Praxis 0
Hilfestellung zur Bearbeitung der praktischen Hausaufgaben

Fachgruppe Telekommunikationsnetze (TKN)

17. Oktober 2023

Im Verlauf des Semesters werden Sie drei Aufgabenstellungen bearbeiten, die jeweils Teil der Portfolioprüfung sind. Auf diesem vorgestellten Aufgabenblatt wollen wir Ihnen die Aufgaben über ein minimales Projektsetup näher bringen, mit einer Probeabgabe die entsprechenden Modalitäten schon einmal zeigen, und ein paar Tipps zur Bearbeitung der folgenden Aufgabenstellungen auf den Weg geben.

1. CMake

Für die Aufgaben verwenden wir CMake[1] als Build-System. Ein minimales CMake Projekt können Sie leicht erstellen, selbst wenn dieses noch keinen Code enthält. Erstellen Sie dazu in einem leeren Ordner eine Datei Namens `CMakeLists.txt`, mit dem folgenden Inhalt:

```
cmake_minimum_required(VERSION 3.5)
project(RN-Praxis)
```

Für dieses (vorerst) leere Projekt können Sie ein `build`-Verzeichnis wie folgt erstellen:

```
cmake -Bbuild
```

2. Probeabgabe

Für alle folgenden Abgaben gelten stets die gleichen Abgabeformalitäten (siehe A). Um Ihnen die Möglichkeit geben sicherzustellen, dass die Abgabe für Sie funktioniert haben Sie die Möglichkeit, dass zuvor erstellte Projekt als Probeabgabe auf ISIS hochzuladen. Bei der Veröffentlichung der ersten Aufgabenstellung werden wir Ihre Abgaben einmal testen, sodass Sie via ISIS sehen ob alles geklappt hat. Beachten Sie, dass es diese Aufgabe nicht verpflichtend ist und *keine* Punkte für die Portfolioprüfung beiträgt.

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

3. Erwartete Fähigkeiten

Die Aufgabenstellungen sind in C zu lösen. C bietet sich einerseits für die systemnahe Programmierung, die in diesem Kurs nötig ist, an. Darüber hinaus wird in der Informatik, und damit für einen Großteil von Ihnen, die Programmierung anhand von C gelehrt.

Wir müssen allerdings auch darauf hinweisen, dass dieser Kurs sich auf Netzwerke, nicht auf Programmierung fokussiert. Entsprechend setzen wir hier eine Grundlage voraus die wir nicht hier vermitteln. Daher werden wir Sie beim Lösen der Aufgaben mit einem Schwerpunkt auf die Netzwerkaspekte unterstützen. Sollten Sie Ihre C-Kenntnisse auffrischen wollen können wir verschiedene Tutorials[2][3] empfehlen, die Sie sich vor der ersten Aufgabenstellung anschauen können.

4. Arbeitsplanung

Die Aufgabenstellungen in diesem Kurs folgen keinem wöchtenlichen Rhytmus, 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 euch 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 beandeln ist.

5. Mit Fehlern umgehen

Nicht alles funktioniert sofort und das ist nicht das Ende der Welt. Allerdings werden Sie in diesem Praktikum lernen müssen, mit den Problemen, auf die Sie stoßen werden, eigenständig umzugehen. Alle Fehler sind im Grunde ein Mismatch zwischen Ihren 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.

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

5.1. Fehlermeldungen

Viele Fehler äußern sich durch eine Fehlermeldung, sei es durch den Compiler, oder auch durch Ihr Programm, selbst wenn es ein `segfault` ist. Lesen Sie diese aufmerksam durch: Auch wenn Fehlermeldungen häufig mehr Boilerplate enthalten als unmittelbar relevant ist, ist außerdem eine Beschreibung des direkten Fehlers vorhanden. Fehlermeldungen zu lesen ist daher immer ein Lernprozess und hängt von den verwendeten Tools, wie z.B. dem verwendeten Compiler, ab. Insbesondere beim Übersetzen von Code kommt es häufig zu einer Menge von Folgefehlern. Für Computer ist es extrem schwierig, fehlerhafte Eingaben zu korrekten zu vervollständigen, daher reicht häufig ein Fehler aus, um eine Litanei von Folgefehlern zu generieren. Lassen Sie sich davon nicht verunsichern, und versuchen Sie zunächst, den ersten Fehler zu lösen. Häufig stellen sich dadurch eine Großzahl oder gar alle weiteren Fehler als Folgefehler heraus, oder man kann die Lösung auf weitere Fehler anwenden.

