SP - Ausschnitte

25.07.2023 - Systemprogrammierung

1. Inhaltsverzeichnis

2. Definitionen

Programm: Folge von Anweisungen, kann von mehreren Prozessen gleichzeitig ausgeführt werden

Prozess: Ein in Ausführung befindliches Programm (und nur genau eines)

Compiler: Erzeugt aus mehreren Programmteilen (Modulen) ein Programm (hier genau auf die Formulierung achten!)

Binder: Erzeugt aus einer oder mehreren Objekt-Dateien ein Programm (laut Pad)

Prozessorkern: Pro Prozessorkern maximal einen laufenden Prozess

3. Adressraum

- Aufteilung des Hauptspeichers eines Prozesses in Segmente
  - Vgl. Vorlesung *A-III, Seite 7f.*

```
static int a = 3; static int b;
static int c = 0; const int f = 42;
const char *s = "Hello World\n";

int main(void) {
    int g = 5;
    static int h = 12;
}
```

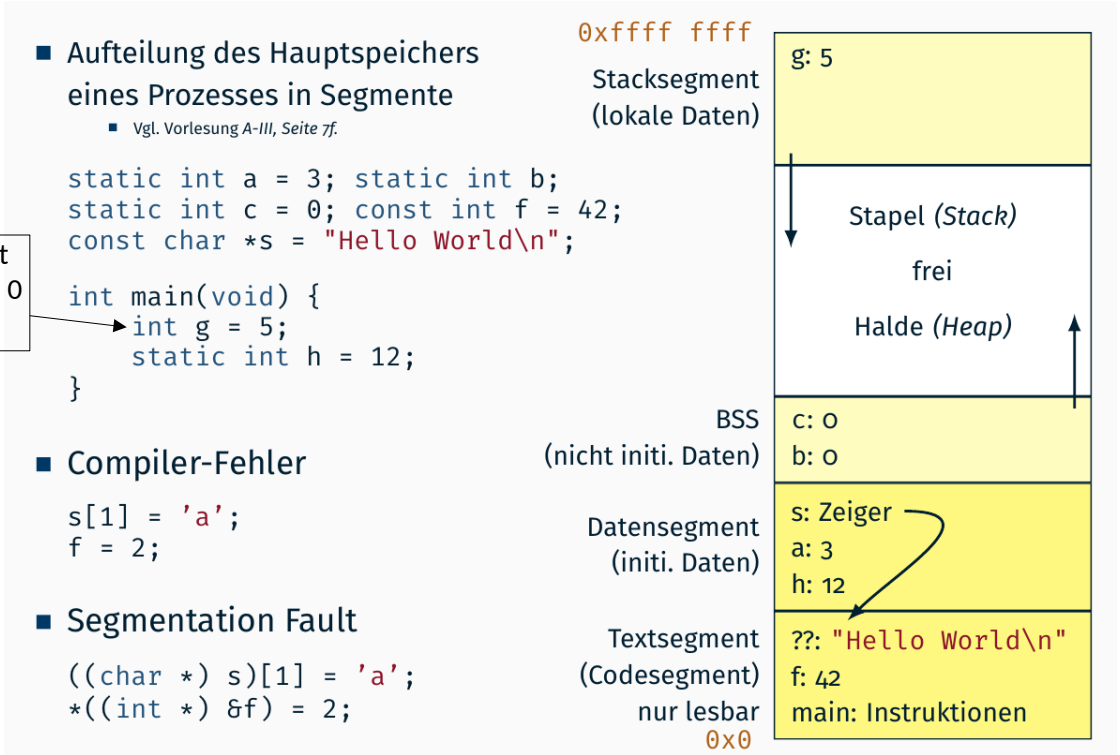
Wird auch nicht automatisch mit 0 initialisiert

- Compiler-Fehler

```
s[1] = 'a';
f = 2;
```

- Segmentation Fault

```
((char *) s)[1] = 'a';
*((int *) &f) = 2;
```



Stack-Frame

Alles was beim Funktionsaufruf und durch die Funktion selbst auf den Stack gelegt wird (Übergabe-Parameter, Rücksprungadresse, lokale Variablen, Caller-Save-Register).

