1. Aufgabenblatt mit Lösung zur Vorlesung

Betriebssysteme WiSe 2017

Basse, Christian, Simon

Abgabe 9 November 2017, 10 Uhr

Aufgabe 1 Architekturen *5 Punkte*

Vorhandene Betriebssystementwürfe können grob in zwei Klassen eingeteilt werden:

- die Makrokernarchitektur,
- die Mikrokernarchitektur.

Disktuieren Sie beide Architekturansätze unter Zuhilfenahme mindestens folgender quellen:

- J. Liedtke, Toward real μ-kernels, Communications of the ACM, 39(9):70–77, September 1996, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.4950&rep=rep1&type=pdf
- C. Maeda, B.N. Bershad, Networking performance for microkernels, Proceedings of Third Workshop on Workstation Operating Systems, 13:154 – 159, April 1992 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.9733&rep=rep1&type=pdf

Lösung:

Aufgabe ist bzgl makro nicht monolithisch → einordnen usw.

μ-Kernel im Bezug zu monolithischen Kernel

- nur IPC, MMU, Scheduler Teil des Kernels
- Interrupts werden ausserhalb des Kernels behandelt
- Protokolle (z.B. UDP) werden ausserhalb des Kernels implementiert
- Implementationen von Protokellen, etc. von monolithischen Kerneln können unter umständen nicht ohne gravierende geschwindigkeits Verluste *direkt* übernommen werden

\+ nur Kernel kann sicherheitskritische Operationen eines Prozessors nutzen, software in user mode nicht

\+ Treiberausfälle etc. sind *nur* Softwarefehler

\+ modularer/flexibel/leicht erweiterbar da nicht Kernel für neue Geräte angepasst/erweitert werden muss

\+ Kernel besser wartbar, da kleiner

\+ Treiber etc. nur Zugriff auf zugewiesenen Speicherbereichen

\- ineffizienter / mehr overhead bei IPCs, Addressraum wechsel etc.

\- Höhere Latenz zum bearbeiten von Daten, da (mindestens) ein Kontextwechsel nach dem Auslösen eines Interrupts stattfinden muss

\- Je nach Hardwarezugriff können nun (leicht austauschbare) Treiber (im user mode) das System korrumpieren

Aufgabe 2 Steuerung von Geräten *15 Punkte*

Implementieren Sie eine `print`-Funktion zum ausgeben von Daten über die serielle Schnittstelle (Debug-Unit, DBGU).

Lösung:

Anleitung zum erstellen eines mit qemu benutzbaren kernels:

(a) Wechseln in den Ordner `GrandiOS`

(b) Zunächst wird mit dem Script `setup_env.sh` die Toolchain von Rust vorbereitet. Unter umständen sind benutzereingaben erwartet, diese können einfach mit Enter bestätigt werden.

(c) Durch starten von `kernel_build.sh` kann nun der Kernel `kernel` erzeugt werden.

(d) Starten von qemu mit dem Kernel: `qemu-bsprak -piotelnet -kernel kernel`

(e) Starten einer telnet Verbindung zu qemu: `telnet localhost 44444`

Das vorbereiten der Toolchain mittels `setup_env.sh` ist nur einmal notwendig.

Für den unwahrscheinlichen Fall, dass aus irgend einem Grund das bauen des Kernels fehlschlägt, ist unter `GrandiOS/kernel.backup` noch ein von uns gebauter Kernel vorhanden.