Übung 01

Freie Universität Berlin | Betriebs- und Kommunikationssysteme | Abgabe 30.04.2021, 10:00

Sommersemester 2021

Prof. Dr.-Ing. Jochen Schiller

Allgemeine Hinweise Lesen Sie bitte folgende Hinweise aufmerksam durch. Sie gelten für alle Übungszettel.

- Melden Sie sich bitte im KVV unter **Section Info** zu einer Übungsgruppe an.
- Abgaben sind bis zum Beginn der Vorlesung jeweils Freitags (10:00 Uhr) möglich.
- Bitte geben Sie zu jedem einzureichenden Zettel Ihre Beantwortung als PDF sowie den Quellcode kommentiert und unkompromiert im KVV ab. Die Abgabe in Papierform im Fach des Tutors ist optional.
- Beantworten Sie alle Aufgaben so verständlich wie möglich mit Ihren eigenen Worten. Abgaben sind in Englisch und Deutsch möglich.
- Falls Sie Quellen jenseits der Vorlesungsfolien verwenden, geben Sie diese an.
- Programmieraufgaben nutzen als Referenzsystem die Linux-Pools, werden also dort zum Testen kompiliert und ausgeführt.
- Zum Bestehen eines Zettels muss in jeder Aufgabe mindestens 1 Punkt erreicht werden, *und* mindestens 50 Prozent der Punkte auf dem gesamten Zettel
- Die Punkte sind wie folgt pro Aufgabe definiert:
  - 3 Punkte: Alles perfekt, die Klausur kann kommen.
  - 2 Punkte: Es funktioniert / ist im Wesentlichen korrekt, kleinere Mängel sind mit Kommentaren versehen.
  - 1 Punkt: Es funktioniert nicht, aber richtige Idee mit Fehlerbeschreibung (!) vorhanden oder Abgabe enthält grobe Fehler bzw. ist unzureichend beschrieben – jedoch erkennbarer Aufwand.
  - 0 Punkte: Aufgabe oder unabhängige Teilaufgabe nicht bearbeitet bzw. kein Arbeitsaufwand erkennbar.

    Damit Zettel leider nicht bestanden.

Aufgabe 1: Betriebssystemdesign

Beantworten Sie die folgenden Fragen und geben Sie ggf. Ihre Quellen an.

1. Die Hauptaufgaben für Betriebssysteme sind es, erstens Abstraktionen zu schaffen – die den Anwendungsprogrammen als virtuelle Ressourcen zur Verfügung gestellt werden – und zweitens, die darunterliegenden physischen Ressourcen zu verwalten.

   Wie nennen sich die jwg. virtuellen Ressourcen zu den folgenden physischen Ressourcen und was bieten uns die einzelnen Abstraktionen?

   a) Prozessor
   b) Arbeitsspeicher
   c) Persistenter „Festplatten-“ Speicher
   d) Netzwerk-Interface

2. Wir nutzen Protection Rings um Nutzerprogramme von Betriebssystem abzuschirmen – jedoch welche Komponente „implementiert“ diesen Schutzmechanismus? Wie nennt sich eine Wechsel zwischen zwei Protection Ringen? Nennen Sie *zwei* Beispiele dafür wann ein solcher Privilegienwechsel stattfinden muss.

3. Ein Microkernel ist „kleiner“ als ein monolithischer. Wie wird trotzdem die gleiche Funktionalität geboten? Was bedeutet das für die Implementierung und Performance von Systemaufrufen, – sprich dem Benutzen von Systemfunktionalität durch ein Nutzerprogramm – aber auch für Hardware-Interrupts und Exceptions?

4. Warum passiert bei Modus-Wechseln durch Systemaufrufe nicht zwangsläufig ein kompletter Context-Wechsel (d. i. alle Register werden gesichert, die Funktionalität ausgeführt, und dann wieder hergestellt) und warum bei (asynchronen) Hardware-Interrupts schon?

   Kann es bei Interrupts für Systemaufrufe zu Nested Interrupts kommen (falls die Hardware das überhaupt unterstützt)? Diskutieren Sie!

C Hinweise In dem Veranstaltungteil **Betriebs- und Kommunikationssysteme** wird in den Übungen in der Programmiersprache C programmiert. Da C eine plattformabhängige Sprache ist, sind einige Richtlinien nötig, um die korrekte Funktionalität ihrer Programme auf den Referenzsystemen sicher zu stellen. Es empfiehlt sich ein solches produzierendes System mindestens als virtuelle Maschine einzurichten, oder an den Poolrechnern zu arbeiten. Probleme, die auf die Nichteinhaltung der Anforderungen zurückzuführen sind, können nicht verfolgt werden und führen zu fehlerhafter Abgabe. Anforderungen an ihr produzierendes System sind:

1. Linux 64 Bit (z. B. Linux Mint, Ubuntu, . . .)
2. C-Compiler der GNU Compiler Collection (GCC) mit Standardbibliotheken
3. Shell (z. B. Zsh, Bash, . . .)
4. Editor (z. B. Atom, Emacs, Vim, gedit, . . .)

Ihr Programm muss mit folgenden Compilerflags ohne Warnungen und / oder Fehler compilieren:

```
$ c99 -O2 -Wall -Wextra -pedantic -o output programm.c
```

Danach können Sie das `output` genannte Programm wiefolgt ausführen, diesem Programmargumente übergeben und sich den Exit-Code angucken:

```
$ ./output arg1 arg2 arg3 "argument vier" arg5
Dies ist eine beispielhafte Programmausgabe
$ echo $?
0
```

Aufgabe 2: `cat(1)` mit POSIX-libc

In dem Rechnerarchitekturteil haben Sie bereits ein vereinfachtes Programm `cat` in Assembler mit Syscalls implementiert. Sie sollen nun das gleiche Programm in C schreiben. Jedoch ist es in C nicht direkt möglich Syscalls aufzurufen, jedoch bietet die POSIX-API von Linux die zu den Syscalls `sys_read` und `sys_write` äquivalenten Funktionen `read` und `write` an, die als sehr einfache Wrapper um die eigentlichen Syscalls implementiert sind.

Nachdem Sie die gleiche Funktionalität in C erreicht haben, erweitern Sie Ihr Programm um die eigentliche Funktionalität von „conCATenate“ und lesen Sie – falls Sie Konsolenargumente bekommen haben – an Stelle von der Standardeingabe die Dateien in der angegebenen Reihenfolge. Somit ist die Programmausgabe der Inhalt aller angegebenen Dateien hintereinander.

Dazu benötigen Sie noch die Funktion `open` welche Ihnen aus einem Dateiname einen Filedeskriptor zurückgibt, den Sie dann an `read` übergeben können.

```
$ cat foo.txt bar.txt baz.txt
Ich bin eine Zeile aus foo.txt.
Ich eine Zeile aus bar.txt.
Und ich gleich mehrere ...
... Zeilen aus baz.txt.
```

Hinweis Mit `man 2 read` bzw. `man 2 write` können Sie sich die Signatur der Funktion anzeigen lassen, samt genauer Dokumentation der Funktionsweise. In dieser sog. Man-Page steht auch, in welchem Header die Funktionen deklariert sind.

Nutzen Sie für die eigentliche Kopier-Routine *nicht* die Funktionen aus `stdio.h` (für Fehlerbehandlung o. Ä. ist die Nutzung erlaubt)!