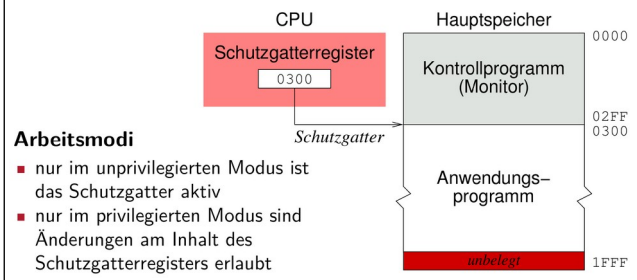
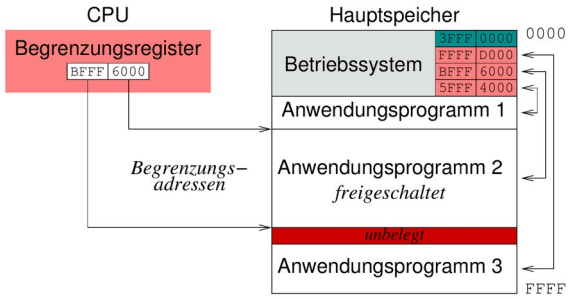
Beim Rücksprung wird dieser Stack-Frame wieder entfernt.

Makros bspw. #define sum(a,b) as a + b

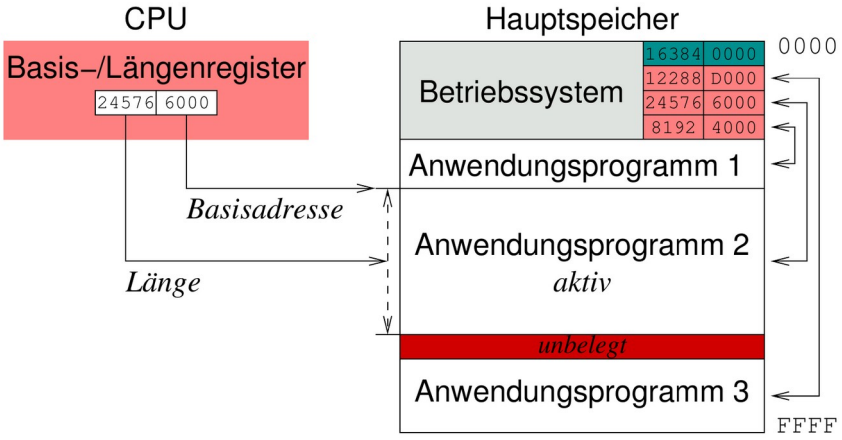
Präprozessor ersetzt einfach das, was links vom *as* steht durch das was rechts vom *as* steht

→ bei Sum oderso wird nicht die Reihenfolge garantiert: sum(4,5) * 3 = 4+5*3 = 19 nicht 9*3 =27

3.2 ADRESSRAUMSCHUTZ

Adressraumschutz durch Abgrenzung/Abteilung (Fencing)	**Adressraumschutz durch Einfriedung/Eingrenzung**
	
Problem: Externe Fragmentierung	Problem: Externe Fragmentierung

Adressraumschutz durch Segmentierung



Problem: Pro Programm nur ein Segment

3.3 Adressraumarten

Virtueller Adressraum

- Abstrahiert ganze Prozesse im Arbeitsspeicher (Swapping möglich)
- Erlaubt es ein Programm laufen zu lassen, obwohl nur Einzelbestandteile von Text und Daten eingelagert sind
- Kann größer sein, als der physikalische Speicher

Logischer Adressraum

- Abstrahiert einzelne Speicherbausteine im Arbeitsspeicher
- Wird vor/zur Laufzeit auf reale Adresse abgebildet
- Lückenloser richtiger Hauptspeicher (gleich groß wie phys.), es gibt aber trotzdem „fake“ Adressen

Paging

Segmentierung

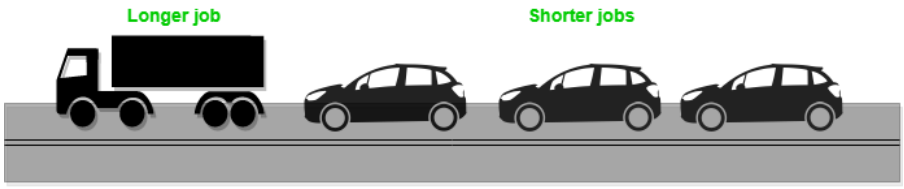
Physikalischer Adressraum

Realer Speicher mit Lücken, etc.

