Universität Bamberg

GRABS: Grundlagen der Rechnerarchitektur und Betriebssysteme

Vorlesung 10: Hardware/Software-Schnittstellen

Literatur:
Arpaci-Dusseau [1]
Kap. 1 und 13

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insbes. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Problem: ***Sicherheit***

- Sind Aufrufe von Betriebssystemfunktionen normale Funktionsaufrufe?
- **Problem:**
  - Nicht alle Anwendungsprogramme sollen alle Funktionen aufrufen dürfen
    - Beispiel: Festplatte formatieren…
- **Lösung:**
  - Einführung von **Privilegienmodi** für Anwendungen und System
  - **Isolation von Adressräumen** im Prozessor
    - Anwendungsprogramme dürfen nicht alle Befehle des Prozessors ausführen
    - Anwendungsprogramme dürfen nur auf eine Teilmenge des Speichers zugreifen
- **Der Betriebssystem-Kern darf aber (fast) alles…**
- Lösung: **Systemaufrufe** erlauben Programmen, Funktionalität zu nutzen, die nur im privilegierten Modus ausgeführt werden darf

Welche Systemaufrufe werden benötigt?

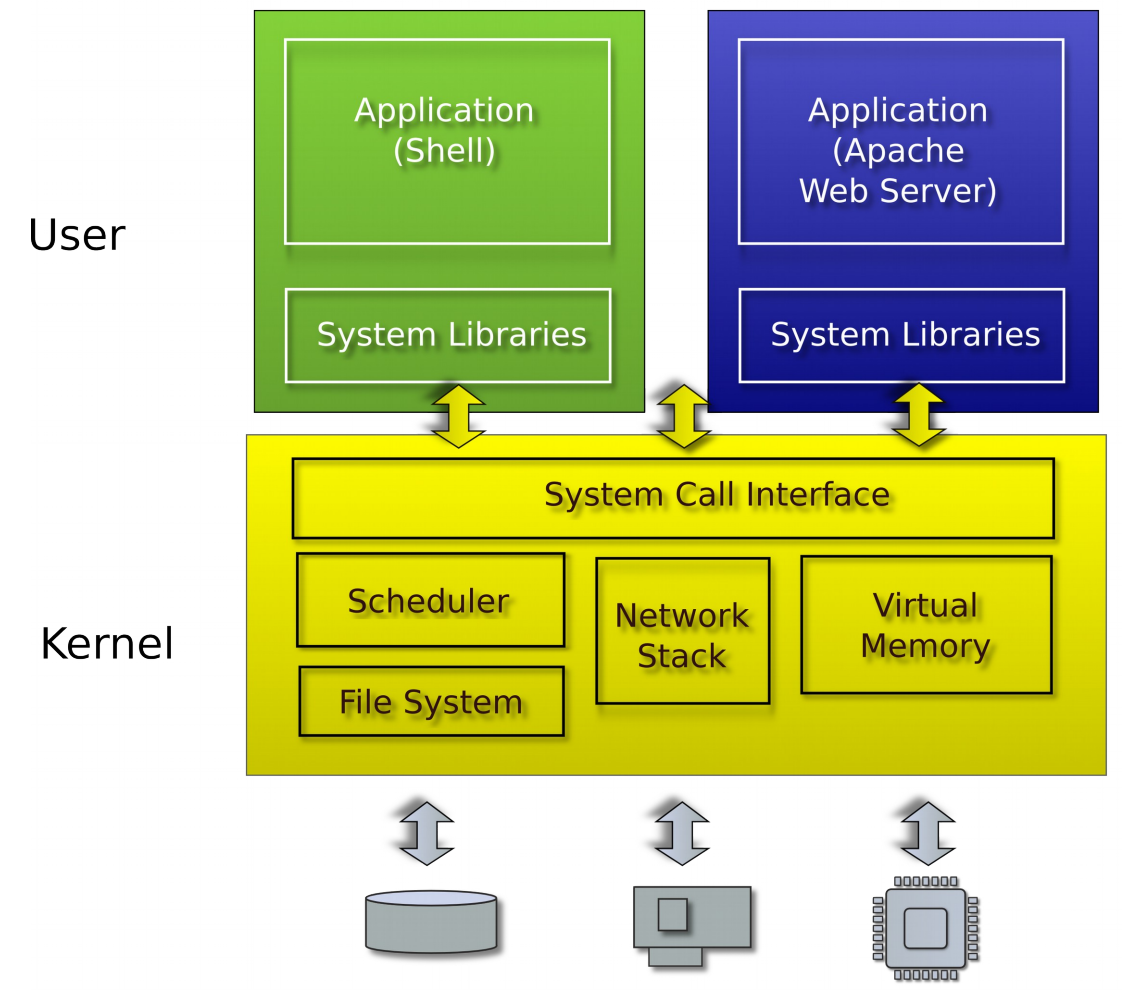
- Systemaufrufe (*system calls*) sind eine abstrakte Schnittstelle zur Hardware des Computers und anderen Ressourcen, wie z.B. Netzwerkverbindungen

