NVS5 Projektübungen – 2

Schuljahr 2020/21

Dr. Günter Kolousek

Version 0.9.4 (2021-02-22)

Inhaltsverzeichnis

Dieses Dokument ist zu lesen!!!

1 Überblick

Es geht bei diesen Projektübungen darum, kleine Projekte mit Prozessen, Threads und Synchronisation sowie **Kommunikation** (basierend auf der **standalone** Version von `asio`) und einfachen **Systemarchitekturen** (Client/Server) für verschiedene Anwendungsszenarien zu entwickeln und damit zu zeigen, dass die praktischen und theoretischen Fähigkeiten zum Implementieren verteilter Systeme erworben wurden.

Die Erstellung der Ausarbeitung soll zeigen, dass der Lehrstoff gemäß des Lehrplanes wie dies im Unterricht durchgenommen worden ist, erlernt worden ist.

Prinzipiell ist diese Angabe strukturell wie die Angabe der Projektübungen 1 in diesem Schuljahr strukturiert und aufgebaut, jedoch wurde noch mehr Wert auf Klarheit gelegt.

Die Ausarbeitung ist insoferne aufgewertet worden, als auch die Aufarbeitung des theoretischen Hintergrundes verpflichtend hinzugekommen ist.

Auch die Benotung ist ähnlich, nur werden die Beispielthemen jetzt gemäß ihres Umfangs und Tiefe nach Basisnoten klassifiziert und als orthogonales Kriterium wird die Art der Kommunikation in die Benotung mit aufgenommen (siehe Abschnitt Benotung).

1.1 Projekteckdaten

- Repository (siehe Abschnitt Repository) inkl. Ausarbeitung in elektronischer Form: 2021-04-06T23:59
- Zwischenstand 2021-03-05T23:59
- Abgabe der Ausarbeitung in schriftlicher Form (hard copy): 2021-04-07
- Prüfungsdatum (siehe Abschnitt Prüfungsgespräch): 2021-04-14
- Feststellungsprüfung: 2021-04-13

Hier folgt eine Liste der weiteren notwendigen Bedingungen:

- Zur Programmierung ist die Version `C++17` zu verwenden und das Projekt muss unter Linux mittels `g++` (Version 10.2+) zu übersetzen sein. Warnungen darf es keine geben! Kann es auf diese Weise nicht übersetzt werden, dann ist die Software nicht funktionsfähig und das Projekt wird mit *Nicht genügend* bewertet. Testen!!!
- Meson (Version 0.56+) ist mit dem von mir zur Verfügung gestellten Template zu verwenden (siehe Abschnitt Repository)!!!
- Das Projekt ist unter die Boost-Lizenz zu stellen.
- Die Funktionsweise der eigentlichen Aufgabenstellung ist **gut** mittels Ausgaben zu **zeigen**!!! Das ist wichtig! Wird *diese* Anforderung nicht erfüllt, dann gilt die Implementierung als **nicht** funktionsfähig!!!!!

- Für das Loggen **ist** die header-only Bibliothek spdlog zu verwenden (siehe `template.tar.gz`).

  Mehr zum Loggen siehe im Abschnitt Fehlerbehandlung und im Abschnitt Ausgaben, Einhaltung der Coding Conventions, Kommentare.

- 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 **muss** die Bibliothek CLI11 verwendet werden (siehe `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`).

- Es dürfen Bibliotheken wie im Abschnitt Software angeführt verwendet werden.

- Will man zusätzliche Bibliotheken verwenden, dann ist dies **vorher** mit mir zu besprechen!

  Bei erfolgter Genehmigung meinerseits, sind die folgenden Richtlinien (wie mit mir besprochen und gemeinsam festgelegt) einzuhalten:

  1. Handelt es sich um eine "Standardbibliothek", die default-mäßig auf einem System installiert ist, dann ist keine weitere Aktion nötig.
  2. Trifft die vorhergehende Bedingung nicht zu, dann sollte es sich vorzugsweise um eine header-only Bibliotheken handeln und diese in einem entsprechenden Verzeichnis eines include-Verzeichnisses direkt im Projekt "installiert" sein!
  3. Steht die Bibliothek nicht als header-only zur Verfügung, dann sollte diese in einem eigenem Verzeichnis zum Projekt hinzugefügt werden.

  Eine zusätzliche systemweite Installation ist nicht gestattet!

- Für einzelne Prozesse sind einzelne Unterverzeichnisse auf der Basis meines `templates` anzulegen (ohne `doc`). Wird eine weitergehende Strukturierung benötigt, dann soll diese entsprechend angelegt werden. Die einzelnen verschiedenen Prozesse sollen einfach gestartet werden können.

