Hochschule RheinMain
DCSM - Informatik
Prof. Dr. Reinhold Kröger

Verteilte Systeme
WS 2015/16

Übungsblatt 3
Praktische Übungen

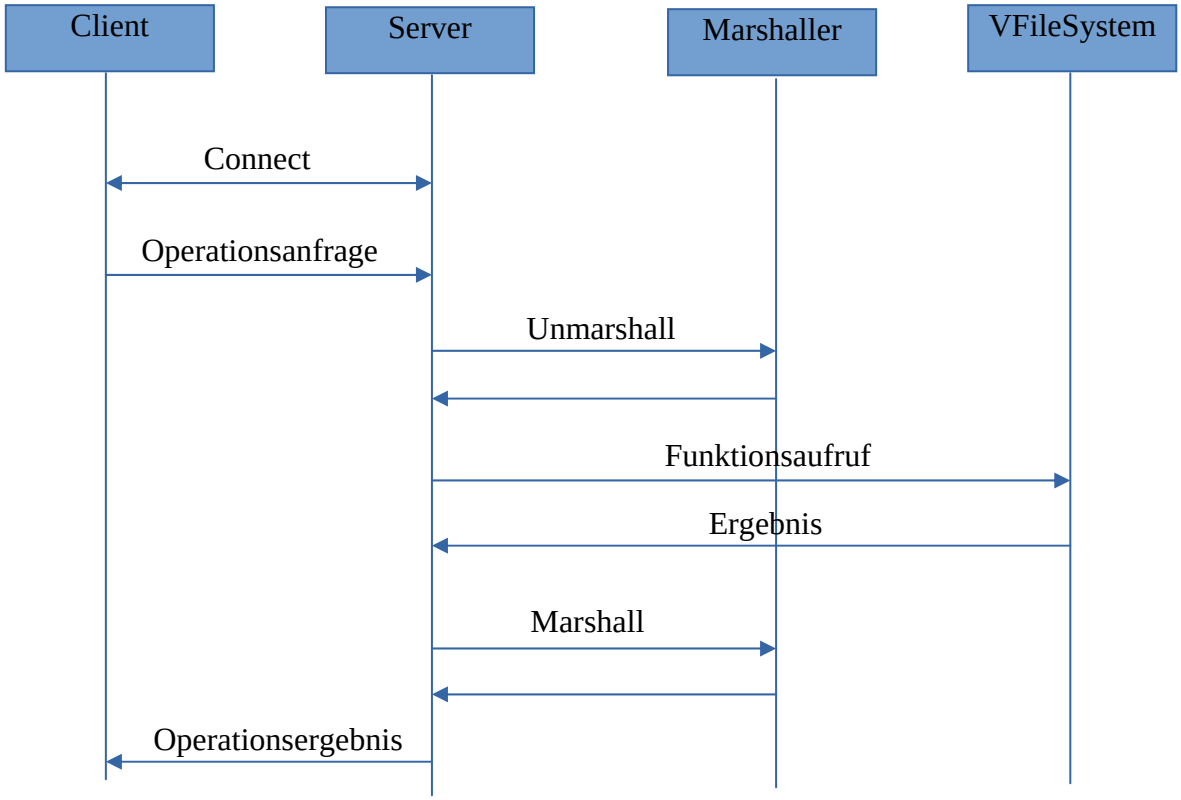
Grundlage dieser Übung ist das für das Aufgabenblatt 1 entwickelte “VFileSystem”. Ziel dieser Aufgabe ist es, das bisher nur lokal nutzbare File System über eine Socket-Schnittstelle zu exportieren, sodass auch entfernte Zugriffe von Clients darauf möglich werden.

Neben den Manual-Seiten zu den in der Vorlesung angegebenen Socket-Funktionen können Sie auch die folgenden Quellen oder andere benutzen:

- Stevens, W. R.: "UNIX Network Programming", Prentice-Hall, 1990 (auch in Deutsch, auch nachfolgende Ausgaben).
- Beej's Guide to Network Programming, Web-Dokument, http://beej.us/guide/bgnet/.
- SUN: "Network Programming Guide", Kap. 10 und 11 (A Socket-Based Interprocess Communications Tutorial).
- eLearning-Kurs zur Socket-Programmierung des Labors für Verteilte Systeme (siehe http://wwwvs.cs.hs-rm.de/lehre/index.html)

Aufgabe 3.1 (Projekt “VFileSystemServer”):

Erweitern Sie die Datei "main.c" des “VFileSystemServer”-Projekts, sodass über einen Socket eine eingehende Verbindung eines Clients angenommen werden kann. Nach Verbindungsaufbau soll eine synchrone Kommunikation zwischen Client und Server stattfinden, bei der der Client VFileSystem-Operationen an den Server schickt, der Server die Operation interpretiert, auf seinem lokalen VFileSystem ausführt und anschließend das Ergebnis an den Client zurückschickt. Die Serialisierung und Deserialisierung der dabei ausgetauschten Nachrichten soll in den Methoden "marshall" bzw. "unmarshall" des "vfilesystem_server_marshaller" umgesetzt werden.



