Praktikum Verteilte Systeme

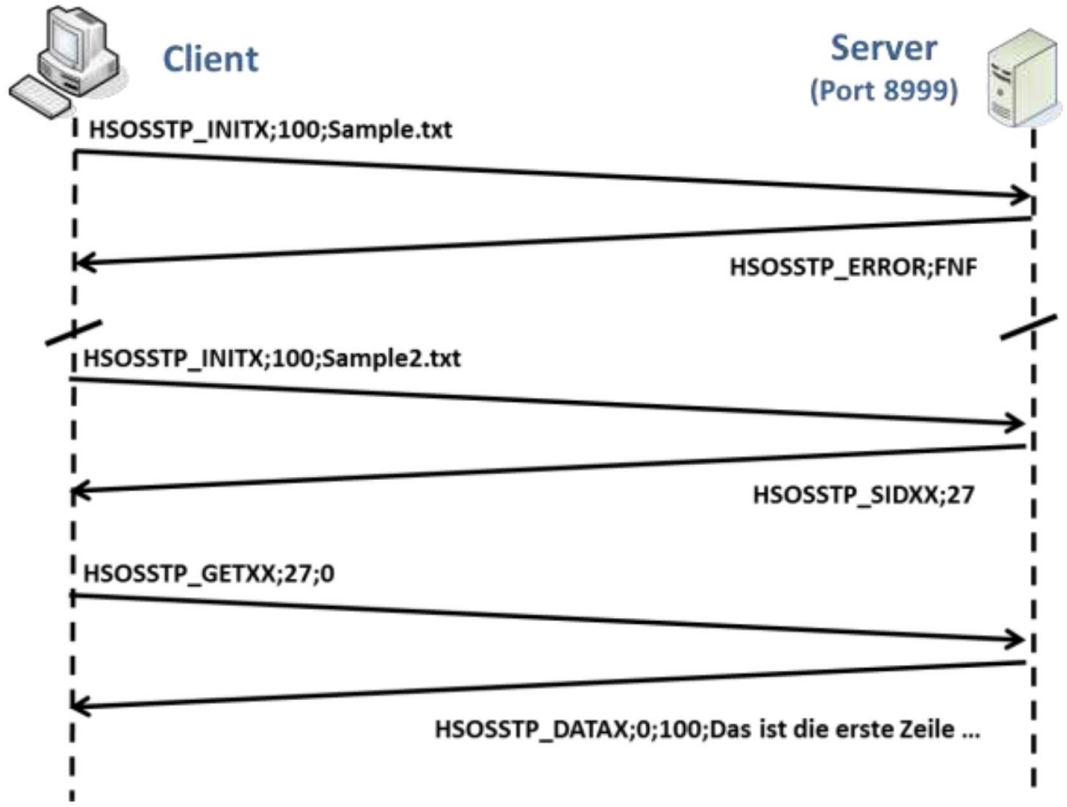
Das Praktikum findet im Pool-Raum SI0024 des Laborbereichs Technische Informatik statt. Für das Praktikum ist eine Vorbereitung erforderlich. Während des Praktikums ist ein Praktikumsprotokoll mit Namen, E-Mail-Adresse, Datum und allen wichtigen Ergebnissen zu erstellen und mit dem Betreuer am Bildschirm durchzusprechen. Das Protokoll soll für 2.1 und 2.2 die für einen Nachweis der Funktion erforderlichen Ein- und Ausgaben enthalten. Kopieren Sie diese (als Text!) aus der Konsole ins Protokoll und beantworten Sie außerdem die Fragen zu 2.3. Laden Sie das Protokoll zusammen mit dem gezippten Source-Code im OSCA-Dokumentenabgabeordner unter praktxx_Name1_Name2.doc hoch (xx=Nr. der Praktikumsaufgabe).

Praktikumsaufgabe 2 (Protokoll zum Datentransfer)

Ziel der Aufgabe ist es, auf Basis eines iterativen UDP-Servers ein einfaches Protokoll für Transfer von Textdateien zu entwickeln und zu testen. Die Aufgabe kann in C oder Java gelöst werden. Sie muss bis zum 25.03, 26.03. bzw. 28.03.2019 fertig gestellt werden. Im Skript und in kap4_beispiele.zip gibt es kein Java-UDP-Server/Client-Beispiel. Sie können sich z.B. an folgenden Vorlagen orientieren:
https://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html
http://openbook.rheinwerk-verlag.de/java7/1507_11_011.html
Es gibt 10 Punkte, von denen mindestens 8 erreicht werden müssen.

2.1 Umsetzung eines Protokolls zum Datentransfer (5 Punkte)

Es soll eine Applikation bestehend aus Client und Server auf Basis von Sockets für den Transfer von Dateien zwischen Rechnern implementiert werden. Der Datentransfer soll dabei in Teilen (Chunks) festgelegter Größe erfolgen. Die Größe der Chunks wird zu Anfang der Kommunikation durch den Client festgelegt. Das folgende Bild zeigt den prinzipiellen Ablauf



