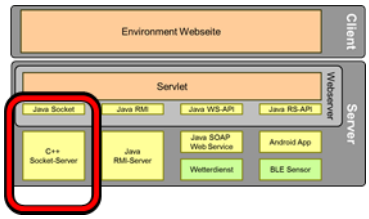
Verteilte Informationssysteme

Person:

Name:

Matrikelnummer:



Übungszettel 1

Ziel ist es, die **Kommunikation** zwischen verteilten Systemen mittels **TCP/IP Sockets** zu realisieren. Dabei soll, ausgehend von einem primitiven Verbindungsaufbau, die Verfahrensweise schrittweise verfeinert werden.

Anmerkung (WINDOWS only …):

- Eine gute Hilfe ist die Webseite www.msdn.com unter Angabe des Suchbegriffes „Winsock Reference“ → Winsock Functions.
- Orientieren Sie sich bei der Implementierung an folgendem Code-Fragment

```
#include <stdio.h>
#include <iostream>
#include <winsock2.h>
#include <ws2tcpip.h>

// gewaehrleistet das Laden der entsprechenden Bibliothek
#pragma comment(lib, "Ws2_32.lib")

// Rueckgabewerte
#define SERVER_SOCKET_ERROR 1
#define SERVER_SOCKET_OK    0

// namespace fuer die Verwendung von cin, cout, cerr
// und endl wird festgelegt
using namespace std;

int main(int _argc, char* _argv[]) {

  // die Version der Socket-DLL festlegen
  WORD    socketDLLVersion = MAKEWORD(2,2);
  WSADATA wsaData;
  // die Socket-Bibliothek aktivieren
  int rVal = WSAStartup(socketDLLVersion, &wsaData);
  if (rVal != 0 ) {
    cerr << "could not find specified socket dll version"
    << endl;
    return SERVER_SOCKET_ERROR;
  }
```

CLion Anmerkungen

Buildfile CMakeList erweitern

```
…
target_link_libraries(TARGET_NAME wsock32 ws2_32)
```

Aufgabe 1a „O“ (5%)

Erstellen Sie anhand der in der Vorlesung vorgestellten Angaben zwei C++ Programme (Server und Client), die sich mittels TCP/IP-Sockets miteinander verbinden. Einen Server, der auf einem bestimmten Port auf die Anfrage eines Clients wartet und einen Client, der sich mit dem Server über den entsprechenden Port verbindet.

Berücksichtigen Sie dabei mögliche Fehlerzustände beim Verbindungsaufbau und testen Sie die Funktionsfähigkeit der Verbindungsaufnahme.

Aufgabe 1b „O“ (5%)

Verfeinern Sie die Programme um folgende Aspekte:

- Übermitteln Sie den Kommunikations-Port als Übergabeparameter beim Start des jeweiligen Programms (Client/Server) über den **`_argv`** Parameter an der Stelle [1] der **`main`** Methode.
- Übermitteln Sie die Server-IP Adresse beim Start des Client Programms über den **`_argv`** Parameter an der Stelle [2]. Überprüfen Sie das Vorhandensein der Werte und geben Sie ggf. eine Bedienungsanleitung aus.
- Lesen Sie die vorhandenen Informationen über den Client (Client-Adresse und Kommunikations-Port) beim Verbindungsaufbau aus und geben Sie diese auf der Konsole aus.

Aufgabe 1c „O“ (5%)

Verfeinern Sie die Programme in folgenden Schritten:

1. Seitens des Clients wird eine Zeile von der Kommandozeile „komplett“ mittels **`cin`** eingelesen und an den Server geschickt.
2. Der Server liest die Informationen aus der Socket aus und gibt sie seinerseits auf der Kommandozeile aus.
3. Der Server beendet sich nicht, wenn der Client die Verbindung abbricht, sondern wartet auf den nächsten Client.
4. Der Client sendet erst dann wieder Daten, wenn er vom Server eine Empfangsbestätigung (ein „**`ACK`**“) erhalten hat.

Aufgabe 2a „M“ (10%)

Entwerfen Sie eine TCP-Server Klasse (inkl. Header-Datei und Konstruktor/Destruktor), die über die gleiche Funktionalität wie der Server aus Aufgabe 1 verfügt und zumindest folgende Methode unterstützt.

```
void InitializeSocket(…) {
    …
    // Socket-Handler
    // bind
    // setSocketOptions
    // listen
    // accept
}
```

Eine Wiederverwendung bzw. Restrukturierung der Ergebnisse aus Aufgabe 1 ist erlaubt.

Konzipieren Sie den Server als „Echo“-Server, der eine Rückmeldung der Form „**ECHO:** some message from client“ zum Client schickt, im Gegensatz zum „**ACK**“ in Aufgabe 1.

Testen Sie die Funktionsfähigkeit Ihres Systems!

