# Übungsblatt 12

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 10. Januar 2024
Abgabe bis
**Freitag, 19. Januar 2024, 23:59 Uhr**
Vorführung vom 22. bis zum 26. Januar 2024

Alle Programme müssen unter **Ubuntu 22.04** kompilierbar bzw. lauffähig und ausreichend kommentiert und mit **Makefile** (C, Assembler) versehen sein, um Punkte zu erhalten. Als Compiler sollen **clang** (C) und **nasm** (Assembler) verwendet werden. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Alle Gruppenmitglieder sollten die Abgabe erklären können. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Die Punkte der Aufgaben sind relevant für die Zulassung. Die Punkte der Bonusaufgaben werden auf Ihren Punktestand addiert, werden aber nicht auf die für die Zulassung benötigten Punkte addiert.

**Abgabestruktur**: Jede Abgabe, die die folgende Struktur nicht umsetzt, wird **NICHT** bepunktet bzw. mit 0 Punkten bewertet

- die zu korrigierenden Lösungen müssen bis zur Deadline auf dem `master`-Branch liegen. Lösungen auf anderen Branches werden nicht gewertet
- alle Lösungen müssen in der vorgegebenen Ordnerstruktur (**blattXX/aufgabeYY**) abgelegt werden, wobei `XX` und `YY` durch die jeweiligen Nummern des Zettels und der Aufgaben ersetzt werden sollen. Der Name soll exakt nur aus diesen Zeichen bestehen und achtet auf Kleinschreibung
- sämtliche Aufgaben, mit Ausnahme der theoretischen Aufgaben, die eine PDF erfordern, benötigen zwingend ein **Makefile**. Abgaben ohne dieses werden nicht gewertet

| | |
|---|---|
| Hinweis: | Manche Browser sind nicht dazu in der Lage die in die PDFs eingebetteten Dateien anzuzeigen. Mittels eines geeigneten Readers, wie dem Adobe Reader, lassen sich diese jedoch anzeigen und verwenden. |

| | |
|---|---|
| Hinweis: | Die Inhalte von Aufgabe 2.2 werden in der Vorlesung am 16. Januar 2024 besprochen. |

# Aufgabe 1 Nachrichtenübertragung mit UDP (4 Punkte)

In dieser Aufgabe soll in C ein System zur Übertragung von Nachrichten implementiert werden. Das System soll aus zwei Anwendungen bestehen, eine Anwendung zum Empfangen von Nachrichten (Empfänger) und eine weitere Anwendung zum Senden von Nachrichten (Sender). Die Kommunikation zwischen den beiden Anwendungen soll über UDP erfolgen.

Sie können Empfänger und Sender jeweils separat auch mit Hilfe der netcat-Experimente aus der vorigen Aufgabe testen.

## Empfänger

Dem Empfänger wird beim Start der Anwendung ein Port vom Betriebssystem zugewiesen. Dieser Port soll direkt nach dem Start auf der Standardausgabe (`stdout`) ausgegeben werden. Benutzen Sie dazu nach dem Aufruf von `bind` die Funktion `getsockname` (siehe Manpages zu `getsockname`). Erhält der Empfänger eine Nachricht, gibt er die IP-Adresse, den Hostnamen und den Port des Senders sowie die übertragene Nachricht aus.

## Sender

Dem Sender wird beim Start der Anwendung vom Benutzer mitgeteilt, an welchen Host und Port gesendet werden soll. Den eigenen Port bekommt er vom Betriebssystem zugewiesen. Das Programm soll die Möglichkeit bieten, wiederholt Textnachrichten von der Standardeingabe (`stdin`) entgegen zu nehmen und diese an den zu Beginn angegebenen Empfänger zu senden.

> Hinweis: Sie können Ihre Programme auch auf einem einzelnen Rechner testen, indem Sie als Hostnamen localhost (IP: 127.0.0.1) verwenden.

Ist es möglich, den korrekten Empfang der Nachrichten zu überprüfen, ohne einen weiteren Mechanismus zu implementieren? Begründen Sie Ihre Antwort in der Datei **protokoll.txt**, die ebenfalls Teil Ihrer Abgabe ist.

# Aufgabe 2 Nachrichtenübertragung mit UDP in einem Programm (6 = 2 + 4 Punkte)

## Aufgabe 2.1

Auf dem vorherigen Übungsblatt haben Sie zwei Programme geschrieben, mit denen Sie Textnachrichten via UDP austauschen können. Vereinigen Sie nun die Funktionalität des Sendens und Empfangens in einem Programm. Nach dem Start des Programms soll zunächst die lokale UDP-Portnummer vom Betriebssystem vergeben und anschließend auf der Standardausgabe ausgegeben werden. Nun kann der Benutzer den Hostnamen und die Portnummer des Kommunikationspartners eingeben. Das Programm soll daraufhin abwechselnd von der Standardeingabe lesen und vom Socket empfangen. Dabei soll per Kommandozeilenparameter bestimmt werden, ob das Programm zuerst von der Standardeingabe oder vom Socket liest.

## Aufgabe 2.2

Ihr UDP-Chatprogramm arbeitet zurzeit mit Blocking I/O. So ist nur ein abwechselndes Empfangen, Schreiben und Senden möglich. Ändern Sie Ihr Programm so ab, dass die Kommunikation mit non-blocking I/O realisiert wird. Das Programm soll in der Lage sein, während es in einer Schleife den Socket auf Daten abfragt, auch eine Eingabe von der Tastatur zu lesen und an die Gegenseite zu senden.

> Hinweis: Schreiben und lesen Sie von der Standardeingabe und Standardausgabe nicht mit den Funktionen für gepufferten I/O (z.B. `printf`/`scanf`, `puts`/`gets` etc.), sondern verwenden Sie die Funktionen `read` und `write`. Die Filedeskriptoren der Standardeingabe und Standardausgabe lauten `STDIN_FILENO` bzw. `STDOUT_FILENO`. Für ein leichtes zeilenweise Einlesen von der Standardeingabe können Sie die Funktion `readline` verwenden, die ähnlich wie `read` funktioniert, dabei aber immer ganze Zeilen (d.h. bis zum nächsten `\n`) liest. Diese Funktion finden Sie eingebettet in die PDF-Datei (**unp_readline.h** und **unp_readline.c**).

Analysieren Sie ihr laufendes Programm mit dem Unix-Kommando `top`. Was fällt auf? Mit welchen einfachen Mitteln können Sie die Situation erheblich verbessern? Die Beantwortung dieser Fragen soll in einer **README.txt** Datei erfolgen.

## Aufgabe 3 Dateien in C (2 Punkte)

In Linux haben Sie grundsätzlich zwei unterschiedliche Möglichkeiten, auf Dateien zuzugreifen und sie aus C heraus zu öffnen, zu lesen und zu schreiben. Einerseits gibt es die POSIX-kompatible API mit Funktionen wie `open`, `read`, `write` und `close`, andererseits gibt es die C-Standardbibliotheksfunktionen mit `fopen`, `fread`, `fwrite` und `fclose`.

### a)

Erklären Sie die Unterschiede zwischen den Funktionen. In welchen Situationen würden Sie eine der beiden bevorzugen? Was für Vor- und Nachteile haben sie jeweils?

### b)

Schauen Sie sich Ihre Abgabe der Aufgabe *Pipes in C* von einem vorherigen Zettel an. Welche der beiden APIs haben Sie hier benutzt? Implementieren Sie die Dateioperationen mit der jeweils anderen Version neu.