- Die Coding Conventions sind einzuhalten!

- Die Abgabe erfolgt indem die Software **spätestens** bis zum Abgabezeitpunkt (siehe Abschnitt Projekteckdaten) im Repository mit dem Tag `v1.0` versehen wird.

1.2 Software

Die nachfolgende Tabelle enthält die zusätzlichen Bibliotheken, die im Zuge dieses Projektes verwendet werden dürfen.

Es handelt sich hier fast ausschließlich um header-only Bibliotheken. Diese Bibliotheken dürfen **nicht** in dem Repository enthalten sein, sondern haben sich außerhalb des Repositories

im Dateisystembaum zu befinden. Das von mir zur Verfügung gestellte Template beinhaltet entsprechende Meson-Optionen. Diese sind zu verwenden!

Die beiden Ausnahmen sind `pystring` und `backward-cpp`: Hier ist sind jeweils die Sourcedateien (`pystring.cpp` bzw. `backward.cpp`) direkt in das `src` bzw. die entsprechenden Headerdateien (`pystring.h` bzw. `backward.hpp`)) in das `include` Verzeichnis zu kopieren.

Um eine klare und definierte Verwendung der Bibliotheken zu erreichen, habe ich die zu verwendende Version bzw. die Art der Einbindung explizit in der nachfolgende Tabelle angegeben.

Software	Zweck	Version	Art des `#include`
asio	Netzwerkprogrammierung	1.18.x (x≥1)	`#include <asio.hpp>`
backward-cpp	stack trace pretty printer	master	`#include "backward.hpp"`
doctest	Unit-Tests	2.4.x (x≥4)	`#include <doctest.h>`
CLI11	CLI	1.9.x (x≥1)	`#include <CLI11.hpp>`
cppfsm	Finite State Machine	master	`#include <fsm.h>`
cpp-httplib	HTTP/HTTPS Bibliothek	0.8.x (x≥2)	`#include <httplib.h>`
cpp-subprocess	Prozesse	master	`#include <subprocess.hpp>`
cpp-peglib	PEG Parser Bibliothek	1.3.x (x≥3)	`#include <peglib.h>`
criterion	Benchmarking	master	`#include <criterion/criterion.hpp>`
fmt	Formatieren von Strings	7.1.x (x≥3)	`#include <fmt/core.h>`
glob	File globbing...	master	`#include <glob.h>`
inja	Template Engine	3.1.x (x≥0)	`#include <inja.hpp>`
json	JSON for Modern C++	3.9.x (x≥1)	`#include <json.hpp>`
magic_enum	Verbesserte enums	0.7.x (x≥2)	`#include <magic_enum.hpp>`
pprint	Ausgabe von C++ Typen	master	`#include <pprint.hpp>`
pystring	String-Funktionen	master	`#include "pystring.h"`
rang	Färbige Ausgabe	3.1.x (x≥0)	`#include <rang.hpp>`
spdlog	Logging	1.8.x (x≥2)	`#include <spdlog/spdlog.h>`
[Boost::ext].SML	State Machine Language	master	`#include <sml.hpp>`
tabulate	Tabellen im Terminal	1.4	`#include <tabulate.hpp>`
taskflow	Parallele Tasks	3.0.x (x≥0)	`#include <taskflow/taskflow.hpp>`
toml++	Konfig. m. TOML-Dateien	2.3.x (x≥0)	`#include <toml.hpp>`
tfile	Handling von Dateien	master	`#include <tfile.h>`
sqlite_orm	ORM für SQLite	1.6.x (x≥0)	`#include <sqlite_orm/sqlite_orm.h>`

Bei den verwendeten Bibliotheken kann es sein, dass es zu Warnungen kommt, wie z.B. bei der Bibliothek `tabulate`. Hier ist folgendes Idiom einzusetzen bei der Verwendung einzusetzen:

```
//ignore warning "-Wnon-virtual-dtor" from extern library "tabulate"
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wnon-virtual-dtor"
#include "tabulate.hpp"
#pragma GCC diagnostic pop
```

Damit wird dem Compiler über die `#pragma` Präprozessordirektive mitgeteilt, dass die Warnung `non-virtual-dtor` ignoriert werden soll. Und zwar nur für die Headerdatei `tabulate.hpp`.

Achtung: Das Zielsystem ist Linux! Wenn unter einem anderen Betriebssystem wie z.B. MacOS oder Windows entwickelt wird, dann kann es durchaus sein, dass auf Grund unterschiedlicher Compiler- bzw. Bibliotheksimplementierungen, sich das Projekt nicht ohne weitere Intervention unter Linux übersetzen lässt.