- Typische Unix-Systemaufrufe fallen in eine von vier Gruppen:
  - Prozesse
    - Erzeugen, Beenden, Warten, Abbrechen
  - Speicher
    - Anforderung (Allokation), Freigabe (Deallokation)
  - Dateien, Verzeichnisse und Dateisysteme
    - Öffnen, Schließen, Lesen, Schreiben, Anlegen, Anmelden
  - Interprozesskommunikation
    - Pipes, shared memory (kommt später)

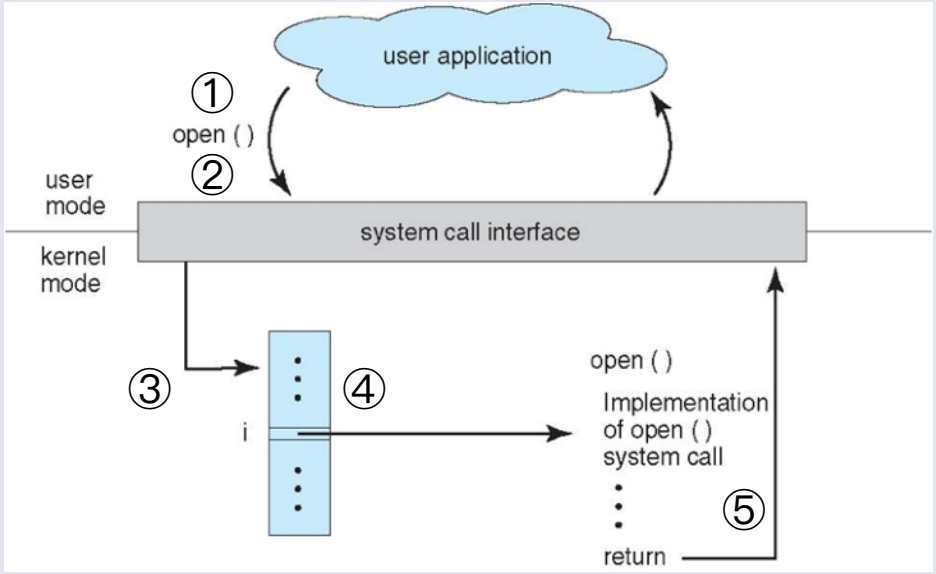
Systemaufrufe in Unix

Systemaufrufe…

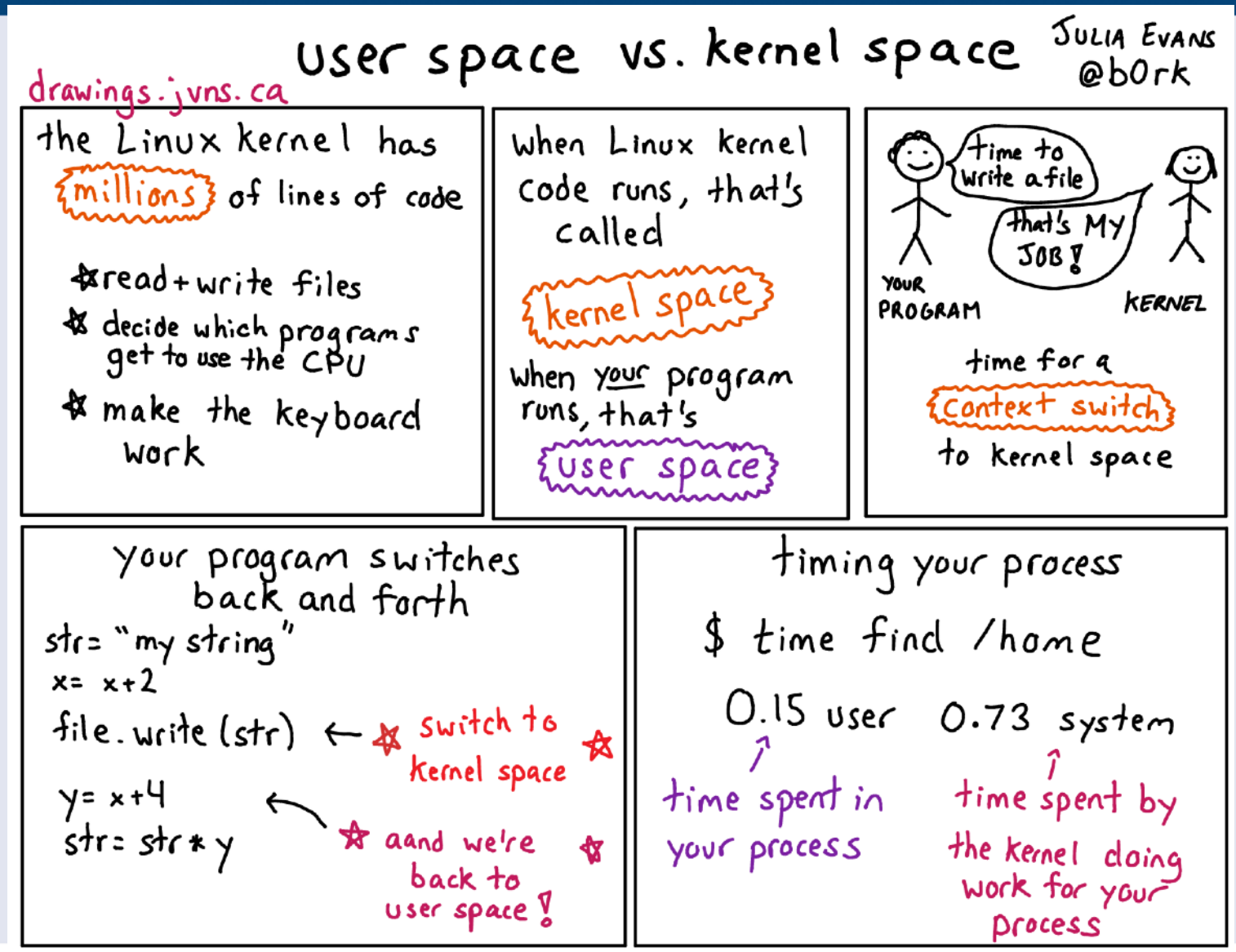
- ermöglichen die Kommunikation von Benutzer- und Kernelmodus
- stellen Aufruf einer Kernel-funktion dar
- implementieren die **Schnittstelle des Betriebs-systems**



Ablauf eines Systemaufrufs



1. Benutzerprogramm ruft eine *Stub-Funktion* in der C-Library (libc) auf, die dem gewünschten Systemaufruf entspricht (hier: **open()**)
2. Die Stub-Funktion belegt die Prozessorregister mit den Systemaufrufparametern und der Systemaufrufnummer
3. Die Stub-Funktion führt eine *spezielle Maschineninstruktion* aus, um in den privilegierten Kernelmodus zu wechseln
4. Der Kernel überprüft die Parameter und holt die Adresse der dem Systemaufruf zugehörigen Kernelfunktion aus einer Tabelle (Index = Systemaufrufnummer)
5. Die Funktion wird ausgeführt und springt mit einer anderen *speziellen Maschineninstruktion* zurück zum Benutzerprogramm



https://jvns.ca/

Typische Systemaufrufe

- Beispiel aus dem Lehrbetriebssystem xv6 des MIT [5]
- Diese Aufrufe entsprechen den Systemaufrufen von 6th Edition Unix von 1976
- Die meisten auch in aktuellem Unix/Linux noch vorhanden

