Rechnernetze und verteilte Systeme
Praxis 0: Hausaufgaben Guide

Fachgruppe Telekommunikationsnetze (TKN)

22. Oktober 2024

Im Verlauf des Semesters werden Sie drei Aufgabenblätter zu Netzwerktechnik mit C bearbeiten, die jeweils Teil der Portfolioprüfung sind. In diesem Blatt 0 lernen Sie, wie Sie ihr Projekt einrichten, wie Sie die Aufgaben effizient bearbeiten können, und wie Sie ihre fertigen Abgaben richtig einreichen.

1. Projekt-Setup (CMake)

C ist eine kompilierte Sprache. Das bedeutet, dass Ihr geschriebener C-Code nicht direkt ausgeführt werden kann, sondern in eine ausführbare Maschinencode-Datei übersetzt werden muss. Dieser Prozess wird *Kompilieren* oder im Englischen *building* genannt.

Für die Aufgaben verwenden wir CMake[1] als Build-System. CMake erspart uns sehr viel kleinteilige Konfigurationsarbeit, indem wir für unser gesamtes Projekt nur eine globale Konfigurations-Datei, die `CMakeLists.txt`, erstellen müssen.

Neues Projekt erstellen

Das Erstellen eines neuen CMake Projektes ist leicht. Für dieses Beispiel gehen wir davon aus, dass Sie eine C-Datei `hello_world.c` haben, die Sie in eine ausführbare Datei `hello_world` kompilieren wollen (ausführbare Dateien besitzen keine Dateiendung). Falls Ihre Code-Datei anders heißt, ersetzen Sie `hello_world.c` durch den Namen ihrer Datei. Das gleiche gilt analog für den Namen der ausführbaren Datei.

Erstellen Sie in dem gleichen Ordner, indem sich Ihr C-Code befindet, eine `CMakeLists.txt` Datei mit dem folgenden Inhalt:

```
project(RN-Praxis) # Der Name unseres Projekts
cmake_minimum_required(VERSION 3.5) # Minimale CMake-Version 3.5
set(CMAKE_C_STANDARD 11) # Wir verwenden C-Version C11

# Diese Zeile ist die Anweisung, dass wir eine ausführbare Datei erstellen wollen
add_executable(hello_world hello_world.c)
```

[1] `https://cmake.org/`

```
7
8  # Wenn Sie mehrere Code-Dateien haben, werden diese Leerzeichen getrennt angehängt:
9  # add_executable(executable first.c second.c third.c)
10
11 # Diese Zeilen sind für das Erstellen der Abgabedatei relevant
12 set(CPACK_SOURCE_GENERATOR "TGZ") # Abgabe soll als .tar.gz erstellt werden
13 # Die fertige Abgabe enthält nur den Quellcode und nicht ihr Build-Verzeichnis
14 set(CPACK_SOURCE_IGNORE_FILES ${CMAKE_BINARY_DIR} /\\..*$)
15 set(CPACK_VERBATIM_VARIABLES YES) # Variablen sollen nicht optimiert werden
16 include(CPack) # Wir nutzen CPack um das Archiv zu erstellen
```

Code kompilieren

Mit dieser `CMakeLists.txt` Datei können Sie nun erst ihr `build`-Verzeichnis, und dann ihre ausführbare Datei wie folgt erstellen. Falls nicht anders spezifiziert, werden Konsolenbefehle in ihrem Projektverzeichnis ausgeführt(Verzeichnis in dem sich ihre `CMakeLists.txt` befindet):

```
1 cmake -B build -D CMAKE_BUILD_TYPE=Debug # Erstellt das Build-Verzeichnis
2 make -C build # Kompiliert in 'build' wie per add_executable spezifiziert
3 # Alternativ kürzer: 'cmake -B build -D CMAKE_BUILD_TYPE=Debug && make -C build'
```

Sobald Sie Änderungen an ihrem Code vorgenommen haben, können Sie die ausführbare Datei mit dem gleichen Befehl neu erstellen. Falls Sie eine IDE wie CLion verwenden, können Sie die ausführbare Datei auch direkt mit der IDE builden. Dafür müssen Sie nur den Pfad des CMake Build-Verzeichnisses in den Einstellungen als „build“ angeben(*Build, Execution, Development -> CMake -> Build directory*).

