Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Georg Hinkel

Verteilte Systeme

SS 2024
LV 4132

Übungsblatt 3
Praktische Übungen
Bearbeitungszeit: 2 Wochen
Abgabe: 19.05.2024, 23:59 Uhr MESZ

Aufgabe 3.1 (Projekt „Mehrbenutzerversion Hamsterasyl“, 20 Punkte):

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 neu implementieren zu lassen ist günstiger, zumal die IT-Verantwortliche beim Surfen sogar einen fertigen Java-Client gefunden hat. Es wird also nur noch der Server benötigt!

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 (optional 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 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.

Update des Repositories

Wie im ersten praktischen Übungsblatt angeregt, sollten Sie für jedes praktische Übungsblatt einen eigenen Branch anlegen. Damit Sie diesen Branch nicht auf Grundlage Ihrer Lösung des ersten Praktikumsblatts anlegen[1], sollten Sie zunächst in den (beinahe leeren) `main`-Branch wechseln und von da aus einen neuen Branch erzeugen:

```
$ git checkout main
$ git checkout -b 3-hamsterrpc
```

Anschließend laden Sie sich bitte zunächst wieder die Materialien dieses Übungsblatts in den lokalen Branch:

```
$ git pull --rebase vs 3-hamsterrpc/java
```

Ersetzen Sie hierbei wieder `java` durch `c` oder `csharp`, wenn Sie mit C oder C# arbeiten möchten.

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. 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!*

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. 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 `git pull --rebase` 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 in der `HamsterTestDataStore.java` die Attribute `pathToHamsterExe` für den Pfad zum `hamster`-Programm und `pathToSUT` für den Pfad zu dem Binary Ihres Servers anpassen. In der letzten Version der Vorlagen sind die Pfade so eingestellt, dass Sie die Tests auch manuell bequem aus einer Entwicklungsumgebung wie IntelliJ heraus starten können.

Wenn Sie die Tests manuell starten, sollten Sie auch darauf achten, dass Sie die Pfade zu dem Hamster Menüprogramm, dem Dateinamen der Daten-Datei sowie den Pfad zu Ihrem Hamster Server anpassen. Wenn Sie mit Java oder mit C# auf Linux arbeiten, sollte der Pfad die komplette notwendige Kommandozeile enthalten (also mit „`java -jar \` oder „`dotnet ` beginnen).

[1] Die Lösungen bauen nicht aufeinander auf, um Ihnen zu ermöglichen, Lösungen für spätere Übungsblätter zu bearbeiten, selbst wenn Sie die ersteren Übungsblätter nicht bewältigt haben sollten.

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

Am einfachsten orientieren Sie sich an der Konfiguration für die CI-Funktionalität des GitLab-Servers (in der Datei .gitlab-ci.yml), darin ist in jedem Übungsblatt ein jeweils korrekter Aufruf der Testsuite enthalten, den GitLab auch tätigen wird, um Ihre Lösung zu prüfen.

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 Module aufzuspalten. Legen Sie falls notwendig entsprechende Dateien an. Vergessen Sie dann nicht, dass neue Dateien mit `git add` dem Repository hinzugefügt werden müssen!!

Allgemeine Hinweise:

- Sie finden eine Dokumentation für das Hamsterprotokoll in den Vorlagen. Für C lässt sich die Dokumentation wieder aus dem Sourcecode mit Doxygen bauen, bei den Vorlagen für Java und C# sind die generierten HTML-Dateien im Verzeichnis `doc` mit in der Vorlage enthalten.

- Schauen Sie sich immer an, wie die Request-Nachrichten und Response- Nachrichten für jeden Request auszusehen haben, welche Informationen da in welcher Reihenfolge vorkommen müssen und wieviele Bytes dafür veranschlagt werden.

- Falsche oder fehlende Benutzereingaben müssen abgefangen und mit einer Fehlermeldung zurückgewiesen werden. Ihr Programm sollte in solchen Fällen eine kurze Bedienungsanleitung („RTFM-Text“) ausgeben. Die Vorlagen beinhalten aber bereits eine Implementierung einer passenden Kommandozeilenschnittstelle, am besten verändern Sie diese nicht.

- Falsche Parameter (z.B. zu lange Besitzer- oder Hamsternamen) werden von der Bibliothek zurückgewiesen. Das Hamsterprotokoll sieht für Fehlerfälle aber fest definierte Fehlercodes vor. In C können Sie die Fehlercodes der Bibliothek direkt in die Antwortnachricht packen, in Java und C# müssen Sie die Exceptions fangen und jeweils passende Fehlercodes zurückliefern.

- Die Kodierung von Ganzzahlendatentypen erfolgt in der üblichen Netzwerkdatendarstellung (Big Endian). Sowohl Java, C, als auch C# verwalten Ganzzahlen aber auf der x86 oder x64 Architektur in Little Endian. Auf dieser Architektur wird Gitlab auch Ihren Code testen. Sie müssen also konvertieren. Eine Übersicht der Konvertierungsfunktionen finden Sie in den programmiersprachenspezifischen Hinweisen.

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[3]. Weiter unterstützt das Plugin das Filtern von Paketen 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.

Hinweise C

In Ihrem Git-Repository finden Sie im Verzeichnis `include` die Datei `hamsterprotocol.h` in der das RPC-Protokoll als Doxygen-Docu spezifiziert ist. Die bereits bekannte Hamsterlib liegt wieder (bzw. immer noch) 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[4].

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.

Für die Konvertierung zwischen Little Endian und Big Endian können Sie die Funktionen `htons`, `htonl`, `ntohs` und `ntohl` verwenden. Die Funktionsnamen erklären sich hierbei, dass die Funktionen entweder *s*hort oder *l*ong zwischen *h*ost und *n*etwork hin und herkonvertieren.

Hinweise Java

Java hat zwei verschiedene Socket-Klassen, eine allgemeine Klasse und eine spezifische `ServerSocket`-Klasse. Um die Methode `accept` aufrufen zu können, benötigen Sie eine Instanz der Klasse `ServerSocket`, die Ihnen dann mit dem Aufruf von `accept` eine Instanz von `Socket` zurückliefert, die den Kanal zu Ihrem Client repräsentiert.

Für die Konvertierung zwischen Big Endian und Little Endian greifen Sie am besten auf die Klasse `java.nio.ByteBuffer` zurück.

Hinweise C#

Anders als Java verwendet .NET nur eine einzige Klasse `Socket`, aber sehr ähnlich liefert Ihnen `Accept` eine neue `Socket`-Instanz, die die Verbindung zu Ihrem Client repräsentiert.

In .NET finden Sie für jede Methode von `Socket` eine synchrone und eine asynchrone Variante vor. Für die Bearbeitung des Übungsblatts reicht es, wenn Sie die synchronen Versionen aufrufen,

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

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

in der Praxis sind allerdings mittlerweile die asynchronen Varianten verbreiteter, da sie keinen Thread blockieren.

Für die Konvertierung zwischen Big Endian und Little Endian eignet sich in .NET am besten die Klasse `BinaryPrimitives`. Diese kann die „Endianness“ wechseln oder eine Zahl in Big Endian in ein Byte-Array schreiben oder daraus lesen.

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

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

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