NVS5 Projektübungen 2018/19

Version 0.9

Dr. Günter Kolousek

2019-01-01

1 Überblick

Es geht darum eine TCP/IP Kommunikation basierend auf der **standalone** Version von `asio` zu implementieren (siehe bereitgestelltes Template).

Obwohl die Programme auf dem "localhost" zur Ausführung kommen, sollen diese *prinzipiell* auf getrennten Hosts ausgeführt werden können (d.h. entsprechend programmieren)!

1.1 Projekteckdaten

- Repository (siehe auch Abschnitt Repository):
  - 3.3.2019 Vorabgabe als Archiv `<nachname.tar.gz>`
  - 10.3.2019 (inkl. Ausarbeitung in elektronischer Form) als `<nachname.tar.gz>`
- Ausarbeitung in schriftlicher Form (hard copy): 12.3.2019
- Prüfungsdatum (siehe Abschnitt Prüfungsgespräch): 14.3.2019
- Feststellungsprüfung: 11.4.2019

Hier folgt eine Liste von weiteren notwendigen Bedingungen:

- Zur Programmierung ist die Version `C++17` zu verwenden! Warnungen sollte es keine (möglichst wenige) geben.
- Die Projekte sind unter die Boost-Lizenz zu stellen.
- Die Beispiele sollen so gestaltet sein, dass diese jeweils die Funktionsweise der eigentlichen Aufgabenstellung **gut** mittels der Ausgaben **zeigen**!!! Das ist wichtig!!!!!

  Die Formatierung jeglicher Ausgabe kann mittels der Bibliothek fmt erfolgen (siehe zur Verfügung gestelltes `template.tar.gz`) zu erfolgen.
- Eine der Aufgabenstellung entsprechende textbasierte Benutzerschnittstelle ist zur Verfügung zu stellen. Das bedeutet, dass die Programme mittels Kommandozeilenparametern gesteuert bzw. mittels Kommandozeilenoptionen konfiguriert werden. So eine Art der Benutzerschnittstelle setzt natürlich auch eine Hilfe mittels der Optionen `-h` bzw. `--help` inklusive einer aussagenkräftigen Fehlermeldung im Fehlerfall voraus!

  Es *kann* die Bibliothek clipp oder die Bibliothek CLI11 verwendet werden (siehe zur Verfügung gestelltes `template.tar.gz`).

- Für das Loggen **ist** die header-only Bibliothek spdlog zu verwenden (siehe zur Verfügung gestelltes `template.tar.gz`).
- JSON ist entweder in der Konfiguration des Programmes oder zur Kommunikation zu verwenden. Dafür ist die header-only Bibliothek JSON for Modern C++ zu verwenden (siehe zur Verfügung gestelltes `template.tar.gz`).
- Meson (Version 0.49+) ist mit dem von mir zur Verfügung gestellten Template zu verwenden (siehe Abschnitt Repository)!!!
- Die Abgabe erfolgt wiederum als `<nachname>.tar.gz` wobei sich in diesem Archiv ein Verzeichnis `<nachname>` befindet. Bei der Abgabe muss das Verzeichnis `build` leer sein und es dürfen keine zusätzlichen SW Ordner (`asio`, `json`, `spdlog`, `fmt`) in diesem Archiv enthalten sein!!! Außerdem dürfen **keine** versteckten "macOS" (oder ähnliche) Dateien darin enthalten sein.

2 Benotung

Die zu erreichende Note hängt von den folgenden Faktoren ab:

- Beispielkategorie
- Art der Kommunikation
- Umfang und Tiefe der Ausführung
- Fehlerbehandlung
- Ausgaben und Kommentare
- Repository
- Ausarbeitung
- Prüfungsgespräch

2.1 Beispielkategorie

Jedes Beispiel wurde von mir nach einer zu erreichbaren Basisnote (siehe Abschnitt Beispielthemen) klassifiziert, die prinzipiell vergeben wird, wenn die Anforderungen **vollständig** umgesetzt wurden.

Eine **Verbesserung** ist durch die Art der Kommunikation (siehe Abschnitt Art der Kommunikation) möglich. Außerdem kann eine **weitere Verbesserung** um eine ½ Note erreicht werden, wenn die restlichen Faktoren außergewöhnlich gut erfüllt werden (d.h. **deutlich** mehr als gefordert).

