Beispiel `12_time`

Dr. Günter Kolousek

21. Januar 2019

1 Allgemeines

- **Backup** nicht vergessen!
- Hier nochmals zwei Erinnerungen:
  - Regelmäßig Commits erzeugen!
  - Backup deines Repos nicht vergessen (am Besten nach jedem Beispiel)!!!
- In diesem Sinne ist jetzt ein neues Verzeichnis `11_time` anzulegen.

2 Aufgabenstellung

Dieses Beispiel behandelt die einfache Stream-orientierte Kommunikation über Sockets an Hand des time-Protokolls. Ein Client nimmt Kontakt zu einem Server auf, empfängt die Serverzeit über TCP und gibt diese aus. Die Kommunikation findet *binär* statt, wobei die Zeit als Anzahl der Sekunden seit dem 1.1.1900 UTC definiert ist und die Übertragung in 4 Bytes durchgeführt wird, die in network byteorder übertragen werden (big endian).

Behandlung der Kommandozeilenparameter und Loggen ist wiederum einzubauen!

Los geht's!

3 Anleitung

1. Entwickle einen Time-Client auf TCP Basis, der sich zu einem lokalen Time-Server (siehe `TimeServer.class`) mit einem konfigurierbaren (Kommandozeilenparameter!) Port verbindet, die aktuelle Zeit erfragt und danach diese auf der Konsole ausgibt. Die Übertragung des Zeitwertes findet wie oben beschrieben statt.

Dieses Programm `time_client` ist in einem **eigenem** Verzeichnis `time_client` im Verzeichnis `src` zu entwickeln. Als Programmname für den eigentlichen Sourcecode bietet sich `main.cpp` an.

Der zur Verfügung gestellte Time-Server kann folendermaßen gestartet werden:

```
$ java TimeServer 1137
just before accept
```

erfolgen. Die Ausgabe "just before accept" gibt lediglich an, dass der Server auf eine Verbindungsanfrage wartet und dient dazu die Funktionsbereitschaft des Servers zu zeigen.

Der Client soll einfach einen Stream zum Server aufbauen, 4 Bytes lesen und diese als Anzahl der Sekunden seit Beginn der Epoche interpretieren, wobei diese in network byte order vom Server übertragen werden.

Fehlerbehandlung ist wie im vorherigen Beispiel einzubauen.

Folgende Punkte sind wichtig:

- Der Beginn der Epoche wird als 1.1.1970 UTC angegeben. Darauf bezieht sich im wesentlichen die Zeitrechnung im Unix-Systemen und hat sich auch für andere Systeme durchgesetzt.
- Die Anzahl der Sekunden vom 1.1.1900 bis zum Beginn der Epoche sind 2208988800.
- Mit `strm.read(reinterpret_cast<char*>(&x), sizeof(x));` kann ein binärer Wert von einem Stream gelesen werden. Die Zeile bedeutet, dass vom Stream die Anzahl der Bytes (korrekt die Anzahl der Vielfachen von `char`) der Variable `x` in die Variable `x` gelesen werden. Der `reinterpret_cast` ist notwendig, da die Methode `read` sich einen `char*` erwartet, aber die Variable `x` in der Regel einen anderen Wert enthält.
- Der Typ der Variable `x` soll bei uns so bemessen sein, dass dieser 4 Bytes beinhalten kann und mit diesem gerechnet werden kann. Nun ist es in `C++` aber so, dass die Größen der Datentypen `char`, `int`, `long`,... nicht exakt spezifiziert sind. Wie sollen wir dann genau 4 Bytes angeben? Hier hilft der Typ `uint32_t` aus der Headerdatei `<cstdint>`.
- Beachte weiters, dass network byte order (big endian) für die Übertragung spezifiziert ist. Intel Systeme verwenden allerdings little endian. Abgesehen davon kann man sich nicht sicher sein unter welchen System das Programm letztendlich zur Ausführung gelangen soll. Hier hilft die Funktion `ntohl` (network to host long) weiter, der man einen 32 Bit Integerwert übergeben kann, dieser in network byte order interpretiert und in host byte order umgewandelt wird. Diese nützliche Funktion ist in `<netinet/in.h>` zu finden.

- Mittels der folgenden Zeile

  ```
  chrono::time_point<chrono::system_clock> now{
      chrono::seconds{time_in_s_since_epoch}};
  ```

  kann der Zeitpunkt bestimmt werden, der dem Abstand der Sekunden seit Beginn der Epoche darstellt, wenn in `time_in_s_since_epoch` die Anzahl der Sekunden seit dem Beginn der Epoche gespeichert sind.

- Die Ausgabe des Wertes kann wie gewohnt mit meinem überladenen Operator `<<` aus der Headerdate `timeutils.h` vorgenommen werden.

Nachdem der Server läuft, kann der Client gestartet werden und die Ausgabe des Clients sollte jetzt folgendermaßen aussehen:

Beispiel:

```
$ time_client
2016-01-20 22:09:04
```

Der Server wird danach wieder ”just before accept” ausgeben.

Übungszweck

- Wiederholung Socketverbindung zu lokalem Host aufbauen.
- Übertragen von Binärdaten
- Erkennen der Problematik von big endian vs. little endian und Kennenlernen des Begriffes network byte order.

2. Weiter mit einem eigenen Time-Server, der für einen Time-Client die lokale Zeit zur Verfügung stellt.

   Füge deinem Projekt ein **weiteres** Unterverzeichnis `time_server` im Verzeichnis `src` hinzu und passe `meson.build` entsprechend an. Dann werden zwei ausführbare Programme gebaut, vorausgesetzt in `time_server` befindet sich auch eine Sourcecode-Datei wie z.B. `main.cpp`.

   Der Server soll an sich genau so funktionieren, wie unser Daytime Server, nur dass dieser das Time-Protokoll implementiert.

   Folgendes ist zu beachten:

   - Mittels `chrono::system_clock::now().time_since_epoch()` bekommt man eine `duration`, die die Zeitdauer seit Beginn der Epoche angibt.
   - Mittels `chrono::duration_cast<chrono::seconds>(d).count()` kann man von einer `duration d` die Anzahl der Sekunden ermitteln.

- Analog zum TimeClient kann man mit `htonl` die network byte order eines Wertes bekommen.
- Die Methode `write` eines Streams funktioniert anolog zur Methode `read` eines Streams.

Übungszweck

- Server-seitige Umsetzung des Time-Protokolls analog zur Client-Seite.