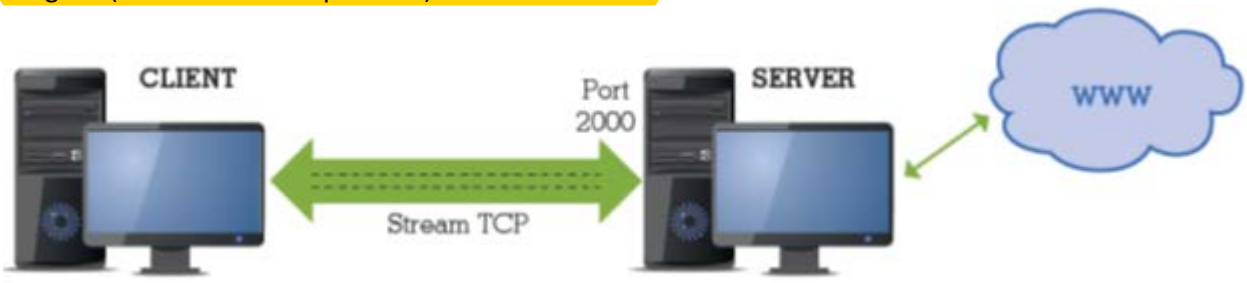
Übungen Client/Server

In diesem Übungsblatt sollten Client/Server Anwendungen mithilfe von Sockets geschrieben werden. Für jede Anwendung sollte vor der Programmierung ein Protokoll für die Kommunikation überlegt werden. Dabei muss klar hervorgehen welche Nachrichten nötig sind und welches Format diese haben. Dabei sollte es eine abstrakte Klasse „Message“ geben und für jede Nachricht eine konkrete Klasse die davon erbt. Des Weiteren muss das Verhalten des Servers und des Clients mithilfe eines Zustandsdiagramms beschrieben werden. Dabei muss klar hervorgehen welche Nachricht für den Zustandswechsel benötigt wird. Bei allen Aufgaben sollten die Exceptions korrekt und sinnvoll abgefangen und die Ressourcen nach der Verwendung wieder freigegeben werden.

1) Erstelle eine Client/Server Anwendung in Java, die für Clients einfache Berechnungen durchführt. Dabei sollte die Addition, Subtraktion, Multiplikation und Division mit Integer-Zahlen ermöglicht werden. Der Client wählt eine Operation aus und schickt dem Server eine Anfrage. Der Server führt diese Anfrage aus und schickt dem Client das Ergebnis zurück.

2) Aufgabe 1 soll mit deiner Benutzerauthentifizierung erweitert werden. Der Client muss sich vor der Anfrage authentifizieren. Nur bei erfolgter Authentifizierung ist es dem Client möglich den Dienst zu nutzen. Überlege dir hierfür einen Ansatz und erweitere das Protokoll von Aufgabe1 demensprechend.

3) Erstelle eine Client/Server Anwendung in Java, die den Download eine Webseite ermöglicht. Der Client spezifiziert die URL und sende sie an den Server. Der Server downloadet die Seite, speichert sie als html-File lokal ab und schickt sie an den Client. Welchen Vorteil hat dieses Vorgehen und welche Aufgabe (als Netzwerkkomponente) erfüllt der Server?



4) Erstelle eine Client/Server Anwendung in Java die das Spiel „Vier gewinnt“ für zwei Spieler realisiert. Regeln laut https://de.wikipedia.org/wiki/Vier_gewinnt

   Das Spiel wird auf einem senkrecht stehenden hohlen Spielbrett gespielt, in das die Spieler abwechselnd ihre Spielsteine fallen lassen. Das Spielbrett besteht aus sieben Spalten (senkrecht) und sechs Reihen (waagerecht). Jeder Spieler besitzt 21 gleichfarbige Spielsteine. Wenn ein Spieler einen Spielstein in eine Spalte fallen lässt, besetzt dieser den untersten freien Platz der Spalte. Gewinner ist der Spieler, der es als erster schafft, vier oder mehr seiner Spielsteine waagerecht, senkrecht oder diagonal in eine Linie zu bringen. Das Spiel endet unentschieden, wenn das Spielbrett komplett gefüllt ist, ohne dass ein Spieler eine Viererlinie gebildet hat.

5) Realisiere eine Client/Server Anwendung in Java, die einen File-Transfer vom Server zum Client ermöglicht. Der Client kann irgendeinen Pfad zu einer Datei oder einem Ordner angeben der dann vom Server zum Client übertragen wird. Dabei sollen auch parallele Downloads möglich sein.

6) Realisiere eine Client/Server Anwendung in Java. Der Server stellt zwei Funktionen (getRandom() und getTime()) zur verfügen, die eine lange Berechnung simulieren sollen. Für diese Aufgabe sollten zwei verschiedene Serverversionen erstellet werden. Eine sequenzielle Version, die nur einen Thread verwendet und somit die Clients nacheinander abarbeitet und eine parallele Version die mehrere Threads verwendet und die Clients parallel abarbeitet. Teste beide Server und dokumentiere deine Beobachtungen. Welche Einschränkungen und welches Leistungsvermögen haben die Server?

7) Erweitere Aufgabe 6 so, dass der Server einen sauberen Shutdown durchführt. D.h. das alle bereits gestarteten Anfragen sollten korrekt beendet, aber keine neue mehr gestartet werden. Alle auftretenden Exceptions sollten dabei korrekt behandelt werden. Verwende dabei das Interface ExecutorService.
(https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html)

8) Realisiere eine Client/Server Anwendung in Java, die einen Chat zwischen mehreren Clients ermöglicht. Verwende dabei UDP-Datagramme.