Z.B. kann es durchaus sein, dass ein alleiniges `#include tfile.h` in einer Implementierung funktioniert, aber unter Linux zuerst ein `#include <cstring>` geschrieben werden muss. Zweifellos ist dies in gewisser Hinsicht ein Fehler von `tfile`, aber...

Deshalb: Neu (unter Linux) clonen und danach testen!!!

2 Benotung

Die zu erreichende Note hängt prinzipiell von der

- Beispielkategorie und der
- Art der Kommunikation

ab. Gemäß der zu erreichenden Note wird wieder mit dem folgenden Schema benotet:

- Funktion, Umfang und Tiefe der Implementierung (30%)
- Fehlerbehandlung (10%)
- Ausgaben, Einhaltung der Coding Conventions, Kommentare (10%)
- Repository: Commits, Issues (10%)
- Ausarbeitung (30%)
- Einhaltung der Richtlinien (10%)

Zu vorgetäuschten Leistungen siehe Abschnitt Vorgetäuschte Leistungen.

Ein optionales Prüfungsgespräch (siehe Abschnitt Prüfungsgespräch) wird hinzugerechnet, wenn ich dieses auf Grund des Projektverlaufes oder des Projektergebnisses als notwendig erachten muss.

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 (auf Grund der Beispielkategorie) 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

D.h. wird ein Beispiel mit der Note 3 gewählt (siehe Abschnitt Beispielthemen) und dies mittels Kommunikationsart C (gemäß obiger Tabelle) implementiert, dann ergibt sich daraus die Note $3 + (-1) = 2$, unter der Voraussetzung, dass die Implementierung gemäß der Angabe funktionsfähig ist und die Anforderungen dieses Dokumentes eingehalten worden sind.

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 3.12.4+ verwendet werden.

2.2.2 gRPC

Das in Abschnitt Google Protocol Buffers Gesagte gilt analog!

gRPC muss in der Version 1.34.0+ 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, wie z.B. HTTP. Will man in diesem Fall Google Protocol Buffers oder gRPC verwenden, dann ist eine zusätzliche Komponente zu entwerfen und implementieren! Dabei könnte es sich um eine Steuerkomponente handeln, die die Konfiguration des HTTP Servers ermöglicht oder mit der man Steuerkommandos an den Server schicken kann.

2.3 Funktion, 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 und LBVO). Erfüllt die Abgabe nicht diese Anforderung, dann führt dies automatisch zu einer **negativen** Beurteilung (siehe §15 LBVO)!

Der Umfang jeder Aufgabenstellung ist abhängig von der Beispielkategorie (siehe Abschnitt Beispielthemen) und der Art der Kommunikation.

Abgesehen davon: Je mehr *Umfang* und je mehr an *Tiefe*, desto besser für die Note!!!

- ad *Funktion*: Es wird davon ausgegangen, dass die implementierte Software gemäß der Angabe funktioniert. Das ist eine Grundanforderung (siehe Abschnitt Projekteckdaten)!
- ad *Umfang*: Eine minimale Umsetzung hinsichtlich des Umfanges liegt vor, wenn genau die Anforderung aus dem Abschnitt Beispielthemen umgesetzt wird. In diesem Fall kann man davon ausgehen, dass die Anforderungen wie in §15 LBVO in den wesentlichen Bereichen überwiegend erfüllt worden sind. Natürlich kann man selbst den Umfang erhöhen, indem man weitere (sinnvolle) Features einbaut.

  Werden von mir angegebene Bibliotheken richtig eingesetzt, dann zählt dies *ebenso* als eine Vergrößerung des *Umfanges*!

  Hier ein paar Beispiele für den Einsatz von Bibliotheken:

  - Es kann durchaus sinnvoll sein, die Optionen zusätzlich oder auch alternativ zur Spezifikation über die Kommandozeile in einer Konfigurationsdatei abzulegen. Dafür könnte man TOML (siehe `toml++`) oder auch JSON (siehe `json`) verwenden.
  - Ergebnisse oder auch den Zustand eines Prozesses könnten in einer Datei (siehe `tfile`) oder auch in einer Datenbank (siehe `sqlite_orm`) abgelegt werden.
  - Die Ausgabe könnte färbig gestaltet werden (siehe `rang`) oder auch Tabellen (siehe `tabulate`) ausgegeben werden.
  - Es könnte durchaus Sinn machen zwei verschiedene Implementierungen einer Lösung bzgl. Laufzeit zu vergleichen und dafür die Software `criterion` zu verwenden.
  - Zur Ausgabe von Testdaten (z.B. beim Logging) kann man für `C++` Datentypen die `pprint` einsetzen.
  - Kommt es zu unerwarteten Abstürzen, dann wäre `backward-cpp` durchaus hilfreich.
  - ...
- ad *Tiefe*: Man kann sich natürlich weiterführend in die *Thematik* einarbeiten, dies in der Ausarbeitung beschreiben (Zitierungen nicht vergessen) und in der Implementierung umsetzen.

