Client-Server mit Socket und API von Berkeley

LaTeX

Projektbereich

Deutsche Sprache

Klasse 3F

Schuljahr 2015/2016

Inhaltsverzeichnis

EINFÜHRUNG

Wir sind die Schüler der 3F von dem GALILEO GALILEI Institut. Wir haben zusammengearbeitet um ein Client-Server zu schaffen, das eine mathematische Berechnung machen muss. In diesen Seiten findet ihr die Dokumentation.

1 NETZWERKPROTOKOLLE

Die *Protokolle* sind Regeln, die die zwei Machinen für die Informationen benutzen, um zu wechseln und sagen, wann und wie sie kommuniziert werden müssen. Wenn die zwei Machinen entfernt sind, spricht man von Neztwerkprotokollen. Ein gemeinsames Merkmal von den Neztwerkprotokollen ist die Schichtstruktur. Die höchste Ebene führt die Dienste von der unter liegenden Ebene aus. Die gleichen Ebenen kommunizieren durch verschiedene Machinen dank dem gleichen Protokoll. Ein Vorteil der Netzwerkeprotokolle ist das Design von einem Niveau, das nur einige Aspekte von den Problem analysiert. Jedes Niveau ist unabhängig und bietet gemeinsame Dienste der ober liegenden Ebene.

1.1 TCP/IP

Die zwei Protokolle *TCP* und *IP* wurden von Bob Kahn und Vinton Cerf 1974 erfunden. Das Ziel war die Vermaschung der Rechner. Die Protokolle stellen eine Reihe von Kommunikationsregeln im Internet und basieren auf der IP.

1.1.1 Die Niveaus

- **Physisches Niveau**: Es handelt sich um die Schnittstelle zwischen Computer und Netz und definiert die Eigenschaften des Computers, dessen Signale, Zeichengebung und Daten
- **Network Access Niveau**: Es beschreibt den Austausch von Daten zwischen einem Knoten und seinem Netz. Die Software, die in diesem Niveau benutzt ist, hängt von dem Netz ab. Es definiert auch den Empfänger und die Qualität des Services.

- **Internetworking Niveau**: Es beschreibt die Übertragung zwischen zwei Knoten im Netz, die die Paketschaltung sind und die Übertragung die nicht zuverlässig ist. Definiert das Format von der Paket, es ist eines System von Global Adressierung und einem Mechanismus von Routing von Paket.

- **Übertragungsniveau**: Es beschreibt die Kommunikation zwischen zwei Knoten und erlaubt die richtige Reihenfolge von den Paketen und eine zuverlässige Datenübertragung. Der Übertragungsniveau bietet einen Kommunikationskanal end-to-end für Pakete. Es nutzt die Protokolle TCP und UDP:

  1. **TCP**: Das TCP ist ein Netzwerkprotokoll und bietet eine zuverlässige Übertragungsniveau. Es wird für seine Sicherheit verwendet.
  2. **UDP** bietet eine unzuverlässig Übertragungsniveau. Es wird für seine Geschwindigkeit verwendet.

- **Anwendungsniveau**: Es beinhaltet die Logik, die man für die verschiedenen Anwendungen braucht. Er bietet verschiedene Netzdienste wie Login, Dateiübertragung, EMail und Web. Ein Programm wirkt mit einem Protokoll mit, um Daten zu senden und empfangen.

1.2 Netzwerk-Anwendung

Die *Anwendungen von Netz* bestehen aus verschiedenen Elementen, die unabhängig arbeiten und sie Infos austauschen können. Die Anwendungen sind kommunizierende und verteilte Prozesse. Die Kommunikation erfolgt, wenn die Dienste des Untersystems angeboten werden. Die Mitarbeit kann man durch verschiedene Modelle implementieren. Das Meist verwendete Modell ist das *client/server*.

1.3 Die Signale

Die Server entwickelt einen *Sohn-Prozess*, um den Anschluss zu betreiben. Wenn der Sohn-Prozess den Anschluss mit dem Client endet, es sendet ein Signal zum *Vater-Prozess* und der “Sohn” endet. Der Sohn-Prozess verwandelt sich in einen *Zombie-Prozess*, dann wird der “Vater” ihn enden durch die wait() und waitpid() Funktionen. Sondern endet der Zombie-Prozess , wenn auch der “Vater gekillt” wird .

2 CLIENT-SERVER

2.1 Definitionen

Das Client-Server-Modell beschreibt eine Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Die Aufgaben werden von Programmen erledigt, die in Clients und Server unterteilt werden. Der Client kann auf Wunsch einen Dienst vom Server anfordern. Der Server, der sich auf demselben oder einem anderen Rechner im Netzwerk befindet, beantwortet die Anforderung; üblicherweise kann ein Server gleichzeitig für mehrere Clients arbeiten.

2.2 Server

Ein *Server* ist ein Programm, das mit einem Client kommuniziert, um ihm Zugang zu einem Dienst zu geben. Ein Computer kann nämlich ein Server und Client zugleich sein: Pee-to-Peer.