Analog kann es auch zu einer **Verschlechterung** der Note kommen, wenn die Anforderungen der anderen Faktoren **nicht hinreichend** erfüllt werden. Bei **Nichterreichung** des Projektzieles wird mit *Nicht genügend* benotet.

Andererseits bedeutet dies allerdings auch, dass die Anforderungen mit der zu erreichenden Note steigen (für ein Befriedigend höher als für ein Genügend,…) (siehe §15 LBVO)!

2.2 Art der Kommunikation

Jedes Beispiel lässt sich mit verschiedenartiger Kommunikation implementieren und bewirkt, dass eine prinzipielle *Veränderung* (Summand) der Basisnote erreicht werden kann:

	Art der Kommunikation	Note
A	stream-basierte und textbasierte Kommunikation	0
B	synchrone Kommunikation basierend auf TCP mit Google Protocol Buffers (proto3)	-½
C	synchrone Kommunikation basierend auf TCP mit Google Protocol Buffers (proto3) & **zusätzlich** Goolge gRPC	-1

2.2.1 Google Protocol Buffers

Es wird davon ausgegangen, dass dieses Produkt am System installiert ist. Das kann entweder mit einem Paketmanager erledigt werden oder aus dem Sourcecode selbst übersetzt und installiert werden. Wichtig ist, dass die Installation so durchgeführt wird, dass das Produkt **systemweit** installiert ist.

Das Template ist für Google Protocol Buffers entsprechend angepasst. Ist Google Protocol Buffers richtig installiert, dann findet Meson die Installation selbständig.

Google Protocol Buffers muss in der Version 3.6.1+ verwendet werden.

2.2.2 gRPC

Das in Abschnitt Google Protocol Buffers Gesagte gilt analog!

gRPC muss in der Version 1.17.2+ verwendet werden.

Das Template ist in diesem Fall selbständig zu erweitern.

Bitte beachten: Es gibt einige Beispiele, die eine Verbesserung auf diese Art und Weise vom Prinzip her nicht erlauben, da ein zusätzlicher Einsatz von protobuf oder gRPC nicht möglich ist (z.B. HTTP). Will man in diesem Fall Google Protocol Buffers oder gRPC verwenden, dann ist eine zusätzliche Komponente zu entwerfen und implementieren!

2.3 Umfang und Tiefe der Ausführung

Ich gehe davon aus, dass das Programm so entwickelt wird, dass es dem Niveau eines 5. Jahrganges entspricht (siehe Lehrplan). Erfüllt die Abgabe nicht diese Anforderung, dann führt dies automatisch zu einer **negativen** Beurteilung.

Abgesehen davon: Je mehr Umfang und desto mehr an Tiefe, desto besser für die Note.

2.4 Fehlerbehandlung

Die Programme sollten, wenn möglich, mit jeder Art von Fehlersituation zurecht kommen und je nach Art des Fehlers diesen entweder

- maskieren und loggen
- loggen und nochmals probieren
- loggen und abbrechen

Für das Loggen ist die header-only Bibliothek spdlog zu verwenden.

2.4.1 The eight fallacies of distributed computing

Bedenke, dass Fehler immer auftreten können und beachte diesbezüglich die "The eight fallacies of distributed computing":

> Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.
>
> 1. The network is reliable
> 2. Latency is zero
> 3. Bandwidth is infinite
> 4. The network is secure
> 5. Topology doesn't change
> 6. There is one administrator
> 7. Transport cost is zero
> 8. The network is homogeneous
>
> Peter Deutsch

2.5 Ausgabe und Kommentare

Es sind sinnvolle Ausgaben vorzunehmen und Logginginformationen auszugeben, die die Funktionalität der Anwendung **klar** demonstrieren!

Weiters sind im Code aussagekräftige Kommentare zu verfassen, wo dies sinnvoll ist.

Ausgaben und Kommentare können entweder in Deutsch oder Englisch verfasst werden. Weder bei den Ausgaben noch bei den Kommentaren dürfen jeweils die Sprachen gemischt werden.

