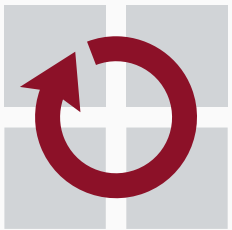
Übungen zu Systemprogrammierung 1

Ü1 – Speicherverwaltung

Sommersemester 2025

Luis Gerhorst, Thomas Preisner, Tobias Häberlein, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Agenda

2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden

Agenda

2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden

Portable Programme

- Entwicklung portabler Programme durch Verwendung definierter Schnittstellen

ANSI C11

- Normierung des Sprachumfangs der Programmiersprache C
- Standard-Bibliotheksfunktionen, z. B. `printf`, `malloc`

Single UNIX Specification, Version 4 (SUSv4)

- Standardisierung der Betriebssystemschnittstelle
- Wird von verschiedenen Betriebssystemen implementiert:
  - Solaris, HP/UX, AIX *(SUSv3)*
  - Mac OS X *(SUSv3)*
  - … und natürlich Linux *(SUSv4, aber nicht offiziell zertifiziert)*

… und was ist mit Windows?

ANSI C11

- Von Microsoft Visual C/C++ nicht unterstützt :-(
- GCC läuft auch unter Windows :-)

Single UNIX Specification, Version 4 (SUSv4)

- Von Microsoft Windows nicht unterstützt :-(
- UNIX-Kompatibilitätsschicht für Windows: Cygwin (`http://cygwin.com/`)
  - … ist aber eher frickelig :-|

- Ab Windows 10 (seit 1607): Windows Subsystem for Linux (WSL)
  - Damit vollwertiges Linux unter Windows verfügbar
  - Aber: nicht von uns unterstützt
  - CIP ist weiterhin Referenzsystem!

Agenda

Anforderungen an abgegeben Lösungen

- C-Sprachumfang konform zu ANSI C11
- Betriebssystemschnittstelle konform zu SUSv4
- `gcc` übersetzt Abgabe **warnungs-** und **fehlerfrei** im CIP-Pool mit folgenden Optionen:

```
-std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror
         -fsanitize=undefined -fno-sanitize-recover -g
```

  - `-std=c11 -pedantic`: nur ANSI-C11-konformer C-Quellcode
  - `-D_XOPEN_SOURCE=700`: nur SUSv4-konforme Betriebssystemaufrufe
- **Korrekte** Fehlerbehandlung (dazu später mehr)
- Keine globalen Variablen, `static` Variablen nur falls nötig
- Korrekturrichtlinien beachten!

  `https://sys.cs.fau.de/lehre/current/sp1/uebung#korrektur`

Agenda

Compiler und Optionen

- Übersetzen einer Quelldatei mit `gcc`:

```
gcc -o test test.c
```

  - Zur Erinnerung: Starten der ausführbaren Datei `test` mit `./test`

- Verhalten des `gcc` kann durch Optionen beeinflusst werden

`-g`	Erzeugt Debug-Symbole in der ausführbaren Datei
`-c`	Übersetzt Quellcode in Maschinencode, erzeugt aber kein ausführbares Programm
`-Wall`	aktiviert weitere Warnungen, die auf mögliche Programmierfehler hinweisen
`-Werror`	`gcc` behandelt Warnungen wie Fehler

Demo: Gängige Compiler-Warnungen

```
student@cip:~ $ gcc -o test test.c
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

- Bibliotheksfunktion: Entsprechendes `#include` fehlt.
  Manual-Page gibt Auskunft über Namen der nötigen Headerdateien:

```
$ man 3p printf
PRINTF(3POSIX)        POSIX Programmer's Manual        PRINTF(3POSIX)
NAME
printf - print formatted output
SYNOPSIS
#include <stdio.h>
int printf(const char *restrict format, ...);
```

- eigene Funktion: Forward-Deklaration fehlt

```
test.c: In function 'foo':
test.c:3:1: warning: control reaches end of non-void function
```

- in der Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung

UndefinedBehaviorSanitizer

Undefined Behavior Sanitizer (UBSan)

UBSan modifiziert Programme zur Übersetzungszeit, um verschiedene Arten von undefiniertem Verhalten zur Laufzeit zu erkennen.

- Instrumentierung wird automatisch durch Übersetzer eingefügt
- zusätzliche Laufzeitbibliothek wird beim Binden eingefügt
- Verschiedene Behandlungsoptionen:
  - `-fsanitize-recover=undefined` (default):
    Fehler wird ausgegeben, Ausführung wird fortgesetzt
  - `-fnosanitize-recover=undefined`:
    Fehler wird ausgegeben, Ausführung wird abgebrochen
  - `-fsanitize-trap=undefined`:
    keine Fehlerausgabe, Ausführung wird abgebrochen,
    Laufzeitbibliothek wird nicht benötigt
- **nicht alle Fehler können vom UBSan erkannt werden**

UndefinedBehaviorSanitizer - Beispiel