Prozesse

Speicher

Dateien, Verzeichnisse, Dateisysteme

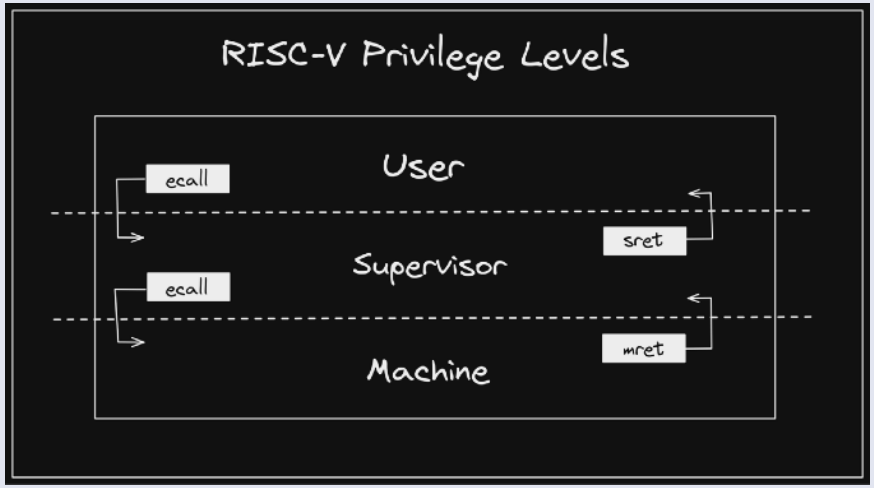
Interprozesskommunikation

Nr	Name	Funktion
1	fork	Neuen Prozess erzeugen
2	exit	Aktuellen Prozess beenden
3	wait	Auf Prozess warten
4	pipe	Interprozesskommunikation
5	read	Daten von Dateideskriptor lesen
6	kill	Prozess terminieren (Signal senden)
7	exec	Neuen Prozess laden
8	fstat	Status einer Datei prüfen
9	chdir	Aktuelles Verzeichnis ändern
10	dup	Dateideskriptor duplizieren
11	getpid	Prozess-ID des aktuellen Prozesses holen
12	sbrk	Mehr Speicher anfordern
13	sleep	Prozess eine Zeit lang schlafen legen
14	uptime	Laufzeit des Systems abfragen
15	open	Datei öffnen, gibt Deskriptor zurück
16	write	Daten in Dateideskriptor schreiben
17	mknod	Neue Spezialdatei anlegen
18	unlink	Datei löschen
19	link	Verweis auf Datei anlegen
20	mkdir	Verzeichnis anlegen
21	close	Datei schließen

RISC-V Privilegienmodi

- **RISC-V unterstützt drei Privilegienmodi**
  - **U**ser – Anwendungen
  - **S**upervisor – Betriebssystem
  - **M**achine – voller HW-Zugriff

picture by Daniel Mangum [3]



- **Umschalten** zwischen Modi kann **explizit** oder **implizit** geschehen
  - Explizit: durch Software veranlasst ("**ecall**" Assemblerinstruktion): ***Systemaufrufe***
  - Implizit: durch Hardwareinterrupt oder als Seiteneffekt der Ausführung einer Instruktion (z.B. ungültige Instruktion/Adresse)
- **Rückkehr** zum Programm erfordert spezielle Instruktion
  - **mret** (return from machine mode) oder **sret** (return from supervisor mode)

Systemaufrufe bei RISC-V

- Wie können wir Systemaufrufe realisieren?
  - Die C-Library (libc) stellt ***Stub-Funktionen*** zur Verfügung
    - Diese sind die Schnittstelle zu den Systemaufrufen
    - Rufen die `ecall`-Instruktion mit passenden Parametern auf
  - Werden wie normale Funktionen aufgerufen
    - Also: Parameter in `a0`, `a1`, …, Rückgabewert in `a0`
- **Besonderheiten:**
  - **Nummer des Systemaufrufs** *(system call* oder *syscall)* in Register `a7`
  - Also nur sieben Parameter in Registern übergebbar (Rest: Stack)

