Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Robert Kaiser

Verteilte Systeme
SS 2021
LV 4132

Übungsblatt 3
Praktische Übungen
Bearbeitungszeit: 2 Wochen
Abgabe: 17.05.2021, 04:00 Uhr MESZ

Aufgabe 3.1 (Projekt „Mehrbenutzerversion Hamsterasyl“):

Das zuvor entwickelte IT-System für das westhessische Hamsterverwahrungsunternehmen ist bei den Mitarbeitern sehr gut angekommen. Durch dieses wichtige Hilfsmittel konnten weitere Kunden gewonnen werden und das Unternehmen ist gewachsen. Inzwischen reicht ein Mitarbeiter und Arbeitsplatz nicht mehr aus, um die ganzen Anfragen der Kunden zu bearbeiten. Es besteht daher Bedarf an einer Version über die mehrere Mitarbeiter mit ihren PCs gleichzeitig auf das IT-System zugreifen können. Das existierende System soll aber nicht abgelöst, sondern aus Kostengründen nur mit Netzwerkfunktionalität erweitert werden.

Die IT-Verantwortliche möchte, dass das Frontend praktisch gleich bleiben kann und zusätzlich die Möglichkeit besteht mit moderneren Programmiersprachen neue Frontends zu entwickeln. Auf einer Fortbildung hat Sie gelernt, dass RPCs der letzte Schrei sind und man das unbedingt machen muss, um ein erfolgreiches und hippes Unternehmen zu sein.

Praktischerweise haben die Entwickler der Open-Source-Bibliothek „Hamsterlib“ ein passendes RPC-Protokoll für ihre Bibliothek spezifiziert. Die Implementierung des Protokolls wird allerdings kommerziell vertrieben, damit die Entwickler ihre Miete bezahlen können. Dafür ist das Hamsterverwahrungsunternehmen allerdings zu geizig. Es von Ihnen zu implementieren zu lassen ist günstiger, zumal beim Surfen die IT-Verantwortliche sogar einen fertigen Java-Client gefunden hat. Es wird also nur noch der Server benötigt!

In Ihrem SVN-Repository finden Sie den neuen Ordner „3“. In diesem ist im Verzeichnis `include` die Datei `hamsterprotocol.h` in der das RPC-Protokoll als Doxygen-Docu spezifiziert ist. Die bereits bekannte Hamsterlib liegt wieder als Source-Code im Verzeichnis `libsrc` vor.

Für die Erzeugung der Doxygen-Dokumentation rufen Sie im Projektverzeichnis `make` auf. Wie bisher wird diese dann im Verzeichnis `doc/html` als html-Dateien angelegt[1].

Im Verzeichnis `src` finden Sie eine Mustervorlage für ein einfaches Hauptprogramm für den RPC-Server. Erstellen Sie daraus ein C-Programm `hamster_server`, das mit einem Kommandoparameter `-p PORT` aufgerufen wird, um den Port für den Server festzulegen.

Der Server soll als Default nur lokal auf „localhost“ also der IP-Adresse `127.0.0.1` laufen. Dieses Vorgehen ist generell immer sinnvoll. Es verhindert, dass Dienste aus Versehen über das Netzwerk oder gar das Internet zugegriffen werden können. Mit dem optionalen Kommandoparameter `-h IP-ADRESSE` soll dann die IP-Adresse festgelegt werden auf der der Server seinen Dienst anbietet

[1] Wie bei Open-Source-Projekten üblich, ist die Dokumentation wieder in Englischer Sprache verfasst.

(optinal im Sinne von: wenn nicht angegeben, wird die Default-Adresse gewählt). Halten Sie sich für die Syntax wieder an die rtfm()-Funktion.

Ihre Aufgabe ist nun: Bauen Sie einen Server, der die Hamsterlib-Funktionen über das das Hamster-RPC-Protokoll bereitstellt. Das heißt, ein Client schickt eine Anfrage, um eine Funktion aufzurufen, Ihr Server empfängt und dekodiert diese, ruft dann die entsprechende Hamsterlib-Funktion auf, packt das Ergebnis in eine Antwort und schickt diese an den Client.