4. Prozesseinplanung (Ausschnitt)

4.1 Konvoi-Effekt

Längere Prozesse zögern kürzere Prozesse unnötig hinaus.



4.2 Round Robin

FCFS aber mit Verdrängung: Jeder Prozess bekommt Zeitscheibe die er arbeiten darf, periodische Umplanung der Prozesse, damit kurze Prozesse früher dran sind

→ Immernoch etwas Konvoieffekt weil kurze Prozesse ihre Zeitscheibe nicht ausnutzen aber besser

Virtual Round Robin

Kürzere Prozesse (die ihre Zeitscheibe nicht voll ausgenutzt haben) werden gezielt bevorzugt. Z. B. mit variabler (kürzerer) Zeitscheibe, schnellerer Verdrängung oder früheren Plätzen in der (FCFS) Warteliste.

4.3 Echtzeitbetriebssystem

Eigenschaften

- Einhaltung vorgegebener Termine
- Rechtzeitigkeit anstelle von Geschwindigkeit
- Vorhersagbarkeit
- Überwachung / Interaktion mit physikalischer Welt

Verhaltensunterschiede bei Terminvorgaben

Feste Terminvorgaben (firm) – Terminverletzungen tolerierbar

- Abbruch der Berechnung (durch Betriebssystem) → Ergebnis wertlos
- Start der nächsten Berechnung (durch BS)
- Transparent für die Anwendung

Harte Terminvorgaben (hard) – Terminverletzung keinesfalls tolerierbar

- Betriebssystem löst Ausnahmesituation aus
- Ausnahmebehandlung führt System in sicheren Zustand
- Intransparent für die Anwendung

