Beispiel `08_daytime`

Dr. Günter Kolousek

18. Januar 2021

1 Allgemeines

- Im ersten Beispiel gibt es genaue Anweisungen zum Aufbau und der Durchführung eines Beispiels. Bei Bedarf nochmals durchlesen!
- Trotzdem hier noch zwei Erinnerungen:
  - **Regelmäßig** Commits erzeugen!
  - **Backup** nicht vergessen!
- In diesem Sinne ist jetzt ein neues Verzeichnis `08_daytime` anzulegen.

2 Aufgabenstellung

Dieses Beispiel behandelt die einfache Stream-orientierte Kommunikation über Sockets an Hand des daytime-Protokolls. Ein Client nimmt Kontakt zu einem Server auf, empfängt die Serverzeit über TCP und gibt diese aus. Die Kommunikation findet zeichenbasiert statt, wobei das Format der Zeit nicht spezifiziert ist.
Los geht's!

3 Anleitung

1. In System für den Produktiveinsatz, speziell in Serverprogrammen ist das Logging eine essentielle Tätigkeit. Wir werden dafür die header-only Bibliothek `spdlog` verwenden. "Installiere" diese daher wie gewohnt und binde diese in dein Projekt (`meson.build` und `meson_options.txt`) ein!
2. Wir schreiben jetzt in unserem Projekt zwei Programme, nämlich einen Client und einen Server. Dies werden wir so organisieren, dass wir in unserem `src` Verzeichnis zwei Unterverzeichnisse, nämlich `daytime_client` und `daytime_server` anlegen. In diesen Unterverzeichnissen werden die jeweiligen `C++` Dateien abgelegt. Die ausführbaren Programme sollen `client` und `server` heißen.

Beginnen wir mit der Datei `main.cpp` für den Client: Jetzt geht es darum `spdlog` zu verwenden. Schaue dir dazu die bereitgestellte Dokumentation oder die Homepage von `spdlog` an und zeige in `main.cpp` mittels Testanweisungen, dass du `spdlog` verwenden kannst! Ich hätte gerne die Ausgaben färbig!

D.h. jetzt soll das Executable "funktionieren"

3. Jetzt kannst du die Testausgaben unter Kommentar setzen!

4. Der Client muss mit dem Server kommunizieren. Dafür wird eine Bibliothek benötigt, da es derzeit keine entsprechende Funktionalität in der Standardbibliothek von `C++` gibt. Wir verwenden dafür `asio`. D.h. "installiere" wie gewohnt die header-only Bibliothek `asio`! D.h. anpassen von `meson_options.txt` und `meson.build` ist notwendig.

5. Entwickle jetzt die Funktionalität für einen Daytime-Client auf TCP Basis, der sich zu einem lokalen Daytime-Server (zu Testzwecken kann meine Implementierung in Form der Java-Class-Datei `DaytimeServer.class` verwendet werden).

   Getestet werden kann der Client werden, indem der von mir bereitgestellte Server auf folgende Art und Weise verwendet werden kann:

   ```
   $ java DaytimeServer 1113
   just before accept
   ```

   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 sich mit dem Port 1113 verbinden und dort das daytime Protokoll abwickeln. Die erhaltene Zeitinformation soll auf `stdout` ausgegeben werden. Danach beendet sich der Client.

   Die Ausgabe des Clients sollte jetzt folgendermaßen aussehen:

   Beispiel:

   ```
   $ daytime_client
   Tue Nov 25 12:07:57 CET 2003
   ```

   Eine Fehlerbehandlung wird jetzt noch nicht eingebaut.

   Warum wird nicht der Standardport 13 des daytime Protokolls verwendet?

6. Jetzt bitte ich darum, dass der Client in gewohnter Weise noch mit einer Kommandozeilenschnittstelle bedient werden kann.

   XXX

7. Beende den Server und starte den Client neu. Wenn du absolut keine Fehlerbehandlung eingebaut hast, wird einfach eine Leerzeile ausgegeben werden. Warum?

Erweitere den Client, dass jetzt sowohl das korrekte Aufbauen der Verbindung überprüft wird. Kann keine Verbindung aufgebaut werden, dann soll eine Fehlermeldung auf `stderr` ausgegeben werden.

