Prof. Dr. A. Schütte
Prof. Dr. J. Reichardt

Fachbereich Informatik
Fachhochschule Darmstadt

Praktikum

VERTEILTE SYSTEME

Im Praktikum (3 Termine à 4 Stunden) werden Programmieraufgaben mit unterschiedlicher Technik bearbeitet.

In jedem Praktikum ist eine Client/Server Anwendung zu realisieren. Dabei soll immer eine Telefonbuchverwaltung mit den Funktionen

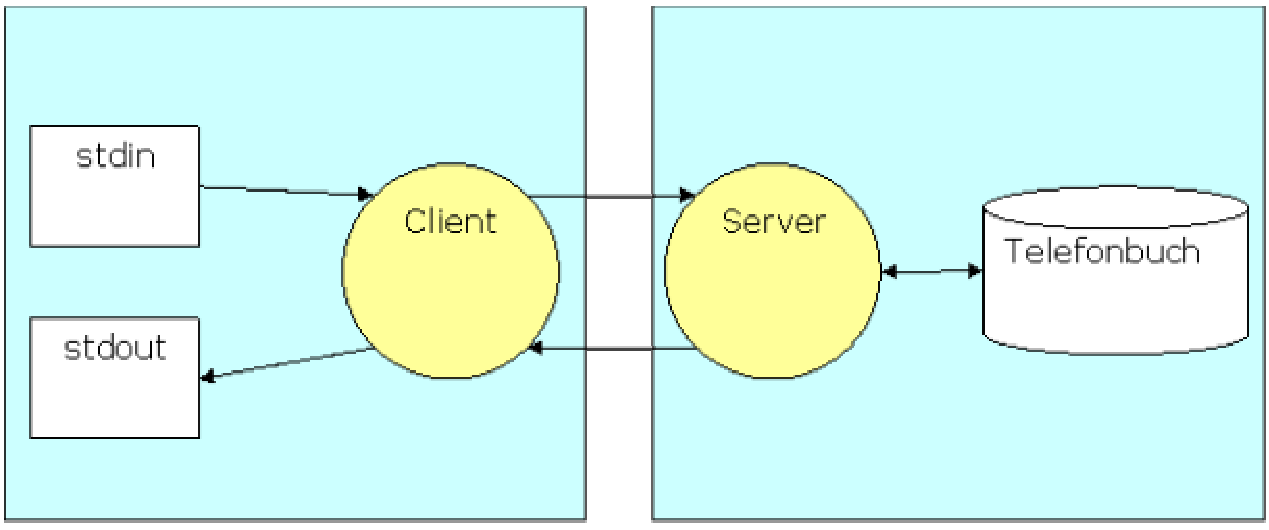
- Eintrag hinzufügen
- Eintrag suchen
- Einträge listen (nur Praktikum 3)

mit den Techniken

- Sockets oder Named Pipes (Praktikum1)
- RPC (Praktikum 2)
- CORBA (Praktikum 3)

umgesetzt werden.

Das Telefonbuch wird vom Server verwaltet. Die Clients stellen Anfragen an den Server.



1. Praktikumsaufgabe (Socket-Kommunikation und IPC)

Als Basis für die Realisierung können entweder Named Pipes **oder** Sockets verwendet werden. Die dazu erforderlichen Programme sind in ein eigenes Verzeichnis zu kopieren und zunächst zu übersetzen und auszuführen. Wenn das funktioniert hat, sind die Anpassungen vorzunehmen, so dass die Telefonverwaltung realisiert ist.

Die Quellen befinden auf dem Server ulab sich unter: `/share/vs/namedPipes` bzw. `/share/vs/sockets`

oder sind als zip Dateien (NamedPipes, C++-Sockets, JavaSocket) verfügbar.

Die Telefonbuchverwaltung soll folgende Funktionen haben:

- addEntry(Name, Telefonnummer) speichert Name und Telefonnummer im Telefonbuch
- searchEntry(Name) liefert die Telefonnummer zum Namen aus dem Telefonbuch

Dabei sind folgende Aufgaben zu erfüllen:

- Festlegen eines Protokolls zwischen Client und Server, so dass die Anfragen behandelt werden können.
- Erweitern des vorgegebenen Client- und Serverprogramms so dass die o.a. Aufgabe mit dem Protokoll erfüllt wird.

