Technische Universität München
Lehrstuhl Informatik VIII
Prof. Dr.-Ing. Georg Carle
Dipl.-Ing. Stephan Günther, M.Sc.
Johannes Naab, M.Sc.

Programmieraufgaben zur Vorlesung Grundlagen Rechnernetze und Verteilte Systeme

Programmieraufgabe 4 – GRNVS Transfer Protocol (2 Punkte)
(29. Juni – 19. Juli 2015)

Abgabe via SVN bis Sonntag, 19. Juli 2015, 23:59:59.99 Uhr (MESZ)

~~**Hinweis:** Am Samstag, den 4.7., und Sonntag, den 5.7., finden für den Campus Garching Arbeiten an der Stromversorgung statt. Dies bedeutet, dass unsere Server (SVN (**`vcs.net.in.tum.de`**), alle VMs, **`asciiart.grnvs.net`**) heruntergefahren werden müssen und von Freitag, den 3.7., 14:00 Uhr bis voraussichtlich Montag, den 6.7., Vormittag nicht erreichbar sind.~~

1 GRNVS Transfer Protocol

Das GRNVS Transfer Protocol (kurz GTP) ist ein textbasiertes Protokoll für ASCII Art. Einige Aspekte des Protokolls sind an das File Transfer Protocol (FTP) angelehnt. Das Protokoll umfasst die Kommunikation zwischen einem Client und einem von uns bereitgestellten Server. Im Verlauf des Protokolls ist es notwendig, manuell Netzwerk-Sockets zu erstellen, um sich sowohl zu dem designierten Server zu verbinden als auch eingehende Verbindungen annehmen zu können. Zur Übertragung der Nachrichten werden reguläre TCP-Sockets verwendet. Erfolgreich übertragene Nachrichten sind unter `http://asciiart.grnvs.net/` einsehbar. Benutzername für die Webseite ist **`grnvss15`**, das Passwort lautet **`asciiart`**

Das Protokoll verwendet zur Übertragung zwei TCP-Verbindungen: Die Steuerkanal dient der Aushandlung eines dedizierten Datenkanals. Die eigentliche Nachricht wird über diesen Datenkanal übertragen. Das Protokoll lässt sich in drei Teile gliedern:

1. Aufbau des Steuerkanals und Aushandlung des Datenkanals
2. Übertragung der Nachricht über den Datenkanal
3. Bestätigung der Übertragung und Abbau des Steuerkanals.

1.1 Aufbau des Steuerkanals und Aushandlung des Datenkanals

1. Der Client baut eine Verbindung zum Server auf TCP 1337 auf. Diese Verbindung ist der Steuerkanal.
2. Der Client sendet nun die Initialisierungssequenz **`"C GRNVS V:1.0"`** und erwartet vom Server die Nachricht **`"S GRNVS V:1.0"`**. Diese Nachrichten dienen der Identifikation des verwendeten Protokolls.
3. Nun sendet der Client einen über die Kommandozeile spezifizierten Nick (Kurznamen) an den Server: **`"C <nick>"`**. Dieser wird zur Zuordnung auf dem Webinterface verwendet.
4. Der Server antwortet mit einem Token **`"S <token>"`**, welches der Client für den zweiten Teil des Protokolls speichert.
5. Der Client öffnet nun einen selbst einen TCP-Port, auf dem er eingehende Verbindungen erwartet.
6. Dessen Portnummer sendet der Client in der Form **`"C <dport>"`** and den Server. Im zweiten Teil des Protokolls wird sich der Server auf diesen Port verbinden.

