Tutorium

Grundlagenpraktikum: Rechenarchitektur (IN0005)

Lukas Ketzer

Lehrstuhl für Rechnerarchitektur & Parallele Systeme (Prof. Schulz)
TUM School of Computation, Information and Technology
Technische Universität München

22. Mai 2024

Allgemeines

- System-Design-Zweig
- Meine Tutorien:
  - Gruppe 18: Do. 10:00 - 12:00 (00.08.059)
  - Gruppe 25: Do. 16:00 - 18:00 (01.06.011)
- Kommunikation
  - Zulip: öffentliche GRA-Streams
  - Zulip DMs: Lukas Ketzer (nur in dringenden Fällen)
  - lukas.a.ketzer@gmail.com (nur in dringenden Fällen)

Tutoriums-Webseite

Abbildung 1 home.in.tum.de/ketz/

Zulip

Abbildung 2 Zulip Gruppe 18 (Do. 10:00 - 12:00

Abbildung 3 Zulip Gruppe 25 (Do. 16:00 - 18:00

Recap

- Valgrind
  - Advanced Tool um den Speicher zu kontrollieren
  - `usage: valgrind ./bin`
  - Am besten mit `-g` compilen, für mehr info
- Makefile
  - Praktische Files, in denen Compile-Befehle stehen
  - **Targest** mit verschiedenen Konfigureationen können definiert werden
- File IO
  - Dateien öffnen, lesen und schreiben
  - `fopen` gibt einen Pointer zurück -> **Null-Pointer-Check**

Systemdesign

Agenda für heute

- C++ und System-C installieren
- Einführung in C++
- Einführung in System-C

System C und C++ installieren

- Kein Windows
- Nur WSL, MacOS (brew) oder Linux

C++ Einführung: Konstruktoren

Bisher

```
class Player {
public:
    char* m_name;
    int m_health;
    int m_strength;
    Player(char* name, int health, int strength) {
        this->m_name = name;
        this->m_health = health;
        this->m_strength = strength;
    }
};
```

C++ Einführung: Konstruktoren

Neu

```
class Player {
public:
    char* m_name;
    int m_health;
    int m_strength;
    Player(char* name, int health, int strength):
        m_name(name),
        m_health(health),
        m_strength(strength)
    {}
};
```

C++ Einführung: Referenzen

Bisher

```
1  class Player {
2  public:
3      int health;
4      Player(int health): health(health) {}
5  };
6  void attack(Player *p) {
7      p->health--;
8  }
9
10 int main() {
11     Player p = Player{5};
12     attack(&p);
13 }
```

C++ Einführung: Referenzen

Neu (und besser)

```
1  class Player {
2  public:
3      int health;
4      Player(int health): health(health) {}
5  };
6  void attack(Player &p) {
7      p.health--;
8  }
9
10 int main() {
11     Player p = Player{5};
12     attack(p);
13 }
```

Aufgabe 1: Pets

Die Vorlage beinhaltet ein simples C++ Programm mit dem Struct

```
struct Pet {
    const char* name;
    const char* species;
    int age;
};
```

Implementiere einen Constructor, damit eine Instanz des Structs folgendermaßen erstellt werden kann:

```
Pet myDog("Snoopy", "Dog", 74);
```

Implementiere außerdem eine Methode info(), die beim Aufrufen beispielsweise folgenden Text ausgibt:

```
My name is Snoopy, I am a Dog and I am 74 years old.
```

Aufgabe 1: Pets (Lösung)

```
1
2  #include <iostream>
3
4  struct Pet {
5    /* attrs */
6    Pet(const char* name, const char* species, int age):
7        name{name},
8        species{species},
9        age{age}
10     {}
11     void info() {
12         std::cout << "My name is " << this->name << " I am a " <<
     this->species << " and I am " << this->age << " years old." <<
     std::endl;
13     }
14 };
```

Einführung SystemC: Installation

- Systemdesign > Lectures > Einführung C++ und SystemC > Makefile
- Bevorzugten Ordner für SystemC Projekte auswählen
- `make` ausführen

Erwartete Orderstruktur

```
workspace
    systemc
        ...
    aufgabe-1
        main.cpp
        Makefile
    aufgabe-2
        main.cpp
        Makefile
```

Aufgabe 2: sc_bv

Installiere SystemC wie es im Vorlesungsmaterial beschrieben wurde.
Erweitere dann das Grundgerüst des gegebenen Codes, indem du ein Signal mit dem Typ `sc_bv<8>` definierst. Dieser Typ stellt einen Bit-Vektor aus 8 Bits dar.
Schreibe dann den Wert `99` in das Signal. Starte die Simulation, lies das Signal aus, und gib den Ausgabewert in der Konsole aus.
Tipp: Bit-Vektoren werden verwendet, um Zahlen als Strings aus Bits darzustellen. Dabei kann auf die individuellen Bits einzeln zugegriffen werden, oder der Vektor wird als Ganzes beschrieben oder gelesen. Wir haben `sc_bv<n>` noch nicht behandelt. Versuche also, in der Dokumentation von SystemC eine beispielhafte Verwendung des Datentyps zu finden.

Aufgabe 2: sc_bv - Lösung

```
#include <systemc>
#include "systemc.h"
using namespace sc_core;

int sc_main(int argc, char* argv[]) {
  sc_signal<sc_bv<8>> signal;
  signal.write(99);
  sc_start();
  std::cout << signal.read() << std::endl;
  return 0;
}
```

Vielen Dank für euer Aufmerksamkeit!