Der Server soll als minimale Anforderung als einfacher sequenzieller Server aufgebaut sein. Beachten Sie, dass der Client für jeden RPC-Aufruf eine neue Verbindung aufbauen kann und Ihr Server natürlich in der Lage sein muss mehrere Anfragen nacheinander abarbeiten zu können.

Tests

Zum Testen stehen Ihnen im Verzeichnis `tests` zwei Java-Clients zur Verfügung. Das Erste (`TestRunnerHamster.jar`) können Sie direkt mit `make test` bzw. `java -jar` starten. Dieses führt eine vollständige Test-Suite aus, die auch für die Bewertung der Aufgabe verwendet wird. Der zweite Client (`CLIClient.jar`) kann mit `java -jar` gestartet werden und bietet Ihnen eine einfache CLI zum inkrementellen Testen des Servers[2]. Beide Clients liegen im Ordner `HamsterRPC_Client` auch als Eclipse-Projekt vor, damit Sie im, Bedarfsfall zum Beispiel eigene Debug-Ausgaben hinzufügen können. *Es ist jedoch nicht Ihre Aufgabe, den Client zu verändern! Änderungen können Sie auch nicht im Subversion commiten.*

Des Weiteren gibt es unter `/HamsterRPC_Client/src/de/hsrm/cs/wwwvs/hamster/rpc/client/Client.java` einen Client-Stub, den Sie anpassen können, um gezielt Funktionen zu testen.

Für das Testen und die Bewertung werden die im Client-Projekt enthaltenden JUnit-Tests verwendet. Diese werden automatisch beim Start der `TestRunnerHamster.jar` ausgeführt.

Wenn möglich sollten Sie auch keine Änderungen an den Testfällen in `de.hsrm.cs.wwwvs.hamster.tests.suite` vornehmen. Sie können in diesem Projekt keine Änderungen committen (aka. „auf den Server hochladen“). Es kann gut sein, dass sich an den Testfällen noch Änderungen oder Verbesserungen ergeben, die dann von den Lehrbeauftragten in dem Projekt verfügbar gemacht werden. Bei einem `svn update` werden diese automatisch übernommen.

Die Testfälle bauen auf Ihrem Server und dem ursprünglichen `hamster` CLI Programm aus Aufgabe 1 auf. Eine fertige Musterlösung liegt im Ordner `tests`. Wenn Sie die Testfälle bei sich über Eclipse ausführen wollen, müssen Sie die Pfade ggf. entsprechend anpassen.

Dazu können Sie entweder (unschön) in der `HamsterTestDataStore.java` die Attribute `pathToHamsterExe` für den Pfad zum `hamster`-Programm und `pathToSUT` für den Pfad zu dem Binary Ihres Servers anpassen. Besser ist es eine RunConfiguration für den JUnit-Test (`HamsterTestSuite.java`) zu erstellen und dort die Pfade als „VM arguments“ zu übergeben [3]. Verwenden Sie dazu (Achtung Pfad in Anführungszeichen angeben!):

- `-DhamsterPath="PFAD_HAMSTER"` für den Pfad zum `hamster`-Programm
- `-DserverPath="PFAD_SERVER"` für den Pfad zu dem Binary Ihres Servers

[2] CLI Client erwartet Server an 127.0.0.1 auf Port 2323.

