Technology
Arts Sciences
TH Köln

# Betriebssysteme und verteilte Systeme

## Praktikumsdokumentation

ausgearbeitet von

Lukas Momberg
Dennis Goßler
Jona Siebel
Daniel Mentjukov

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang
Allgemeine Informatik

Eingereicht bei: Frau Ekellem
Technische Hochschule Köln

Gummersbach, im Juni 2021

# Inhaltsverzeichnis

# 1 Einleitung

In dem folgenden Kapitel wird die Aufgabenstellung aufgegliedert und erläutert. Die Aufgabenstellung umfasste die Entwicklung eines Serverprogramms in C oder C++. In dem Serverprogramm sollten mehrere Clients über eine Netzwerkverbindung Daten ablegen und abfragen können. Des Weiteren sollten die Clients gleichzeitig per TCP verbunden sein und mit einem gegebenen Befehlsset Daten auf den Server schreiben, abfragen und löschen können. Dabei war es wichtig, dass die Daten dabei auf den Server konsistent gehalten werden.

## 1.1 Phase 1

Der erstellte Socket Server sollte mit beliebig vielen Clients über den TCP Port 5678 kommunizieren können. Die Datenhaltung sollte auf Grundlage des Key-Value-Stores erfolgen. Das Datenhaltungssystem sollte folgende Funktionen **PUT**, **GET** und **DELETE** bereitstellen. Die Funktionsaufrufe sollten zum Beispiel mit Call-by-Reference erfolgen. *key* und *value* akzeptieren nur alphanumerische Zeichen und keine Sonderzeichen. Dazu darf der *key* keine Leerzeichen enthalten. Im Folgenden kommen nun die Funktionsbeschreibungen.

Die **get()** Funktion sollte einen Schlüsselwert in der Datenhaltung suchen und den hinterlegten Wert zurückgeben. Wenn der Wert nicht vorhanden ist, wird durch einen Rückgabewert, der kleiner 0 ist, darauf aufmerksam gemacht. Wenn der Wert gefunden wurde, sollte eine 1 zurückgegeben werden.

Die **put()** Funktion sollte einen Wert (value) mit dem Schlüsselwert (key) hinterlegen. Wenn der Schlüssel bereits vorhanden ist, soll der Wert überschrieben werden.

Die **del()** Funktion sollte einen Schlüsselwert suchen und zusammen mit dem Wert aus der Datenhaltung entfernen.

Die Zeichenketten, die ein Client über TCP schickt, stellen im wesentlichen Befehle für den Key Value Server dar. Die Zeichenkette sollte folgende Struktur aufweisen:

**[Befehl]** **[key]** **[value]**

Des Weiteren sollte man den Befehl **QUIT** implementieren, mit dem der Client dafür sorgen kann, dass der Server die Verbindung zum Client beendet, sobald der Client diese Zeichenkette sendet.

## 1.2 Phase 2

Die Phase 2 beinhaltete das Programm gegen mögliche Race Conditions abzusichern. Dazu sollte man sich entsprechende kritische Situationen im Programm anschauen und gemeinsam in der Gruppe Möglichkeiten zur Lösung des Problems suchen. Dazu sollte man passende Mittel der Prozesssynchronisation nutzen. Eine weitere Aufgabe bestand darin die konsistente Datenhaltung für einen exklusiven Zugriff auf den Key-Value Store zu für Clients zu ermöglichen. Dazu sollte ein Client einen alleinigen Zugriff auf den Key-Value-Store anfordern können, sodass nur dieser Zugriff hat und alle anderen Clients blockiert werden. Dafür sollte man das durch den Befehl **BEG** und **END** realisieren. Der erste Befehl dient dazu, um den exklusiven Zugriff zu beginnen und der zweite um den exklusiven Zugriff zu beenden. Während ein Client exklusiven Zugriff auf den Key-Value-Store besitzt, dürfen alle anderen Clients keinen Zugriff erhalten und müssen warten. Aber weiterhin sollten auch nicht exklusive Interaktionen möglich sein, wenn die beiden Befehle nicht verwendet werden. Des Weiteren sollten wir eine Möglichkeit realisieren, dass ein Client sich auf Veränderung der Werte eines Schlüssels registrieren kann. Ein Client sollte sich mit dem Befehl **SUB** für einen Schlüsselwert subscriben können. Sobald eine Änderung für diesen Schlüssel durch einen anderen Client stattfindet, also "published", sollten alle verbundenen Clients für den entsprechenden Schlüssel durch den Server informiert werden. Die subscriber erhalten die gleichen Ausgaben wie der Client, der die Befehle selbst ausgeführt hat. Ein Client sollte sich auf mehrere Keys registrieren können. In der Art eines Chats mit verschiedenen Chaträumen.