4.4 Kriterien für die Prozesseinplanung

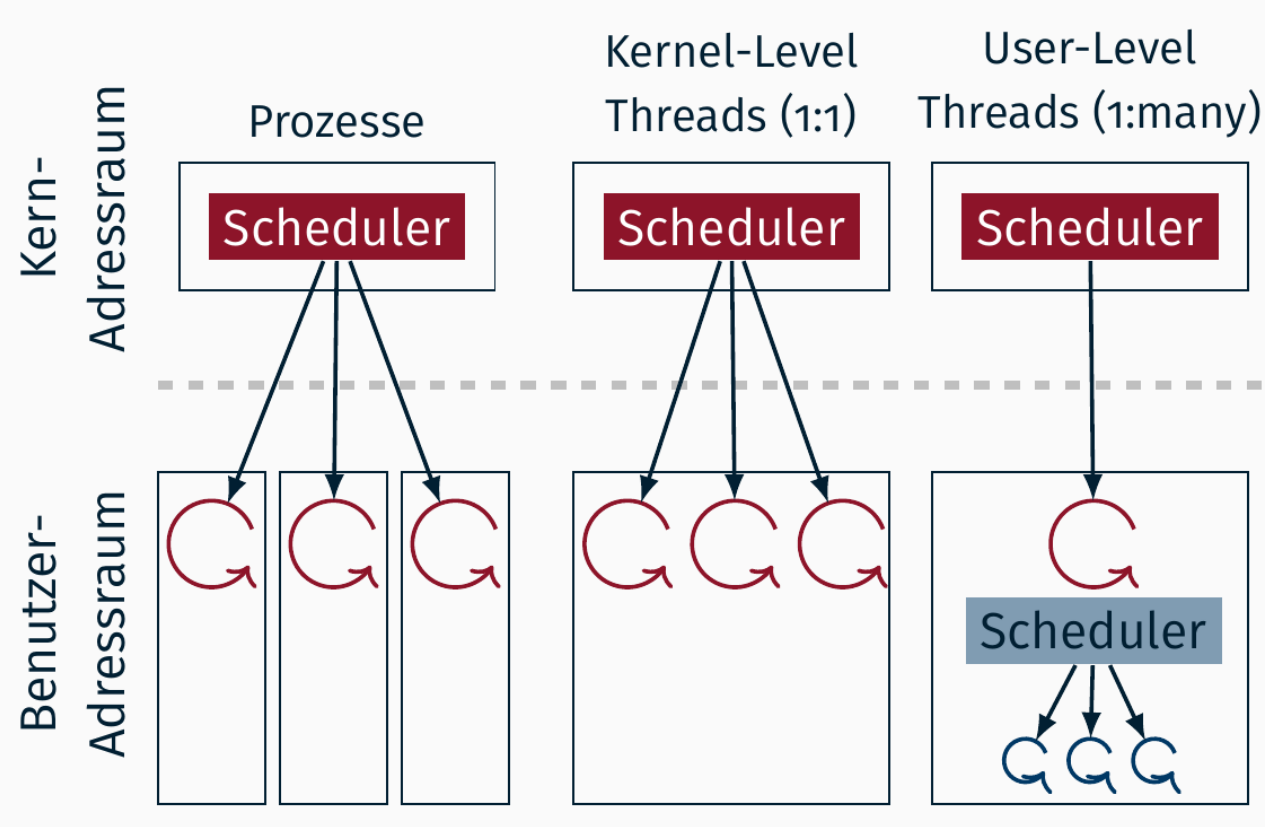
Benutzerorientiert	Systemorientiert
Das vom Nutzer wahrgenommene Verhalten Bspw. eine möglichst kurze Antwortzeit bei System-aufrufen, DurchLaufzeit des ges. Prozesses	Eine effektive Auslastung von Betriebsmitteln Bspw. ein möglichst hoher Durchsatz an vollendeten Prozessen, gleichmäßige Prozessauslastung
– Antwortzeit – Durchlaufzeit (Zeit vom Start bis zur Beendigung von einem Prozess minimieren) – Termineinhaltung – Vorhersagbarkeit	– Durchsatz – Prozessorauslastung – Gerechtigkeit – Dringlichkeit – Lastausgleich

Prioritäten je nach Betriebsart:

Allgemein	Stapelbetrieb	Dialogbetrieb	Echtzeitbetrieb
Gerechtigkeit, Lastaus-gleich	Durchsatz, Durchlaufzeit, Prozessorauslastung	Antwortzeit	Dringlichkeit, Termin-einhaltung, Vorhersagbar-keit
			Konflikte: Gerechtigkeit, Lastausgleich

5. MULTI-THREADING

Prozesse	Kernel-Level-Threads (Schwergewichtig)	User-Level-Threads (Leichtgewichtig)
• Ausführungsumgebung mit einem Aktivitätsträger • Keine parallelen Abläufe innerhalb eines logischen Adressraums auf Multi-prozessorsystemen	• Teurer als User-Threads • Gruppe von Threads nutzt gemeinsam die Betriebs-mittel eines Prozesses • Schedulingstrategie vom System vorgegeben • Systemkern sieht jeden Kernel-Thread als eigenen Aktivitätsträger: → Multi-Threading	• Extrem billig zu erzeugen • Schedulingstrategie vom Programmierer vorgege-ben • Realisierung auf Anwen-dungsebene • Systemkern sieht nur ei-nen Kontrollfluss: → Ein blockierender User-Thread blockiert alle User-Threads → Kein Multi-Threading
fork();	*pthread_create();* *(pthread_detach();* Wenn der Rückgabewert egal ist und der Speicher sofort nach Be-endigung freigegeben werden soll)	



5.2 Probleme bei FORK und EXEC:

- Nur der aufrufende Thread wird geklont (alle anderen Threads sind im Child nicht mehr vorhanden)
- Gelockte Mutexe bleiben gelockt und können nicht freigegeben oder zerstört werden
- Kind kann inkonsistenten Zustand kopieren
- Dateideskriptoren werden bei `fork();` vererbt, und bei `exec();` auch nicht geschlossen =>

```
int fd = open("index.html", O_RDONLY | O_CLOEXEC);
FILE *fp = fdopen(fd, "r");
```

Aber: Bei *exec* werden alle Mutexe zerstört und alle weitere Threads verschwinden

5.3 Volatile

Variable wird immer neu aus Speicher geholt und nicht lokal aus Cache oder so

→ Man bekommt Änderungen mit, aber nur an dieser Variable (vgl Java)

5.4 Verklemmungen (Deadlock)

Eine irreversible gegenseitige Blockierung von Prozessen, bei der das Prozessssystem keinen Fortschritt mehr macht (still steht).