1. Der Client initiiert den Transfer. Der Client sendet dazu eine Nachricht: **HSOSSTP_INITX;<*chunk size*>;<*filename*>** an den Server. Hierin gibt **<*chunk size*>** die Größe der transferierten Datenblöcke in Bytes an. Der zweite Parameter definiert den Namen der Datei, die vom Server zum Client zu transferieren ist.
2. Es soll zu Anfang ein Sitzungsschlüssel zwischen Client und Server vereinbart werden. Der Server sendet nach der initialen Anfrage des Clients den Sitzungsschlüssel in einer Nachricht: **HSOSSTP_SIDXX;<*session key*>** an den Client. Als Sitzungsschlüssel wird ein Integer-Wert verwendet, der während der Laufzeit des Servers eindeutig sein soll (er kann also z.B. als Zähler realisiert werden). Der Sitzungsschlüssel wird intern im Server mit dem Zustand des zu der Sitzung gehörenden Datentransfers verknüpft (Filedeskriptor der zu transferierenden Datei und Anzahl der übertragenen Chunks).
3. Kann ein Transfer nicht initiiert werden oder tritt während des Transfers ein Fehler auf, so wird an den aufrufenden Prozess eine Fehlermeldung **(HSOSSTP_ERROR;<reason>)** ausgegeben.
4. Der Client sendet nun iteriert Anfragen der Art **HSOSSTP_GETXX;<session key>;<chunk no>** an den Server. Die Chunks werden beginnend mit 0 durchnummeriert. Da die Größe der Chunks bei den **HSOSSTP_GETXX** Nachrichten nicht mitgeschickt wird, muss im Server eine Zuordnung dieses Wertes zu dem Sitzungsschlüssel mitgespeichert werden.
5. Der Server sendet daraufhin das entsprechende Paket vereinbarter Größe zurück. Diese werden mit einem Prefix **HSOSSTP_DATAX;<chunk no>;<actual chunksize>;<data>** versehen. **<chunk no> und <chunk size>** geben die Nummer und die Größe des gerade transferierten Chunks an; **<data>** die transferierten Daten.
6. Die Daten werden durch den Client gelesen und in der lokalen Datei gespeichert.

Realisieren Sie die o.g. Funktionen in Form eines iterativen UDP-Servers und eines Clients. Der Server soll in der Lage sein, mehrere Transfersitzungen gleichzeitig zu bedienen. Der Client soll mit der IP-Adresse des Servers, der Größe der Chunks und dem Namen der vom Remote-System zu transferierenden Datei als Parameter aufgerufen werden können. Ein beispielhafter Aufruf lautet:

```
HSOSSTP_client 131.173.110.1 100 Sample.Txt#
```

Mit dem Shell-Befehl **diff** können Sie die transferierten und Original-Dateien miteinander vergleichen.

2.2 Erweiterung durch Abfangen von Fehlerfällen (2 Punkte)

Aus Gründen der Kompatibilität (sie sollen auch die Server der anderen Teilnehmer verwenden können) seien die folgenden Fehlermeldungen vorgegeben:

- **HSOSSTP_ERROR;FNF**: angeforderte Datei kann nicht gefunden werden
- **HSOSSTP_ERROR;CNF**: angeforderter Chunk kann nicht transferiert werden
- **HSOSSTP_ERROR;NOS**: keine Session vorhanden

Sie sollen die Funktion des Systems bei Verteilung des Clients und Servers auf unterschiedliche Hosts demonstrieren. Testen Sie mit Ihrem Client mindestens einem Server einer anderen Gruppe!

2.3 Fragen zur Lösung (3 Punkte)

Beantworten Sie im Protokoll die folgenden Fragen zu Ihrer Lösung:

- Was ist speziell bei Verwendung von UDP als Transportprotokoll zu berücksichtigen?
- Warum sollte für die Umsetzung des Protokolls UDP und nicht TCP verwendet werden?
- Wie kann man das Ende einer Datentransfer-Sitzung erkennen? Beschreiben Sie die bei Ihrer Lösung ausgetauschten Nachrichten durch ein Sequenzdiagramm. Orientieren Sie sich dabei am Bild auf der ersten Seite.
- Wie werden Datentransfer-Sitzungen terminiert?
- Was passiert bei Wiederaufnahme einer Sitzung?
- Was passiert beim Zugriff auf eine nichtexistierende Sitzung?
- Wie groß kann die Chunk-Size maximal eingestellt werden?

Hinweise bei der Realisierung in C:

Beschreiben Sie, warum in Client und Server die Dateigröße unwichtig ist und nicht ermittelt werden muss (und darf) und ob das Programm bei großen Dateien mehr Heap-Speicher als bei kleinen benötigt. In dieser Aufgabe sind folgende C-Funktionen zu verwenden:

- **`FILE *fopen (const char *filename, const char *mode);`**
  Öffnen einer Datei zum Lesen (Server) oder Schreiben (Client).
- **`int fclose(FILE *stream);`**
  Schließen einer Datei.
- **`size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`**
- **`size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE* stream);`**
  Lesen von und Schreiben auf eine Datei.
- **`char *strtok( char *string1, const char *string2 );`**
  Parsen von Nachrichten zwischen Client und Server

Details zu den Funktionen finden Sie ein den man-Pages Versuchen Sie nicht, diese Funktionen selbst nachzubauen oder die Aufgabe mit anderen als den angegebenen Funktionen zu lösen.

Fehlersuche in C:
Kompilieren Sie stets mit dem **`gcc`** und der Option –g zur Erzeugung von Debug-Informationen.
Das korrekte Arbeiten der dynamischen Speicherverwaltung kann mit dem Befehl **`valgrind`** überprüft werden: **`valgrind –v <ausführbare Datei>`**
Auf manchen Linux-Systemen muss der Aufruf mit **`/usr/bin/valgrind`** erfolgen.