NVS5 Projektübungen – Probleme

Schuljahr 2020/21

Dr. Günter Kolousek

2021-01-26

Inhaltsverzeichnis

In diesem Dokument habe ich die wichtigsten (aufgetretenen) Probleme zusammengefasst!

1 Build-Prozess

- `meson.build`
  - die erlaubten header-only Bibliotheken sind in einem `include` Ordner (und daher im Repo) und daher fehlt `get_option(...)`.
  - kein vernünftiger Name für Executable, wie z.B. `EXENAME` oder `project1`. Dem Programm kann man doch einen vernünftigen Namen spendieren, der einen Hinweis auf dessen Funktion liefert, das ist man dem Programm einfach schuldig.
  - absolute Pfade verwendet, wie z.B. `/src/main.cpp`. So kann ich nicht übersetzen!
  - Wenn ich das Projekt in Zukunft nicht ohne weitere Intervention builden kann, dann wird es mit *Nicht genügend* beurteilt!

    Zum Beispiel:

    * Kein `include` Verzeichnis angelegt (nach dem clonen), aber in `meson.build` angegeben. Daher wird der Build-Prozess nicht funktionieren.

    * Auf `MacOS` entwickelt und auf Grund von unterschiedlichen Compiler- bzw. Bibliotheksimplementierungen unter Linux nicht ohne weitere Interventionen zu übersetzen.

  Deshalb: Neu (unter Linux) clonen und danach testen!!!

- `meson_options.txt`
  - bitte alle *nicht* verwendeten Optionen herauslöschen.
- Warnungen treten auf und/oder Warnungen wurden ausgeschaltet

  Bei selbst entwickelten Code darf keine Warnung auftreten und bei verwendeten Bibliotheken ist folgendes Idiom einzusetzen:

```
//ignore warning "-Wnon-virtual-dtor" from extern library "tabulate"
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wnon-virtual-dtor"
#include "tabulate.hpp"
#pragma GCC diagnostic pop
```

  Damit wird dem Compiler über die `#pragma` Präprozessordirektive mitgeteilt, dass die Warnung `non-virtual-dtor` ignoriert werden soll. Und zwar nur für die Headerdatei `tabulate.hpp`.
- *falsche* Include-Direktiven, d.h.: `#include "../include/xxx.h"`

2 Repo & github

- Dokumentation nicht in einem eigenen Dokumentationsverzeichnis. Meine präferierte Namensgebung ist `doc`.
- generierte Dateien sind im Repo!!! z.B. im Ordner `doc` Dateien mit der Endung `.aux`,... (außer `.pdf`!). D.h. bitte die Datei `.gitignore` von Anfang an warten!!!
- `build` Ordner fehlt oder ist nicht leer. Leere Verzeichnisse sind in den vielen Versionsverwaltungssystemen nicht möglich, daher bitte darin (also in `build`) eine eine leere Datei `.gitkeep` oder `.keep` anlegen.
- Verzeichnisse und Dateien von IDEs sind im Repo wie z.B. von VSCode. Unter Linux kann man sich auch die versteckten Dateien im aktuellen Verzeichnis mit `ls -a` anzeigen lassen.
- "Meine" Dateien wie z.B. `spdlog.cpp` aus dem Template sind im Repo vorhanden. Bitte löschen, benötigt werden!
- Keine Issues vorhanden, d.h. Entwicklungsprozess ist extrem fraglich...
- Commit-Messages: *Form* und *Anzahl* mangelhaft. Es handelt sich um ein Projekt, das das Ziel hat, Sie in die Entwicklung von verteilten Systemen einzuführen. Eine Entwicklung sollte so stattfinden, dass im Großen und Ganzen jede zusammenhängende (funktionale) Änderung als ein Commit im Repository abgelegt werden soll. Die Commit-

Nachrichten sollten sinnvoll sein und gleichartig aufgebaut (siehe `revision_control_mercurial.p` Folien 35 bis 37).

- Groß/Kleinschreibung bei Dateinamen nicht beachtet oder Leerzeichen in Dateinamen enthalten. Z.B.: `Meine neuen Tasks.txt` anstatt `meine_neuen_tasks.txt`.

  Ja, ich weiß, dass diese Konventionen nicht für jede Person die optimale Form darstellt, aber hier gebe ich die Vorgaben an und in einer Firma ist es eben die Firma...

  Hintergrund für diese Vorgaben:

  - Auf der Kommandozeile unter Betriebssystemen mit Unterscheidung der Groß- und Kleinschreibung, ist die Bedienung einfacher.
  - Keine Leerzeichen, weil erstens wieder leichter einzugeben und es noch immer Programme gibt, die mit Leerzeichen in Dateinamen nicht zurechtkommen.

3 Source-Code

- `UTF-8` zur Zeichenkodierung *nicht* verwendet. D.h. Editor oder IDE entsprechend konfigurieren.
- `NL` als Zeilentrennzeichen nicht korrekt eingestellt. Tritt speziell dann auf, wenn nicht unter Linux entwickelt wird. Dann ist wieder der Editor bzw. die IDE entsprechend zu konfigurieren.
- Coding Conventions nicht eingehalten
  - Zeilenlänge grob nicht eingehalten!
  - Position von {, (,..., wie z.B.:

```
void check_counter ()
{
    if(empty ){
        // do something
    }
}
```

  - Namensgebung der Bezeichner unsinnig oder nicht entsprechend der Coding Conventions, wie z.B.:

```
void f() {
    bool x;
    int Counter{};
    // ...
}
```

4 Ein/Ausgaben

- Aus den Ausgaben lässt sich das Verhalten des Programmes nicht oder nur schwer nachvollziehen.
- Ausgaben nicht synchronisiert.
- `spdlog` nicht verwendet: das ist von meiner Seite ein ”muss”!!!!
  - sollte so verwendet werden, dass man das Programm zusätzlich zu den normalen Ausgaben besser **nachvollziehen** kann bzw. Debug-Ausgaben zu sehen sind. Logging ist speziell in verteilten System (aber nicht nur dort) extrem wichtig. Man möchte wissen was, wann und wo passiert ist, damit man das Funktionieren oder Nicht-Funktionieren nachvollziehen kann. Auch aus Sicherheitsaspekten ist Logging ein wichtiges Tool!
- `CLI11`
  - nicht ausreichend eingesetzt, z.B. zum Steuern des Loglevels (siehe `spdlog`). Speziell für Serverprogramme ist die Steuerbarkeit beim Aufruf und auch eine flexible Konfiguration mittels Umgebungsvariable, Konfigurationsdateien und Kommandozeilenparameter essentiell.
  - Langoption mit einem Buchstaben verwendet, z.B. `--k`

5 Dokumentation

- Sourcedateien wie `.tex` fehlt. Ja, ich weiß, dass man aus den Sourcedateien ein `.pdf` generieren kann, aber hier möchte ich das `.pdf` bitte auch im Repo haben.
- Struktur und Umfang nicht ausreichend.
- zu viele Rechtschreibfehler vorhanden (wahrscheinlich nie durchgelesen).

6 Programmierstil und Programmierfehler

- Speicher nicht freigegeben, d.h. `delete` fehlt
- `condition_variable` falsch verwendet
- Race Conditions treten auf
- Fehlerbehandlung fehlt!!!
- Spaghetticode vorhanden
- globale Variable verwendet
  - wenn es schon nicht geht, dann wenigstens lokale Variable in `main()`
- Initializer-list bei Klassen nicht verwendet