DHBW Ravensburg
Vorlesung Verteilte Systeme
Kurs TIT12, 6. Semester

Nachprüfung

Bearbeitungshinweise

Die Prüfungsleistung für die Vorlesung Verteilte Systeme wird durch einen Programmentwurf mit Schwerpunkt Unix Socket Programmierung erbracht.

Wenn Sie Programmfragmente aus der Literatur, dem Internet oder von anderen Quellen verwenden, ist die Quelle als Kommentar im Quelltext kenntlich zu machen und zusätzlich in der Datei `README.txt` anzugeben.

Programmierumgebung

Verwenden Sie als Programmiersprache C und als Betriebssystem Ubuntu 14.04 auf einer x86 Architektur.[1] Andere Programmiersprachen dürfen nicht verwendet werden.

Das von Ihnen erstellte Programm darf neben der Standard C Bibliothek nur die während der Vorlesung vorgestellte Bibliothek `libsockets` sowie die ebenfalls bereit gestellte Bibliothek `libdebug` einbinden. Eigene Erweiterungen dieser Bibliotheken sind zulässig, aber für die Aufgabenstellung eigentlich nicht erforderlich.

Abgabe und Bewertung

Der Programmentwurf ist als gezipptes Archiv im tar-Format (tar.gz) per Email an `kontakt@ralfreutemann.name` sowie in Kopie an Prof. Fahr zu schicken. Der Abgabetermin wird Ihnen von der DHBW mitgeteilt. Abzugeben ist

- der Quelltext sämtlicher Module im Unterverzeichnis `src`,
- das ausführbare Programm `logproxy` im Unterverzeichnis `build`,
- eine Datei `README.txt` mit Hinweisen zum Programm,
- eine Datei mit der Beschreibung und dem Protokoll (bestehend z.B. aus Request- und Response-Header) der durchgeführten Tests.

[1] ISO/IEC 9899:1999 mit sog. GNU-Extensions. Die hierfür notwendige `gcc`-Option ist in dem Ihnen zur Verfügung gestellten `Makefile` bereits enthalten (`-std=gnu99`).

Beachten Sie bitte, dass Sie für die Vollständigkeit des abgegebenen Quellcodes und die Lesbarkeit der Dateien verantwortlich sind.

Der Programmentwurf wird anhand der folgenden Kriterien bewertet:

1. Funktionalität, Korrektheit, Effizienz und Robustheit
2. Modularisierung, Verständlichkeit und Kommentierung
3. Testdurchführung und -beschreibung

Sollte während des Programmlaufs ein Speicherzugriffsfehler auftreten, wird die Punktzahl um 50% reduziert.

Das Unterstützungspaket für diesen Programmentwurf ist Bestandteil des folgenden `git`-Repository:

`https://github.com/rdrcode/distsys`

Hinweis: Sie dürfen lediglich die Dateien im Unterverzeichnis `src` ändern oder dort neue Dateien ergänzen. Die Verzeichnisstruktur darf sonst nicht verändert werden. Ausnahmen müssen vor dem Abgabetermin vom Dozenten genehmigt werden.

Aufgabenstellung

Bei der Entwicklung verteilter Systeme ist es oftmals hilfreich, die Kommunikation zwischen Client und Server zu protokollieren. Dies kann mit einem sogenannten Proxy-Server geschehen, der als transparenter Server Anfragen von Clients an den eigentlichen Server weiterleitet, die Rückantwort des Servers wieder an den Client zurücksendet und den Kommunikationsinhalt in eine Datei protokolliert.

Implementieren Sie einen verbindungsorientierten Proxy-Server `logproxy`, der auf einem konfigurierbaren Port auf Verbindungen wartet und alle dort eintreffenden Anfragen an einen (entfernten) Server weiterleitet. Die Rückantwort des Servers wird dann an den entsprechenden Client zurücksendet. Sämtliche Anfragen und Antworten die zwischen Client und Server über den Proxy ausgetauscht werden, sind in eine Datei zu schreiben. Der Name der Datei ist über einen Parameter wählbar. Neben dem lokalen Port ist der entfernte Server (Name bzw. IP-Adresse) über einen Kommandozeilenparameter einstellbar.