9) Realisiere eine Client/Server Anwendung in C. Dabei sollte der Server die Lawinengefahr mehrerer Ortschaften berechnen und dem Client zurückgeben.

10) Schreibe eine parallele Client/Server Anwendung in C. Dabei können zwei beliebige Funktionen verwendet werden. Es ist auch möglich dieselben Funktionen wie bei Aufgabe 6 zu verwenden. Wichtig ist allerdings wie bei Aufgabe 6, dass die Berechnungen lange dauern. Vergleiche die parallele Lösung mit der sequenziellen und dokumentiere deine Beobachtungen.

11) Bei dieser Aufgabe sollte überprüft werden, ob der Client und der Server plattformunabhängig sind.

    a) Realisiere einen File-Server in C der unter Linux läuft. Er soll eine Verbindung öffnen können und auf eine Anfrage mit dem Format **GET filename.ext** mit dem Inhalt der angefragten Datei filename (mit der Dateierweiterung ext) antworten (falls diese vorhanden ist). Bei allen Anfragen die nicht dem Schema **GET filename.ext** entsprechen, soll eine Fehlermeldung auf der Konsole ausgegeben **und** an den Client geschickt werden. Auch im Falle, dass die Datei nicht existiert, ist dies entsprechend zu behandeln. Test der Implementierung: Verbinden dich via telnet mit deinem Server und versuche verschiedene Dateianfragen und teste deine Fehlermeldungen!

    b) Realisiere des Weitern für die Überprüfung der Plattformunabhängigkeit einen Client in Java der auf einem Windows Rechner läuft. Verbinde dich zum File-Server aus a) und lade eine Datei herunter. Das Programm soll drei Parameter von der Kommandozeile verarbeiten (IP, Port, Dateiname), um einen Aufruf auf folgende Art und Weise zu ermöglichen:
    java FileClient 192.168.0.1 22222 testfile.txt

    Der Client kann nach der Verbindung mit dem Server einfach den Ausdruck GET testfile.ext an den Server schicken und die komplette Rückmeldung soll in einer Datei gespeichert werden. Weiters soll

der Client eventuelle Fehlerfälle (Ausnahmen bei falscher IP oder falschem Port) entsprechend abfangen und behandeln!

c) Teste die gesamte Implementierung:
• Verwende den Client und verbinden dich mit dem Server und transferiere eine Datei, die existiert.
• Versuche eine Datei zu empfangen, die nicht existiert um die Fehlerbehandlung im Client zu zeigen.

12) **Zusatzaufgabe:** Realisiere ein P2P Netzwerk mit Java oder C Sockets. Dabei können beliebige Funktionalitäten angeboten werden. Beim einem P2P Netzwerk sind alle Rechner gleichgestellt, d.h. alle Rechner können als Client und Server agieren. Durch diese Eigenschaft ist bei einem P2P Netzwerk die Adresse und der Port des Servers nicht bekannt, deshalb muss ein ServiceAnnouncer und ein ServiceLocator implementiert werden. Der ServiceAnnouncer wartet auf ein Ping und schickt, sobald er dieses empfängt ein Pong zurück mit diesem er dem Sender des Ping mitteilt, welchen Service er anbietet. Der ServiceLocator schickt einen Ping an die Broadcastadresse und wartet auf die Antworten. Diese beiden Klassen sollen eine Instance von der Klasse DatagramSocket verwenden um die Funktionalitäten zur Verfügung zu stellen. Beschreibe welchen Vor- und Nachteil dein Protokoll und die Verwendung der Klasse DatagramSocket haben.

Abgabe:

Alle Aufgaben müssen selbständig gelöst und kommentiert werden. Bei identischen Aufgaben bekommen die betreffenden Schüler eine stark negative Note.
Abzugeben sind pro Aufgabe: Protokoll, Zustandsdiagramm, Test der Funktionsweise (z.B. Screenshot, Kommandozeilenausgabe), Quellcode gut dokumentiert (u.a. wie das Programm kompiliert und ausgeführt werden kann), Executable Java File (für Java Projekte) oder eine ausführbare Datei (für C Programme) für alle Clients und Server. Das Protokoll, Zustandsdiagramm und der Test müssen in einer Datei namens readMe abgespeichert werden. Bei den Aufgaben wo auch Fragen zu beantworten sind, sollten auch diese in der readMe-Datei mit den entsprechenden Antworten vorhanden sein.
Eine weitere readMe-Datei aus der klar hervorgeht welche Aufgaben gelöst und warum evtl. Aufgaben nicht gelöst wurden (1 Absatz pro Aufgabe) muss vorhanden sein. In der readMe-Datei soll sich jeder Schüler auch selbst bewerten (Note von 1-10 geben).
Der Schüler sollte endsprechend seiner Note die Programme bei einer Prüfung erklären können. Eine Aufgabe zählt nur dann als abgegeben, wenn alle Dateien dieser Aufgabe vorhanden sind.

Abgabetermin: 17.02.2017 um 24 Uhr mit Git. Das Repository muss ClientServer genannt werden. Für jede Übung muss ein Unterordner mit den entsprechenden Files erstellt werden. Dieser Ordner muss so genannt werden, dass eindeutig hervorgeht um welche Aufgabe es sich handelt (z.B. 01_Berechnungen für Aufgabe 1). Die readMe-Datei aus der hervorgeht welche Aufgaben gelöst wurden und der vorgeschlagenen Note, muss in der Wurzel-URL liegen. In den Unterordnern soll die readMe-Datei der entsprechenden Aufgaben vorhanden sein. Alle readMe-Dateien müssen als **PDF** abgegeben werden.