Mir ist durchaus bewusst, dass die Problemstellungen innerhalb einer Kategorie nicht alle exakt den gleichen Schwierigkeitsgrad aufweisen. Dies wird von mir bei der Benotung berücksichtigt!

2.4 Fehlerbehandlung

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

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

Für das Loggen siehe den folgenden Abschnitt.

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 Ausgaben, Einhaltung der Coding Conventions, Kommentare

Es sind sinnvolle Ausgaben vorzunehmen und Logginginformationen (spdlog ist zu verwenden) auszugeben, die die Funktionalität der Anwendung **klar** demonstrieren!!!!!

Zu Logging:

- Aspekte unter denen Logging sinnvoll ist:
  - Erkennen des Verhaltens des Systems, z.B. Messwerte der Latenz, Anzahl der Zugriffe, (fehlgeschlagene) Anmeldeversuche,...
  - Nachverfolgung auftretender Fehler: Fehler tritt auf (wo, wann, welcher, unter welchen Umständen) und wie kommt es dazu.
  - Erkennung und Rekonstruktion von sicherheitsrelevanten Ereignissen:

  - Unterstützung bei der Analyse des Verhaltens bei steigender Last, Lastschwankungen, Stresssituationen,...
- Ort des Logging:
  - `stderr`: einfach, kann direkt in andere Prozesse mittels "pipe" weitergeleitet werden
  - Dateien (mehrere im Rotationsverfahren, alte Versionen löschen): persistent, kann mittels eines Netzwerkdateisystem auch auf anderen Hosts liegen (aber achten auf zeitliche Verzögerungen!)
  - Datenbank: strukturiert und daher einfache Auswertung!
  - via Netzwerk auf andere Hosts: siehe Aspekt der Sicherheit
- Art des Lesbarkeit:
  - maschinenlesbar
  - Lesbarkeit durch Menschen
- Steuerung und Konfiguration des Logging
  - Kommandozeile
  - Umgebungsvariable
  - Konfigurationsdateien

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

Die vorgegebenen Coding Conventions sind einzuhalten (wie dies im Unterricht besprochen wurde).

2.6 Repository

Das Repository (privat!) muss auf github gehostet werden und folgendermaßen benannt sein: `<nachname>_project_2` (alles in Kleinbuchstaben und ohne Sonderzeichen**!!!**). Eine entsprechende Einladung muss an mich geschickt werden!

Dieses Repository muss ein Meson-Projekt beinhalten, das auf dem von mir zur Verfügung gestellten Template basiert, wobei der Ordner `build` **leer** zu sein hat.

Überhaupt sind keine generierten Dateien (außer das PDF der Dokumentation) sowie keine von Tools (wie einer IDE) erzeugten Dateien im Repository zulässig!

In der von mir zur Verfügung gestellten Datei `meson_options.txt` sind bitte alle *nicht* verwendeten Optionen herauslöschen. Alle anderen sind genau so zu verwenden!

Von mir zur Verfügung gestellten Dateien (siehe Template), die nicht benötigt werden, sind nicht in das Repository aufzunehmen.

Der Ordner `build` sollte vorhanden, aber bis auf eine leere Datei `.gitkeep` oder `.keep` leer sein.

Dateien (und damit auch Verzeichnisse) sind konsequent in Kleinbuchstaben, ohne Sonderzeichen und ohne Leerzeichen zu benennen.

Der Inhalt jeder Datei hat in `UTF-8` kodiert zu sein und als Zeilentrennzeichen ist `NL` zu verwenden (Linux!).

Bzgl. Commits (siehe `revision_control_mercurial.pdf` Folien 35 bis 37):

- 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".

Issues (z.B. Fehler, geplante Features, Meilensteine,...) sind anzulegen und zu bearbeiten!!!

Es ist ein Readme anzulegen, das kurz das Projekt beschreibt.