Die Parameter für den Programmaufruf sind in der folgenden Tabelle definiert:

Option	Bedeutung
`-s host:port`	Definiert die Socketadresse des entfernten Servers, an den alle Anfragen weitergeleitet werden.
`-f file`	Protokolliert sämtliche Anfragen und Antworten in die Datei `file`. Wird als Dateiname "–" verwendet, so ist die Standardausgabe (`stdout`) zu verwenden.
`-p port`	Definiert den lokalen Port, auf dem der Server auf Anrufe wartet.
`-h`	Gibt eine Übersicht der gültigen Parameter. Diese Meldung soll auch ausgegeben werden, wenn keine oder falsche Parameter angegeben wurden.

Nach dem Start des Proxy-Servers soll eine kurze Meldung ausgegeben werden, dass Anfragen von Clients an dem angegebenen Port akzeptiert werden. Der erfolgreiche Verbindungsaufbau zu einem entfernten Server soll ebenfalls angezeigt werden. Soweit nicht anders angegeben, erfolgen Ausgaben auf die Standardausgabe (`stdout`). Als Beispiel:

```
Accepting client requests on port 5000.
...
Connected to host obelix (192.168.0.1).
```

Realisieren Sie den Proxy-Server so, dass Anfragen von Clients quasi-parallel mit jeweils einem Prozess (Proxy-Prozess) pro Client abgearbeitet werden. Verwenden Sie für die Implementierung dieser Funktionalität Unix Prozesse (siehe `fork(2)`). Der Masterprozess nimmt Verbindungen von Clients entgegen, die dann jeweils durch einen eigenen Kindprozess bearbeitet werden. Der Masterprozess ist dafür zuständig, beendete Kindprozesse über einen Signal-Handler wieder einzusammeln.

Nachdem die Verbindung zwischen Client und Server über den Proxy aufgebaut ist, kann die Kommunikation in beliebiger Reihenfolge erfolgen. D.h. der Proxy muss gleichzeitig auf dem Lesekanal des Clients als auch auf dem des Servers auf Daten warten. Verwenden Sie hierfür Multiplexed-I/O (siehe `select(2)` und Vorlesungsskript Client/Server Socket Programmierung).

Der Proxy beendet die Verbindung zwischen Client und Server erst dann, wenn *beide* die Verbindung beendet haben. Bei einer beendeten Verbindung liefert `read()` den Rückgabewert Null zurück. Dies muss der Proxy an den anderen Kommunikationspartner weiterleiten, indem der Proxy den Schreibkanal zu dieser Verbindung einseitig schliesst (siehe `shutdown(2)`.

Sendet ein Client einen Request, der aus einer Textzeile mit einem einzelnen ”.” besteht (siehe REQUEST #2 im folgenden Beispiel) an den Proxy-Server, so beendet dieser die Verbidnung mit dem Client. Hierzu als Beispiel ein `echod`-Request:

```
=== (127.0.0.1:30168) REQUEST #1, length 7 byte(s), Tue May 01 02:37:44 2001
0000: 48 61 6c 6c 6f 0d 0a                              | Hallo..

=== (192.168.0.1:19845) RESPONSE #1, length 7 byte(s), Tue May 01 02:37:44 2001
0000: 48 61 6c 6c 6f 0d 0a                              | Hallo..

=== (127.0.0.1:30168) REQUEST #2, length 3 byte(s), Tue May 01 02:37:46 2001
0000: 2e 0d 0a                                          | ...
<REQUEST 7 byte(s) total>
<RESPONSE 7 byte(s) total>
```