2.3 Client

Ein *Client* kann einen Dienst bei dem Server anfordern, welcher diesen Dienst bereitstellt.

3 SOCKET

3.1 Was ist der Socket?

In Informatik ist ein *Socket* eine Software, die das *Operativ System* verwaltet. Der Socket ist einen Netz-Kommunikationskanal zwischen einem Prozess und eine Ressource. Also der Socket ist ein bestimmtes Objekt, auf dem man die Daten lesen,schreiben,übertragen und erhalten kann.

3.2 Funkionalität und Nutzung des Sockets

Mit dem *Socket* können wir die *API* verwenden, die durch das Netz verteilt sind. Mit dem Socket kann man die Daten zwischen entfernte Host und lokale Prozesse senden und erhalten. Die lokalen und entfernte Socket bilden ein Paar, das aus der Adresse und dem Ëingang"von Client/Server besteht. Üblicherweise das O.S. bietet einige API um die Anwendungen mit der Kontrolle und der Verwendung des Netz-Sockets zu helfen.
Die Protokolle, mit denen wir die Socket implementieren können sind:

- TCP (Transfer Control Protocol)
- UDP (User Datagram Protocol)

Diese Protokolle basieren auf dem IP (Internet Protocol). Man Kann die Sockets mit verschieden Methoden implementieren, wie C, C++ und Java.

3.3 Die Enstehung des Sockets

Der **Socket** entstand 1983, als er in dem BSD in California eingeführt wurden. Die Berkeley Universität, dank der Finanzierung, implementiert TCP-IP im O.S. Unix. Die Forscher von Berkeley entwickelten die Funktionsgruppe, die "**Socket-Schnittstelle**" hiessen.

3.4 Socket-Schnittstelle

Die Schnittstelle mit den Anwendungen wird nicht immer in den Kommunikationsprotokollen definiert, sondern es ist Teil des Opertiv-Systems, das die Protokolllinien enthält. So definieren die API des O.S. der Name und das Thema von der Relativfunktion. Trotzdem diese Freiheit der Wahl, haben viele O.S. die API, die Socket genannt ist, entschieden. Viele Rechnerproduzenten haben das System BSD für die Basis der Entwicklung von den neuen O.S. gewählt, dadurch ist die Socket-Schnittstelle Marktstandard geworden.

3.5 API Socket

Man sagt, dass die Anwendung von Client-Server durch die Transportprotokolle kommunizieren. Diese Anwendung müssen viele Informationen an der Protokoll geben. Wie zum Beispiel Server oder Client werden. Also,

die *Anwendung* brauchen die *„Interfaccia con le Applicazioni“ (API)*, die definieren, welche Funktion sie machen sollten. Die API bestimmen nicht nur die verfügbare Werkzeuge für die Kommunikation, sondern auch die Schwierigkeit, die sie in dem Programm zu schreiben haben.

4 MAKEFILE

4.1 Geburt der Makefile

Schreieben ein Programm, das tunlich von verschiedenen File ist, bedeutet um die Daten einzeln zu kompilieren und danach in der fertiger File verbinden. Wenn die Daten viele sind machen wir diese Verfahren:

- eine Datei mit einem Rechner zu benutzen
- die Methode um es zu benutzen
- die Daten zu kompilieren
- die Ausführbaresdatei

Das ist der Anhalt der Makefile.

4.2 Der Makebefehl

Mit diesem Befehl kann man große Programme entwickeln. während das, man erinnert die Stucke des Programms, die änderungen haben, nur diese werden kompiliert.
Jetzt haben wir ein Beispiel:

```
PROGS=error my_io my_signal str_srv_tris str_cli_tris tris_client tris_server
CC = gcc
CCN = gcc -std=c99

all:$(PROGS)

error:  error.c
        ${CC} -c error.c

my_io : my_io.c
        ${CC} -c my_io.c

my_signal: my_signal.c
        ${CC} -c my_signal.c

str_srv_tris: str_srv_tris.c
        ${CC} -c str_srv_tris.c

str_cli_tris: str_cli_tris.c
        ${CCN} -c str_cli_tris.c

tris_client:tris_client.o str_cli_tris.o my_io.o error.o
        ${CC} -o $@ $^

tris_server:tris_server.o str_srv_tris.o my_signal.o my_io.o error.o
        ${CC} -o $@ $^

clean:
        rm -f *.o
        rm -f *.*~
        rm -f *~
shish:
        rm -f *.o
        rm -f *.*~
        rm -f *~
        rm tris_client
        rm tris_server
```

4.3 Andere Befehle

Der folgenden Befehl braucht man um nutzlose Datei zu loschen.

```
clean:
        rm -f *.o
        rm -f *.*~
        rm -f *~
```

In den Unterrichten haben wir den folgenden Befehl um nutzlose Datei zu loschen, ohne ein zu ein zu wählen.

```
shish:
        rm -f *.o
        rm -f *.*~
        rm -f *~
        rm tris_client
        rm tris_server
```