Bedingungen

Notwendig:

1. Ausschließlichkeit der Betriebsmittelnutzung (mutual exclusion)
2. Nachforderung von Betriebsmitteln (hold and wait)
3. Unentziehbarkeit der Betriebsmittel (no preemption)

Hinreichend & Notwendig:

4. Zirkuläres Warten auf die Betriebsmittel

→ Für eine Verklemmung müssen alle 4 erfüllt sein

5.5 Verklemmungsvorbeugung (Deadlock prevention)

- Nicht-blockierende Synchronisation
- Alle benötigten Betriebsmittel atomar anfordern
- Betriebsmittel virtualisieren → Die realen Betriebsmittel können entzogen werden
- Betriebsmittel in fester Reihenfolge zuteilen

5.6 Verklemmungsvermeidung (Deadlock avoidance)

- Nur zirkuläres Warten wird durch laufende Analyse verhindert
- Vor jeder Betriebsmittelanforderung wird geprüft, ob System in einem unsicheren Zustand wäre

Bankiersalgorithmus

- System weiß genau welcher Prozess welche Betriebsmittel anfordern will/schon hat (unrealistisch). Der Algorithmus kann für so einen Fall Testen ob Deadlocks auftreten würden.
- → Falls ja wird die Anforderung der Mittel zurückgewiesen

Betriebsmittelgraph

- System weiß welche Betriebsmittel ein Prozess anfordert und erstellt daraus einen Graphen
- Durch Analyse dieses Graphen können Zyklen erkannt werden

5.7 Nicht-blockierende Synchronisation (CAS)

Vorteile	Nachteile
Rein auf Anwendungsebene, keine teuren Systemaufrufe Geringere Mehrkosten als bei Locking, wenn die CAS-Operation auf Anhieb funktioniert Konkurrierende Threads werden vom Scheduler nach dessen Kriterien eingeplant Keine Abhängkeit vom Halter eines Locks Verklemmungsfrei	Schwere algorithmische Umsetzung Verhungern (komplexe Änderung kann immer wieder von kürzerer Änderung ungültig gemacht werden) Braucht Hardware-Unterstützung

Wird v.a. bei „**optimistischen Ansätzen**“ verwendet, bei denen davon ausgegangen wird, dass Prozesse nicht gleichzeitig eintreffen, es also wenig Konkurrenz gibt. Falls es doch dazu kommt werden Konflikte nachträglich behandelt mittels hardwaregesteuertem wechselseitigem Ausschluss (CAS, TAS, FAA).

5.8 Sonstiges

Bei Monoprozessorsystemen ohne Verdrängung braucht man logischerweise keine Synchronisation

Pessimistischer Ansatz: Vmtl. viel Konkurrenz zwischen Prozessen, deswegen blockierende Synchronisation im wechselseitigen Ausschluss

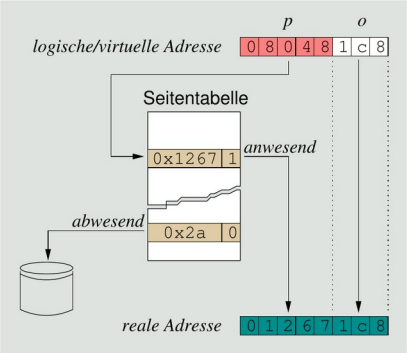
Mehrseitige Synchronisation: Alle beteiligten Prozesse sind betroffen

