Betriebs- und Kommunikationssysteme

WS 15/16

Prof. Dr-Ing. Volker Roth

Freie Universität Berlin

3. Übung

Ausgabe	Abgabe
30.10.15	13.11.15

Bitte bei der Abgabe Name 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. Nutzen Sie dazu die Flags `-std=c99`, `-Wall` und `-pedantic`. Es sollten keine Warnungen auftauchen.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Behauptungen über Betriebssysteme (2 Punkte)

Beurteilen Sie den Wahrheitsgehalt der folgenden Aussagen und begründen Sie Ihre Entscheidung. Wenn Sie Spezialfälle kennen, die dem Standardfall widersprechen, so schreiben Sie diese mit auf.

- Ein Microkernel beschränkt sich auf grundlegende Speicherverwaltung und Kommunikation zwischen Prozessen.
- Im Kernelmodus muss das Betriebssystem darauf achten, dass die Speicherbereiche verschiedener Prozesse voneinander isoliert sind.
- Interrups können nicht unterbrochen werden.
- Bei Microkerneln kommt es häufiger zu einem Kontextwechsel als bei monolithischen Kerneln.
- Die erste Implementierung von Syscalls (siehe Folie 2.21) ist für ein Microkernel geeignet.
- Die zweite Implementierung von Syscalls (siehe Folie 2.22) ist für einen monolithischen Kernel geeignet.
- In der dritten Implementierung von Syscalls (siehe Folie 2.23) werden Syscalls durch einen Nachrichtenausausch realisiert.
- In der vierten Implementierung von Syscalls (siehe Folie 2.24) werden Syscalls durch einen Nachrichtenausausch realisiert.

Aufgabe 2: C Programmierung (3 Punkte)

a) Wählen sie sich eine shell (bash, csh, etc.) und machen Sie sich mit den Grundlagen, wie z.B. Dateien erzeugen, kopieren, verschieben, Anzeigen von Dateiinhalten und Auflisten von Ordnerinhalten, vertraut.

b) Machen Sie sich mit Low-Level-IO Systemaufrufen unter Linux vertraut. Für diesen Teil der Aufgabe benötigen Sie nur `open()`, `close()`, `read()` und `write()`. Verwenden sie **nicht** die Standardbibliotheksfunktionen für IO! Schreiben Sie eine Funktion

```
int copy(char *sourcename, char *targetname);
```

Die Funktion soll den Inhalt einer Datei kopieren und zwar nur dann, wenn es noch keine Datei mit dem Namen der Zieldatei existiert. Eine Datei darf nicht einfach überschrieben werden. Verwenden Sie zum kopieren einen Puffer (char buffer[BUFFSIZE]). (Experimentieren Sie wenn sie wollen mit unterschiedlichen Puffergrößen und vergleichen sie die benötigte Zeit.)

c) Verwenden Sie Ihre copy-Funktion um einen einfachen Papierkorb zu implementieren. Der Papierkorb soll drei Befehle unterstützen:

```
./trashcan -p filename
```

PUT: Eine Datei innerhalb des Verzeichnisses in dem trashcan ausgeführt wird, soll in einen versteckten Ordner `".ti3_trash"` verschoben werden.

```
./trashcan -g filename
```

GET: Eine Datei innerhalb von `".ti3_trash"` soll wiederhergestellt werden, im selben Verzeichnis, in dem trashcan ausgeführt wird.

```
./trashcan -r filename
```

REMOVE: Die Datei filename in dem Ordner `".ti3_trash"` wird endgültig gelöscht.

Das Verzeichnis soll automatisch beim ersten Aufruf von `trashcan` erzeugt werden. Finden Sie zum Erstellen eines Ordners sowie für das Löschen von Dateien entsprechende Systemaufrufe. In keinem Fall darf eine existierende Datei überschrieben werden. Überlegen Sie sich sinnvolle Fehlerbehandlungen. Sie können das Grundgerüst `trashcan_framework.c` von der Veranstaltungsseite verwenden.