

Vertiefung Software-Entwicklung in C++

Prof. Dr. rer. nat. Peter Gerwinski

14. Januar 2019

Vertiefung Software-Entwicklung in C++

<https://gitlab.cvh-server.de/pgerwinski/cpp.git>

- 1 Einführung
- 2 Wiederholung: Programmieren in C
- 3 Einführung in C++
- 4 Standard-Bibliotheken (STL)
- 5 C++11
 - ...
 - 5.4 R-Wert-Referenztypen
 - 5.5 Lambda-Ausdrücke
- 6 Die Boost-Bibliothek
 - 6.1 C++11-Erweiterungen für ältere Compiler
 - 6.2 Spracherweiterungen
 - 6.3 Zeitangaben
 - 6.4 Interprozeßkommunikation
 - 6.5 Asynchroner Input/Output
- 7 Plug-In-Architekturen



Änderungen
vorbehalten

5 C++11

5.5 Lambda-Ausdrücke

Übergabe von Funktionszeigern

```
int is_smaller (int a, int b)
{
    return a < b;
}
```

```
sort (numbers.begin (), numbers.end (), is_smaller);
```

Stattdessen: *Lambda-Ausdrücke*

6 Die Boost-Bibliothek

Überblick: <http://dieboostcppbibliotheken.de/>

6.1 C++11-Erweiterungen für ältere Compiler

C++11

```
for (auto &i : container)
[](int i){ cout << i << endl; }
#include <cstdlib>
#include <random>
...
```

Boost

```
BOOST_FOREACH (int &i, container)
cout << boost::lambda::_1 << "\n"
#include <boost/cstdint.hpp>
#include <boost/random.hpp>
...
```

6 Die Boost-Bibliothek

6.2 Spracherweiterungen

- „Duck Typing“:
`boost::any`

6 Die Boost-Bibliothek

6.2 Spracherweiterungen

- „Duck Typing“:
`boost::any`
- Coroutinen:
kooperatives Multitasking (Alternative zu Threads)
anhalten und Wert-Rückgabe über `push_type`-Objekt,
aufrufen und Wert-Abfrage über `pull_type`-Objekt

6 Die Boost-Bibliothek

6.2 Spracherweiterungen

- „Duck Typing“:
`boost::any`
- Coroutinen:
kooperatives Multitasking (Alternative zu Threads)
anhalten und Wert-Rückgabe über `push_type`-Objekt,
aufrufen und Wert-Abfrage über `pull_type`-Objekt
- Parameter-Namen:
`BOOST_PARAMETER_NAME()`, `BOOST_PARAMETER_FUNCTION()`

6 Die Boost-Bibliothek

6.3 Zeitangaben

- Kalender (Schaltjahr-Problematik usw.)
- formatierte Ein-/Ausgabe
- Zeitzonen
- Zeitmessungen
- Messung der Code-Ausführungsgeschwindigkeit
- betriebssystemunabhängiges Warten:
`this_thread::sleep_for (chrono::milliseconds (137));`

6 Die Boost-Bibliothek

6.4 Interprozeßkommunikation

- Gemeinsamer Speicher: *Shared Memory*
- Kernel verwaltet benannte Speicherbereiche
- Existenz persistent, unabhängig von Prozessen
- Spezialfall von *Mapped Region*
- in C (Unix): `shm_open()`, `mmap()`, Bibliothek: `-lrt`
- in C++ (Boost): betriebssystemunabhängig
- ... außer, man verwendet MS-Windows-spezifischen Shared Memory, der beim Beenden des Prozesses automatisch entfernt wird

6 Die Boost-Bibliothek

6.4 Interprozeßkommunikation

- Verwalteter gemeinsamer Speicher: *Managed Shared Memory*
- Ein C++-map-Container verwaltet benannte Variable innerhalb des Shared Memory.
- erzeugen mit `construct<>`, verwenden mit `find<>`, beides mit `find_or_construct<>`
- wieder entfernen mit `destroy<>`, `destroy_ptr<>`
- Speicherüberlauf: `bad_alloc`-Exception

6 Die Boost-Bibliothek

6.4 Interprozeßkommunikation

- Verwalteter gemeinsamer Speicher: *Managed Shared Memory*
- Ein C++-`map`-Container verwaltet benannte Variable innerhalb des Shared Memory.
- erzeugen mit `construct<>`, verwenden mit `find<>`, beides mit `find_or_construct<>`
- wieder entfernen mit `destroy<>`, `destroy_ptr<>`
- Speicherüberlauf: `bad_alloc`-Exception
- spezielle Typen: `boost::interprocess::string` und Container (enthalten Zeiger → müssen auf Shared Memory verweisen)

6 Die Boost-Bibliothek

6.4 Interprozeßkommunikation

- Verwalteter gemeinsamer Speicher: *Managed Shared Memory*
- Ein C++-`map`-Container verwaltet benannte Variable innerhalb des Shared Memory.
- erzeugen mit `construct<>`, verwenden mit `find<>`, beides mit `find_or_construct<>`
- wieder entfernen mit `destroy<>`, `destroy_ptr<>`
- Speicherüberlauf: `bad_alloc`-Exception
- spezielle Typen: `boost::interprocess::string` und Container (enthalten Zeiger → müssen auf Shared Memory verweisen)
- Synchronisation: *Mutex*-Variable `interprocess_mutex`, `named_mutex`

6 Die Boost-Bibliothek

6.5 Asynchroner Input/Output

- serielle Schnittstelle, TCP/IP, ...
aber auch: Timer
- mit Callbacks
statt mit blockendem I/O und `select()`
- Timer: Zeitpunkt, Wartezeit
- TCP/IP: Resolver, Verbindungsaufbau, Daten
- statt Schleife: Callback re-installiert sich selbst

Vertiefung Software-Entwicklung in C++

<https://gitlab.cvh-server.de/pgerwinski/cpp.git>

- 1 Einführung
- 2 Wiederholung: Programmieren in C
- 3 Einführung in C++
- 4 Standard-Bibliotheken (STL)
- 5 C++11
 - ...
 - 5.4 R-Wert-Referenztypen
 - 5.5 Lambda-Ausdrücke
- 6 Die Boost-Bibliothek
 - 6.1 C++11-Erweiterungen für ältere Compiler
 - 6.2 Spracherweiterungen
 - 6.3 Zeitangaben
 - 6.4 Interprozeßkommunikation
 - 6.5 Asynchroner Input/Output
- 7 Plug-In-Architekturen



Änderungen
vorbehalten