Die durch Doppelpunkt getrennten Zahlen geben die Socketadresse (IP Adresse sowie Port Nummer) des entfernten Servers (Response) bzw. entfernten Clients (Request) an (siehe hierzu `getsockname(3)`). Für die Ausgabe der übertragenen Daten steht Ihnen das Modul `hex_print` zur Verfügung. Orientieren Sie sich für das Ausgabeformat an dem folgenden Beispiel eines HTTP-Requests:

```
=== (127.0.0.1:31811) REQUEST #1, length 16 byte(s), Tue May 01 01:45:44 2001
0000: 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 30 0d 0a | GET / HTTP/1.0..

=== (127.0.0.1:31811) REQUEST #2, length 2 byte(s), Tue May 01 01:45:45 2001
0000: 0d 0a                                           | ..

=== (192.168.0.1:22751) RESPONSE #1, length 2158 byte(s), Tue May 01 01:45:45 2001
0000: 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d | HTTP/1.1 200 OK.
0010: 0a 44 61 74 65 3a 20 54 75 65 2c 20 33 30 20 41 | .Date: Tue, 30 A
0020: 70 72 20 32 30 30 31 20 32 33 3a 34 35 3a 34 34 | pr 2001 23:45:44
0030: 20 47 4d 54 0d 0a 53 65 72 76 65 72 3a 20 41 70 |  GMT..Server: Ap
0040: 61 63 68 65 2f 31 2e 33 2e 31 32 20 28 55 6e 69 | ache/1.3.12 (Uni
0050: 78 29 20 20 28 53 75 53 45 2f 4c 69 6e 75 78 29 | x)  (SuSE/Linux)

...

0860: 42 4f 44 59 3e 0a 3c 2f 48 54 4d 4c 3e 0a       | BODY>.</HTML>.
<REQUEST 18 byte(s) total>
<RESPONSE 2158 byte(s) total>
```

Programmierrichtlinien

Es wird von Ihnen erwartet, dass Sie die bereits aus anderen Vorlesungen bekannten, allgemein üblichen Prinzipien der Softwareentwicklung hinsichtlich der Modularisierung und Kommentierung von Programmen beachten. Ferner sind die folgenden Programmierrichtlinien zu erfüllen:

1. Die Programmodule müssen unter der Verwendung *aller* folgenden Optionen ohne Fehlermeldungen bzw. Warnungen mit dem `gcc` C Compiler übersetzt werken können:
   - `-Wall`
   - `-Werror`
   - `-Wunreachable-code`
   - `-Wswitch-default`
   - `-std=gnu99`
   - `-pedantic`
   - `-O2`
2. Alle Fehlermeldungen sind auf die Standardfehlerausgabe (`stderr`) zu schreiben.
3. Liefert eine Funktion (z.B. `malloc()`) einen für den Programmablauf relevanten Fehlercode zurück, so ist dieser zu überprüfen und geeignet zu reagieren.
4. Die Verwendung der `goto`-Anweisung ist *nicht* erlaubt. Ausserdem sollen `break` und `continue` in Schleifenkonstrukten nur ausnahmsweise verwendet und dann aber entsprechend kommentiert werden.
5. Vermeiden Sie die Verwendung von globalen Variablen soweit wie möglich.
6. Die Anweisungsblöcke nach `if`, `else` usw. müssen stets in geschweiften Klammern stehen, auch dann, wenn die Blöcke nur eine einzelne Anweisung enthalten.
7. Am Kopf eines Moduls (`*.c`, `*.h`) sind der Modulname, die Namen der Autorinnen/Autoren, sowie der Zweck des Moduls anzuführen.
8. Vor jeder Funktion muss ein Kommentarblock stehen, der die folgenden Punkte enthält:
   - Name und Zweck der Funktion,
   - Beschreibung der Parameter (Input, Output),
   - Beschreibung der Verwendung globaler Veriablen,
   - Beschreibung des Rückgabewertes.