Code ausführen & testen

Nachdem Sie Ihren Code kompiliert haben, können Sie ihn ausführen und testen. Falls Sie z.B. eine ausführbare Datei `hello_world` erstellt haben, und zusätzlich noch eine IP-Adresse „172.0.27.14“ als erstes Argument übergeben wollen, können Sie das mit dem folgenden Befehl tun:

```
1 ./build/hello_world 172.0.27.14 # Führt hello_world aus mit IP-Adresse als argv[1]
```

IDEs bieten hierfür auch einige hilfreiche Extra-Funktionen in Form von Run-Configurations[2].

Zu jeder Abgabe werden Ihnen zusätzlich Python-Tests zur Verfügung gestellt, die die Korrektheit Ihrer Implementierung testen. Um diese zu nutzen, müssen Sie zunächst Python und pytest installieren.

```
1 sudo apt update # Aktualisiert die Paketliste
2 sudo apt install python3 python3-pip # Installiert Python und pip (Paketmanager)
3 pip3 install pytest # Installiert pytest mit pip
```

Nun können Sie die Tests mit den folgendenen Befehlen ausführen:

[2] `https://www.jetbrains.com/help/clion/running-applications.html`

```
# Führt alle Tests im Projekt aus:
pytest
# Führt nur die Tests in einem testfile test_praxis0.py aus:
pytest test/test_praxis0.py
# Führt nur einen spezifischen Test aus:
pytest test/test_praxis0.py -k test_executable_exists
```

Abgabedatei erstellen

Um Ihre fertige Abgabe einzureichen, **müssen** Sie die Abgabedatei mit folgenden Befehl erstellen:

```
make -C build package_source # Erstellt ihre Abgabedatei im 'build' Ordner
```

2. Probeabgabe

Für alle folgenden Abgaben gelten stets die gleichen Abgabeformalitäten (siehe A). Um Ihnen die Möglichkeit zu geben zu testen, dass das Kompilieren und Erstellen der Abgabedatei für Sie funktioniert, gibt es eine Probeaufgabe 0.

Ihre Aufgabe ist es, ein einfaches „Hello World“ Programm zu schreiben, dass bei Aufruf den Text „Hello World!“ auf der Konsole ausgibt. Erstellen Sie dafür manuell ein CMake Projekt wie oben beschrieben und eine C-Datei(oder auch mehrere), die ihren Code enthält.

Nutzen Sie `hello_world` als Output-Namen des `add_executable` Befehls in Ihrer `CMakeLists.txt` Datei. Stellen Sie sicher, dass Sie die Zeilen zur Abgabenerstellung korrekt in Ihrer `CMakeLists.txt` Datei eingefügt haben, und erstellen Sie die Abgabedatei. Diese können Sie dann in ISIS hochladen.

Nach Bearbeitung dieses Praxis 0 Zettels, werden wir Ihre Abgaben einmal testen, sodass Sie via ISIS sehen, ob alles geklappt hat. Wir empfehlen dringend, die Probeabgabe durchzuführen, um spätere Probleme (und ggf. eine 0 Punkte Bewertung) bei den späteren Praxis Aufgaben zu vermeiden.

3. Voraussetzungen für das Modul

Rechnernetze ist kein Programmiersprachenmodul, sondern lehrt im Kern Netzwerktechnik. Das bedeutet für Sie, dass benötigte C-Grundlagen für dieses Modul vorausgesetzt, und daher nicht gelehrt, werden. Wir werden Sie beim Lösen der Aufgaben mit einem Schwerpunkt auf die Netzwerkaspekte unterstützen. Um Ihre C-Kenntnisse potentiell aufzufrischen, können wir verschiedene Tutorials[3][4] empfehlen, die Sie sich vor dem ersten Aufgabenblatt anschauen sollten.

[3] `https://www.w3schools.com/c`
[4] `https://www.learn-c.org`

Die verschiedenen Netzwerkpraktiken, wie das Erstellen von, und Kommunizieren mit Sockets, werden im "Beej's Guide to Socket Programming"[5] übersichtlich erklärt. Hier sind vor allem Kapitel 5 mit den einzelnen benötigten Funktionsaufrufen und Kapitel 6 mit anschaulichen Beispielen sehr hilfreich.