Das sollte dann in etwa so aussehen:

```
Could not connect to server!
```

8. Ok, das ist gut, aber unter Umständen interessieren uns auch noch weitere Informationen, die in weiterer Folge *geloggt* werden sollen. In komplexeren Programmen und speziell in Netzwerkprogrammen muss immer wieder geloggt werden. Dazu ist es sinnvoll sich eine entsprechende Unterstützung zu organisieren. Wir werden dazu die Header-only Bibliothek `spdlog` verwenden.

   Auch diese Bibliothek wird von mir in Form eines Archives am ifssh zur Verfügung gestellt. Die Verwendung ist analog zu `asio` (im speziellen ist auch diese Bibliothek **nicht** im Abgabeordner abzulegen!)

   `spdlog.cpp` aus dem Template enthält dafür auch Code, der zeigt wie man damit auf primitiver Ebene umgehen kann.

   Für die gesamte Dokumentation siehe spdlog!

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

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

   Der Server soll vorerst nur einmal eine Verbindung akzeptieren (Port 1113) und danach die aktuelle Zeit ermitteln und als Zeile an den Client zurücksenden. Fertig.

   ```
   $ daytime_client
   2016-01-13 23:27:07
   ```

   Für die Übertragung der aktuellen Zeit kann wiederum die von mir bereitgestellte Headerdatei `timeutils.h` verwendet werden oder – noch einfacher – die mittels `system_clock::now()` ermittelte Zeit in den Ausgabestrom (vom Typ `tcp::iostream`) geschoben (also mit `<<`) werden.

   Der Server wird sich danach beendet haben.

10. Erweitere den Server, sodass sich dieser nicht mehr beendet, sondern nach erfolgter Bearbeitung der Anfrage wieder bereit für eine erneute Verbindung ist.

11. Erweiterung um Verarbeitung von Kommandozeilenparametern. Dazu ist entweder die header-only Bibliothek `clipp` oder die header-only Bibliothek `CLI11` zu verwenden! Wie "gewohnt" stelle ich auch hierfür eine Version zur Verfügung, die an einem entsprechenden Ort zu entpacken ist.

- Der Port an dem der Server lauscht *muss* als Kommandozeilenparameter übergeben werden. Eine Hilfe soll *optional* angefordert werden können.

  Beispielausgabe sollte in etwa folgendermaßen aussehen (kann auch mit CLII11 realisiert werden):

  ```
  $ daytime_server
  SYNOPSIS
          daytime_server -p <port> [-h]

  OPTIONS
          <port>      server port
          -h, --help  help
  ```

- Weiters soll in diesem Zusammenhang auch der Client angepasst werden, dass dieser als Kommandozeilenparameter *optional* den Port akzeptiert. Wird kein Port angegeben, dann soll 1113 zum Einsatz kommen. Eine Hilfe soll optional angefordert werden können.

  Beispiel (kann auch mit CLI11 realisiert werden):

  ```
  $ daytime_client -h
  SYNOPSIS
          daytime_client [-p <port>] [-h]

  OPTIONS
          <port>      port to connect to
          -h, --help  help
  ```

8. Erweitere jetzt sowohl den Client als auch den Server jetzt dass dieser jetzt auch mit Fehlern umgehen kann. Setze dazu die Möglichkeiten von `spdlog` ein!

   Fehler, die behandelt werden sollen:

   - Client kann sich nicht verbinden
   - Client kann keine Daten einlesen
   - Server kann Verbindung nicht fehlerfrei annehmen
   - Server kann nicht fehlerfrei senden

   Prinzipiell gibt es die Möglichkeit mittels `try/catch` oder mit zusätzlichen Parameter vom Typ `error_code`. Beide Möglichkeiten sind gleichwertig, aber die Lösung mittels `try/catch` für diese Situation besser geeignet, nicht wahr?

4 Übungszweck

- externe header-only Bibliothek ”installieren” und Buildsystem entsprechend konfigurieren
- Socketverbindung zu lokalem Host aufbauen. Übertragung von Zeichendaten (Client-Seite) mittels stream-basierter Kommunikation.
- Server-Socket anlegen und Client-Requests annehmen (blocking, single-threaded). Übertragung von Zeichendaten (Server-Seite) mittels stream-basierter Kommunikation.
- single-threaded Server für beliebig viele Requests.
- Verarbeitung von Kommandozeilenparameter für Server- und Client-Programme.
- Einfachste Fehlerbehandlung kennenlernen.