1.2 Übertragung der Nachricht über den Datenkanal

1. Der Server versucht eine neue TCP-Verbindung auf dem zuvor ausgetauschten Port zum Client aufzubauen.
2. Nach erfolgreichem Verbindungsaufbau sendet der Server über den Datenkanal zunächst **`"T GRNVS V:1.0"`**. Diese Nachricht dient der Identifikation des Protokolls.
3. Der Client bestätigt dem Server, dass er das Protkoll unterstützt, indem er den zuvor bereits über den Steuerkanal gesendeten Nick noch einmal über den Datenkanal in der Form **`"D <nick>"`** wiederholt.
4. Anschließend identifiziert sich der Server gegenüber dem Client, indem das Token des Steuerkanals wiederholt wird.
5. Nachdem sich der Client sicher ist, dass die Verbindung vom ursprünglichen Server stammt, sendet er die eigentliche Nachricht: **`"D <msg>"`**. Der Inhalt der Nachricht wird dem Client entweder als Kommandozeilenparameter oder in Form einer Textdatei übergeben.
6. Der Server bestätigt den Erhalt, indem er ein neues Token (Datentoken) **`"T <dtoken>"`** über den Datenkanal sendet.
7. Daraufhin wird der Datenkanal geschlossen. Der dritte Teil des Protokolls nutzt den verbleibenden Steuerkanal.

1.3 Bestätigung der Übertragung und Abbau der Steuerkanal

1. Der Server sendet auf dem Steuerkanal die Länge der abgerufenen Nachricht in der Form **"S <msglen>"**.
2. Stimmt die Länge mit den gesendeten Daten überein, bestätigt der Client dies durch Senden des Datentokens **"C <dtoken>"**.
3. Der Server speichert die Nachricht und bestätigt dies in Form von **"S ACK"**.
4. Der Steuerkanal wird daraufhin geschlossen.

1.4 Protokollablauf

Eine grafische Übersicht über den Protokollablauf ist in Abbildung 1 und 2 abgebildet.

