UNIVERSITÄT LEIPZIG

Institut für Informatik
Lehrstuhl für Rechnernetze und Verteilte Systeme

Programmierübung
Kommunikationssysteme
WS 19/20

Betreuer: **M.Sc. Sebastian Günther**

Ziele: **Die Aufgaben 1 und 2 der Programmierübungen vertiefen die Inhalte der Vorlesung und dienen der Vorbereitung von Prüfungen. Es wird vorausgesetzt, dass die Aufgaben selbständig bearbeitet werden.**

Gruppen: **Entgegen der Aussage in der Einführungsveranstaltung sollen die Aufgaben in Zweiergruppen bearbeitet und vorgestellt werden.**

Zulassung Klausur:

Die erfolgreiche Bearbeitung der Programmierübungen sowie das erfolgreiche Absolvieren des Reviewgesprächs sind Voraussetzung für die Zulassung zur Prüfungsklausur. Für das Reviewgespräch müssen beide Partner in der Lage sein, beide Aufgaben/Lösungen zu erklären.

Programmiersprache:

Die Programmierung muss in Java erfolgen. Die zu implementierende Netzwerkkommunikation muss mittels der Socket-API des Java SE erfolgen!

Dokumentation:

Der Code soll so dokumentiert sein, dass Außenstehende, die die verwendeten Programmierparadigmen und das API nicht kennen, sich problemlos einlesen können.

Abgabe:

Ihr Quellcode muss in geeignet gepackter Form (.zip/.tar.gz) bis zum 19.01.2020 (23:59 Uhr) im Moodle Kurs abgegeben werden. Bitte vermerken Sie auch ihre Gruppenpartnerin/ihren Gruppenpartner bei der Abgabe. Ein Upload je Gruppe ist ausreichend.

Im Anschluss (ab 20.01.) findet eine Präsentation der Lösung, sowie ein Reviewgespräch statt. Die Terminvergabe finden Sie im neuen Jahr ebenfalls im Moodle-Kurs.

Aufgabe 1: Client/Server Paradigma

Programmieren Sie einen stark vereinfachten **Dateiserver** und einen dazu passenden **Client**!

Der **Server** soll Zugriff auf ein Verzeichnis mit mehreren (kleinen) Textdateien haben. Mittels Anfragen soll er das Auflisten der Dateien im Verzeichnis sowie die Rückgabe eines Dateiinhaltes ermöglichen.

Der **Client** soll als Konsolenanwendung das Verbinden zum Server, Absetzen der Anfragen/Befehle sowie die Ausgabe der Antworten ermöglichen.

Folgende Befehle sollen unterstützt werden:

`LIST`	Dateiauflistung: Rückgabe aller Dateinamen im Verzeichnis
`GET <Dateiname>`	Anfordern der Textdatei <*Dateiname*> im Verzeichnis
`QUIT`	Schließen der Verbindung

Zusätzliche Anforderungen:

1. TCP muss als Transportprotokoll eingesetzt werden.
2. Der Client soll mehrere Befehle absetzen können und die Verbindung soll aufrechterhalten werden.
3. Der Server soll Anfragen von mehreren Clients sequentiell verarbeiten können. Ein paralleler Zugriff mehrerer Clients ist nicht notwendig.
4. Fehler (z.B. Datei nicht vorhanden, Server nicht erreichbar) sollen von Server und Client abgefangen und behandelt werden (Ausgabe von Fehlermeldung).

Hinweise:

Client und Server können auf demselben Gerät laufen und über Ports oberhalb von 50000 kommunizieren
GUI-Anwendungen werden **nicht** gefordert (Konsolenanwendungen sind ausreichend!)

Aufgabe 2: Peer-to-Peer Paradigma

Programmieren Sie ***einen*** Client für ein Peer-to-Peer System (nur eine ausführbare Datei).

Der Client soll als Teil eines P2P-Wetterdatennetzwerks seine Statusinformationen mit anderen Clients austauschen. Dazu nimmt er zwei Aufgaben gleichzeitig wahr:

1. **„Advertise“:** Der Client sendet periodisch Anfragen an andere Clients innerhalb der Portrange.
2. **„Discover“ und Reply:** Der Client bearbeitet ankommende Pakete.
    a. Beim Empfang von Anfrage-Paketen übermittelt er seine Statusinformationen (als Status-Pakete) an den Absender des Pakets.
    b. Beim Empfang von Status-Paketen übernimmt der Client die darin enthaltenen Informationen in den eigenen Bestand (sofern aktuell, etc.).

Der Clientstatus enthält folgende Daten: Standort, Temperatur, Luftfeuchtigkeit, Timestamp

Zusätzliche Anforderungen:

1. UDP muss als Transportprotokoll eingesetzt werden.
2. Beim Start eines Clients soll dessen Standort eingegeben werden. (z.B. „Gipfel“, „Rathaus“, „Markt“, …)
3. Temperatur und Luftfeuchtigkeit eines Clients können zufällig gewählt werden und sollten sich periodisch ändern.
4. Clients sollen sich jeweils nur den aktuellen Status je Standort merken.
5. Clients sollen jeweils alle bekannten Statusinformationen teilen, sodass auch nach dem Ausfall eines Clients dessen zuletzt übermittelter Status weiter im Netz verbreitet wird.
6. Verwenden Sie eine feste Portrange (50001-50010) für die Kommunikation.
7. Clients sollen sich selbstständig aus dieser Range einen freien Port suchen.

Testszenario:

1. Starten zweier Clients (C1, C2)
2. Warten, bis diese erfolgreich ihren Status ausgetauscht haben
3. Beenden von Client C1
4. Starten eines weiteren Clients (C3)
5. C3 sollte nun den Zustand von C1, C2 und C3 kennen!