NVS PS 2016: Programmieraufgabe

Thomas Samy Dafir, Dominik Baumgartner, Vivien Wallner

1 Aufgabe

Erstellen zweier Programme, von denen eines als Sender, das andere als Empfänger agiert. Der Sender soll mittels UDP Pakete an den Empfänger senden. Dabei sollen die ersten vier Byte jeder Nachricht eine fortlaufende Nummer enthalten.

2 Dokumentation (Java-Version)

Dokumentation: siehe auch kommentierten Sourcecode

2.1 Funktion allgemein

Der Sender erstellt einen Datagram-Socket, dessen Adresse nicht bekannt sein muss, da der Sender nur sendet und keine Antwort erhält (es werden also keine Nachrichten an den Sender gesendet). Der Sendevorgang geht folgendermaßen vonstatten: Die Nachricht liegt als String vor. Sie wird in ein Byte-Array konvertiert, indem der ASCII-Wert jedes Zeichens des String als Byte repräsentiert wird. Aus diesem Array und der Empfänger-Info (IP, Port) wird dann ein UDP-Paket erstellt. Dieses wird dann dem Socket übergeben und versendet. Der Empfänger erstellt einen Socket, um UDP-Pakete empfangen zu können. Die Port-Nummer des Socket muss hier jedoch bekannt sein, da der Sender Pakete an die Kombination aus IP-Adresse (in unserem Fall 127.0.0.1) und Port-Nummer sendet. Da die Nachricht, die ein UDP-Paket enthält eine Abfolge von Bytes ist, muss ein Buffer-Byte-Array erstellt werden, um empfangene Daten zu speichern. Empfangsvorgang: Ein UDP-Paket kommt am Socket an und wird mittels *receive* gespeichert. Die Nachricht kann dann mittels *getData* extrahiert werden. Ergebnis ist ein Array von Byte Werten. Dieses muss nun interpretiert werden, da eine Byte-Folge je nach Kodierung verschiedene Bedeutungen haben kann. Da wir wissen, dass

es sich um eine Nachricht handelt, die ursprünglich als String von ASCII-Zeichen vorlag, können wir nun aus diesen Bytes einen String bauen und erhalten damit die gesendete Nachricht im lesbaren Format.

2.2 Empfänger

Generell müssen im Empfänger folgende Schritte ausgeführt werden:

1. Erstelle Datagram-Socket.
2. Erstelle Byte-Array als Buffer.
3. Erstelle Datagram-Paket Variable unter Verwendung des Buffers.
4. Empfange über den Socket Pakete.

2.2.1 Socket erstellen

Ein Datagram Socket wird in Java durch ein Objekt des Type *DatagramSocket* repräsentiert. Dieses Objekt wird unter Angabe einer Port-Nummer erstellt. Im Falle des Empfängers muss diese bekannt sein, da der Sender Pakete an diesen Port sendet.

2.2.2 Byte-Array und Datagram-Paket erstellen

Nachrichten, die als UDP-Pakete versendet werden, werden durch eine Abfolge von Bytes (einem Byte-Array) repräsentiert. Jedes Paket enthält also ein Byte-Array mit der Nachricht in Bytes. Dieses Array wird jetzt verwendet, um eine Paket-Variable zu definieren, der jedes empfangene Paket zugewiesen wird.

2.2.3 Pakete empfangen und zählen

Um eine unbekannte Anzahl an Paketen zu empfangen, wird *socket.receive*() in einer Endlosschleife verwendet. *receive*() ist eine "blockingMethode, die die Ausführung des aktuellen Thread so lange blockiert, bis ein Paket empfangen wird. Um diese Endlosschleife wieder verlassen zu können, wird ein Timeout für den Socket erstellt.
Dieser veranlasst den Socket, eine *SocketTimeoutException* zu werfen, sobald die festgelegte Zeit verstrichen ist, ohne dass ein Paket empfangen wurde. Diese Exception wird in einem *catch* nach der Schleife aufgefangen und die Anzahl an empfangenen Paketen ausgegeben. Danach wird an das Ende der Schleife zurückgesprungen und nur noch der Socket geschlossen.

2.2.4 Empfangenen Nachrichten speichern oder ausgeben

Der Sender nimmt während des Sendevorgangs keine Rücksicht auf den Empfänger d.h. der Sender sendet alle zu sendenden Pakete so schnell wie möglich, ohne die Zeit zu berücksichtigen, die der Empfänger eventuell braucht, um diese zu verarbeiten. Genau hier entsteht ein Problem, wenn man die empfangenen Pakete speichern oder die Nachrichten ausgeben will. In unserem Fall werden nach dem Empfang eines Pakets alle verbleibenden Anweisungen in der while-Schleife ausgeführt. Während diese ausgeführt werden, kann natürlich kein weiteres Paket empfangen werden. Während der Ausführung dieser Operationen sendet der Sender aber weiter, benötigen diese jetzt also zu viel Zeit, können die währenddessen gesendeten Pakete nicht empfangen werden und somit werden einige Pakete übersprungen und nicht empfangen. Dies führt jetzt zu dem Problem, dass eine direkte Ausgabe der enthaltenen Nachrichten nicht möglich ist , da die Ausgabe mit *print* auf cmd zu langsam ist (aufgrund des Terminals). Um hier kein Problem beim Empfangen zu verursachen, schreibt unser Empfänger die erhaltenen Nachrichten in die Datei *receive.txt*. Datei-Schreibe-Operationen sind nicht vom langsamen Terminal beeinflusst, somit sehr schnell und führen nicht zu einer großen Verzögerung, sodass alle Pakete ohne Verlust empfangen werden und der Inhalt gespeichert wird.