Einseitige Synchronisation: Nur einer der beteiligten Prozesse betroffen

Kategorien von Betriebsmitteln

Wiederverwendbare Betriebsmittel	Konsumierbare Betriebsmittel
Begrenzter Zugriff	Unbegrenzter Zugriff
Existieren dauerhaft (persistent)	Existieren vorübergehend (flüchtig)
Sind entweder teilbar oder unteilbar sein	Werden erzeugt und wieder zerstört
Beispiele in Hardware: CPU, RAM, GPU	Beispiele in Hardware: Signale, Traps
Beispiele in Software: kritischer Abschnitt, Variable	Beispiele in Software: Meldungen, Datenstrom

6. PAGING



6.1 SEITENDESKRIPTOR

Page Table: Seitennummer → Seitendeskriptor

Seitendeskriptor:

- Attribute: anwesend, referenziert, modifiziert, schreibbar, ausführbar
- Kachelnummer/-adresse im Hauptspeicher (falls anwesend, eingelagert) oder Blocknummer in der Ablage (falls abwesend, ausgelagert)

6.2 ABLAUF BEIM PAGE FAULT

1. MMU erkennt nicht gesetztes *present-bit* → Trap
2. Wenn Seite ungültig: SIGSEGV (Prozess in beendet)
3. Wenn Seite gültig: Seite wird angefordert und eingelagert (Prozess in blockiert):
   - Falls Seite im Freiseitenpuffer, muss Seite nicht vom Hintergrundspeicher geladen werden
   - Falls keine freie Seite im Arbeitsspeicher: Ungenutzte Seite auslagern (nach Seitenersetzungsstrategie)
   - Seite in freien Seitenrahmen einlagern und als *present* markieren (Prozess in bereit)
4. Prozess wird vom Scheduler in laufend überführt: Zugriff wiederholen

6.3 LEAST RECENTLY USED

Legt fest welche Page (?) aus dem Cache entfernt wird wenn er voll ist. Also hier die bei der der letzte Zugriff am längsten her ist. Grundsätlich immer mit FIFO Liste der Elemente

Zweite Chance (Basiert auf FCFS):

Used Bit welches periodisch auf 0 gesetzt wird und wenn es benutzt wird auf 1

→ Dann aus FIFO Liste in der alle drin sind das erste Element mit Used Bit = 0 entfernen

Dritte Chance:

Betrachte (Used Bit, Dirty Bit) wobei Dirty Bit: 1 falls Seite beschrieben wurde, wird nicht genullt

→ Dann am besten was mit (0,0) entfernen sonst (0,1) und dann (1,0) dann (1,1)

6.4 SONSTIGES

- Seitenflattern: Seitenein-/auslagerungen dominieren die Systemaktivität
- Freiseitenpuffer: Puffert freie Seiten damit man schneller eine freie Seite findet
- Demand-Paging: Page wird erst eingelagert wenn sie benötigt wird
- Anticipatory-Paging: Pages werden auch schon vorher (je nach Zugriffsmuster) geladen

6.5 Externe und Interne Fragmentierung

Interne Fragmentierung (Paging)	Externe Fragmentierung (Segmentierung)
• Alle Blöcke gleich groß • Innerhalb der Blöcke ist der Speicher nur teilweise befüllt → Kein Zugriffsfehler bei Zugriff auf ungenutzten Bereich	• Blöcke unterschiedlich groß • Zwischen den Blöcken ist Speicher ungenutzt

7. Speicherverwaltung

7.1 Dateisysteme – Gemischtes

Namensräume

- Namensraum: Baumstruktur bei der Objekte über Pfadnamen eindeutig angesprochen werden
- Flacher Namensraum: Nur ein Verzeichnis → Eindeutigkeit der Pfadnamen geht nur mit eindeutigen Objektnamen
- Hierarchischer Namensraum: Verschiedene Verzeichnisse → In verschiedenen Verzeichnissen gleicher Name erlaubt

Indiziertes Speichern:

- Speicher in gleich große Blöcke aufgeteilt
- Pro Datei ein (evtl. mehrfach verketteter) Index-Block (Inode) der auf die Datenblöcke in der richtigen Reihenfolge zeigt.
- Extents: Indexblock zeigt auf Datenblock aber diese Daten (die als Daten gelistet sind) sind Pointer auf anderen Speicher → neues Konzept, bei modernen Systemen teilweise eingeführt

