TI III: Operating and Communication Systems WS 2010/11 Übungsblatt Nr. 2

Dipl.-Inform. Heiko Will, AG Technische Informatik, Freie Universität Berlin

Ausgabe am 29.10.2010 — Abgabe spätestens 05.11.2010, 10:00 Uhr

Bitte bei der Abgabe beide Namen/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/der richtigen Tutorin abgegeben werden.
Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind.
Zu spät abgegebene Lösungen werden nicht mehr berücksichtigt!

---

1. Aufgabe: Protection Rings

Beschreiben Sie in kurzen Sätzen die Idee hinter den Protection Rings. Wieviele Ringe werden von gängigen Betriebssystemen zu welchem Zweck verwendet?

2. Aufgabe: Beurteilen Sie

Beurteilen Sie den Wahrheitsgehalt der folgenden Aussagen. Begründen Sie Ihre Entscheidung. Wenn Sie Spezialfälle kennen, die dem Standardfall wiedersprechen, 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 von einander isoliert sind.
- Wenn Interrupt Service Routinen nebenläufig abgearbeitet werden, muss das Betriebssystem darauf achten, dass der Prozessor beim Timer-Interrupt nicht in den Benutzermodus wechselt.
- Bei Microkerneln kommt es häufiger zu einem Kontextwechsel als bei monolithischen Kerneln.

3. Aufgabe: Syscalls

Beschreiben Sie in wenigen Sätzen die vier in der Vorlesung vorgestellten Implementierungen von Systemaufrufen bei monolithischen Kernel und Microkernel in Hinsicht auf den Ablauf der Wechsel zwischen User- und Kernelspace.

4. Aufgabe: Programmieren in c

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 system calls 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 soll nicht einfach überschrieben werden.
Sie werden zum kopieren einen Buffer (char buffer[BUFFSIZE]) verwenden müssen. (Experimentieren Sie wenn sie wollen mit unterschiedlichen Buffergrößen und vergleichen sie die benötigte Zeit.)

c) Verwenden Sie Ihre copy-Funktion um einen einfachen Papierkorb zu implementieren. Er soll drei Befehle verstehen:
./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 ".ti_trashcan" 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 system calls.
In keinem Fall soll eine existierende Datei überschrieben werden. Überlegen Sie sich sinnvolle Fehlerbehandlungen.
Sie können das Grundgerüst trashcan_framework.c von der Veranstaltungsseite verwenden.