Optional kann folgende Erweiterung realisiert werden:

Die Serveranfragen sind in eine Datei zu protokollieren. Die Protokolldatei soll folgenden Aufbau haben:

```
[Mon Jan 28 19:08:43 2002] Nachricht 1
[Mon Jan 28 19:08:45 2002] Nachricht 2
```

zur Durchführung

- Das Praktikum 1 beginnt mit der Socket-Variante. Verbleibt nach der erfolgreichen Bearbeitung dieser Aufgabe noch genügend Zeit, ist anschließend die NamedPipe-Variante zu implementieren.
- Alle Programme sind zunächst gemäß der README-Datei in Ihr Home-Verzeichnis zu kopieren, in einen lauffähigen Zustand zu bringen und zu testen.
- Zum Verständnis des Quellcodes der Client- und Server-Programme ist dieser an allen wesentlichen Stellen zu kommentieren. Verwenden Sie hierzu auch die sogenannten ‚man pages', sowie die Zusatzblätter in der Anlage. Insbesondere sind die Socket-Funktionen (bzw. NamedPipe-Funktionen) detailliert zu beschreiben.
- Erstellen Sie ein Konzept für die Anpassung von Client und Server an die spezielle Telefonbuchaufgabe. Hierzu sind zunächst die Stellen im Quellcode zu identifizieren, die von den Änderungen betroffen sind. Desweiteren ist die Telefonbuchfunktion zu entwerfen und das Übertragungsprotokoll festzulegen.
- Die Server-Funktionen ‚addEntry' und ‚searchEntry' sind möglichst einfach zu realisieren, etwa mit statischen Tabellen.
- Da die Client-Server-Kommunikation auch plattform-übergreifend erfolgen soll, ist im Anschluß an die gefundene lokale Lösung in gemeinsamer Absprache mit allen Prakti-

kumsgruppen ein plattform-übergreifender Standard für das Telefonbuchprotokoll zu bestimmen.

- Für die plattform-übergreifende Client-Server-Kommunikation sind auch die IP-Adressen in den Client- und Server-Programmen von lokal (127. ... .1) auf beliebige Adressen im Bereich 141.100.42.x umzustellen, wobei x das 2-stellige Ende der jeweiligen Rechnernummer ist. Beachten Sie auch die Port-Nummern.

ANHANG

Sequence of Socket Procedure Calls

As the figure shows, the server calls seven socket procedures and the client calls six. The client begins by calling library procedures *gethostbyname* to convert the name of a computer to an IP address and *getprotobyname* to convert the name of a protocol to the internal binary form used by the socket procedure. The client then calls *socket* to create a socket and *connect* to connect the socket to a server. Once the connection is in place, the client repeatedly calls *recv* to receive the data that the server sends. Finally, after all data has been received, the client calls *close* to close the socket.

