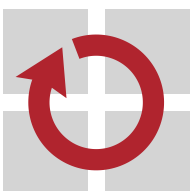
Übungen zu Systemprogrammierung 1

Ü1-2 – Speicherverwaltung

Sommersemester 2018

Christian Eichler, Jürgen Kleinöder

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 Gelerntes anwenden

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 Gelerntes anwenden

Compiler und Optionen

- Übersetzen einer Quelldatei mit `gcc`:

  ```
  user@host:~$ 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

Gängige Compiler-Warnungen

```
chris@legio:~$ 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 den Namen der nötigen Headerdateien:

  ```
  $ man 3 printf
  SYNOPSIS
    #include <stdio.h>

    int printf(const char *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

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 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 :-|

- Neu in Windows 10 als **Beta**: **unfertiges** Ubuntu-Subsystem
- Ihr wollt eure SP-Programme unter Linux entwickeln. Wirklich.

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 Gelerntes anwenden

Anforderungen an abgegeben Lösungen

- C-Sprachumfang konform zu ANSI C11
- Betriebssystemschnittstelle konform zu SUSv4
- **warnungs-** und **fehlerfrei** im CIP-Pool mit folgenden `gcc`-Optionen übersetzen:
  `-std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror`
  - `-std=c11 -pedantic` erlauben nur ANSI-C11-konformen C-Quellcode
  - `-D_XOPEN_SOURCE=700` erlaubt nur SUSv4-konforme Betriebssystemaufrufe

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 Gelerntes anwenden

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

Dynamische Speicherverwaltung

- Beispiel in Java: Aufgabe 10.1 (Menschenkette) in AuD SS 2011

```
WaitingHuman somebody = new WaitingHuman("Mrs. Somebody");
WaitingHuman nobody = new WaitingHuman("Mr. Nobody");

somebody.add(nobody);
```

  - 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(3)`

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

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

Dynamische Speicherverwaltung

- Explizite Initialisierung mit definiertem Wert: `memset(3)`
  ```
  memset(newElement, 0, sizeof(struct listelement));
  ```
- Mit `0` vorinitialisierter Speicher kann mit `calloc(3)` angefordert werden
  ```
  struct listelement *newElement;
  newElement = calloc( 1, sizeof(struct listelement) );
  if ( newElement == NULL ) { /* Fehler */ }
  ```
- Im Gegensatz zu Java gibt es in C keinen Garbage-Collection-Mechanismus
  - Speicherbereich muss von Hand mittels `free(3)` freigegeben werden
  - Nur Speicher, der mit einer der Funktionen `malloc(3)`, `calloc(3)` oder `realloc(3)` angefordert wurde, darf mit `free(3)` freigegeben werden!
  - Zugriff auf freigegebenen Speicherbereich ist undefiniert

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 Gelerntes anwenden

Einfach verkettete FIFO-Liste

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

```
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(3)`) ein Fehler auftreten, sind keine Fehlermeldungen auszugeben.

Agenda

1.1 Übersetzen von Programmen

1.2 Portable Programme

1.3 Anforderungen an Abgaben

1.4 Verkettete Listen

1.5 Aufgabe 1: lilo

1.6 Gelerntes anwenden

Aktive Mitarbeit erforderlich!

„Aufgabenstellung“

- Optional: Programm schreiben, welches „Hallo Welt!“ ausgibt
- Sieb des Eratosthenes implementieren
  - Erstes Argument gibt an, bis zu welcher Zahl geprüft werden soll
- Programme übersetzen