Aufgabe 11: IPC und Threads – Server

Nachdem Sie in Aufgabe 09R den `MyGroup-client` gebaut haben, ist es das Ziel dieser Aufgabe, den `MyGroup-server` zu implementieren.

Für die Lösung der folgenden Aufgaben ist eine doppelt-verkettete Liste im Modul `reads_list` und ein Ring-Buffer im Modul `ring_buffer` vorgegeben. Anders als bei bisherigen Aufgaben dürfen und sollen diese vorgegebenen Module ergänzt werden. Änderungen an *main.c* sind hingegen **nicht** erlaubt.

Hilfreich für die Lösung sind zudem folgende Manpages: **socket(3)**, **bind(3)**, **connect(3)**, **listen(3)**, **accept(3)**, **htons(3)**, **recv(3)**, und **send(3)** sowie **pthread_create(3)**, **pthread_rwlock_wrlock(3)**, **sem_overview(7)** und **pthread_mutex_init(3)**. Beachten Sie, dass die meisten API-Funktionen fehlschlagen können. Fangen Sie daher alle Fehler ab und beenden Sie den Server mit einem der Situation angemessen Exit-Code. Wenn ein Client die Verbindung beendet, dann schließen Sie den zugehörigen Datei-Descriptor und beenden Sie den Thread, der mit dem Client kommuniziert.

Ein Nichtbeachten von Fehlern führt zu Punktabzug bei jeder Teilaufgabe! Zur Vereinfachung kann jedoch die Behandlung von Fehlern für alle `lock`, `unlock`, `post`, `wait`, `signal` und `broadcast` Funktionen ignoriert werden.

TIPP: Versuchen Sie frühzeitig, Ihre Lösung mit dem Thread-Sanitizer zu debuggen. Diesen können Sie aktivieren, indem Sie in der *CMakeLists.txt* die Variable `sanitizeThread` auf `ON` setzen. Beachten Sie, dass dadurch der Address-Sanitizer deaktiviert wird. Informationen zum Thread-Sanitizer finden Sie unter

`https://clang.llvm.org/docs/ThreadSanitizer.html`

und `https://github.com/google/sanitizers/wiki/ThreadSanitizerCppManual`.

Zum Testen Ihres Servers ist eine Referenzimplementierung des Clients `myGroup-client` auf allen Rechnern im CIP-Pool (Raum G40) installiert. Sie können diesen Client starten, indem Sie einfach in der Konsole *myGroup-reference-client* eingeben (ohne `./` vorweg). Ebenfalls auf den Rechnern vorinstalliert ist das Programm *myGroup-test-client*. Dieser Client simuliert verschiedene Situationen, mit denen Ihr Server zurechtkommen muss, und berechnet zugleich die vorläufig erzielten Punkte. Eine kurze Beschreibung, wie dieser Test-Client zu bedienen ist, bekommen Sie zu sehen, wenn Sie das Programm mit der Option `-h` starten. **Beachten Sie bitte, dass wir die Tests mit aktivierten Thread-Sanitizier im Server durchführen werden und jeder Fehler in der Synchronisation des Servers einen Punktabzug bringt**. Diese Tests werden ebenfalls in der GitLab CI ausgeführt, auch dort bekommen Sie eine vorläufige Angabe über die erzielten Punkte. Zusätzlich gibt es in der GitLab CI einen Benchmark, der Informationen zur Performance von Client- und Server-Kommunikation liefert; dies ist jedoch nicht relevant für die Bewertung.

11.1 Server initialisieren (2P)

Implementieren Sie in der Funktion `initialize_server()` einen Socket, auf dem der Server auf eingehende Clientverbindungen wartet. Benutzen Sie hierfür den vorgegebenen Port (=`SERVER_PORT`). Die maximale Anzahl der noch nicht akzeptierten Verbindungen ist in dem Makro `MAX_SOCKET_QUEUE` definiert.

11.2 Verbindungen annehmen (2P)

Implementieren Sie in der Funktion `accept_connections()` die Annahme von Client-Verbindungen. Dies ist die Kernaufgabe des Main- bzw. Server-Threads und soll daher kontinuierlich durchgeführt werden. Erstellen Sie für jede Client-Verbindung einen eigenen Client- bzw. Worker-Thread, der die Funktion `handle_connection()` ausführen soll. Achten Sie darauf, dass keine Data-Races entstehen, und wählen Sie einen geeigneten *Detachstate* für die Threads.

11.3 Synchronisation (18P)

Die Kommunikation mit den Clients unterscheidet sich nicht wesentlich von Aufgabe 09R und ist daher in der Funktion `handle_connection()` vorgegeben. In dieser Implementierung fehlt jedoch die komplette Synchronisation. Diese soll nun von Ihnen implementiert werden. Hierfür müssen zuerst die Strukturen `ring_buffer_element` und `reads_list_element` ergänzt sowie die Funktionen `reads_list_increment_all()`, `reads_list_decrement()` und `reads_list_get_reads()` implementiert werden. Widmen Sie sich danach der eigentlichen Synchronisierung des Ring-Buffers.

Als kleine Hilfestellung finden Sie Hinweise in den Kommentaren des vorgegebenen Codes. Da jedoch bei dieser Aufgabe verschiedene Lösungen möglich sind, kann es auch sinnvoll sein, Code-Stellen zu ergänzen, die nicht mit einem Hinweis versehen sind. Ein Verändern von bereits existierenden Code ist hingegen **nicht** notwendig, soweit dies nicht ausdrücklich durch Kommentare gekennzeichnet ist.

Tipps zum Lösen des Aufgabenteils 11.3

- Zum Lösen dieser Aufgabe ist es wichtig, dass Sie die Struktur des Servers verstehen und wissen, wie dieser zu synchronisieren ist. **Schauen Sie sich hierfür die Übungsfolien zum Thema IPC und Multithreading nochmal genau an.**
- Bevor Sie dann mit der Implementierung beginnen, sollten Sie sich mit dem vorgegeben Code vertraut machen. Schauen Sie sich die Implementierung genau an und werden Sie vertraut mit allen verwendeten Funktionen aus den Modulen (`reads_list` und `ring_buffer`).
- **Sollten Sie irgendwas nicht verstehen, fragen Sie die Übungsbetreuer.**
- Wenn Sie mit der Implementierung beginnen, konzentrieren Sie sich zuerst auf die Synchronisation des Ring-Buffers unter Verwendung eines einzelnen Clients. Beachten Sie, dass hierzu auch das korrekte An- und Abmelden des Clients vom Server gehört.
- Stellen Sie dann mit Hilfe des *Thread-Sanitizers* sicher, dass keine Wettlaufsituationen existieren.
- Wenn der Server mit einem Client fehlerfrei funktioniert, erweitern Sie Ihre Implementierung, sodass auch mehrere Clients gleichzeitig mit dem Server korrekt funktionieren. Achten Sie auch hier darauf, dass sich Clients jederzeit an- und abmelden können und neue Wettlaufsituationen möglich sind.

Abgabe bis 01.02.2019, 23:59 Uhr.

Einhaltung der Coding Guidelines: 2 Punkte
Zu erzielende Minimalpunktzahl: 12 Punkte