```c
int main(void) {
  int values[] = { 13, 37 };
  for (int i = 0; i <= 2; i++) { printf("Value %d: %d\n", i, values[i]); }
}
```

Ohne UBSan:

```
student@cip:~ $ gcc test.c -g -o test && ./test
Value 0: 13
Value 1: 37
Value 2: 2
```

Mit UBSan:

```
student@cip:~ $ gcc test.c -fsanitize=undefined -g -o test && ./test
Value 0: 13
Value 1: 37
test.c:6:37: runtime error: index 2 out of bounds for type 'int [2]'
test.c:6:3: runtime error: load of address 0x7fff5f1fc39c with insufficient
     space for an object of type 'int'
0x7fff5f1fc39c: note: pointer points here
  25 00 00 00 02 00 00 00  60 02 06 e6 c8 7f 00 00  c8 c4 1f 5f ff 7f 0...
             ^
Value 2: 2
```

Agenda

Verkettete Liste

- Wann setzt man eine verkettete Liste ein?
- Anforderungsanalyse für verkettete Liste
  - Wieviele Listenelemente gibt es maximal?
  - Welche Lebensdauer muss ein Listenelement besitzen?
  - In welchem Kontext muss ein Listenelement sichtbar sein?
- Wir brauchen einen Mechanismus, mit dem Listenelemente...
  - ... in a-priori nicht bekannter Anzahl
  - ... zur Laufzeit des Programmes erzeugt und zerstört werden können

Warum geht das nicht auf dem Stack?

$\rightarrow$ vgl. Vorlesung A-II-50, A-II-85

Dynamische Speicherverwaltung

- Beispiel in Java: Einfache verkettete Liste

```
ListElement first = new ListElement("first element");
ListElement second = new ListElement("second element");
first.enqueue(second);
```

  - In Java: Neues Listenelement wird mit Hilfe von new instanziiert
    - ⇒ Reservieren eines Speicherbereiches für das Objekt
    - ⇒ Initialisieren des Objektes durch Ausführen des Konstruktors

- In C: Anlegen eines Listenelementes mittels `malloc(3p)`

```
struct listelement *newElement;
newElement = malloc(sizeof(*newElement));
if (newElement == NULL) {
    // Fehlerbehandlung
}
```

  - Zurückgegebener Speicher hat undefinierten/zufälligen Wert
    - ⇒ Inititialisierung muss per Hand erfolgen

Dynamische Speicherverwaltung

- Explizite Initialisierung mit definiertem Wert: `memset(3p)`

```
memset(newElement, 0, sizeof(struct listelement));
```

- Mit `0` vorinitialisierter Speicher: `calloc(3p)`

```
struct listelement *newElement;
newElement = calloc(1, sizeof(*newElement));
if (newElement == NULL) { /* Fehler */ }
```

- Im Gegensatz zu Java kein Garbage-Collection-Mechanismus in C
  - Speicherbereich muss von Hand mittels `free(3p)` freigegeben werden
  - Nur Speicher, der mit `malloc(3p)`, `calloc(3p)` oder `realloc(3p)` angefordert wurde, darf mit `free(3p)` freigegeben werden!
  - Zugriff auf freigegebenen Speicherbereich ist undefiniert

Agenda

Einfach verkettete FIFO-Liste

- Zielsetzungen
  - Kennenlernen der Umgebung und Entwicklungswerkzeuge
  - Dynamische Speicherverwaltung und Umgang mit Zeigern
  - Verwendung des Abgabesystems
- Strukturdefinition

```c
struct listelement {
  int value;
  struct listelement *next;
};
typedef struct listelement listelement; // optional
```

Schnittstelle

- Nur folgende Funktionen zu implementieren
  - `insertElement()`: Fügt einen neuen, nicht-negativen Wert in die Liste ein, wenn dieser noch nicht vorhanden ist. Tritt ein Fehler auf, wird -1 zurückgegeben. Ansonsten wird der eingefügte Wert zurückgegeben.
  - `removeElement()`: Entfernt den ältesten Wert in der Liste und gibt diesen zurück. Ist die Liste leer, wird -1 zurückgeliefert.
- Keine Listen-Funktionalität in der `main()`-Funktion
  - Allerdings: Erweitern der `main()` zum Testen erlaubt und **erwünscht**
- Sollte bei der Ausführung einer verwendeten Funktion (z. B. `malloc(3p)`) ein Fehler auftreten, sind keine Fehlermeldungen auszugeben. Im Fehlerfall wird -1 zurück gegeben.

Agenda

2.1 Portable Programme

2.2 Anforderungen an Abgaben

2.3 Übersetzen von Programmen

3.1 Verkettete Listen

4.1 Aufgabe 1: lilo

5.1 Gelerntes anwenden

Aktive Mitarbeit erforderlich!

„Aufgabenstellung“

- Programm schreiben, welches ähnlich wie `echo` funktioniert
  - Alle Argumente werden ausgegeben
  - Jedes Zeichen in einem Argument wird auf eigener Zeile ausgegeben
- Programme übersetzen