Fehlermeldungen erlauben Ihnen außerdem, Lösungen im Internet zu recherchieren, z.B. mit einer generischen Suchmaschine, oder aber bei StackOverflow[4]. Dabei kann es hilfreich sein, selektiv nach Teilen der Fehlermeldung zu suchen. Häufig enthalten diese Details eures Programms, wie Namen von Variablen oder Pfade, welche Suchmaschinen verwirren. Bei möglichen Lösungen die Sie finden sollten Sie versuchen die Systematik nachzuvollziehen. Nicht alle Lösungen lassen sich direkt anwenden, können aber die richtige Idee enthalten. Darüber hinaus lassen sich ähnliche Fehler in der Zukunft vermeiden, wenn Sie das Problem verstanden haben.

5.2. Tools

Verschiedene Debuggingtools können Ihnen helfen Fehlerquellen zu finden.

Ein interaktiver *Debugger* wie `gdb`[5] erlaubt euch, die reguläre Ausführung eures Programms zu unterbrechen und interaktiv den Zustand im Detail zu untersuchen. So können Sie das Verhalten zeilenweise überprüfen und damit herausfinden ab wann, und warum, etwas Unerwartetes passiert. Damit die Struktur des Programms exakt dem Quellcode entspricht, und das Programm mit Informationen entsprechenden Zeilen im Quellcode annotiert ist, muss das Programm mit Debugging Symbolen, und ohne Optimierungen kompiliert sein. Dies können Sie beim Erstellen des Build-Verzeichnis' via CMake konfigurieren:

```
1 cmake -Bbuild -DCMAKE\_BUILD\_TYPE=Debug
```

Viele IDEs integrieren Debugger und erleichtern so deren Verwendung. Es gibt allerdings auch standalone Tools wie `gdbgui`[6] die eine ähnliche Oberfläche bieten.

Die manuelle Speicherverwaltung in C ist eine häufige Fehlerquelle. Um die Speicherverwaltung zu untersuchen können Sie Tools wie `valgrind`[7] verwenden. Valgrind überwacht die Interaktion Ihres Programms mit dem Speicher und kann so auf akute oder

[4] `https://stackoverflow.com/`
[5] `https://www.gnu.org/software/gdb/`
[6] `https://www.gdbgui.com/`
[7] `https://valgrind.org/`

potentielle Probleme hinweisen. Auch ohne einen konkreten Fehler zu verfolgen lohnt es sich, das Programm mit valgrind zu testen — im besten Fall um sich zu bestätigen, dass alles in Ordnung ist.

5.3. Best practices

Der Fehler mit der simpelsten Lösung ist der, der nie gemacht wird. Um Fehler zu vermeiden, also fehlerfreien Code zu schreiben gibt es eine Reihe an Best Practices. Diese sind natürlich kein Garant für ein perfektes Programm, und zu jeder Regel gibt es eine sinnvolle Ausnahme, aber für die grundlegende Herangehensweise ans Programmieren sind sie sehr nützlich.

Damit euer Programm sich möglichst leicht verwenden, verändern, und erweitern lässt, sollte es sich möglich erwartbar verhalten. Ein wichtiger Aspekt davon ist, *lesbaren Code* zu schreiben. Code wird im Allgemeinen nicht einmal geschrieben, sondern wird immer wieder verändert. Dies trifft insbesondere auch in diesem Kurs zu, da Sie Ihre Implementierung im Verlauf des Semesters sukzessive erweitern. Durch lesbaren Code erleichtern Sie sich das Zusammenarbeiten im Team, das wieder Anfangen nach dem Wochenende, sowie die Fehlersuche.

Was genau lesbarer Code ist, ist ein kontroverses Thema, als grundsätzliche Richtlinie sollte sich die Semantik möglichst leicht erschließen. Das bedeutet auch, dass nicht immer der kürzeste, interessanteste, oder coolste Weg der beste ist, sondern der langweiligste. Außerdem ist Dokumentation ein wichtiges Werkzeug um Code um Informationen ergänzen, die dieser nicht schon beinhaltet. Insbesondere können Sie damit unintuitives Verhalten erklären. Je seltener Sie dies benötigen, desto besser. Im besten Fall ist der Code über adäquate Bezeichnerwahl und angemessene Modularisierung verständlich genug um ohne Kommentare verständlich zu sein. Sie können im Internet verschiedenste Regelwerke[8] finden, die beim schreiben von lesbarem Code helfen.