Aufgabe 2b „O“ (10%)

Entwickeln Sie neue Server und Client Klassen, die die Funktionalität eines „Echo“-Servers auf Basis einer UDP-basierenden Kommunikation realisieren und testen Sie diese.

Aufgabe 2c „O“ (5%)

Erweitern Sie ihren TCP-Server um die Verwendung von IPv6 Adressen und testen Sie ihn durch die Verwendung entsprechender Adressen.

Aufgabe 2d „O“ (15%)

Erweitern Sie den Server um die Möglichkeit „gleichzeitig“ mehrere Clients zu bedienen. Verwenden Sie hierfür Threads. Lagern Sie hierfür den **send**/**rcv**-Anteil der **StartServer**-Methode in die statische **ClientCommunication**-Methode („Thread-Methode“) aus.

Testen Sie die Funktionsfähigkeit Ihres Systems durch die gleichzeitige Verwendung mehrerer Clients!

Aufgabe 2e „O“ (5%)

Verwenden Sie den Synchronisationsmechanismus „Semaphor“, um die Anzahl der maximal, gleichzeitig kommunizierender Clients auf 3 zu beschränken. Implementieren Sie hierfür auf der Server Seite zwei Methoden **IncrCounter()** und **DecrCounter()**, die die Anzahl der aktuell verbundenen Clients ausgeben und den Semaphor inkrementieren bzw. dekrementieren. Rufen sie die beiden Methoden in der **ClientCommunication** am Anfang bzw. am Ende auf und testen Sie somit die korrekte Funktionsweise des Semaphors, in dem Sie versuchen mit mehr als drei Clients mit Ihrem Server zu kommunizieren.

Aufgabe 3a „M“ (20%)

Entwickeln Sie ein Kommunikationsprotokoll zwischen Client und TCP-Server, das folgende Aspekte berücksichtigt:

- Ein Request/Response-Paar für die Abfrage der verfügbaren Sensoren:

Client Request	Server Response
`sensortypes#`	`<sensortype>;…;<sensortype>#`
`sensortypes#`	`light;noise;air#`

- Ein Request/Response-Paar für die Abfrage konkreter Sensorwerte:

Client Request	Server Response
`sensor;<sensortype>#`	`<timestamp>|<value>[;<value>]*#`
`sensor;air#`	`123456789|123;235;23#`

- Ein Request/Response-Paar für die Abfrage aller Sensorwerte:

Client Request	Server Response
`sensor;ALL#`	`<timestamp>|<sensor>;<value>[|<sensor>;<value>]*#`
`sensor;ALL#`	`123456789|light;123|noise;42|air … #`

Simulieren Sie seitens des Servers das Vorhandensein entsprechender Sensoren und manipulieren Sie die entsprechenden Sensorwerte mittels eines Zufallszahlengenerators. Beachten Sie, dass einige Sensoren nur einen Datenwert liefern, andere hingegen mehrere.

Testen Sie die Funktionsfähigkeit des Systems mit Hilfe des C/C++ Clients, wobei eine rein kommandozeilenbasierte Variante (ohne Menü oder ähnliches) ausreichend ist!

Aufgabe 3b „O“ (5%)

Entwickeln Sie auf Basis von Java das bereits bekannte Client/“Echo“-Server Tupel und testen Sie die Funktionalität.

Aufgabe 3c „M“ (10%)

Entwickeln Sie einen auf Java basierenden Client, der über eine Socket mittels des unter Aufgabe 3a angegebenen Protokolls mit Ihrem TCP-Server kommuniziert.

Eine rein kommandozeilenbasierte Variante (ohne Menü oder ähnliches) ist ausreichend!

Aufgabe 3d „M“ (5%)

Stellen Sie auf der Clientseite eine „Environment Data“-API mit folgendem Aussehen zur Verfügung:

```
public interface IEnvService {
  public String[]  requestEnvironmentDataTypes();
  public EnvData   requestEnvironmentData(String _type);
  public EnvData[] requestAll ();
}
```

Lassen Sie ihren Java-Client die Schnittstelle implementieren und testen sie die Funktionsfähigkeit des Systems mit nachfolgendem Code.

```
public static void main(String[] _argv) {

  IEnvService service = new MyJavaClient();
  while (true) {
    String[] sensors = service.requestEnvironmentDataTypes();
    for (String sensor : sensors) {
      EnvData dataO = service.requestEnvironmentData(sensor);
      System.out.print(dataO)
      System.out.println();
      System.out.println(*****************************);
    }// for sensor
    System.out.println();
    System.out.println();
    EnvData[] dataOs = service.requestAll();
    for (EnvData dataO : dataOs) {
      System.out.println(dataO);
    } // for data
    try {
      Thread.sleep(1000);
    } catch (Exception _e) {
      _e.printStackTrace();
    }
  } // while true
}
```