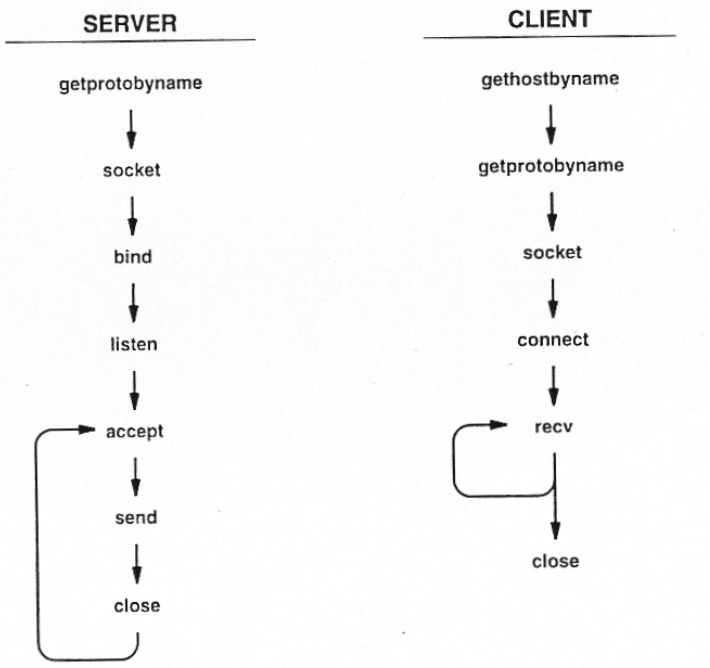


Figure 28.1 The sequence of socket procedure calls in the example client and server. The server must call *listen* before a client calls *connect*.

The server also calls *getprotobyname* to generate the internal binary identifier for the protocol before calling *socket* to create a socket. Once a socket has been created, the server calls *bind* to specify a local protocol port for the socket and *listen* to place the socket in passive mode. The server then enters an infinite loop in which it calls *accept* to accept the next incoming connection request, *send* to send a message to the client, and *close* to close the new connection. After closing a connection, the server calls *accept* to extract the next incoming connection.

2. Praktikumsaufgabe (Remote Procedure Call)

Die Aufgabenstellung entspricht der des Praktikums 2. Für die Implementierung ist der Remote Procedure Call Mechanismus von Sun zu verwenden.

Die Quellen befinden auf dem Server ulab sich unter: `/share/vs/rpc`

oder sind als zip Datei (RPC) verfügbar.

Optional kann folgende Erweiterung realisiert werden:

Optional soll der Server die Zeit, die beim Protokollieren der Anfragen in die Protokolldatei geschrieben wird von einem Zeitserver erfragen. Verwendet werden soll der Rechner „trex“, der über den Port 13 auf Anfragen mit der aktuellen Zeit antwortet.

Die Quellen der Programme für den Zeitserver sind im Verzeichnis

Die Quellen befinden sich auf dem Server `ulab` unter: `/share/vs/timeDate`

oder sind als zip Datei (timeDate) verfügbar.

zur Durchführung:

- Die Praktikumsaufgabe soll Sie mit dem RPC-Mechanismus der Firma SUN vertraut machen. Der RPC baut auf der Socket-Kommunikation auf, die Sie im ersten Versuch kennengelernt haben.
- Wie im ersten Versuch ist auch hier wieder der Telefonbuch-Service zu implementieren. Ausgangspunkt der Implementierung ist die Schnittstellen-Beschreibung der Service-Prozeduren.
- Ein Beispiel einer solchen Schnittstellen-Beschreibung finden Sie in der Datei *addiere.x*. Diese Datei ist zunächst von /share/vs/rpc in Ihr Home-Verzeichnis zu kopieren. Ausgehend von dieser Beispieldatei ist dann die gesamte Implementierung des Telefonbuch-Service durch entsprechende Modifikation zu entwickeln.
- Der RPC-Compiler *rpcgen* generiert mit dem Aufruf ***rpcgen –a addiere.x*** aus der Schnittstellen-Beschreibung die Client- und Server-Stubs, eine Header-Datei, die Parameter-Konvertierungsroutinen, ein Makefile, sowie zwei Beispielprogramme für Client bzw. Server.
- Das Client-Programm ist mit dem Aufruf der *addiere*-Funktion zu ergänzen, das Server-Programm mit der Implementierung der *addiere*-Funktion (siehe auch beiliegende Skriptseiten). Danach ist das Programm mit ***make –f Makefile.addiere*** zu compilieren und mit ***addiere_server&*** und ***addiere_client localhost*** zu testen. Statt *localhost* kann als Clientparameter auch eine passende IP-Adresse angegeben werden.
- Analog zur *addiere*-Funktion sind nun die Telefonbuch-Funktionen *addEntry* und *searchEntry* zu implementieren. Dabei ist insbesondere zu beachten, dass Funktionsparameter wie im *addiere*-Beispiel stets zu einer einzigen Parameter-Struktur zusammengefasst werden müssen. Wandeln Sie dazu das *addiere*-Beispiel zunächst in die *searchEntry*-Funktion um.
- Für die Erweiterung der Telefonbuch-Schnittstelle um eine weitere Funktion ist sowohl die Schnittstellen-Beschreibung als auch die Client-Struktur anzupassen. In einer Endlosschleife soll der Client Kommandos akzeptieren und die entsprechende Proze-

dur aufrufen (verwenden Sie ggfs. Ihren Clientcode aus dem ersten Versuch). Auf der Serverseite sind die Funktionen zu implementieren.