## 1.3 Phase 3

Die Phase 3 besteht darin das Programm so zu erweitern, um Clients die Möglichkeit zu geben, bestimmte Systemprogramme auf dem Server aufrufen zu können, um dann deren Ausgabe in den Key-Value Store zu laden. Dafür sollte man den Befehl **OP** implementieren. Der Befehl soll als ersten ein Argument, einen Key und als Zweites einen der folgenden Systembefehle entgegennehmen können:

**date**

**Who**

**uptime**

Die Zeichenkette des Befehls hat so auszusehen:

**OP [key] [systembefehl]**

Des Weiteren sollte man das Programm aus der Perspektive der Nachhaltigkeit betrachten. Das bedeutet das man sich die Systemressourcen besser einteilt mit der Hilfe zum Beispiel von Shared Memory, Semaphore, Message Queue und Filedescriptoren. Dazu sollte man die Freigabe aller Ressourcen, die nur für einen Client nötig waren, sobald dieser Client mit der Zeichenkette **QUIT** signalisiert hat, dass er die Verbindung beenden möchte. Dazu sollte man noch eine Funktion **freeResourceAndExit()** implementieren, um alle weiteren nicht mehr benötigten Ressourcen freizugeben und deren Server-Prozess zu beenden. In der Dokumentation sollte man dann erörtern, wie und wann diese Funktion aufgerufen werden sollte.

# 2 Unser Projekt

Im Folgenden stellen wir unsere Umsetzung der einzelnen Phasen vor und dazu erklären und erörtern wir unsere Entscheidungen. Das Programm wurde in C umgesetzt.

## 2.1 Phase 1

In unserem Programm gibt es am Anfang die Möglichkeit den vorgegebenen TCP Port 5678 zu verwenden oder einen eigenen zu wählen. Das bringt den Vorteil das Programm auf einem anderen Port zu laufen zum Beispiel, wenn dieser Port von einem anderen Programm schon verwendet wird. Dazu noch gibt es die Möglichkeit bei uns im Programm, ob man im Server oder Client Modus starten will. Wenn man sich für einen der beiden Modi entschieden hat, gibt es die Möglichkeit zwischen IPv4 oder IPv6. An unserem Server können sich beliebig viele Clients verbinden wie das auch in der Aufgabenstellung gefordert war. Statt Forking haben wir Threading verwendet das gab uns den Vorteil das Programm auf Windows, Linux (Ubuntu) und macOS ausführbar zu machen und das parallele ausführen. Dazu haben wir noch eine Möglichkeit implementiert das User Manual über den Browser aufrufen zu können. Des Weiteren haben wir die Befehle **PUT**, **GET** und **DELETE** implementiert. Zusätzlich haben wir ein User Interface implementiert, wo die Befehle farblich hervorgehoben werden.
Im Folgenden sieht man Bildschirmaufnahmen vom User Interface.