Weiter ist **zwingend** eine Datei `CHANGELOG.org`, das sich bezüglich Intention und Aufbau an keep a changelog zu halten hat. Lediglich das Format wird von mir abgeändert (ja, ich bin der prof `;-)`):

Jeden Montag muss ein Eintrag mit einer Versionsnummer und einem Datum vorhanden sein, das die Änderungen der letzten Woche in einem für Menschen lesbarer Form enthält, also z.B.

```
* Changelog
** [Unreleased]
*** Added
- xxx
- xxx
*** Changed
- xxx
- xxx
*** Removed
- xxx
** [0.0.2] - 2020-02-08
*** Changed
- Show an example of the order of changelog entries inside CHANGELOG.org
** [0.0.1] - 2020-02-01
*** Added
- Create repository and send invitation to prof
- Instantiate given template
- Fill .gitignore
- Create empty doc file
```

Der textuelle Inhalt kann auch in Deutsch verfasst sein (nicht jedoch die Überschriften.

Es ist sinnvoll und ratsam die Änderungen zeitnah in dieser Datei zu dokumentieren, d.h. unter dem Header `[Unreleased]` und damit hergehend auch im Repository zu "committen".

Die Dokumentation (also das `.pdf` und auch die Sourcedatei, meist `.tex`) ist in einem Verzeichnis `doc` abzulegen.

2.7 Ausarbeitung

- Inhalt: Kurze Beschreibung des Hintergrunds (um was geht es, Aufbereitung der *theoretischen Grundlagen*, wie sieht die Lösung aus), des Aufbaus (wie wurde die Lösung umgesetzt, UML, zumindest Klassendiagramm) und der Bedienung in eigenen Worten und kommentierten Source-Code-Beispielen.
- Umfang: Deckblatt mit Titel, Autor, Klasse und Katalognummer, Beispielname und Beispielnummer, Inhaltsverzeichnis, Inhalt, ggf. Literaturverzeichnis
- Dateiformat: Textformat in einer Auszeichnungssprache: LaTeX oder Emacs Org-Mode (z.B. auch mittels pandoc). **Zusätzlich** ist die Ausarbeitung elektronisch als PDF *und* als hard-copy abzugeben (siehe Abschnitt Projekteckdaten).

2.8 Einhaltung der Richtlinien

Werden die Richtlinien grob missachtet, dann wird die Beurteilung der gesamten Projektarbeit mit *Nicht genügend* erfolgen. Anderenfalls wird entsprechend den Beanstandungen Abzüge im Sinne des unter Benotung angeführten Schlüssels geben.

2.9 Vorgetäuschte Leistungen

Werden vorgetäuschte Leistungen gefunden, wie z.B. Plagiate (Internet, andere Arbeiten), dann wird das Projekt nicht beurteilt und es werden entsprechende Leistungsfeststellungen notwendig. Es erfolgt keine Feststellung der Verschuldensfrage von meiner Seite, d.h. unabhängig davon, ob dieses Plagiat aktiv oder passiv zustandegekommen ist, wird das Projekt nicht beurteilt.

2.10 Prüfungsgespräch

Ich werde, wenn **notwendig**, d.h. *in Abhängigkeit* des Projektverlaufes, 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 zeitgerecht (d.h. gemäß den rechtlichen Rahmenbedingungen) mitgeteilt.

Wird auch die verlangte Leistungsfeststellung nicht abgelegt, dann muss eine Feststellungsprüfung über die nicht erbrachten Leistungsfeststellungen 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

Bei den meisten Themen handelt es sich um Simulationen. Simulationen operieren oft mit zufälligen Daten und zufälligen Zeiten. Dafür sind sinnvolle Bereiche zu wählen und die Generierung entsprechend den gelernten Möglichkeiten von `C++17` zu implementieren (kein `rand()`!).

Im Zuge der Kommunikation kann es unter realen Bedingungen zu Paketverlusten oder Verfälschungen (Fehlern) auftreten. Dies wird am lokalen Host nicht passieren. Deshalb soll ein "Kanal" (siehe meine zur Verfügung gestelltes Template der Klasse `Pipe` als Vorlage) implementiert werden, der zufällige Fehler (Verfälschungen, Verluste, Verzögerungen) in zufälligen Abständen einbaut (simulierte Fehlerquelle im Übertragungskanal). Alle zufälligen Zeiten (und Werte) sind prinzipiell sinnvoll zu wählen, sodass die Simulation das gewünschte Verhalten zeigt.

Prinzipiell sind alle Programme so zu entwickeln, dass diese prinzipiell auf mehreren Hosts zum Einsatz gebracht werden können, auch wenn wir diese nur am lokalen Host ausführen werden. Dafür sind unter anderem die Ports geeignet zu wählen bzw. zu konfigurieren (z.B. in einer JSON Datei oder in eine TOML Datei).

Bei Fragen bitte mich über die üblichen und vereinbarten Kommunikationskanäle kontaktieren!