4. Arbeitsplanung

Die Aufgabenstellungen in diesem Kurs folgen keinem wöchentlichen Rhythmus, sondern haben eine Bearbeitungszeit von ca. einem Monat. Der Arbeitsaufwand für die Bearbeitung ist allerdings dennoch vergleichbar. Daher ist es wichtig, kontinuierlich an den Aufgaben zu arbeiten, erwarten Sie nicht, eine Lösung innerhalb nur einer Woche zu erarbeiten!

Um Ihnen die Lösung zu erleichtern, enthalten die Aufgabenstellungen viele Hinweise, und Verweise auf Lösungsansätze. Lesen Sie diese daher aufmerksam, am besten vollständig, bevor Sie mit der Lösung beginnen. Erfahrungsgemäß lässt sich dadurch einige Zeit einsparen, die sonst auf frustrierende Fehlersuche verwendet wird.

Das zu erarbeitende Programm ist vergleichsweise umfangreich. Es lohnt sich, vor dem Programmieren einen groben Plan zu überlegen, wie Ihre Implementierung funktionieren soll. Dadurch können Sie beispielsweise vermeiden, einen Fall zu übersehen, der im Nachhinein schwer adäquat zu behandeln ist.

5. Fehler beheben

Nicht alles funktioniert sofort, und das ist nicht das Ende der Welt. Allerdings werden Sie in diesem Praktikum lernen müssen, mit Fehlern, auf die Sie stoßen, eigenständig umzugehen. Alle Fehler sind im Grunde ein Mismatch zwischen der Erwartungen an das Verhalten eines Systems mit der Realität. Die Fehlersuche ist dabei stets ein Prozess der Konkretisierung dieses Mismatches: Von „Das Programm sollte nicht abstürzen“ zu „Diese Variable sollte einen validen Pointer enthalten“. Für diese Konkretisierung gibt es verschiedene Herangehensweisen, und Tools, die Ihnen zur Verfügung stehen.

5.1. Fehlermeldungen verstehen

Viele Fehler äußern sich durch eine Fehlermeldung. Fehlermeldungen bestehen meistens aus einem *Stacktrace*, und einer Beschreibung des Fehlers. Der erste Schritt bei der Fehlersuche ist, die Fehlermeldung zu lesen.

Der Stacktrace ist die Liste von aufeinander folgenden Funktionsaufrufen, die zu dem Fehler geführt haben. Diese Liste ist in der Regel von oben nach unten aufgebaut, wobei die unterste Funktion den Fehler verursacht hat. Mithilfe des Stacktraces können Sie so herausfinden, an welcher Stelle im Code der Fehler aufgetreten ist. Lassen Sie sich nicht von der Menge an Informationen abschrecken, die der Stacktrace enthält. Häufig

[5] `https://beej.us/guide/bgnet/pdf/bgnet_a4_c_1.pdf`

ist nur ein kleiner Teil davon für uns relevant, und Sie können sich auf die *letzten Zeilen* konzentrieren.

Die Beschreibung des Fehlers gibt Ihnen eine Idee, was schief gelaufen ist. Versuchen Sie, die Fehlerbeschreibung in dem Kontext zu verstehen, in dem der Fehler aufgetreten ist. Zusätzlich können sie die Fehlerbeschreibung in eine Suchmaschine eingeben, um mehr Informationen, meist in Form von Stackoverflow-Posts[6], zu erhalten. Dabei sollten sie personalisierte Details aus der Fehlermeldung wie z.B. Dateipfade oder Variablennamen entfernen, um die Suche zu erleichtern. So können sie eine genauere Vermutung darüber entwickeln, wodurch der Fehler möglicherweise verursacht wurde.

5.2. Tool-Support

Verschiedene Tools können es dramatisch einfacher machen Fehlerquellen zu finden.

Ein interaktiver *Debugger* wie `gdb`[7] erlaubt Ihnen, die Ausführung Ihres Programms zu pausieren und den Zustand des Programs im Detail zu untersuchen. So können Sie ihr Programm zeilenweise überwachen und damit exakt lokalisieren, wo genau etwas Unerwartetes passiert. Mit dieser mächtigen Funktionalität ist der Debugger ein must-have für die Fehlersuche.