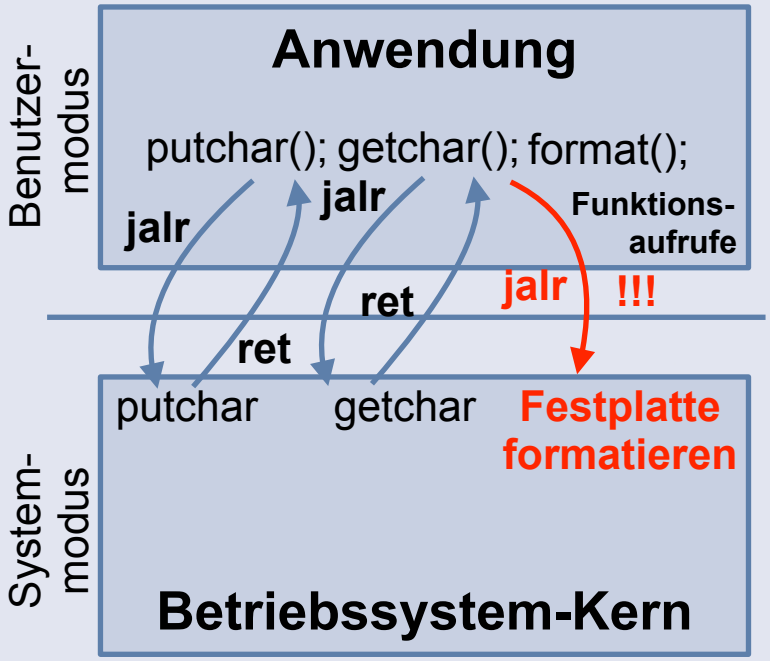
```
#define SYS_sleep 13
    .global sleep
sleep:
    li a7, SYS_sleep
    ecall
    ret
```

Systemaufrufe vs. Funktionsaufrufe

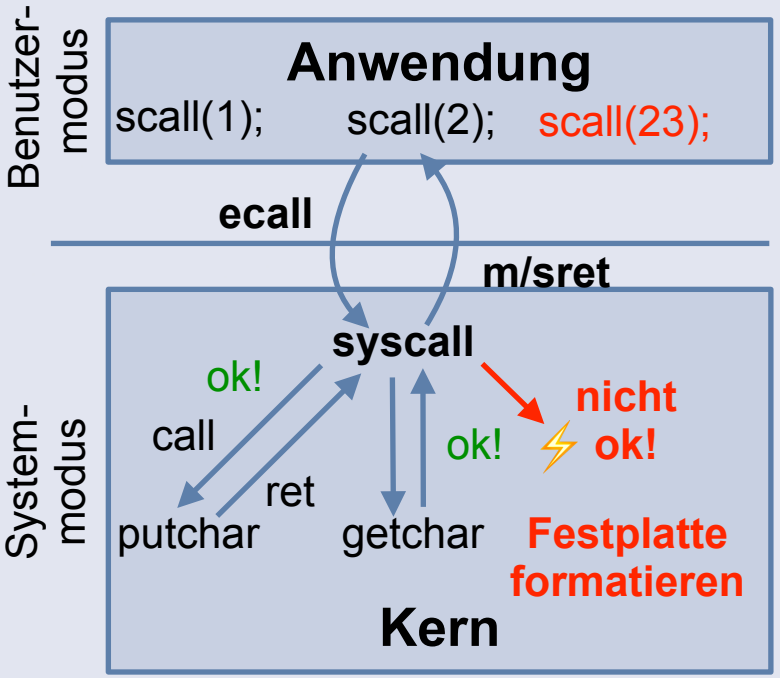
Universität Bamberg

- **Sicherstellen eines *kontrollierten Übergangs* vom Benutzermodus zum Systemmodus (Kernel)**
  - Benutzerprogramme (Anwendungen) dürfen nur zugelassene Funktionen aufrufen
  - Betriebssystem-Kern überprüft aufgerufene Funktionen und Parameter

Direkte Funktionsaufrufe an BS



Verwendung von Systemaufrufen



System- und Libraryaufrufe anschauen

- Es gibt Werkzeuge unter Linux (und ähnliche unter anderen Unix-Systemen – leider ist macOS hier schwierig…) zur Analyse der von einem Programm getätigten System- und Libraryaufrufe (in *shared libraries*):
  - strace für Systemaufrufe
  - ltrace für Library-Aufrufe

- Live-Demo: strace und ltrace in Linux
  ⇒ Hörsaalübung

Referenzen

[1] Remzi Arpaci-Dusseau, Andrea Arpaci-Dusseau
Operating Systems: Three Easy Pieces
https://pages.cs.wisc.edu/~remzi/OSTEP/

[2] **RISC-V ELF psABI** Document
https://github.com/riscv-non-isa/riscv-elf-psabi-doc/

[3] Russ Cox, Frans Kaashoek, Robert Morris
xv6: a simple, Unix-like teaching operating system
https://pdos.csail.mit.edu/6.828/2023/xv6/book-riscv-rev3.pdf