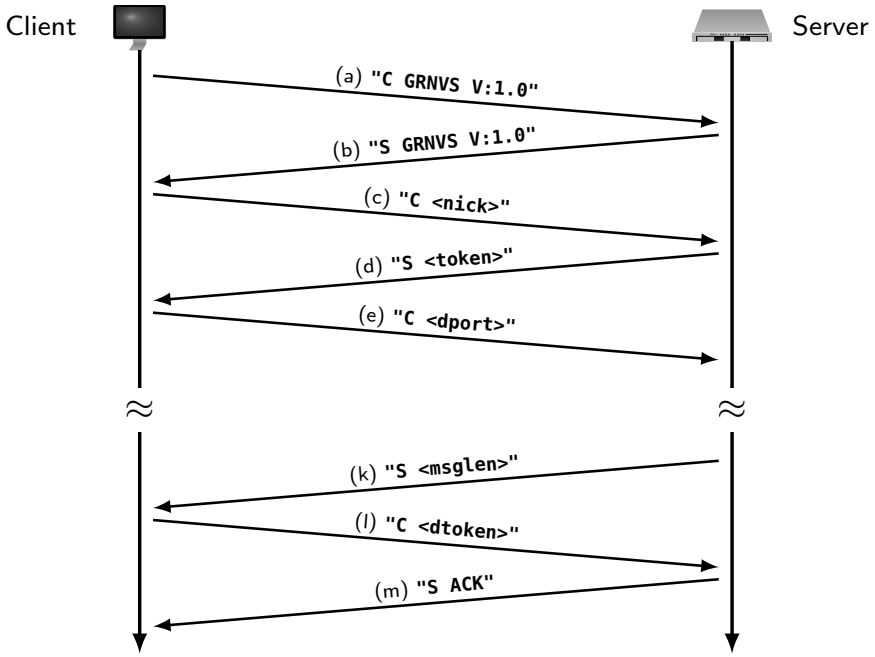

Abbildung 1: Übersicht über den Protokollablauf: Steuerkanal

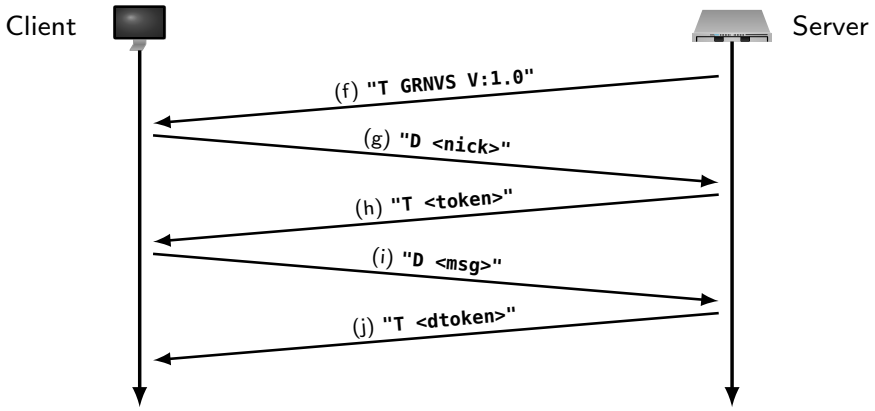

Abbildung 2: Übersicht über den Protokollablauf: Datenkanal

1.5 Fehlerbehandlung

Sollten Client oder Server dem Protokollablauf nicht folgen oder ungültige Nachrichten senden, sendet die jewilige Gegenseite eine Fehlernachricht. Fehlernachrichten haben das folgende Format: **"E <error>"**. Diese enthält eine Beschreibung des aufgetretenen Fehlers. Beim Auftritt eines Fehler werden, nachdem die Fehlernachricht gesendet wurde, alle Verbindungen direkt geschlossen.
Beim Auftreten eines Fehler im Protokollablauf oder Nachrichtenformat *(Netstrings)* (siehe Abschnitt 2) soll der Client eine Fehlermeldung auf **stdout** ausgeben, die Verbindungen schließen und dann terminieren. Diese Fehlermeldung soll am Zeilenanfang mit „**Error:**“ beginnen und durch eine kurze Beschreibung vervollständigt werden, z. B „**Error: invalid message: S: GRNVS V:2.0, expected S: GRNVS V:1.0**“. Dieses Verhalten im Fehlerfall ist für die automatischen Abgabetests zwingend erforderlich.

1.6 Nachrichtenformat

Alle Nachrichten werden mit einem von Absender und Verbindung abhängigen Präfix versehen. Nachrichten des Clients auf dem Steuerkanal beginnen mit „**C** “, die des Servers mit „**S** “. Auf der Datenkanal wird der Client durch „**D** “ und der Server durch „**T** “ identifiziert. Fehlernachrichten beginnen, unabhängig von Absender und Verbindung mit „**E** “. Eine Liste möglicher Nachrichten mit Beispiel ist Tabelle 1 zu entnehmen. Alle Nachrichten werden als *Netstrings* (siehe Abschnitt 2 kodiert.

	Format	Sender	Verbindung	Beispiel
(a)	**"C GRNVS V:1.0"**	Client	Steuerkanal	**C GRNVS V:1.0**
(b)	**"S GRNVS V:1.0"**	Server	Steuerkanal	**S GRNVS V:1.0**
(c)	**"C <nick>"**	Client	Steuerkanal	**C Neo**
(d)	**"S <token>"**	Server	Steuerkanal	**S W@:JFKXT**
(e)	**"C <dport>"**	Client	Steuerkanal	**C 4242**
(f)	**"T GRNVS V:1.0"**	Server	Datenkanal	**T GRNVS V:1.0**
(g)	**"D <nick>"**	Client	Datenkanal	**D Neo**
(h)	**"T <token>"**	Server	Datenkanal	**T W@:JFKXT**
(i)	**"D <msg>"**	Client	Datenkanal	**D Help?**
(j)	**"T <dtoken>"**	Server	Datenkanal	**T E:9A0SLY**
(k)	**"S <msglen>"**	Server	Steuerkanal	**S 5**
(l)	**"C <dtoken>"**	Client	Steuerkanal	**C E:9A0SLY**
(m)	**"S ACK"**	Server	Steuerkanal	**S ACK**

