8. Übung

Ausgabe	Abgabe	Diskussion
06.12.13	13.12.13	16.12.-20.12.13

Bitte bei der Abgabe Name/Matr.Nr. der Mitglieder einer Gruppe, Nummer der Übung/Teilaufgabe und Datum auf den Lösungsblättern nicht vergessen! Darauf achten, dass die Lösungen beim richtigen Tutor abgegeben werden. Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Dateisysteme (2+2=4 Punkte)

In einem hierarchischen Dateisystem werde freier Festplattenplatz in einer Liste verwaltet.

a) Der Zeiger auf die Freispeicherliste sei verloren gegangen. Kann das System die Freispeicherliste wiederherstellen und wenn ja, wie?

b) Schlagen Sie ein Verfahren vor, bei dem der Zeiger nicht aufgrund eines einzigen Speicherfehlers verloren gehen kann.

Aufgabe 2: Dateisysteme (3+2=5 Punkte)

Bei dieser Aufgaben geht es um das FAT16 Dateisystem.

a) Beschreiben Sie kurz das FAT16 Dateisystem.

b) Erklären Sie genau, warum bei einem FAT16 Dateisystem im Wurzelverzeichnis nur eine begrenzte Anzahl Dateien stehen darf, in einem Unterverzeichnis aber fast beliebig viele. Wie viele Dateien passen maximal in ein Unterverzeichnis? Argumentieren sie anhand der FAT16 Spezifikation und nicht durch Suchen der Lösungen im Internet!

Aufgabe 3: Buffer Overflow (2+3* Punkte)

Beschreiben Sie was unter einem Buffer Overflow zu verstehen ist und wie dieser genutzt werden kann, um die Programmausführung zu beeinflussen.

*Bonusaufgabe: Geben Sie ein kleines Beispielprogramm zu einem Buffer Overflow an.

Aufgabe 4: C-Programmierung, der C-Präprozessor (1+1+1+1+1=5 Punkte)

Bevor der C-Compiler einen Programmtext kompiliert, durchläuft ein *Präprozessor* den Quelltext. Anweisungen für den Präprozessor werden stets in eine eigene Zeile geschrieben und mit dem Zeichen „#“ eingeführt. Die einfache Anweisung

```
#include "filename"
```

haben Sie bereits kennengelernt, sie fügt die angegebene Datei an der Stelle der Anweisung ein. Eine weitere sehr nützliche Präprozessoranweisung ist

```
#define symbol entsprechung
```

Wird ein Symbol definiert, ersetzt der Präprozessor jedes Vorkommen von `symbol` durch `entsprechung` (nur im Quelltext, d.h. nicht in Kommentaren und nicht in Zeichenketten). Der Text der eingesetzt wird ist durch das Zeilenende nach der `#define`-Anweisung begrenzt. Ist also in einem Quelltext z.B. die Anweisung:

```
#define MAX 100
```

enthalten, wird bei jedem Vorkommen von `MAX` die Zeichenkette `100` eingesetzt.

Mittels `#define` ist es auch möglich, Makros zu definieren. In diesem Fall hat das Symbol das Aussehen einer Funktion, d.h. in Klammern wird eine Liste von Parametern angegeben. Einen Datentyp haben diese jedoch nicht, da nur eine textuelle Ersetzung stattfindet!

Folgende Beispiele verdeutlichen die Nutzung von Parametern:

```
#define SUMME(a,b) a+b   /* Summe aus den Parametern berechnen */
#define square(x) x*x    /* Quadrat von x                      */

/* Makro zur Ausgabe eines Integers.                           */
/* ACHTUNG: keine TypprÃ¼fung, da Textersetzung!                */
#define PRINT_INTEGER(i) printf("%d",i);
```

Ob ein Symbol definiert wurde oder nicht, kann mittels Präprozessoranweisung abgefragt werden. Daraus ergeben sich Möglichkeiten für eine bedingte Übersetzung des Source-Codes. Die einzelnen Anweisungen bilden dabei auch die Begrenzung für den zu kompilierenden bzw. nicht zu kompilierenden Code:

```
#ifdef symbol          -> wenn symbol definiert wurde (egal wie)
... code               -> Code, der in diesem Fall uebersetzt wird
#endif                 -> Ende der bedingten Kompilierung
```

Statt `#endif` kann auch `#else` stehen. Dann folgt ein Block, der kompiliert wird, wenn `symbol` nicht definiert ist. Der Block nach `#else` wird dann mit `#endif` beendet. Neben `#ifdef` für „ist definiert“ kann auch `#ifndef` verwendet werden („ist nicht definiert“). Eine Schachtelung ist zulässig. Verwendet wird die bedingte Kompilierung z.B. zum Ausschluss von Testausgaben aus fertigen Programmen.

Der C-Compiler selber (also nicht der Präprozessor!) kennt noch die Ausdrucksform der bedingten Auswertung. Die Syntax der bedingten Auswertung hat folgendes Aussehen, wobei `e1,e2` und `e3` Ausdrücke sind:

```
e1 ? e2 : e3
```

Wird ein solches Konstrukt durchlaufen, so findet zunächst eine Auswertung von `e1` statt. Wird `e1` als wahr angenommen (`e1` ungleich 0), wird der Ausdruck `e2` ausgewertet und ist das Ergebnis der bedingten Auswertung. Ist `e1` unwahr (also gleich 0), wird `e3` ausgewertet und als Resultat genutzt.

Beispiel: Das Maximum zweier Zahlen `a` und `b` soll einer Variable `z` zugewiesen werden. Ohne bedingte Auswertung sieht das so aus:

```
if (a>b)
    z=a;
else
    z=b;
```

Mit bedingter Auswertung:

```
z = (a>b) ? a : b;
```

Laden Sie von der Webseite zur Vorlesung die Datei `u8_4.c` herunter und testen Sie das darin enthaltene Programm.

a) Warum funktioniert `square(1+1)` nicht?

b) Definieren Sie ein Makro `isdigit(c)`, welches `TRUE` (einen Wert ungleich 0) liefert, wenn das übergebene Zeichen eine Ziffer ist.

c) Was läuft bei `max(a,b++)` falsch?

d) Was sind Vorteile von Makros? Was sind Nachteile?

e) Was passiert, wenn die Zeile „`#define VERBOSE`“ entfernt wird?