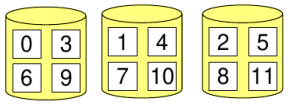
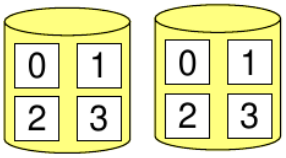
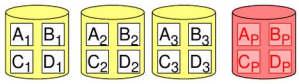
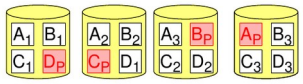
Journaling File-Systems

- Alle Änderungen an Daten & Meta-Daten werden vor der Änderung als Transaktionen (mit Informationen zum Rückgängigmachen und Wiederholen) protokolliert
- Wurden sie erfolgreich durchgeführt, wird dies (erst danach) auch protokolliert
- Bei einem Absturz kann das Dateisystem wieder in einen konsistenten Zustand gebracht werden
- Nach dem Erreichen bestimmter Checkpoints wird das Log-File gelöscht

Dateien löschen: Zugriffsrechte *read*, *write*, *exec* auf übergeordnetem Ordner notwendig aber nicht unbedingt auf Datei selbst!!

Dateideskriptor: Eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei, ein Gerät, einen Socket oder eine Pipe benutzen kann.

7.2 RAID

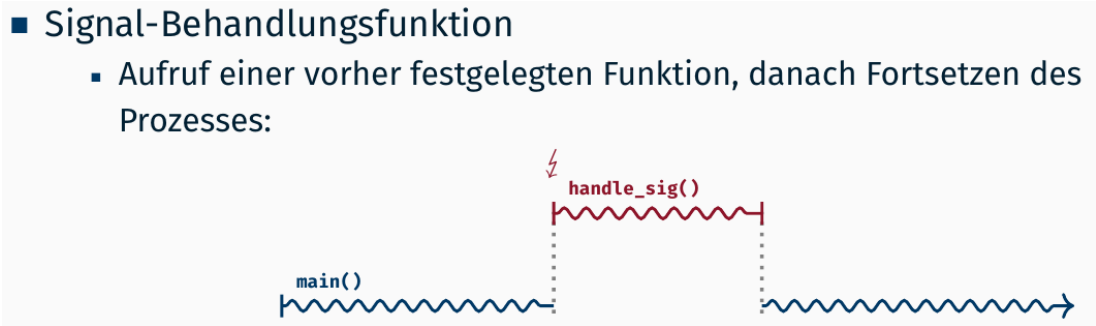
RAID 0	RAID 1	RAID 4	RAID 5
		 Paritätsblock ist byteweises XOR	 RAID 6: Doppelte Paritätsblöcke

Allgemein sind alle Verfahren schneller als einzelne Festplatten, da die Daten auf mehrere Platten verteilt sind.

7.3 Static- & Dynamic Binding

Static Libraries	Dynamic Libraries (Shared Libraries)
• Dateiendung: *.a* • Linker bindet alle .o-Dateien aus der Library, die bis dahin unaufgelöste „Commands“ enthalten (die Reihenfolge der Libraries kann somit eine Rolle spielen) • Library wird zur Ausführung des Programmes nicht mehr benötigt	• Dateiendung: *.so* • Code enthält relative Adressen/Platzhalter (Position-Independent Code, (`-fPIC`)) • Adressen werden erst beim Programmstart/ Laden gebindet (*dynamic linker*) • Vorteil: Speicherplatzersparnis & Updates

8. Signale



Es entsteht (nicht-echt-parallele) Nebenläufigkeit, bspw. mit der *errno*! (Der Signalhandler selber kann aber nicht unterbrochen werden und läuft immer komplett durch)

8.2 Wichtige Signale

- SIGINT: Ctrl+C
- SIGTERM: kill();
- SIGPIPE: Schreiben auf Socket, nachdem Gegenseite geschlossen wurde
- SIGSEGV: Segmentation Fault
- SIGABORT: abort();
- SIGFPE: Floating-Point-Exception (Überlauf, Geteilt durch 0)
- SIGCHLD: Child ist zum Zombie geworden