Versionsneutrale Socket-Programmierung

In dieser Aufgabe lernen Sie Anwendungen zu programmieren, die über das Internet kommunizieren. Hier steht praktisches Erarbeiten der Netzwerkprogrammierung in C im Vordergrund – mit einem dialogorientierten Protokoll ("Request-Response") als Anwendung.

Aufgabe

Implementieren Sie in der Sprache C eine Anwendung zwischen Clients und einem multiclientfähigen Server unter Verwendung von TCP. Diese sollen das folgende vereinfachte Anwendungsprotokoll im http-Stil realisieren.

Client Kommando	Server Antwort

`List`	Liste aller verbundenen Clients der Form:
	`<Clienthostname>:<Clientport>`
	`<N> Clients verbunden`

`Files`	Liste aller Dateien im Server-Verzeichnis in der Form:
	`<Dateiname> <Datei-Attribute: last modified, size>`
	`<N> Dateien`

`Get <dateiname>`	`<Datei-Attribute: last modified, size>` Inhalt von `<Dateiname>`

`Put <dateiname>`	Nach vollständigem Empfang und Speicherung der Datei:
	`OK <Serverhostname>`
	`<Benutzte Server-IP-Adresse>`
	`<Datum + Uhrzeit>`

`Quit`	Clientseitige Beendigung der Verbindung.

`Get` und `Put` sollen Textdateien aus dem Verzeichnis übertragen, indem der entsprechende Server, bzw. Client, gestartet wurde. Die Anwendung soll **protokollunabhängig**, also sowohl für IPv4 als auch IPv6, lauffähig sein.

Entwicklungsumgebung

Die Aufgabe soll auf den Laborrechnern lauffähig sein. Achten Sie darauf, dass sich die APIs auf Linux und Windows unterscheiden! Wenn Sie ihr Programm basierend auf der Windows API entwickeln, funktioniert es daher nicht zwangsweise auch auf dem Linux-Server. Obwohl

macOS und Linux deutlich näher beieinander sind bei vielen Netzwerk spezifischen APIs, kann es auch hier Unterschiede in den Details geben.

Tools: Linux, C Compiler (gcc, clang), CMake, MANPAGEs, Debugger (gdb, lldb), tshark, Handbuch[1]

Hinweise zur Implementierung

Als Anfang für die Implementierung nutzen Sie bitte das **hier** verlinkte Projekt. Es liefert eine Ausgangsimplementierung für einen Server und Client in C. Außerdem sind in der CMake Konfiguration einige Optionen eingestellt, die Sie bitte beim Entwickeln beibehalten (z.B. den *Address Sanitizer*).

Die Übertragung der Befehle, Antworten und Dateien soll als Strom geschehen und nicht durch die Größe lokaler Buffer limitiert sein. Um festzustellen, dass Sie das Ende eines Stroms erreicht haben, markieren Sie diese Stelle mit dem *End of Transmission* (EOT) ASCII Character[2].

Um Interoperabilität zu gewährleisten, müssen die Kommandos wie in der Aufgabe beschrieben versendet werde. Außerdem können sie keine besonderen Antworten oder Abkürzungen benutzen, die nur Ihre Anwendungen verstehen.

Client-Details

Dem Client soll beim Aufruf auf der Kommandozeile der Server-Kontakt mitgegeben werden, also wahlweise der DNS-Name oder die IPv4/6 Adressen sowie der Port des Servers. Im Anschluss nimmt der Client in einer Schleife Befehle entgegen und führt diese aus.

Tipps: Beim Übertragen der Daten kann `sendfile` helfen, schauen Sie für Details in die MANPAGE.

Server-Details

Der Server bleibt single-threaded, multiplexed aber eingehende Verbindungen im Reactive Pattern mithilfe des `select` Posix-Calls. Initialisieren Sie hierzu im Zustand 'Listen' ein `fd_set`, rufen hierauf `select` auf und steuern das Annehmen eingehender Verbindungen (`accept`) über aktive Deskriptoren im `fd_set`.

Nehmen Sie als default Port die 0. Dann bekommt der Socket einen Port vom Betriebssystem zugewiesen, den Sie anschließend auslesen und ausgeben könnten. Zusätzlich soll der Server ein CLI-Argument entgegennehmen, um einen spezifischen Port zu wählen.

Tipps: Mit "`netstat -lnt`" können Sie schauen auf welcher welcher Adresse ihr Server lauscht.

Versionsneutrale Programmierung

- Erweitern Sie die Programme um eine DNS-Namensabfrage mithilfe des Aufrufs `getaddrinfo` (s. MANPAGE). Dieser Call erzeugt die für den `socket`-Aufruf notwendigen Adressstrukturen in folgender Gestalt:

[1] Als praktisches Handbuch sei Beej's Guide to Network Programming empfohlen.
[2] https://en.wikipedia.org/wiki/End-of-Transmission_character

  - Er liefert einen Pointer auf eine verkettete Listen von `addrinfo` Adressstrukturen (s. MANPAGE).
  - Mithilfe einer Indirektion liefert `addrinfo` transparenten Zugriff auf die protokollabhängigen `sockaddr*`-Strukturen.

- Für die Socketerzeugung iterieren Sie über die Ergebnisliste und benutzen diejenige Adressstruktur, welche als erste funktioniert *und* den Anforderungen der Aufgabe genügt.

- Ergänzen Sie nun Ihr Programm für den Zugriff mithilfe von Namen und IP-Adressen aus der Kommandozeile. IP-Adressen können Sie mit `inet_pton` (s. MANPAGE)in die Netzwerkdarstellung, `getaddrinfo` kann mit der Eingabe einer IP-Adresse ebenfalls benutzt werden.

Testen

Testen Sie, dass Ihr Server sowohl über IPv4 als auch über IPv6 erreichbar ist. Stellen Sie außerdem sicher, dass Ihr Client einen Server über den DNS Namen finden kann.[3]

Neben dem Erstellen der Verbindung testen Sie bitte auch, dass Sie das Protokoll richtig implementiert haben, also Dateien entsprechend der Spezifikation übertragen werden. Die Befehle sollen den Vorgaben entsprechend als Text übertragen werden. Mit Tools wie Wireshark oder tshark können Sie sehen, ob die Befehle entsprechend in den TCP-Paketen übertragen werden. Da Sie nur Textdateien versenden, können Sie das CLI-Tool `diff` verwenden, um zu überprüfen, ob die Dateien gleich sind.

Stellen Sie außerdem sicher, dass ihre Programme keine Segfaults erzeugen und bei einer falschen Eingabe nicht hängen bleiben. Wenn ein Client oder der Server unerwartet beendet werden, sollte die Gegenseite dies ohne unvorhergesehenes Verhalten abfangen (Hinweis: Schauen Sie in die MANPAGE von recv).

Da das Protokoll rein textbasiert ist, sollte es mit den Programmen anderer Gruppen zusammen funktionieren.

Abgabe

Reichen sie wohldokumentierten C-Code zusammen mit einem Protokoll Ihrer Praxistests ein.

Tipps: Verschicken Sie den Code und Protokoll bitte zusammen in einem Zip Archiv. Bevor Sie das Projektverzeichnis komprimieren, löschen Sie zunächst den `build` Ordner.

[3] Im Labor gibt es Namenseinträge nur für IPv4, das IPv6-Netz `fd32:6de0:1f69:16::8c/64` ist hinter den Interfaces /eth1 verfügbar – s. Netzplan.