2.3 Sender

Um UDP-Pakete zu senden, müssen folgende Schritte ausgeführt werden.

1. Erstelle Datagram-Socket.
2. Erstelle Byte-Array als Buffer.
3. Erstelle ein InetAddress Object, das die Empfänger-IP repräsentiert.
4. Erstelle zu sendende Nachricht.
5. Erstelle und sende Pakete.

2.3.1 Socket erstellen

Anders als im Empfänger wird der Socket hier unter Verwendung des leeren Standard-Konstruktors erstellt. Damit wird der Socket keiner bestimmten, sondern dem erst-freien Port zugewiesen. Die Port-Nummer ist damit nicht bekannt, wird jedoch nicht gebraucht, da vom Empfänger keine Antwort erwartet wird.

2.3.2 Byte-Array erstellen

Auch hier wird ein Byte-Array erstellt, das später die zu sendende Nachricht enthalten und zur Erstellung der Datagram-Pakete benötigt wird.

2.3.3 InetAdress Objekt erstellen

Um ein Paket an den Empfänger zu senden, wird natürlich dessen IP-Adresse benötigt. Diese liegt hier als Hostname oder IP-Adresse im String-Format vor. Dieser String wir nun mittels $getByName()$ zu einer $InetAddress$ konvertiert, um später für das Erstellen der Pakets verwendet werden zu können.

2.3.4 Nachricht erstellen

Eine Nachricht wird unter Verwendung des aktuellen Schleifen-Index erstellt. Dieser wird mittels $String.format$ so formatiert, dass der Index immer mit führenden Nullen so aufgefüllt wird, dass ein 4 Symbole (4 Byte) langer String entsteht. An diesen String werden zusätzlich noch einige Symbole als Nachricht angefügt.

2.3.5 Pakete erstellen und senden

Nun muss die Nachricht in ein Paket gepackt werden. Dazu wird zuerst der String in ein Byte-Array konvertiert und danach mit diesem Array ein Paket erstellt. Anders als beim Empfänger müssen hier zusätzlich Empfänger-Adresse und -Port angegeben werden, da das Paket hier nicht nur als Empfänger dient. Weiters werden beim Senden keine Adresse und Port angegeben, sämtliche Information ist im Paket gespeichert. Das erstellte Paket kann nun unter der Verwendung von $socket.send()$ verschickt werden. Dieser Vorgang wird wiederholt, bis der in Schritte aufgeteilte Sendevorgang beendet ist und keine Pakete mehr gesendet werden müssen.

3 Dokumentation (C-Version)

Dokumentation: siehe auch kommentierten Sourcecode

3.1 Funktion allgemein

Aufgrund der in C nicht vorhandenen Objektorientierung gestaltet sich die Implementierung komplizierter als in Java. Anstatt praktischer vorhandener Klassen werden hier in Libraries vordefinierte Struct-Konstrukte in Zusammenhang mit Library-definierten Funktionen verwendet. Im Sender müssen

zuerst ein Socket und ein Adressen-Konstrukt, welches die Empfängerdaten enthält erstellt werden. Außerdem muss noch ein Character-Array erstellt werden, das die zu sendenden Nachrichten enthalten wird. Danach werden Nachrichten mit fortlaufender Nummer erstellt und im Buffer gespeichert un dann unter Angabe des Adressen-Konstrukts und des Sockets versendet zu werden. Der Empfänger erstellt ebenfalls einen Socket und ein Adressen-Konstrukt, das unter anderem den die Port-Nummer enthält, auf der der Socket erstellt werden soll. Adresse und Socket sind hier 2 separate Objekte. Unter Verwendung von *bind* werden dem Socket die im Adressen-Objekt gespeicherten Eigenschaften wie Port-Nummer zugewiesen. Danach können unter Verwendung von $revcfrom$ in einer Schleife viele Pakete empfangen werden. Um die "blockingFunktion $recvfrom$ wieder verlassen zu können, wird für den Socket ein Timeout gesetzt. Werden für die Dauer des Timeout keine Objekte empfangen, gibt $recvfrom$ einen Wert kleiner 0 zurück, die Schleife wird abgebrochen und der Empfangsvorgang ist beendet.

3.2 Sender

Um UDP-Pakete zu senden, müssen folgende Schritte ausgeführt werden.

- Socket erstellen.
- *sockaddr_in* struct erstellen und Attribute festlegen.
- In mehreren Schritten mittels *sendto* Pakete senden.

3.2.1 Socket erstellen

3.2.2 *sockaddr_in* struct erstellen und Attribute festlegen

3.2.3 In mehreren Schritten mittels *sendto* Pakete senden

3.3 Empfänger

Um UDP-Pakete zu empfangen müssen folgende Schritte ausgeführt werden.

- Socket erstellen.
- *sockaddr_in* struct erstellen und Attribute festlegen.
- Timeout für Socket festlegen.
- Socket mit Adressen-Objekt mittels *bind* verbinden.
- Mittels $recvfrom$ Pakete empfangen.

4 Auswertung (Java-Version)

5 Auswertung (C-Version)