[3] Rechtsklick, auf `HamsterTestSuite.java`, „Run as“, „JUnit test“. Das geht schief, und dann oben neben dem grünen Dreieck (Auswahlbox), „Run Configurations“. Im neuen Fenster unter „JUunit“ die erstellte Run Configuration auswählen, dann im Reiter auf „Arguments“ und dann in der „VM arguments“ Textbox die Argumente hinzufügen

Sollten Sie den `HamsterTestRunner.jar` nicht mit `make test` aufrufen, können Sie ihm ebenfalls die Pfade als Parameter übergeben:

- `-P PATH_SERVER` für den Pfad zu dem Binary Ihres Servers
- `-H PATH_HAMSTER` für den Pfad zum `hamster`-Programm

Für den Fall, dass Sie Probleme mit einigen Tests bekommen, die darauf zurückzuführen sind, das Ihr Rechner zu langsam ist (bzw. die Tests zu schnell) ausgeführt werden, können Sie in der `HamsterTestDataStore.java` manuell die Attribute `sleepMin`, `sleepMed`, `sleepMax` anpassen und die „Schlafzeiten“ in ms hoch setzen.

Tipps zur Bearbeitung

Machen Sie sich mit dem zu implementierenden Protokoll vertraut. Versuchen Sie insbesondere die notwendigen Abläufe nachzuvollziehen. Machen Sie sich als erstes einen Plan was Sie tun müssen um die RPCs umzusetzen. Verwenden Sie ggf. Pseudocode um die Abläufe festzulegen.

Gehen Sie schrittweise vor! Testen Sie nach jeden Schritt Ihre Lösung. Beginnen Sie mit den Sockets, schauen Sie ob der Client sich verbinden kann. Empfangen Sie dann den Header der Nachricht und dekodieren Sie ihn, etc.

Es kann ggf. sinnvoll sein, die Funktionalität in mehrere C-Module aufzuspalten. Legen Sie falls notwendig entsprechende `.h` und `.c` Dateien in den `include` bzw. `src` Verzeichnissen an. Vergessen Sie dann nicht, dass neue Dateien mit `svn add DATEINAME` dem SVN-Repository hinzugefügt werden müssen!!

Hinweise:

- Falsche oder fehlende Benutzereingaben müssen abgefangen und mit einer Fehlermeldung zurückgewiesen werden. Ihr Programm sollte in solchen Fällen eine kurze Bedienungsanleitung („RTFM-Text“) ausgeben.
- Falsche Parameter (z.B. zu lange Besitzer- oder Hamsternamen) werden von der Bibliothek zurückgewiesen. Der Server soll dann entsprechende Error-Nachrichten erzeugen und sie dem Client schicken.
- Die Kodierung von Ganzzahlendatentypen erfolgt in der Netzwerkdatendarstellung (Big-Endian).

Weiter befindet sich zum Debuggen im Ordner `tests` die Datei `hamster.lua`. Hierbei handelt sich um ein Plugin für das Tool Wireshark[2]. Dieses Plugin erweitert Wireshark, um die Fähigkeit das Hamster Protokoll zu verarbeiten und auf mögliche Fehler hinzuweisen. Eine Anleitung um Wireshark in Ubuntu/Debian zu installieren finden Sie unter [3]. Weiterhin ist es wichtig, dass Ihr Benutzer sich in der Gruppe `wireshark` befindet (`addgroup $USER wireshark`)! Beachten Sie das die Änderungen der Gruppenzugehörigkeiten erst beim erneuten Einloggen übernommen werden. Um das Plugin verwenden zu können, muss dies in den versteckten Ordner `~/.local/lib/wireshark/plugins` kopiert werden. Möglicherweise müssen die entsprechenden Ordner vorher erzeugt werden! Um Debuggen zu können, müssen Sie den Traffic auf dem Loopback `lo` device mitschneiden. Über den Anzeigefilter `hamster` können Sie den Paketmitschnitt nach dem Hamster Protokoll filtrieren[4]. Weiter unterstützt das Plugin das Filtern von Paketen

[4]Server muss auf Port: 2323, 9000, 9001, 9002 oder 9003 laufen!

nach bestimmten Inhalten. Beispielsweise kann mit dem Filter `hamster.rpcCallID == 2 && hamster.flags == 0` nach allen Lookup Requests filtert werden. Nur im C001 ist Wireshark auf den Pool PCs installiert.

[1] http://www.doxygen.org

[2] https://www.wireshark.org/

[3] https://wiki.ubuntuusers.de/Wireshark/