HOCHSCHULE
KONSTANZ
TECHNIK, WIRTSCHAFT
UND GESTALTUNG

Verteilte Systeme
Übung

25. Apr. 2018

Aufgabenblatt 2:
Sockets und Pooling

Abgabe bis: 30. Mai 2018

Prof. Dr. Rainer Mueller

SS 2018

Aufgabe 1: Socket-Spielerei (Wiederholung zur REKO-Auffrischung)

1

a. Übersetzen und probieren Sie das Client-Server-Socket-Basis-Programm aus der Vorlesung. Sie finden die Quellen in Moodle.

b. Bauen Sie das Socket-Programm wie folgt für eine Mehrfachkommunikation zw. Client und Server um:

- Der Client stellt nach dem Start solange automatisch Anfragen an den Server, bis dies vom Anwender durch eine entsprechende Text-Eingabe unterbrochen wird. Die Abstände zwischen Anfragen sind zufällig und jede einzelne Anfrage eindeutig auf Server-Seite identifizierbar. Dokumentieren Sie die Anfragen und Ihre Identifikation auf Client- und auf Server-Seite durch entsprechende Textausgaben:

```
Client: send Test-757

Server: received 'Test-757'
```

- Wenn mehrere Clients gleichzeitig mit dem Server kommunizieren, soll auf Server-Seite unterschieden werden können, von welchem Client die jeweilige gerade empfangene Anfrage kommt. Dokumentieren Sie auch dies durch entsprechende Textausgaben.

Aufgabe 2: Web-Server (Wiederholung zur REKO-Auffrischung)

2

a. Entwickeln Sie einen einfachen **socket-basierten Web-Server**, der eine Web-Seite an einen anfragenden Web-Browser schicken kann.

   Beispiel-Web-Seite:

   “HTTP/1.1 200 OK\r\n\r\n<html><body><h1>Hallo Web-Welt</h1></body></html>“

   Der Web-Server soll in einer Endlosschleife laufen und soll folgende Aufrufparameter besitzen:

   - -port <port>: Die Port-Nummer, unter der der Server angesprochen werden kann
   - -log <filename>: Der Dateiname, in dem der Server alle wesentlichen Aktionen mitloggt.

b. Erweitern Sie den Server so, dass er mit beliebig vielen HTML-Seiten in beliebigen Unterverzeichnissen umgehen kann. Der Web-Browser schickt also eine vollständige URL, aus der Sie Pfad und Dateiname der HTML extrahieren müssen, um nach der passenden HTML-Datei im Verzeichnis des Web-Servers zu suchen. Diese Datei wird dann an den Web-Client geschickt. Diese HTML-Seiten liegen als reale Dateien und nicht als HTML-Code im Server hart kodiert vor.

c. Erweitern Sie den Server so, dass er auch mit fehlerhaften URLs umgehen kann und dem Web-Browser die passende Antwort schickt.

Aufgabe 3: Nebenläufigkeitsgrade

3

Diese Aufgabe baut auf der Aufgabe 2 des Aufgabenblatts 1 auf. Es gibt dafür eine neue Version des Java-Archivs `rm.requestResponse.jar` und der Datei `rm.requestResponse.javadoc.zip`. Beachten Sie dabei die neue Methode `cleanUp` der Klasse `Component`.

a. Rufen Sie sich die Teilaufgabe e. der Aufgabe2 von Aufgabenblatt 1 in Erinnerung. Nennen Sie eine Möglichkeit, wie man das Verhalten bei mehreren nebenläufigen Clients verbessern kann, insb. dann, wenn alle Instanzen, der Server und alle Clients auf derselben Maschine ablaufen.

b. Erweitern Sie Ihre Lösung der Aufgabe 2 von Aufgabenblatt 1 so, dass beliebig viele Clients gleichzeitig mit dem nicht nebenläufigen Server fehlerfrei kommunizieren können. Das bedeutet am Beispiel zweier Clients A und B, dass die Anfragen von B nicht erst dann beantwortet werden, wenn alle Anfragen von A beantwortet wurden.
Verwenden Sie zum Testen beispielsweise 10 Clients mit ähnlichen Einstellungen, die gleichzeitig Anfragen an den Server stellen.

c. Erhöhen Sie die Nebenläufigkeit des Servers maximal, sodass die Wartezeiten der anfragenden Clients minimiert werden.
Wiederholen Sie den Test mit den 10 Clients und beobachten Sie das veränderte Verhalten.

d. Integrieren Sie einen nebenläufigen ThreadCounter in den Server, der immer die gerade aktuelle genutzte Threadanzahl im Server für den Anwender sichtbar in der Console ausgibt. Der Threadcounter soll dabei ohne Sleep-Zeiten auskommen.

Aufgabe 3: Nebenläufigkeitsgrade

3

e. Erweitern Sie den Server aus Teilaufgabe c. und d. so, dass zudem auch noch ein konstanter und ein dynamischer Thread-Pool für die Kontrolle des Nebenläufigkeitsgrads des Servers verwendet werden kann.

   Testen Sie erneut mit 10 Clients.

Aufgabe 4: Bonusaufgabe

4

Diese Aufgabe muss nicht abgegeben werden, um den Schein zur Veranstaltung zu bekommen. Geben Sie diese Aufgabe aber ab, so können Sie sich die Abgabe von insgesamt drei anderen Aufgaben auf diesem und dem nächsten Aufgabenblatt (Ausnahme: Aufgabe 3 auf Aufgabenblatt 2) sparen und bekommen den Schein trotzdem.

Implementieren Sie die Klassen `Component` und `Message` aus `rm.requestResponse.jar` so, dass die Kommunikation zwischen den Component-Instanzen ausschließlich mit Hilfe von Sockets erfolgt. Orientieren Sie sich zur Spezifikation der einzelnen Methoden ggf. am Javadoc zum Archiv.

Stellen Sie sicher, dass Ihre Implementierung von `Component` und `Message` mit Ihren abgegeben Lösungen der Aufgabe 2 auf Aufgabenblatt 1 und Aufgabe 3 auf diesem Aufgabenblatt korrekt funktioniert.

Hinweis: Eine Lösung aus einer Form von erkennbarem Reverse Engineering wird nicht akzeptiert.