Moderne IDEs wie *CLion*[8] oder *VSCode*[9] integrieren den Debugger direkt an den Ort wo Code geschrieben wird und erleichtern so seine Verwendung. Es gibt allerdings auch standalone Tools wie `gdbgui`[10] die eine ähnliche Oberfläche bieten, falls Sie keine IDE verwenden wollen.

Die manuelle Speicherverwaltung in C ist zusätzlich eine häufige Fehlerquelle. Um ihre Speicherverwaltung zu überprüfen, können Sie ein Tool wie `valgrind`[11] verwenden. Valgrind überwacht die Interaktion Ihres Programms mit dem Speicher und kann so auf Fehler hinweisen. Es lohnt sich grundsätzlich ihr Programm mit valgrind zu testen — im besten Fall um sich zu bestätigen, dass alles in Ordnung ist.

5.3. Coding best practices

Der beste Fehler ist der, der nie gemacht wird. Um Fehler so weit wie möglich zu vermeiden gibt es eine Reihe an Best Practices. Zu jeder Regel gibt es natürlich auch sinnvolle Ausnahmen, aber für die grundlegende Herangehensweise ans Programmieren sind sie sehr nützlich.

Lesbarer Code

Damit Ihr Programm sich möglichst leicht verwenden, verändern, und erweitern lässt, sollte es sich möglich erwartbar verhalten. Ein wichtiger Aspekt davon ist, *lesbaren* Code

[6] `https://stackoverflow.com/`
[7] `https://www.gnu.org/software/gdb/`
[8] `https://www.jetbrains.com/clion/download/` - Debugger Tutorial: `https://youtu.be/5wGsRdumueU`
[9] `https://code.visualstudio.com/download`
[10] `https://www.gdbgui.com/`
[11] `https://valgrind.org/`

zu schreiben. Code wird im Allgemeinen nicht einmal geschrieben, sondern wird immer wieder verändert und insgesamt sehr häufig gelesen. Dies trifft insbesondere auch in diesem Kurs zu, da die Aufgaben auf einander aufbauen und Sie Ihre Implementierung im Verlauf des Semesters sukzessive erweitern. Wenn sie Best Practices zu lesbarem Code folgen, vereinfacht sich das Zusammenarbeiten im Team, das wieder Anfangen nach dem Wochenende, sowie die Fehlersuche.

Lesbarer Code bedeutet, dass sich seine Bedeutung möglichst leicht und ohne vorausgesetztes Kontextwissen erschließen lässt. Das bedeutet, dass der kürzeste oder spannendste Weg etwas zu programmieren häufig nicht der beste ist. Auf den zweiten Blick entpuppt sich der kürzeste Weg oft als der unverständlichste.

Dies gilt genauso für Variablennamen, Funktionsnamen, und Kommentare: Benennen sie Variablen und Funktionen so, dass ihr Zweck klar wird, auch ohne den Code selbst geschrieben haben zu müssen. Verzichten sie auf Abkürzungen, die nicht allgemein bekannt sind, auch wenn das bedeutet, dass ihre Variablennamen länger werden.

Sie können im Internet verschiedenste ausführlichere Regelwerke[12] finden, die beim schreiben von lesbarem Code helfen.

Dokumentation

Die Dokumentation ist ein wichtiges Werkzeug um Code um Informationen wie unintuitives Verhalten zu ergänzen, die nicht direkt eindeutig sind. Dokumentation wird häufig in Form von Funktionsbeschreibungen geschrieben. Funktionsbeschreibungen sollten den Zweck der Funktion, die Parameter, und den Rückgabewert beschreiben. Gleichzeitig ist ihr Code im besten Fall mit angemessener Benennung leserlich genug um ohne Zeilenkommentare verständlich zu sein.

Nicht nur das Schreiben von Dokumentation ist wichtig, sondern auch das Lesen derselben für Funktionen die sie verwenden wollen. Insbesondere in C sind einige der Standardfunktionen nicht hundertprozentig intuitiv zu verwenden. Im POSIX Interface beispielsweise sind viele Namen abgekürzt und erschließen sich nicht unmittelbar (`sockaddr_in` steht selbstverständlich für `socket address internet`).