Das Nachrichtenformat sieht dabei wie folgt aus:

uint32_t	uint8_t	uint8_t[]
Nachrichtenlänge	Nachrichtentyp	Payload

Die Nachrichtenlänge enthält die Größe des ihr folgenden Datenpakets in Bytes. Sie setzt sich aus einem Byte für den Nachrichtentyp und der Payload-Größe in Bytes zusammen. Die Größe des Payloads variiert je nach Nachrichtentyp. Die unterschiedlichen Nachrichtentypen sind in "vfilesystem_server_messages.h" (NEW_FILE_REQUEST, …) definiert. Zu jedem Nachrichtentyp gehört eine entsprechende Payload-Struktur (NewFileRequest, …).

Bei der Serialisierung bzw. Deserialisierung der Payload-Struktur müssen darin enthaltene Referenzen aufgelöst werden. Das folgende Beispiel veranschaulicht die Serialisierung anhand einer File-Info-Operation:

Der Client schickt die folgende Nachricht an den Server:

Nachrichtenlänge	Nachrichtentyp	handle
uint32_t	uint8_t	uint32_t
5	7	10

Darin steht in den ersten vier Bytes, dass der Rest der Nachricht 5 Byte groß ist. Im nächsten Byte steht die Information, dass es sich um den Nachrichtentyp 7, also einen "FILE_INFO_REQUEST", handelt. Dementsprechend handelt es sich beim Payload um eine Serialisierung der Struktur "FileInfoRequest", die nur ein einziges Feld für das angefragte File Handle beinhaltet.

Der Server interpretiert die Nachricht, führt die Funktionen fs_get_file_parent, fs_get_file_size, fs_get_file_name_length und fs_get_file_name mit den entsprechenden Parametern auf seinem lokalen VFileSystem aus und antwortet mit der folgenden Nachricht, die das Ergebnis der Funktionsaufrufe beinhaltet:

Nach-richten-länge	Nach-richten-typ	parent	size	name_ length	name[0]	name[1]	name[2]	name[3]
uint32_t	uint8_t	uint32_t	uint32_t	uint8_t	uint8_t	uint8_t	uint8_t	uint8_t
14	8	8	64	4	't'	'e'	's'	't'

Darin steht in den ersten vier Bytes, dass der Rest der Nachricht 14 Bytes groß ist. Im nächsten Byte steht die Information, dass es sich um den Nachrichtentyp 8, also eine "FILE_INFO_RESPONSE", handelt. Dementsprechend handelt es sich beim Payload um eine Serialisierung der Struktur "FileInfoResponse", die aus der FolderID des Parents, der Dateigröße, der Länge des Dateinamens und dem Dateinamen besteht. Beachten Sie, dass die Antwort-Nachricht kein terminierendes Null-Byte für den Namen enthält.

Für den Fall eines Fehlers (z.B. die Datei existiert nicht) soll der Server mit einer Nachricht des Typs "ERROR_RESPONSE" und entsprechendem "ErrorResponse"-Payload antworten.

Achten Sie darauf, dass die Daten in Network Byte Order und Strings als US-ASCII kodiert werden!

Aufgabe 3.2 (Test Ihres “VFileSystemServer”):

In Ihrem Projekt finden Sie eine Java-basierte Kommandozeilenanwendung "Client.jar", mit der Sie ihre Implementierung testen können.

Aufgabe 3.3 (Multiklientenfähigkeit des “VFileSystemServer”):

Erweitern Sie ihren Server so, dass er in der Lage ist, mehrere Clients gleichzeitig zu bedienen. Dazu soll der Hauptprozess für jede angenommene Verbindung einen Posix-Thread erzeugen, der die Verarbeitung der Anfragen des Clients übernimmt, bevor er wieder auf neue Verbindungen wartet (vgl. Folien Kap. 2.7: Multithreaded Server).