Nicht nur das Schreiben von Dokumentation ist ein wichtiger Aspekt, auch das Lesen derselben. 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`). Lesen Sie daher die Dokumentation für Funktionen die Sie verwenden aufmerksam. Insbesondere die `manpages`[9] sind hier hilfreich.

Versionskontrolle ist ein mächtiges Tool, das das längerfristige Arbeiten an einem Projekt erleichtert. Durch Versionskontrolle ist es möglich, einen Überblick über Änderungen am Code zu behalten. Das kann zum Verständnis dienen, wenn sich nachvollziehen lässt, wie eine komplexe Struktur im Laufe der Zeit gewachsen ist, der Kollaboration, da sich Versionen mit Teammitgliedern austauschen lassen, oder auch vor Datenverlust durch Versehen, Stromausfall, oder Defekten schützen. `git`[10] hat sich hier als quasi Standardtool etabliert. Der Umgang mit git kann unintuitiv sein, daher gibt es eine Vielzahl an

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

GUIs und Tutorials[11][12] die die Verwendung erleichtern und erklären.

Die Entwicklung von größeren Projekten kann durch eine IDE erleichtert werden. Eine IDE erlaubt das Entwickeln, Debuggen und tracken von Revisionen innerhalb eines Tools. Durch die vielen Möglichkeiten sind IDEs aber auch stets mit einer Lernkurve verbunden. Die richtige Umgebung 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[13]

5.4. Hilfe suchen

Wenn Sie selbst nicht weiterkommen, fragen Sie andere: Ihre Kommilitonen, das Internet, oder uns. Häufig reicht schon das formulieren der Frage, um selbst auf die Lösung zu kommen[14]. Ihre Kommilitonen bearbeiten das gleiche Thema und haben vielleicht bereits eine Lösung für das gleiche Problem, oder einen Ansatz der euch noch nicht eingefallen ist Außerdem bieten wir im Verlauf des Semesters Sprechstunden an, in denen Sie die Aufgaben bearbeiten und Hilfe bei Problemen erhalten.

[11] `https://www.w3schools.com/git/`
[12] `https://www.youtube.com/watch?v=HkdAHXoRtos`
[13] `https://www.jetbrains.com/clion/` (Vollversion via Universitäts-Mail), `https://youtu.be/5wGsRdumueU`
[14] `https://en.wikipedia.org/wiki/Rubber_duck_debugging`

A. Abgabeformalitäten

Die Aufgaben sollen von Ihnen in Gruppenarbeit mit jeweils *bis zu drei Kursteilnehmern gelöst* werden. Um die Verwaltung auf ISIS einfach zu halten muss *jedes Gruppenmitglied seine Abgabe* hochladen. Fügen Sie Ihrer Abgabe eine Datei `group.txt` hinzu die Name und Matrikelnummer aller Gruppenmitglieder enthält. Dadurch bleibt die reguläre Gruppenarbeit bei unserem Plagiarismuscheck unbeachtet.

Ihre Abgaben laden Sie auf ISIS bis zur entsprechenden *Abgabefrist* hoch. Beachten Sie bei der Abgabe, dass die Abgabefrist fix ist und es *keine Ausnahmen für späte Abgaben* gibt. Planen Sie also einen angemessenen Puffer zur Frist hin ein um Eventualitäten, die Ihre Abgabe verzögern könnten, vorzubeugen. In Krankheitsfällen kann die Bearbeitungszeit angepasst werden, sofern diese ärztlich belegt sind.

Abgaben werden nur im `.tar.gz`-Format akzeptiert. Sie können ein entsprechendes Archiv erstellen, indem Sie das folgende Snippet an Ihre `CMakeLists` anhängen und im Build-Ordner `make package_source` ausführen:

```
# Packaging
set(CPACK_SOURCE_GENERATOR "TGZ")
set(CPACK_SOURCE_IGNORE_FILES ${CMAKE_BINARY_DIR} /\\..*$ .git .venv)
set(CPACK_VERBATIM_VARIABLES YES)
include(CPack)
```