Insbesondere die `manpages`[13] sind hier hilfreich. IDEs integrieren die Dokumentation häufig direkt, sodass diese beim hovern des Mauszeigers über einem Funktionsnamen angezeigt wird.

Versionskontrolle

Versionskontrolle ist ein mächtiges Tool, das das längerfristige Arbeiten oder Arbeiten als Team an einem Projekt deutlich erleichtert. Durch Versionskontrolle ist es möglich, einen Überblick über Änderungen am Code zu behalten. Das kann zum Verständnis dienen, da sich nachvollziehen lässt, wie eine komplexe Struktur im Laufe der Zeit gewachsen ist, der Kollaboration, da sich Versionen mit Teammitgliedern austauschen lassen, und vor

[12] z.B. `code.tutsplus.com/de/tutorials/top-15-best-practices-for-writing-super-readable-code--net-8118`
[13] z.B. `https://linux.die.net/man/7/socket`

Datenverlust schützen. `git`[14] hat sich hier als quasi Standardtool etabliert. Der Umgang mit git kann anfangs unintuitiv sein, daher gibt es eine Vielzahl an GUIs und Tutorials[15][16] die die Verwendung erleichtern und erklären.

Entwicklungsumgebung

Die Entwicklung von größeren Projekten kann durch eine IDE erleichtert werden. Eine IDE erlaubt das Entwickeln, Debuggen, Versionieren und Verwenden der Konsole, alles innerhalb eines Programms. Durch die vielen Möglichkeiten sind IDEs aber auch stets mit einer initialen Lernkurve verbunden. Zum Beispiel müssten Sie, falls sie eine IDE auf Windows verwenden, nicht nur WSL installieren und für alle Konsolenbefehle nutzen, sondern auch die Toolchain der IDE auf WSL ändern[17]. Die richtige Entwicklungsumgebung für Alle gibt es hier nicht, Sie sollten für sich entscheiden, womit Sie sich am wohlsten fühlen. Das kann die Entwicklung in der Kommandozeile mit `vim`, `gdb` und `git` sein, oder aber eine IDE wie CLion[18]

5.4. Hilfe suchen

Wenn Sie selbst nicht weiterkommen, fragen Sie andere: Ihre Kommilitonen, das Internet, oder uns. Manchmal reicht schon das Formulieren der Frage, um selbst auf die Lösung zu kommen[19]. Ihre Kommilitonen bearbeiten das gleiche Thema und haben vielleicht bereits eine Lösung für das gleiche Problem, oder einen Ansatz der Ihnen noch nicht eingefallen ist. Zusätzlich bieten wir regelmäßige Sprechstunden an, in denen Sie die Aufgaben bearbeiten und Hilfe bei Problemen erhalten können.

[14] `https://git-scm.com/`
[15] `https://www.w3schools.com/git/`
[16] `https://www.youtube.com/watch?v=HkdAHXoRtos`
[17] `https://www.jetbrains.com/help/clion/how-to-use-wsl-development-environment-in-product.html`
[18] `https://www.jetbrains.com/clion/` (Vollversion via Universitäts-Mail)
[19] `https://en.wikipedia.org/wiki/Rubber_duck_debugging`

A. Abgabeformalitäten

Die Aufgaben können in Gruppen aus **ein bis drei Mitgliedern** bearbeitet (und abgegeben) werden. Dazu wählen Sie jeweils in der ersten Woche der Bearbeitungszeit eine (neue) Abgabegruppe. Wir haben dafür entsprechende Gruppenwahl-Module in ISIS eingerichtet. Auch wenn Sie die Aufgaben alleine (d.h. in einer Gruppe mit einem Mitglied) bearbeiten möchten, müssen Sie (jeweils) eine solche Abgabegruppe wählen. Die getätigte Gruppenwahl gilt jeweils für den gesamten Bearbeitungszeitraum eines Praxis-Zettel.

Ab der jeweils zweiten Woche der Bearbeitungszeit können Sie Ihre Lösung abgeben. Ab diesem Zeitpunkt kann die Gruppenwahl nicht mehr verändert werden. Die Gruppenabgabe muss von einem der Gruppenmitglieder in ISIS hochgeladen werden und gilt dann für die gesamte Gruppe. Somit erhalten alle Gruppenmitglieder dieselbe Bewertung. Ohne eine Abgabe auf ISIS erhalten Sie keine Punkte!