Siehe auch Zusatzblätter zum RPC (siehe Skript von Prof. Schütte)

3. Praktikumsaufgabe (CORBA)

Die CORBA Anwendung der Vorlesung (Telefonbuchverwaltung) soll als Basis verwendet werden.

Die Quellen befinden auf dem Server ulab sich unter: `/share/vs/corba`

oder sind als gzip Datei (CORBA) verfügbar.

Kopieren Sie zunächst die alle Dateien des o.a. Quellverzeichnisses in ein eigenes Verzeichnis und führen Sie das Kommando „make“ aus. Damit die richtige Umgebung zum Arbeiten mit dem ORB eingestellt ist, sollten Sie in Ihr Profile (Datei $HOME/.profile) die Zeile „**. /share/mico/admin/mico-setup.sh**“ eintragen und sich neu anmelden.

Danach testen Sie die Anwendung, so wie in der Vorlesung gezeigt lokal auf Ihrem Rechner.

Nun sind die Ergänzungen vorzunehmen. Realisieren Sie eine neue Client Anwendung zum **Zählen der Einträge** des Telefonbuchs. Die dazu erforderlichen Erweiterungen:

- Ergänzen des Interface (IDL)
- Erweitern des Serverprogramms
- Neues Client Programm

sind zu realisieren.

Weiterhin soll das Telefonbuch ist in einer ASCII Datei abgespeichert werden.

Optional können folgende Erweiterungen realisiert werden:

- Neues Client Programm zur Auflistung der gesamten Telefonbuchdatei.
- Erweiterung des Servers, so dass alle Aktivitäten in eine Log Datei geschrieben werden.

Zur Durchführung:

- In diesem Praktikumsversuch soll die Telefonbuch-Anwendung auf der Basis der Common Object Request Broker Architecture implementiert werden.
- Im Verzeichnis */share/vs/corba* liegt ein vollständiges Beispiel vor, welches zunächst zu studieren ist und danach zu testen.
    - Kopieren der Dateien */share/vs/corba/** in das eigene Home-Verzeichnis.
    - Die kopierten Source-Files sind zu analysieren und der CORBA-Architektur zuzuordnen (Client, Server, Stubs, POA, etc.). Diese Zuordnung ist im Praktikumsbericht zu dokumentieren. Insbesondere ist auch anzugeben, wie die einzelnen Komponenten programmtechnisch miteinander verbunden sind (z.B. Server mit POA, Server mit Server-Methoden).
    - Mit *make* sind die ausführbaren Files zu erzeugen.
    - Mit *nsd –ORBIIOPAddr inet:localhost:5000 &* ist der Name-Service zu starten.
    - Die Adresse *inet:localhost:5000* ist auch beim Server-Start und bei jedem Client-Aufruf anzugeben (Option *–ORBNamingAddr*).
    - Die weiteren Client-Parameter sind aus dem Client-Sourcecode zu ermitteln.

    - Mit Hilfe des Admin-Programms *nsadmin* ist die vom Server zum Name-Service exportierte Serveradresse (Interoperable Objekt-Referenz, IOR) zu ermitteln und zu dekodieren. Was sind ihre Bestandteile?
- Abschließend ist die Telefonbuch-Anwendung um eine weitere Funktion zu ergänzen, die die Einträge im Telefonbuchzählt und die Anzahl auf der Client-Seite ausgibt.
- Optional kann die Abspeicherung des Telefonbuchs in einer ASCII-Datei realisiert werden.

Siehe auch Skript „Verteilte Systeme“ von Prof. Schütte, Kap. CORBA