Tabelle 1: Liste der Nachrichten

2 Netstrings

Die im Protokollablauf ausgetauschten Nachrichten werden als *Netstrings*[1] kodiert. Netstrings nutzen das folgende Format:

<Laenge> ":" <Nachricht> ","

Ein Netstring besteht aus der Länge der Nachricht, gefolgt von einem Doppelpunkt, der Nachricht und einem abschließendem Komma. **Laenge** ist hierbei die Anzahl der Bytes in der Nachricht selbst, d. h. ohne die Längenangabe, den Doppelpunkt und das Komma. Die Länge ist eine ASCII kodierte Dezimalzahl. Die Nachricht „Dies ist eine Beispielnachricht.“ wird als Netstring wie folgt kodiert:

32:Dies ist eine Beispielnachricht.,

Da die Längenangabe vor der Nachricht gelesen wird, eignen sich Netstrings auch dazu, zu große Nachrichten zu erkennen noch bevor diese vollständig eingelesen worden sind. Im Rahmen des Protokolls wird der Server keine Nachrichten länger als 256 B senden. Der Server nutzt diese Eigenschaft, um zu lange Nachrichten direkt zu verwerfen und mit einer Fehlernachricht zu beantworten.

3 Hinweise

Wie in den letzten Programmieraufgaben werden auch dieses Mal wieder Rahmenprogramme für C und Java über das SVN Repository unter **assignment/assignment4/** zur Verfügung gestellt. Es bietet sich an, die Rahmenprogramme als Grundlage für die Abgabe zu verwenden. Stellen, an denen die Programme modifiziert werden müssen, sind mit „TODO“ gekennzeichnet.

[1] http://cr.yp.to/proto/netstrings.txt

Um Ihnen eine Rückmeldung zu Ihrer Abgabe vor der Deadline zu ermöglichen, können Sie wieder das Test-Framework nutzen, welches die eingecheckte Programmversion auf ihre Funktionsfähigkeit testet. Wenn noch Unklarheiten zur Benutzung von Makefiles und Testumgebung bestehen, halten Sie sich bitte an die Angaben vom Aufgabenblatt *assignment1*.

Wichtig:

- Für die Programmieraufgabe werden vom Betriebssystem bereitgestellte TCP-Sockets verwendet. In dieser Aufgabe sind Pakete *nicht* manuell zu erzeugen.
- (Adaptierte) Code Snippets (z. B. von *StackOverflow*, Umfang von mehr als einer Zeile) *müssen* mit der Quelle (URL) versehen sein.
- Die Test-Ergebnisse sind *nicht* die alleinige Bewertungsgrundlage für die Abgaben sondern stellen lediglich eine Orientierung dar, ob die Abgabe den Erwartungen entspricht. Kriterien, die nicht von dem Test-Framework berücksichtigt werden können, sind z. B. die korrekte Berücksichtigung der Byte-Order oder graceful Termination ohne Exceptions. Auch die Qualität des Source Codes wird bei der Bewertung berücksichtigt.

4 Benutzung der Programmierumgebung

Nach dem Auschecken ihres SVN-Repositories erhalten Sie nun die folgende Ordnerstruktur.

```
assignment4
|-- C
|   |-- Makefile
|   |-- src
|       |-- asciiclient.c
|       |-- asciiclient.h
|       |-- assignment4.c
...
```

```
assignment4
|-- java
|   |-- deps
|   |   |-- manifest.txt
|   |   |-- run
|   |-- Makefile
|   |-- src
|        |-- Arguments.java
|        |-- Assignment4.java
...
```

In den Unterordnern **`C`** und **`Java`** finden sich die Umgebungen, die Sie für die Bearbeitung der Aufgabe benötigen. Entscheiden Sie sich für *eine* Programmiersprache und kopieren Sie ***den Inhalt eines dieser Ordner*** in ihr Arbeitsverzeichnis:

`/users/<LRZ-Kennung>/assignment4/abgabe/`

Es sollten sich danach die Ordner **`src, deps`** und die Datei **`Makefile`** in **`abgabe`** befinden. Nachdem das gewünschte Rahmenprogramm nach **`abgabe`** kopiert wurde, muss dort nun **`src/assignment4.c`** bzw. **`src/Assignment4.java`** bearbeitet werden.