2.6 Repository

Das Repository (Mercurial-Repository) muss ein Meson-Projekt basierend auf dem zur Verfügung gestellten Template enthalten, wobei der Ordner `build` **leer** zu sein hat. Verzeichnisname (in Kleinbuchstaben **!!!**): `<nachname>`.

Die Programme müssen unter Linux ohne Fehler übersetzt und ausgeführt werden können.

Bzgl. Commits:

- jeweils eine logische Einheit, also etwas was man als Patch verwenden kann bzw. ein Schritt den man wieder zurücknehmen können soll.
- Commit-Meldungen sollen kurz und prägnant sein und sollen ausdrücken für was dieser Commit steht.

  Der Schreibstil sollte aktiv sein, also: "Add this and that" **anstatt** "This and that was added".

2.7 Ausarbeitung

- Inhalt: Kurze Beschreibung des Hintergrunds, des Aufbaus (UML, **zumindest** Klassendiagramm) und der Bedienung in eigenen Worten und kommentierten Source-Code-Beispielen sowie elektronische Abgabe des Repositiory.
- Umfang: Deckblatt, Inhaltsverzeichnis, Inhalt, ggf. Literaturverzeichnis
- Dateiformat: Textformat in einer Auszeichnungssprache (LaTeX oder OrgMode). **Zusätzlich** ist die Ausarbeitung elektronisch als PDF abzugeben.

2.8 Prüfungsgespräch

Ich werde ein kurzes Prüfungsgespräch mit Ihnen führen, das den abgedeckten Stoff des gewählten Beispieles, die eigentliche Umsetzung zum Inhalt sowie den unter Abschnitt Prüfung und Prüfungsstoff festgelegten Umfang zur Folge hat.

Wird dieses Projekt *nicht* oder *nicht in genügender Form* abgegeben, dann wird eine Leistungsfeststellung in schriftlicher Form und als praktische Arbeit abzulegen sein. Dies wird nach der Vorabgabe des Projektes fixiert.

Wird auch die verlangte Leistungsfeststellung nicht abgelegt, dann muss eine Feststellungsprüfung absolviert werden.

Das Datum für die Leistungsfeststellung bzw. die Feststellungsprüfung sind im Abschnitt Projekteckdaten angeführt.

3 Prüfung und Prüfungsstoff

Stoffumfang der Prüfung ist folgender Teil des heuer durchgenommenen Stoffes:

- Grundlagen und Basiskonzepte, Nachrichtenübertragung, Netzarchitektur (Kap 2, 3, 4 von `distsys1.pdf`)
- Internetprotokoll (Kap 11 von `distsys1.pdf`)
- Transportprotokolle (Kap 12 von `distsys1.pdf`)
- Prozesse und Threads (Foliensätze `processes`, `threads`, `threads2`)
- Synchronisation und parallele Programmierung (Foliensätze `synchronization`, `condition_synchronization`, `sync_mechanisms`, `threadsafe_interfaces`, `dist_sync`, `task_based_programming`, `parallel_programming`, `threads_perfmem`)
- Kommunikation, Serverprogrammierung, verteilte Systeme (Foliensätze `encoding`, `data_interoperability`, `serialization`, `communication`, `server_programming`, `distsys`, `systemarchitecture`)
- TCP/IP Programmierung wie im Beispiel umgesetzt (`tcpip_programming1`, `tcpip_programming2`, `tcpip_programming3`)