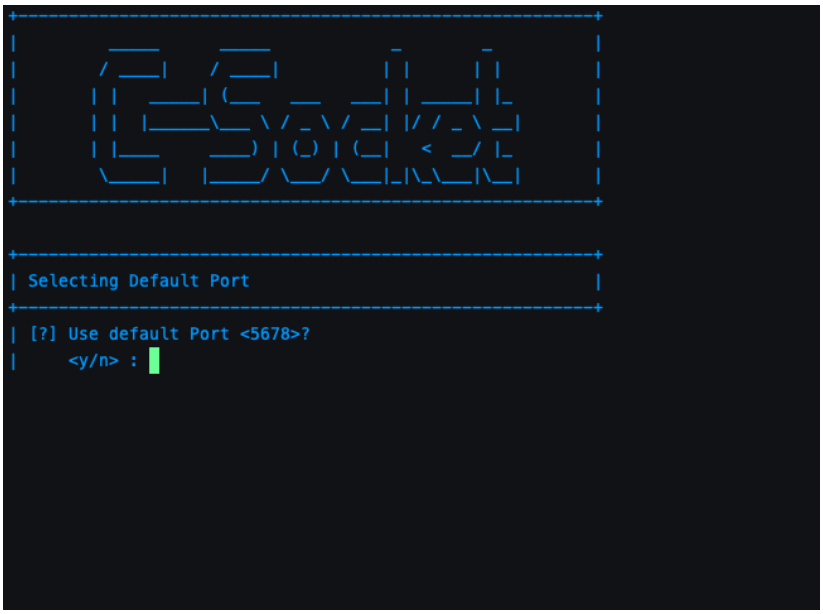


Abbildung 2.1: Screenshot 1 vom User Interface (select Port)

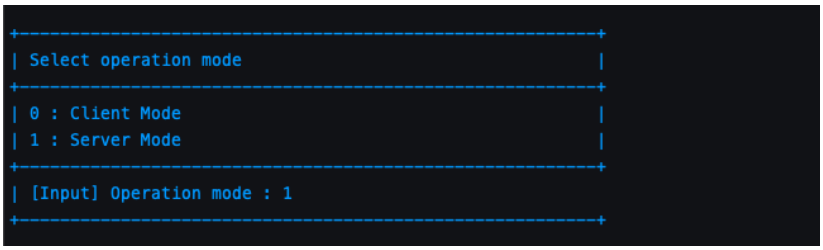


Abbildung 2.2: Screenshot 2 vom User Interface (select operation mode)

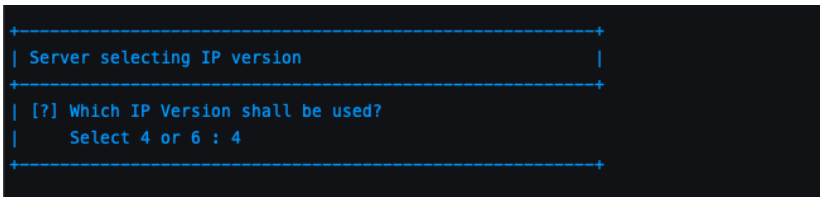


Abbildung 2.3: Screenshot 3 vom User Interface (select IP version)

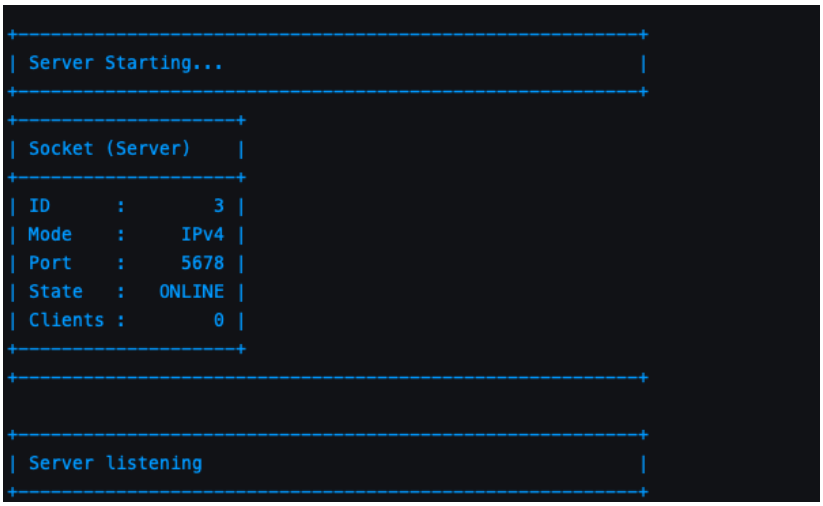


Abbildung 2.4: Screenshot 4 vom User Interface

## 2.2 Phase 2

Um das Programm gegen mögliche Race Conditions abzusichern, haben wir Semaphoren verwendet. Semaphoren geben uns den Vorteil das nie mehr als ein Prozess in einen kritischen Abschnitt eintritt. Dazu haben wir wie gefordert die Befehle **BEG**, **END** und **SUB** implementiert. Dazu haben wir noch eine farbige Konsolenausgabe implementiert. Die Nutzung der Befehle folgt nun in weiteren Bildschirmaufnahmen:

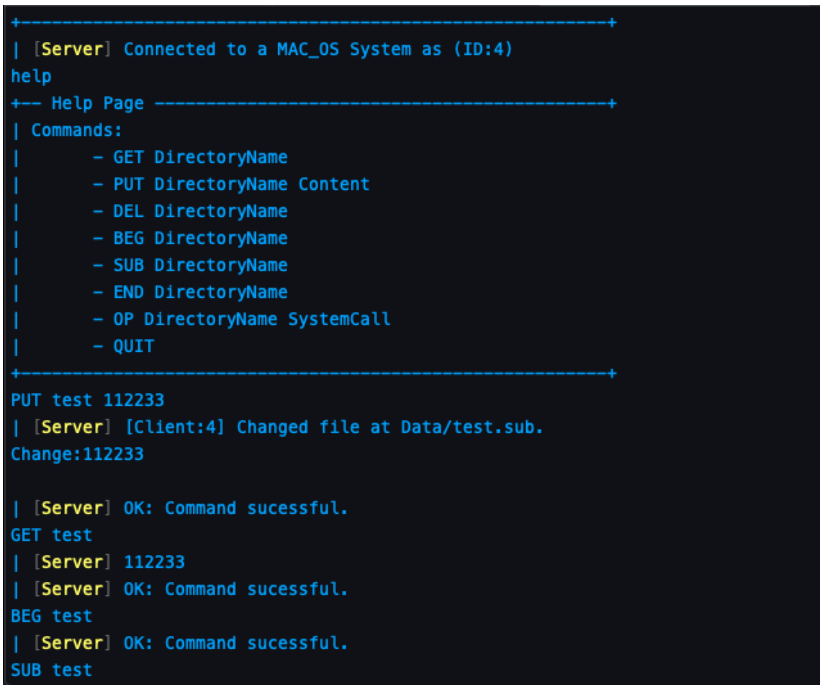


Abbildung 2.5: Screenshot 5 vom User Interface (Befehlsausführung Client)

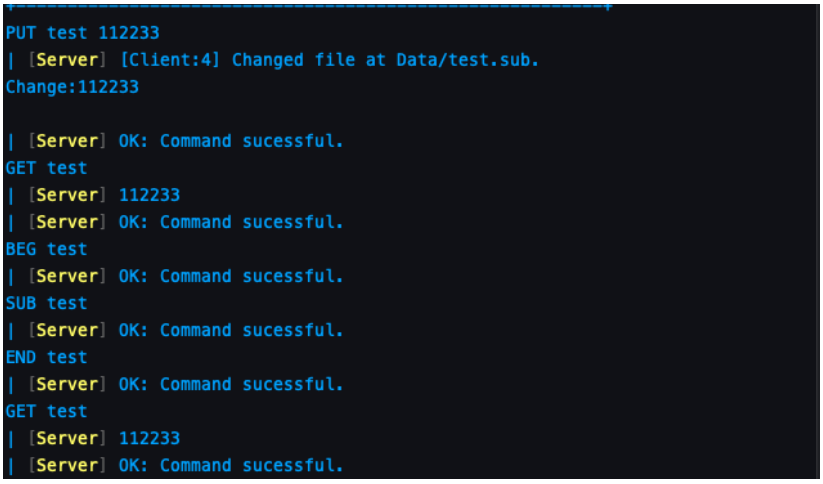


Abbildung 2.6: Screenshot 6 vom User Interface (Befehlsausführung Client)

## 2.3 Phase 3 und Leistungsmatrix

In dieser Phase haben wir das Programm so erweitert, dass wir mit dem Befehl **OP** folgende Systembefehle ausführen können:

**date**

**Who**

**uptime**

Nun folgt unsere Leistungsmatrix:

Tabelle 2.1: Unsere Leistungsmatrix

| % | **Lukas** | **Dennis** | **Jona** | **Daniel** |
|---|---|---|---|---|
| Programmierung | 50 | 40 | 10 | 0 |
| Dokumentation | 0 | 0 | 0 | 100 |
| Anpassung für Linux | 30 | 40 | 30 | 0 |
| Anpassung für macOS | 0 | 10 | 0 | 90 |
| Powerpoint Präsentation | 25 | 25 | 25 | 25 |

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis