) Inhalt der Nachricht (variabel)

Textausgabe:

Nachricht	**Textausgabe**
SV_AMSG	%r - %u: %n • %r: Chat-Raumname • %u: Benutzername • %n: Nachricht

Verbindungsabbau:

Der Abbau der Verbindung wird vom Benutzer auf dem Client mittels der Eingabe des Befehls

/disconnect

eingeleitet. Der Client sendet eine *CL_DISC_REQ* Nachricht an den Server und wartet auf eine entsprechende *SV_DISC_REP* Antwort. Auch diese Prozedur soll im Falle eines Nachrichtenverlustes dreimal, mit einer Wartezeit von fünf Sekunden, wiederholt werden. Der Server verkündet allen Clients, die in gleichen Chat-Räumen wie der verlassende Benutzer waren, dass dieser das Netzwerk verlassen hat. Dies geschieht mittels mehrerer *SV_DISC_AMSG* Nachrichten.

Nachrichtenformat:

Nachricht	**Id**	**Feld (Länge in Byte)**
CL_DISC_REQ	8	-
SV_DISC_REP	9	-
SV_DISC_AMSG	10	Länge des Benutzernamens (4) Benutzername (variabel)

Textausgabe:

Nachricht	**Textausgabe**
CL_DISC_REQ	Beende die Verbindung zu Server %h (%s). • %h: Hostname des Servers • %s: IPv4 Adresse des Servers
SV_DISC_REP	Verbindung erfolgreich beendet. Verbindung nicht erfolgreich beendet. Wartezeit verstrichen.
SV_DISC_AMSG	%u hat das System verlassen. • %u: Benutzername

Viel Erfolg und viel Spaß!