4 Beispielthemen

Nummer	Thema	Note
1	Einfache Client/Server (1 gleichzeitiger Client) Anwendung zur Authentifizierung an einen Server mittels eines vereinfachten CHAP Protokolls. Wobei der "Response Value" als verschlüsselter "Challenge Value" (basierend auf dem XXTEA Verschlüsselungsalgorithmus) ermittelt wird. Der Server kann mittels einer variablen Anzahl an Benutzern (samt Passwörtern) gestartet werden. D.h. der Client wird mit Benutzername und Passwort gestartet, dieser "meldet" sich beim Server an, der letztendlich Erfolg oder Fehler zurückliefert. Das Ergebnis wird dem Benutzer des Clients mitgeteilt.	4
2	Einfache Client/Server (1 gleichzeitiger Client) Anwendung mit Request/Response-Kommunikation mit Fehlererkennung basierend auf zweidimensionalen Paritätsbits (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal). Siehe Folien `encoding.pdf` und das Skript `distsys1.pdf`.	4
3	Einfache Client/Server (1 gleichzeitiger Client) Anwendung mit Request/Response-Kommunikation mit Fehlererkennung basierend auf der 4B5B Blockkodierung (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal). Siehe Folien `encoding.pdf` und das Skript `distsys1.pdf`.	4
4	Einfache Client/Server (1 gleichzeitiger Client) Anwendung mit Rahmen-basierter Request/Response-Kommunikation basierend auf der sentinel-Methode (sowohl BSC als auch HDLC) (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal). Siehe Skript `distsys1.pdf`.	4
5	Einfache Client/Server (1 gleichzeitiger Client) Anwendung zur Simulation der Leitungskodierung AMI kombiniert mit einer Fehlererkennung basierend auf einfacher Parität (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal). Folien `encoding.pdf` und das Skript `distsys1.pdf`.	4
6	Einfache Daemon-Server-Implementierung. Daemon-Server soll mittels JSON konfigurierbar sein (speziell die zu verwaltenden Ports) und Clients sollen entweder mittels JSON, Umgebungsvariable oder Kommandozeilenparameter der Host als auch der Port des Daemon-Server zu konfigurieren sein. An speziellen Diensten soll ein Echo-Server, ein Daytime-Server, ein Time-Server und ein Finger-Server implementiert werden. Für jeden dieser Server ist ein entsprechender Client zu implementieren. Siehe Folien `server_programming.pdf`	4
7	Simulation der Weiterleitung in IP in einem einfachen konfigurierten Netz (ca. 7 Nodes, mittels Konfigurationsdatei (z.B. in JSON)) mit einer Einspeisung und jeweils zufälligem Ziel. Siehe Skript `distsys1.pdf`.	4
8	Implementierung eines Gopher Clients (menü-basiert) und Servers (beliebige Anzahl an gleichzeitigen Clients), der die *item types* 0,1 und 3 unterstützt. Der Server soll die Menüs basierend auf der Verzeichnisstruktur und Gopher-Dateien (gophermaps) zur Verfügung stellen. 2 Gopher-Server sind zu konfigurieren und bereitzustellen (also in der Anwendung zu starten).	4

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
9	Einfache Broker-Implementierung. Broker soll mittels JSON konfigurierbar sein (speziell der Port) und den Hosts soll entweder mittels Umgebungsvariable oder Kommandozeilenparameter der Port des Brokers mitgeteilt werden. Jeder Host meldet sich mit einer ID am Broker an und kann mittels einer ID an den Broker eine Nachricht schicken. Hosts generieren Sprüche und senden diese jeweils an beliebige andere Hosts, die diese ausgeben. Siehe Folien `systemarchitecture`.	4
10		4
11		4
12		4
13	Simulation des Reader/Writer Problems (2 Writer/3 Reader) mit "verteiltem" Read-Write-Lock (eigener Netzprozess ausgeführt als multithreaded Server, da angenommen wird, dass die Überprüfung jedes Request zeitlich ins Gewicht fällt). Siehe Folien `sync_mechanisms.pdf`.	4
14	Simulation des Dining Philosophers Problem (5 Philosophen) auf der Basis eines Butlers, der auf Anfrage jedem Philosoph eine Gabel zuteilt, vorausgesetzt, dass kein Deadlock entsteht (insgesamt 6 Netzprozesse).	4
15	Textbasierte Chat-Applikation basierend auf Channels mit einem Server und einer beliebigen Anzahl an Clients. Ein Client meldet sich meim Server mit einem Namen an, sendet und empfängt Nachrichten und meldet sich auch wieder ab. Die anderen Clients bekommen alle Nachrichten (exkl. der eigenen) sowie alle An- und Abmeldungen mitgeteilt. Jeder Client wird als eigener Thread am Server repräsentiert. Jeder Client kann gleichzeitig Nachrichten empfangen und Eingaben vom Benutzer entgegennehmen. Es sollen Channels angelegt und auch wieder gelöscht werden können (Admin-Funktionalität). Einem Channel kann man beitreten und auch wieder verlassen. Die Channels sind zu dauerhaft abzuspeichern.	4
16	Simulation mit 5 Prozessen mit einer verteilten (wiederverwendbare) Barrier (eigener Prozess, ein Thread je Prozess) (siehe "The Little Book of Semaphores", Seite 43).	4
17	Basisfunktionalität eines Publish/Subscribe Brokers (mit n gleichzeitigen Clients). Clients können beliebige Subjects subskribieren und für subskribierte Subjects Nachrichten senden. Subjects werden am Server konfiguriert. Jeder Client enthält alle Nachrichten der subskribierten Subjects (außer die selber gesendeten). Persistente Speicherung der Nachrichten. Orientierung am Protokoll MQTT!!!	3
18	Wahlalgorithmus basierend auf Bully-Algorithmus (inkl. "theoretischem" Vergleich mit dem Chang-Roberts Algorithmus, siehe Foliensatz `dist_sync.pdf`)	3
19	Funktionsfähiger HTTP 1.1-Client zum Herunterladen/Speichern von beliebigen Dateien (GET, PUT, POST, DELETE, Basisfunktionalität; Steuerung über Kommandozeile; inkl. HTTP Basic-Authentication, nicht auf Header vergessen und inkl. Cookies, http-parser kann verwendet werden)	3

