Übung 06: NIO und Networking

Abgabetermin: 16. 5. 2022, 8:30

Übung 6: FileShare mit Channels und asynchron (24 Points)

Bauen Sie die Applikation `FileShare` aus Übung 5 folgend um:

Verwendung von Channels

Statt mit `ServerSocket` und `Socket` soll die Netzwerkkommunikation nun mit `ServerSocketChannel` und `SocketChannel` realisiert werden.

Auch das Lesen und Schreiben auf Files soll mit `FileChannels` erfolgen.

Asynchrones Eventhandling am Server

Sowohl `accept` von `ServerSocket` als auch die Leseoperationen (Inputs vom Netzwerk) bei den `Sockets` sollen am Server asynchron mit Ereignissen abgearbeitet werden. Das heißt, sowohl der `ServerSocket` als auch die `Sockets` arbeiten *nicht-blockierend*. Es muss daher ein `Selector` mit entsprechender Select-Schleife verwendet werden.

Der Client soll weiterhin synchron (aber mit Channels) arbeiten.

Ausführen von Fileoperationen mit ThreadPool

Am Server sollen alle Fileoperationen, d.h. das Lesen und Senden der Directories, Lesen und Senden der Files in einem Directory und insbesondere der Download eines Files, als asynchrone Tasks ausgeführt werden. Das heißt, wenn der Server ein entsprechendes Kommando erhält, soll ein Task an einen ExecutorService geschickt werden, in dem die Dateioperationen ausgeführt und die Messages zum Client geschickt werden. Setzen Sie dazu einen geeigneten ExecutorService ein, z.B. einen *FixedThreadPool* (`Executors.newFixedThreadPool()`) oder einen *CachedThreadPool* `Executors.newCachedThreadPool()`).

Anleitungen und Hinweise

Bauen Sie auf dem Programm aus Übung 5 auf.

Zusätzlich finden Sie in einem Download auf Moodle eine erweiterte Klasse `Msg` mit Methoden `receiveMsg` und `sendMsg`, die mit `Channel`s arbeiten. Diese können Sie wieder für das Senden und Empfangen von Messages verwenden. Beachten Sie aber, dass das Empfangen von Nachrichten am Server in der Select-Loop durch ein Ereignis ausgelöst wird. Erst wenn ein Ereignis ausgelöst wurde, können die Daten mit `receiveMsg` ausgelesen werden.

Um die Aufgabe einfach zu halten, sollten Sie einen genügend großen `ByteBuffer` verwenden, in dem eine Message auch vollständig übertragen werden kann. Sie können also davon ausgehen, dass die Messages, z.B. die Files in einem Directory, in diesen `ByteBuffer` passt (ansonsten müssten Sie eine Message aus mehreren nacheinander empfangenen Pufferinhalten zusammensetzen).

Optionale Zusatzaufgabe: Upload (6 Zusatzpunkte)

Für die Implementierung eines Uploads vom Client gibt es wieder 6 Zusatzpunkte.

Die Schwierigkeit liegt dabei darin, dass die Daten zum Upload auch über die Select-Ereignisse empfangen werden. Hier müssen Sie einen besonderen Zustand vorsehen, der bestimmt, dass Sie nun Filedaten und keine Messages empfangen.