Nummer	Thema	Note
1	Einfache Client/Server (1 gleichzeitiger Client) Anwendung zur Authentifizierung an einen Server mittels eines *vereinfachten* (nur ASCII oder JSON) CHAP Protokolls. Wobei der "Response Value" als verschlüsselter "Challenge Value" (basierend auf dem XXTEA Verschlüsselungsalgorithmus, Source Code kann direkt verwendet werden) ermittelt wird. Der Server kann mittels einer variablen Anzahl an Benutzern (samt Passwörtern, d.h. zumindest hart kodiert im Sourcecode oder als Kommandozeilenparameter oder in einer (json) Datei) 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 und zusätzlich mittels IPv4 Prüfsummen (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal). Der Client sendet zufällige ASCII Zeichen (Bytes) an den Server, die der Server dekodiert ausgibt. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar. Siehe Folien `encoding.pdf` und das Skript `distsys1.pdf`.	4

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
3	Einfache Client/Server (1 gleichzeitiger Client) Anwendung mit Request/Response-Kommunikation mit Fehlererkennung (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal). Die Datenübertragung erfolgt bitweise mittels der Blockkodierung 4B5B der Leitungskodierung NRZ-I. Der Client sendet zufällige ASCII Zeichen (Bytes) an den Server, die der Server dekodiert ausgibt. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar. 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). Der Client sendet zufällige ASCII Zeichen (Bytes) an den Server, die der Server dekodiert ausgibt. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar. Siehe Skript `distsys1.pdf`.	4
5	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 und ein Time-Server implementiert werden. Für jeden dieser Server ist ein entsprechender Client zu implementieren. Siehe Folien `server_programming.pdf`	4
6	Simulation der Weiterleitung in IP in einem einfachen konfigurierten Netz (ca. 7 Nodes) mit einer Einspeisung und jeweils zufälligem Ziel. Jeder Node stellt einen Prozess dar, der mit anderen Prozessen logisch verbunden ist. Diese Nodes, die Kommunikationsbeziehungen, als auch der Einspeisepunkt sind eben in der Konfigurationsdatei (z.B. in JSON) enthalten. Pakete werden zu zufälligen Zeiten eingespeist und sind über die Ausgabe der Prozesse zu verfolgen.	4
7	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
8	Einfache Super-Server-Implementierung. Super-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 ein Daytime-Server und ein Finger-Server implementiert werden. Für jeden dieser Server ist ein entsprechender Client zu implementieren. Siehe Folien `server_programming.pdf`	4

9	Simulation einer Aufzugssteuerung mit genau 3 Etagen (je ein Prozess) und einem Aufzug. Das System soll mit Hilfe eines eigenen Programmes mittels der Kommandos `CALL <floornr>` (Rufe Aufzug zu Etage `floornr`, `MOVE <floornr>` (Bewege Aufzug zu Etage `floornr`) verfügen (mittels eines REPL-Interpreters). Z.B.
	`>>> CALL 3`
	`>>> CALL 2`
	`>>> MOVE 1`
	`>>> MOVE 3`
	Daraufhin wird der Aufzug in die Etage 3 gerufen, während sich dieser zu Etage 3 bewegt, wird der Aufzug auch zur Etage 2 gerufen, in Etage 3 steigt wer und fährt zur Etage 1, danach bewegt sich der Aufzug zur gewünschten Etage 2, es steigt wer ein und fährt zur Etage 3. Die Zeit zwischen den Etagen soll z.B. 3s betragen (per Kommandozeile konfigurierbar). Zusätzlich sollen auch Fehler auftreten können: der Aufzug kann während des Fahrens zwischen zwei Etage stecken bleiben und auch der Türe kann entweder nicht aufgehen oder nicht schließen. Diese Situationen sind zu beachten und zu behandeln (dem Benutzer und dem Rettungsdienst melden).
		4

10	Simulation des Reader/Writer Problems auf Basis eines Dateiservers (eigener Prozess, multi-threaded Server). Jeder Client verbindet sich mit dem Dateiserver (eigner Thread), sendet zufällig Lese- oder Schreibanfragen (Zeitbereiche sind sinnvoll zu wählen) für die Datei (mit Zeilenbereich und u.U. Inhalt) und bekommt die Ergebnisse zurück. Als Simulation gesehen soll eine Leseoperation 1s und eine Schreiboperation 5s dauern. Siehe Folien `sync_mechanisms.pdf`. Verwendung von `shared_lock` und `unique_lock` erlaubt.	4