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
20	Funktionsfähiger multi-threaded HTTP 1.1-Server, beliebige Dateien (GET, PUT, POST, DELETE, Basisfunktionalität; inkl. HTTP Basic-Authentication, nicht auf Header vergessen und inkl. Cookies, http-parser kann verwendet werden)	3
21	Funktionsfähiges Monitoring System für TCP (Port), UDP (Port), Telnet, HTTP GET & POP3 (libcurl [Systeminstallation] bzw. cURLpp [Installation direkt in das Projekt], curlpp Tutorial); Konfiguration z.B. mittels JSON	3
22	Simulation einer Ampelsteuerung: 2 hintereinander geschaltete Ampeln (je ein Netzprozess) mit je 4 Straßeneinmündungen (je ein Netzprozess) inkl. spezifizierter Auslastung der Straßeneinmündungen; Konfiguration z.B. mittels JSON-Datei; Auswertung (opt. Steuerung der Ampeln auf Grund der Präsenz von Fahrzeugen)	3
23	Schifferl versenken (h2h, optional: h2m), 1 Server, 1 oder 2 Clients, Anmelden, Abmelden, Spielen, Punktestand abfragen (Persistenz in Dateien basierend auf JSON)	3
24	Einfacher POP3 Client `https://tools.ietf.org/html/rfc1939` (Liste anzeigen, Mail herunterladen, Mail löschen, Unterstützung von StartTLS mittels `gnutls-cli` (lokal zu installieren))	3
25	Einfache Client/Server (beliebige Anzahl an Clients) Anwendung, die JSON-RPC implementiert.	3
25	Netzwerk-basiertes RobotGame wie robowiki mit zentralem Server	2
26	Entwurf (API) und Erstellung eines einfachen Redis-Clients (Umsetzung des RESP Protokolls, inkl. Pipelining, Publish/Subscribe, Transactions) sowie Demonstration eines verteilten Locks auf Basis von Redlock (bei Einsatz von gRPC → Proxy, Adapter)	2
27	Erstellung eines einfachen MapReduce-Systems samt Beispielanwendung (inkl. Aufarbeitung des Themas, siehe auch DISCO)	2
28	Sliding Window Algorithmus zur Flusssteuerung in einer P2P Situation bestehend aus 2 Hosts mit simulierten Übertragungsfehlern	2
29	Simulation des Distance Vector Algorithmus für ein konfigurierbares Netzwerk, zufälliger Ausfall von Verbindungen	2
30	Simulation des Link-State Algorithmus für ein konfigurierbares Netzwerk, zufälliger Ausfall von Verbindungen	2
31	Simulation des Spanning Tree Algorithmus für ein konfigurierbares Netzwerk, zufälliger Ausfall von Verbindungen	2
32	Verzeichnissynchronisation basierend auf Datum und Hashwert zwischen zwei Netzprozessen	2