# Aufgabe 9: IPC und Threads – Client

Ein Gruppenkommunikationsdienst ist üblicherweise eine verteilte Anwendung bestehend aus mehreren Clients, die über einen Server miteinander kommunizieren. Je nach Implementierung kann dabei der Server verschiedene Garantien realisieren. Die Hauptaufgabe des Gruppenkommunikationsdienstes `MyGroup` ist es dabei, zu garantieren, dass angemeldete Clients Nachrichten in der gleichen Reihenfolge empfangen. Eine nähere Beschreibung des Aufbaus von `MyGroup` finden Sie in den Folien der großen Übung. **Schauen Sie sich daher bitte diese Folien nochmal genau an, bevor Sie mit der Bearbeitung der Aufgaben beginnen.** Ebenfalls hilfreich sind die Folien der Vorlesung zum Thema Interprozesskommunikation.

Das Ziel dieser Aufgabe ist die Implementierung des `MyGroup-clients`. Zum Testen Ihrer Implementierung ist eine fertig kompilierte Version des `myGroup-servers` im Material zur Übung enthalten. Sie können den Server mittels `./myGroup-server` starten.

Für die Lösung der folgenden Aufgaben werden Hilfsfunktionen benötigt, die Sie im Modul `utils` finden. Hilfreich sind zudem folgende Manpages: **socket(2)**, **connect(3)**, **recv(3)**, und **send(3)**, sowie **pthread_create(3)** und **pthread_mutex(3)**.

**Testen:** Zum Testen Ihrer Implementierung können Sie das mitgelieferte Skript `run_tests.bash` ohne Parameter aufrufen. Dieses Skript startet für Sie Ihren Client `./build/myGroup-client` sowie den mitgelieferten Server `./myGroup-server` und wertet die Ausgaben der beiden Programme aus. Es gibt vier Tests A, B, C und D, welche die Aufgaben 9.1, 9.2, 9.4 und 9.5 testen. Die Bewertung der Aufgaben 9.3, 9.6 und 9.7 erfolgt manuell. Die GitLab CI Tests für diese Aufgabe laufen ebenso ab.

## 9.1 Aufbau einer Socketverbindung (3P)

Implementieren Sie in der Funktion `connect_to_server()` eine stromorientierte Socketverbindung zum Server. Benutzen Sie zum Speichern des *Socket File Descriptors* die globale Variable `client_socket`. Um den Server zu finden, verwenden Sie die IP-Adresse für die lokale Maschine `127.0.0.1` und benutzen Sie den vorgegebenen Port (Variable `constants.SERVER_PORT`). Hilfreiche Manpages sind **inet_addr(3)** und **htons(3)**.

## 9.2 Handshake (3P)

Implementieren Sie in der Funktion `handshake()` einen initialen Nachrichtenaustausch, bei dem der Client an den Server die Nachricht `Hello, I'm the client` sendet und anschließend auf eine Antwort vom Server wartet. Die Antwort vom Server soll über die Standardausgabe ausgegeben werden.

Nutzen Sie für den Nachrichtenaustausch mit dem Server die Funktionen **send(3)** und **recv(3)** ohne Flags. Beachten Sie, dass Nachrichten eine maximale Länge von `MAX_MESSAGE_LENGTH` haben. Dies gilt auch für den Nachrichtenaustausch in Aufgabe 9.4 und 9.5.

## 9.3 Benutzerschnittstelle (2P)

Implementieren Sie die Funktion `send_message()`. Diese soll sich zuerst mittels `prompt_user_input()` eine Eingabezeile vom Benutzer holen, welche die Länge von `MAX_USER_INPUT` nicht überschreiten darf. Abhängig von der Benutzereingabe soll dann entweder `set_request()` oder `get_request()` ausgeführt werden. Orientieren Sie sich hierbei an der Beschreibung des `MyGroup-clients`, die in der großen Übung vorgestellt wurde. Wiederholen Sie danach den Ablauf, ein Verlassen der Funktion soll also nicht stattfinden.

## 9.4 SET-Anfrage (4p)

Implementieren Sie nun das Senden von `SET`-Anfragen in der Funktion `set_request()`. Diese Funktion bekommt die Eingabe des Benutzers übergeben und verschickt eine Nachricht in der Form `s:<Eingabe des Benutzers>` an den Server. Anschließend soll diese Funktion auf eine Antwort des Servers warten und entsprechend der Vorgabe aus der großen Übung auf die Antwort vom Server reagieren. Geben Sie mit `prompt_error()` eine Fehlermeldung aus, falls Ihre Nachricht nicht auf dem Server gespeichert werden konnte. Sollte etwas anderes schief laufen und die Antwort vom Server beinhaltet weder `r:ack` noch `r:nack`, geben Sie die volle Nachricht auf der *Standardfehlerausgabe* aus und beenden Sie den Client über `exit_client()`.

## 9.5 GET-Anfrage (3P)

Implementieren Sie nun analog zu Aufgabe 9.4 die Funktion `get_request()`, die das Versenden von `GET`-Anfragen realisiert. Hierbei soll zuerst eine Nachricht in der Form `g:` an den Server verschickt und danach auf die Antwort gewartet werden. Orientieren Sie sich bei der Antwort vom Server an den Vorgaben aus der großen Übung und geben Sie Nachrichten über die Funktion `print_reply()` auf der Konsole aus.

## 9.6 GET-Anfragen im Hintergrund (3P)

Bisher mussten alle `GET`- und `SET`-Anfragen manuell vom Benutzer des Clients initialisiert werden. Nun soll ein dedizierter Thread parallel zur Benutzereingabe periodisch Get-Anfragen an den Server stellen und die Antwort gegebenenfalls ausgeben.

Erstellen Sie in der Funktion `start_reader_thread()` einen neuen Thread, der die Funktion `read_continously()` ausführt. Implementieren Sie anschließend in der Funktion `read_continously()` ein kontinuierliches Versenden von `GET`-Anfragen. Warten Sie dabei zwischen `GET`-Anfragen eine vorgegebene Zeit lang (s. `READING_INTERVAL`).

Das Versenden einer Anfrage und der Empfang einer Antwort vom Server stellen jeweils einen *kritischen Bereich* dar, weil wir von einem einzelnen Kommunikationskanal ausgehen. Synchronisieren Sie daher Ihren Client entsprechend, sodass Lese- und Schreibanfragen nicht gleichzeitig erfolgen.

## 9.7 Fehlerbehandlung (5P)

Die meisten Bibliotheks-Funktionen, die Sie in den vorherigen Teilaufgaben aufrufen, können fehlschlagen. Außerdem kann es beim Warten auf eine Antwort vom Server zum Verbindungsabbruch kommen. Fangen Sie alle Fehler und den Verbindungsabbruch ab und beenden Sie Ihre Anwendung gegebenenfalls mit `exit_client`, damit alle offenen Sockets geschlossen werden. Der Exit-Code sollte hierbei der Situation angemessen sein.

*Hinweis:* Diese Teilaufgabe können und sollten Sie parallel zu den vorherigen Aufgaben bearbeiten.

## Abgabe bis 18.01.2019, 23:59 Uhr.

**Einhaltung der Coding Guidelines: 3 Punkte**
**Zu erzielende Minimalpunktzahl: 13 Punkte**