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
11	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). Zuerst müssen sich die Philosophen beim Butler anmelden (und ein eigener Thread für den Philosophen gestartet wird), der ihnen dann einen Sitzplatz zuordnet. Haben alle Philosophen Platz genommen, dann kann das Essen beginnen. Dafür fordert der Philosoph vom Butler zuerst die linke und danach die rechte Gabel an. Der Butler kümmert sich darum, dass es zu keinem Deadlock kommt und verzögert bei Bedarf die Zuteilung einer Gabel.	4
12	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
13	Autosimulation mit einer konfigurierbaren Anzahl an Autos (Prozessen), die ein Rennen nach dem anderen fahren. Der Beginn eines Rennens ist, wenn alle Autos sich beim Rennsystem (Prozess! ein Thread je Auto) zur Startlinie begeben haben. Sind alle dort angekommen, dann startet das Rennen (wiederverwendbare Barrier). Sind alle im Ziel angekommen, dann stoppt das Rennen, jedes Auto (d.h. der Prozess) "misst" seine (zufällige) Zeit und sendet diese an das Rennsystem, das eine Rangliste ermittelt und ausgibt. Dann beginnt das nächste Rennen. (siehe "The Little Book of Semaphores").	4
14	Simulation einer verteilten Synchronisation mit einem zentralen Koordinator: Es gibt einen Koordinator- und n Workerprozesse. Jeder Workerprozess will in zufälligen Abständen (zwischen 3 und 5 Sekunden) in den kritischen Abschnitt eintreten. Im kritischen Abschnitt verbleibt ein Worker 4 Sekunden. Die Konfiguration erfolgt über JSON. Der Server soll resistent gegen Abstürze sein, d.h. sein Zustand muss (zumindest) in einer Datei abgelegt werden und beim Wiederstarten von diesem gelesen werden.	4
15	Simulation einer verteilten Synchronisation basierend auf dem verteiltem Algorithmus: Es gibt insgesamt n Workerprozesse. Jeder Worker will in zufälligen Abständen (zwischen 3 und 5 Sekunden) in den kritischen Abschnitt eintreten. Im kritischen Abschnitt verbleibt ein Worker 4 Sekunden. Die Konfiguration erfolgt über JSON.	4

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
16	Simulation einer verteilten Synchronisation basierend auf dem Tokenring Algorithmus: Es gibt insgesamt n Workerprozesse. Jeder Workerthread will in zufälligen Abständen (zwischen 3 und 5 Sekunden) in den kritischen Abschnitt eintreten. Im kritischen Abschnitt verbleibt ein Worker 4 Sekunden. n ist über die Kommandozeile zu konfigurieren.	4
17	Wahlalgorithmus im Ring basierend auf dem Chang-Roberts Algorithmus: Es gibt insgesamt n Workerprozesse. Jeder Worker bekommt beim Starten eine zufällige ID und alle 5 Sekunden wird ein neuer Wahlvorgang gestartet. n ist über die Kommandozeile zu konfigurieren.	4
18	Einfache Client/Server (beliebige Anzahl an gleichzeitigen Clients) Anwendung mit Request/Response-Kommunikation mit zur Datenübertragung von beliebig großen ganzen, vorzeichenbehafteten Zahlen mit Fehlererkennung (DATA, ACK, NAK, zufällige simulierte Fehlerquelle im Übertragungskanal) mittels Signed LEB128. Jede Zahl wird mittels einer Prüfsumme gemäß Fletcher_16 (mit M=255) versehen, die im Zuge der Fehlererkennung überprüft wird.	4
19	Simulation einer Ampelsteuerung: 2 hintereinander geschaltete Ampeln (je ein Netzprozess) mit je 4 Straßeneinmündungen. An jeder Straßeneinmündung ist eine Straße verbunden, wobei es nur eine Straße ist, die die beiden Ampeln miteinander verbindet. Jede Straße stellt ebenfalls einen Netzprozess dar und hat eine spezifizierte Auslastung der Straßeneinmündungen; Konfiguration z.B. mittels JSON-Datei; Auswertung (opt. Steuerung der Ampeln auf Grund der Präsenz von Fahrzeugen)	4
20	Simulation einer Autobahn mit Überwachung der Geschwindigkeit (). Es stehen in der ... 2 Messstellen zur Verfügung, die in einem bestimmten Abstand (z.B. 10km) voneinander aufgestellt sind. Wird eine Geschwindigkeitsübertretung festgestellt, dann wird der Polizei alarmiert. Diese hat eine Alarmierungszeit von 3 Minuten. Der Polizei wird in der Alarmierungsmeldung die voraussichtlichen Position...	4
21	Ein gegebener 5 Bitcode (Konfiguration der Codewörter mittels Datei) soll so erweitert werden (d.h. links um die entsprechenden Bits vergrößert wird), dass eine vorgegebene Hammingdistanz erreicht wird. Dieser Code wird zur Übertragung genutzt. Der Server und der Client werden entsprechend gestartet, wobei der Client zufällige Codewörter an den Client sendet (DATA, ACK, NAK, simulierte Fehlerquelle im Übertragungskanal).	4
22	Einfache Client/Server (1 gleichzeitiger Client) Anwendung, die zufällige ASCII Zeichen mittels zuverlässiger Übertragung (Stop-and-Go Algorithmus) vom Client zum Server überträgt. Bei der Übertragung kann es zu den bekannten 4 Fällen kommen. Der Grundvorrat an ASCII Zeichen (Bytes) ist per Kommandozeile konfigurierbar. Siehe Skript `distsys1.pdf`.	4

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
23	Schifferl versenken (h2h, optional: h2m), 1 Server, 1 oder 2 Clients, Anmelden, Abmelden, Spielen, Punktestand abfragen (Persistenz in Dateien basierend auf JSON), Konfiguration mit TOML	4
24	Einfache Broker-Implementierung. Broker soll mittels TOML 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`. 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
25	Wahlalgorithmus basierend auf Bully-Algorithmus (inkl. ”theoretischem” Vergleich mit dem Chang-Roberts Algorithmus, siehe Foliensatz `dist_sync.pdf`). Konfiguration mit TOML und auch mittels Umgebungsvariablen und Kommandozeilenparameter.	3
26	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
27	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
28	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
29	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
30	Einfache Client/Server (beliebige Anzahl an Clients) Anwendung, die JSON-RPC implementiert. Am Server können mehrere Funktionen mit Tags registriert werden. Ein Verzeichnisdienst, der zu einem gegebenen Tag alle registrierten Funktionen zurückliefert ist zu implementieren.	3

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema	Note
31	Netzwerk-basiertes Iteratives Prisoner's Dilemma mit zentralem Server. Clients können eigene Strategien implementieren und gegen andere antreten.	3
32	Spiel Go (h2h), 1 Server, 1 oder 2 Clients, Anmelden, Abmelden, Punktestand abfragen (Persistenz in Dateien basierend auf JSON), Konfiguration mit TOML, Graphische Darstellung auf der Konsole, Implementierung des Protokolles basierend auf einem endlichen Automaten	3
33	Einfache Client/Server (1 gleichzeitiger Client) Anwendung, die zufällige ASCII Zeichen mittels zuverlässiger Übertragung (Sliding-Window Algorithmus) vom Client zum Server überträgt (basierend auf TCP). Paketverluste, Datenverfälschungen und Fehler bei der Paketreihenfolge sind zu simulieren. Der Grundvorrat an ASCII Zeichen (Bytes) ist per Kommandozeile konfigurierbar. Siehe Skript `distsys1.pdf`.	3
34	Implementierung eines HTTP-Proxy auf Basis von HTTP/1.1, der lediglich für die Methoden GET und HEAD seine Proxy und Caching Funktionalität für Textdateien erfüllt und alle anderen Anfragen und Antwortet einfach durchreicht. Muss funktionsfähig sein (es dürfen keine HTTP-Bibliotheken verwendet werden).	3
35	Netzwerk-basiertes RobotGame wie robowiki mit zentralem Server.	2
36	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
37	Erstellung eines einfachen MapReduce-Systems samt Beispielanwendung (inkl. Aufarbeitung des Themas, siehe auch DISCO)	2
38	Simulation von TCP (d.h. grobe Nachbildung der Funktionalität, inkl. Ports), die in einer P2P Situation zufällige ASCII Zeichen mittels zuverlässiger Übertragung (Sliding-Window Algorithmus) vom Client zum Server überträgt (basierend auf UDP! für die Simulation ist der Port 6666 am Server zu verwenden). Paketverluste, Datenverfälschungen und Fehler bei der Paketreihenfolge sind zu simulieren. Der Grundvorrat an ASCII Zeichen (Bytes) ist per Kommandozeile konfigurierbar. Siehe Skript `distsys1.pdf`.	2
39	Simulation des Distance Vector Algorithmus für ein konfigurierbares Netzwerk, zufälliger Ausfall von Verbindungen	2
40	Simulation des Link-State Algorithmus für ein konfigurierbares Netzwerk, zufälliger Ausfall von Verbindungen	2
41	Simulation des Spanning Tree Algorithmus für ein konfigurierbares Netzwerk, zufälliger Ausfall von Verbindungen	2
42	Verzeichnissynchronisation basierend auf Datum und Hashwert zwischen zwei Netzprozessen	2
43	Implementierung eines Remote Method Invocation Systems basierend auf `asio`	2