Wichtig: Die Verzeichnisstruktur muss exakt eingehalten werden. Lediglich Quelltext im Abgabeverzeichnis wird bewertet.

4.1 Ausführung

Nachdem im Ordner **`abgabe`** **`make`** ausgeführt wurde, finden Sie die ausführbare Binary *asciiclient* in Ihrem Verzeichnis. Diese kann im Ordner **`abgabe`** mit dem Befehl **`./asciiclient -m <message> <nick> <destination>`** ausgeführt werden. Hierzu wird eine Ziel-IP-Adresse benötigt. Diese kann z. B. über das Tool **`host`** abgefragt werden. Der Parameter **`-t AAAA`** gibt an, dass die zugehörige IPv6-Adresse per DNS aufgelöst werden soll.

```
# host -t AAAA asciiart.grnvs.net
asciiart.grnvs.net has IPv6 address 2a00:4700:0:3:216:3eff:feb4:ac14
```

Alternative Parameter sind **`./asciiclient -p <port> -f <file> <nick> <destination>`**. Der übergebene *nick* erscheint nach erfolgreichem Protokollablauf mit der in *file* übergebenen Nachricht auf der Website `http://asciiart.grnvs.net/`.

4.2 Abgabetests

Wenn Sie der Meinung sind, dass Ihr Programm die Anforderungen erfüllt, können sie die Funktionalität durch die bereitgestellte Testumgebung überprüfen lassen. Im Ordner **`assignment4`** befindet sich die Datei **`test-bitte.txt`**. Um einen Test anzufordern, verändern Sie diese Datei und committen sie in das SVN Repository. Führen Sie die Testumgebung erst aus, nachdem Sie das Programm selbst ausreichend getestet haben.

Um die korrekte Ausführung des Testers zu gewährleisten, dürfen die Ordnerstrukturen sowie die Makefile nicht verändert werden. Geben Sie außerdem Logging-/Debugging-Output nur auf **stderr** aus (**`fprintf(stderr, ...)`** bzw. **`System.err.print(...)`**) und stellen Sie sicher, dass auf **stdout** nur das erwartete Ergebnis ausgebeben wird.

Für diese Programmieraufgabe wird kein Ausgabe auf stdout erwartet, wenn eine Nachricht erfolgreich übertragen wurde. Im Fehlerfall soll auf stdout nur der String „Error: <error>“ ausgegeben werden, wobei **<error>** eine kurze Fehlerbeschreibung enthält. Der genaue Inhalt der Beschreibung wird in den Tests nicht überprüft.

Update 8. Juli 2015: Für die Abgabetests werden der Client und ein (modifizierter) Server auf der selben Maschine gestartet. Da sich diese dann den Netzwerkstack teilen ist es notwendig, dass Client und Server verschiedene Portnummern für die Listen-Sockets wählen. Der Server nimmt auch in den Abgabetests Verbindungen immer auf TCP Port 1337 an. Der Client kann für den Datenkanalsomit nicht auch TCP Port 1337 verwenden, sondern hierfür z. B. ein zufällig gewählten Port verwenden.

Sie können nur einen Test alle 6 Stunden ausführen lassen. Falls Sie vor Ablauf der 6 Stunden einen zweiten Test beantragen, wird dieser verzögert, bis die 6 Stunden vergangen sind. Die Testumgebung wird nach dem Commit die Ergebnisse des Tests in den Ordner **results/<revision nr>** schreiben, welchen Sie nach einem **svn update** erhalten.

Status	Beschreibung
Success	Die getestete Abgabe hat den Test erfolgreich bestanden.
Failed	Die getestete Abgabe hat den Test nicht bestanden.
Precondition failed	Der Test wurde nicht ausgeführt, da ein hierfür vorhergehender notwendiger Test nicht erfolgreich war.

Tabelle 2: Beschreibung der Test Status Codes

Der Client wird gegen die folgenden Testfälle ausgeführt.

Testcase	Beschreibung
“asciiclient” compiled and marked as executable	Das Programm kann kompiliert und gestartet werden.
Client supports protocol	Der Client folgt dem vollständigen Protokollablauf. Hierfür wird das unten stehende Log und bei Bedarf eine Fehlermeldung ausgegeben.
Client validates server context	Der Client überprüft den Server Kontext “S”.
Client validates server version	Der Client überprüft die Server Version.
Client validates server context for token	Der Client überprüft den Server Kontext des Token.
Client supports fragmented Netstrings	Der Client kann mit auf mehrere TCP-Segmente aufgeteilten Netstrings umgehen.
Client supports multiple Netstrings per TCP segment	Der Client kann mit mehreren Netstrings in einem TCP-Segment umgehen.
Client rejects leading zeros in Netstrings	Der Client überprüft die Länge der Netstrings auf führende “0”.
Client validate Netstrings format	Der Client validiert die Netstrings auf ihre Gültigkeit.
Client validates server context in data connection	Der Client überprüft den Server Kontext “T” im Datenkanal.
Client validates server version in data connection	Der Client überprüft die Server Version im Datenkanal.
Client validates server token in data connection	Der Client validiert den Server Token im Datenkanal.
Client validates server context for token in data connection	Der Client überprüft den Server Kontext des Token im Datenkanal.
Client validates message length	Der Client überprüft die vom Server gesendete Nachrichtenlänge.

Tabelle 3: Beschreibung der Abgabetests

Log für ein erfolgreichen „Client supports protocol“ Test:

```
>  Client connected to server
a: Client sent protocol version
b: Server sent protocol version
c: Client sent nick
d: Server sent token
e: Client sent port for data connection
>  Established data connection
f: Server sent protocol version on data connection
g: Read nick in data connection
h: Server sent token on data connection
i: Received message
j: Server sent data token
l: Server sent message length on control connection
k: Client echoed data token for message
m: Server acknowledged data token
```

4.3 Hinweise, falls Sie nicht die Rahmenprogramme verwenden

Es ist möglich, die Aufgaben in einer Sprache Ihrer Wahl zu lösen. Es gibt ein paar Dinge, die dabei aber beachtet werden müssen:

- Der Arbeitsaufwand muss vergleichbar sein mit dem der Rahmenprogramme (magic.do_GTP()) währe **nicht** ok. Falls Sie sich unsicher sind, fragen Sie die Übungsleitung.
- Die ausführbare Datei **muss** *asciiclient* heißen.
- Die ausführbare Datei **muss** die Parameter der Rahmenprogramme unterstützen:
  `./asciiclient -m <message> <nick> <destination>`
- Es **muss** eine Makefile existieren. Falls Ihre Lösung nicht kompiliert werden muss, da Sie z.B. eine Scriptsprache verwendet haben, muss dennoch eine Dummy-Makefile vorhanden sein, welche ggf. einfach nichts tut.
- Um Pakete vor dem Test zu installieren müssen die Paketnamen in einer Datei *deps.txt* in dem Ordner *abgabe* stehen.
- Pakete dürfen nur aus Debian Jessie *main* und *contrib* installiert werden.
- Sollte die übergebene Adresse keine IPv6-Adresse sein, so soll das Programm ohne Ausgabe auf stdout direkt terminieren.
- Weitere Nachrichten, Logs, etc. sind **nicht** auf *stdout* sondern auf *stderr* auszugeben.