Es werden nur via CMake erstellte Abgabearchive im `.tar.gz`-Format akzeptiert. Beachten Sie, dass eine Änderung der Dateierweiterung bei anderen Formate (z.B. `.zip`) nicht zwangsläufig auch das unterliegende Format ändert. Sie erstellen ein entsprechendes Archiv, indem Sie `make -C build package_source` ausführen. Wir empfehlen *dringend*, ihr so erstelltes Archiv einmal zu entpacken, und die Tests auszuführen. So können Sie viele Fehler mit ihrer Projektkonfiguration vor der Abgabe erkennen und vermeiden.

Ihre Abgaben können ausschließlich auf ISIS und bis zur entsprechenden *Abgabefrist* abgegeben werden. Sollten Sie technische Probleme bei der Abgabe haben, informieren Sie uns darüber unverzüglich und lassen Sie uns zur Sicherheit ein Archiv Ihrer Abgabe per Mail zukommen. Beachten Sie bei der Abgabe, dass die Abgabefrist **fix** ist und es *keine Ausnahmen für späte Abgaben oder Abgaben via E-Mail* gibt. Planen Sie einen angemessenen Puffer zur Frist hin ein. In Krankheitsfällen kann die Bearbeitungszeit angepasst werden, sofern sie uns baldmöglichst ein Attest zusenden.

B. Tests und Bewertung

Die einzelnen Tests finden Sie jeweils in der Vorgabe als `test/test_praxisX.py`. Diese können mit `pytest` ausgeführt werden:

```
python3 -m pytest test # Alle tests ausführen
python3 -m pytest test/test_praxisX.py # Nur die Tests für einen bestimmten Zettel
python3 -m pytest test/test_praxis1.py -k test_listen # Limit auf einen bestimmten
    Test
```

Sollte `pytest` nicht auf Ihrem System installiert sein, können Sie dies vermutlich mit dem Paketmanager, beispielsweise `apt`, oder aber `pip`, installieren.

Ihre Abgaben werden anhand der Tests der Aufgabenstellung automatisiert bewertet. Beachten Sie, dass Ihre Implementierung sich nicht auf die verwendeten Werte (Node IDs, Ports, URIs, ...) verlassen sollte, diese können zur Bewertung abweichen. Darüber hinaus sollten Sie die Tests nicht verändern, um sicherzustellen, dass die Semantik nicht unbeabsichtigterweise verändert wird. Eine Ausnahme hierfür sind natürlich Updates der Tests, die wir gegebenenfalls ankündigen, um eventuelle Fehler zu auszubessern.

Wir stellen die folgenden Erwartungen an Ihre Abgaben:

- Ihre Abgabe muss ein `CMake` Projekt sein.
- Ihre Abgabe muss eine `CMakeLists.txt` enthalten.
- Ihr Projekt muss ein Target entsprechend der oben genannten Definition haben (z.B. `hello_world`) mit dem selben Dateinamen (z.B. `hello_world`) (case-sensitive) erstellen.
- Ihre Abgabe muss interne CMake Variablen, insbesondere `CMAKE_BINARY_DIR` und `CMAKE_CURRENT_BINARY_DIR` unverändert lassen.
- Ihr Programm muss auf den EECS Poolrechnern[20] korrekt funktionieren.
- Ihre Abgabe muss mit CPack (siehe oben) erstellt werden.
- Ihr Programm muss die Tests vom jeweils aktuellen Zettel bestehen, nicht auch vorherige.
- Wenn sich Ihr Program nicht deterministisch verhält, wird auch die Bewertung nicht deterministisch sein.

Um diese Anforderungen zu überprüfen, sollten Sie:

- das in der Vorgabe enthaltene `test/check_submission.sh`-Script verwenden:

```
./test/check_submission.sh praxisX
```

- Ihre Abgabe auf den Testystemen testen.

Fehler, die hierbei auftreten, werden dazu führen, dass auch die Bewertung fehlschlägt und Ihre Abgabe entsprechend benotet wird.

[20] Die Bewertung führen wir auf den Ubuntu 20.04 Systemen der EECS durch, welche auch für Sie sowohl vor Ort, als auch via SSH zugänglich sind.