

Überblick: Teil D Betriebssystemabstraktionen

15 Nebenläufigkeit

16 Ergänzungen zur Einführung in C

17 Betriebssysteme

18 Dateisysteme

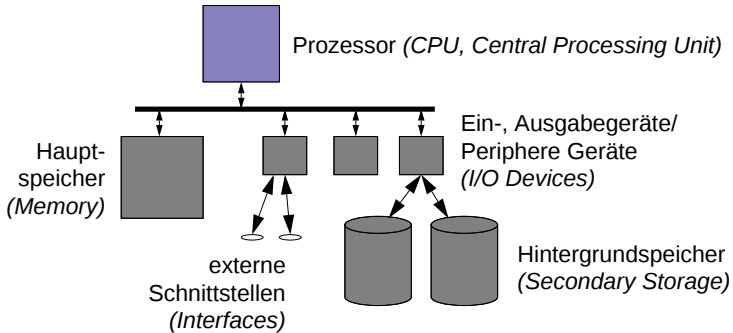
19 Programme und Prozesse

20 Speicherorganisation

21 Nebenläufige Prozesse



■ Einordnung



- Register
 - Prozessor besitzt Steuer- und Vielzweckregister
 - Steuerregister:
 - Programmzähler (*Instruction Pointer*)
 - Stapelregister (*Stack Pointer*)
 - Statusregister
 - etc.
- Programmzähler enthält Speicherstelle der nächsten Instruktion
 - Instruktion wird geladen und
 - ausgeführt
 - Programmzähler wird inkrementiert
 - dieser Vorgang wird ständig wiederholt



Prozessor (2)

■ Beispiel für Instruktionen

```
...  
0010 5510000000    movl DS:$10, %ebx  
0015 5614000000    movl DS:$14, %eax  
001a 8a            addl %eax, %ebx  
001b 5a18000000    movl %ebx, DS:$18
```

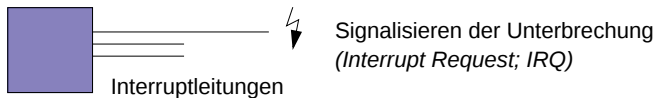
■ Prozessor arbeitet in einem bestimmten Modus

- Benutzermodus: eingeschränkter Befehlssatz
- privilegierter Modus: erlaubt Ausführung privilegierter Befehle
 - Konfigurationsänderungen des Prozessors
 - Moduswechsel
 - spezielle Ein-, Ausgabebefehle



Prozessor (3)

■ Unterbrechungen (*Interrupts*)



■ ausgelöst durch ein Signal eines externen Geräts

➔ asynchron zur Programmausführung

- Prozessor unterbricht laufende Bearbeitung und führt eine definierte Befehlsfolge aus (vom privilegierten Modus aus konfigurierbar)
- vorher werden alle Register einschließlich Programmzähler gesichert (z.B. auf dem Stack)
- nach einer Unterbrechung kann der ursprüngliche Zustand wiederhergestellt werden
- Unterbrechungen werden im privilegierten Modus bearbeitet



Prozessor (4)

- Ausnahmesituationen, Systemaufrufe (*Traps*)
 - ausgelöst durch eine Aktivität des gerade ausgeführten Programms
 - fehlerhaftes Verhalten
(Zugriff auf ungültige Speicheradresse, ungültiger Maschinenbefehl, Division durch Null)
 - kontrollierter Eintritt in den privilegierten Modus
(spezieller Maschinenbefehl - *Trap* oder *Supervisor Call*)
 - Implementierung der Betriebssystemschnittstelle
 - ➡ synchron zur Programmausführung
- Prozessor schaltet in privilegierten Modus und führt definierte Befehlsfolge aus (vom privilegierten Modus aus konfigurierbar)
 - Ausnahmesituation wird geeignet bearbeitet (z. B. durch Abbruch der Programmausführung)
 - Systemaufruf wird durch Funktionen des Betriebssystems im privilegierten Modus ausgeführt (partielle Interpretation)
 - Parameter werden nach einer Konvention übergeben (z.B. auf dem Stack)



Programme, Prozesse und Speicher

- **Programm:** Folge von Anweisungen
(hinterlegt beispielsweise als ausführbare Datei auf dem Hintergrundspeicher)
- **Prozess:** Betriebssystemkonzept
 - Programm, das sich in Ausführung befindet, und seine Daten
(Beachte: ein Programm kann sich mehrfach in Ausführung befinden)
 - eine konkrete Ausführungsumgebung für ein Programm
Speicher, Rechte, Verwaltungsinformation (verbrauchte Rechenzeit,...),...
- jeder Prozess bekommt einen eigenen virtuellen Adressraum zur Verfügung gestellt
 - eigener (virtueller) Speicherbereich von 0 bis 2 GB (oder mehr bis 4 GB)
 - Datenbereiche von verschiedenen Prozessen und Betriebssystem sind gegeneinander geschützt
 - Datentransfer zwischen Prozessen nur durch Vermittlung des Betriebssystems möglich



Prozesse

- ▲ Bisherige Definition:
 - Programm, das sich in Ausführung befindet, und seine Daten

- eine etwas andere Sicht:

- ein virtueller Prozessor, der ein Programm ausführt
 - Speicher → virtueller Adressraum
 - Prozessor → Zeitanteile am echten Prozessor
 - Interrupts → Signale
 - I/O-Schnittstellen → Dateisystem, Kommunikationsmechanismen
 - Maschinenbefehle → direkte Ausführung durch echten Prozessor
oder partielle Interpretation von Trap-Befehlen
durch Betriebssystemcode



■ Mehrprogrammbetrieb

- mehrere Prozesse können quasi gleichzeitig ausgeführt werden
- steht nur ein echter Prozessor zur Verfügung, werden Zeitanteile der Rechenzeit an die Prozesse vergeben (**Time Sharing System**)
- die Entscheidung, welcher Prozess zu welchem Zeitpunkt wieviel Rechenzeit zugeteilt bekommt, trifft das Betriebssystem (**Scheduler**)
- die Umschaltung zwischen Prozessen erfolgt durch das Betriebssystem (**Dispatcher**)
- Prozesse laufen nebenläufig
(das ausgeführte Programm weiß nicht, an welchen Stellen auf einen anderen Prozess umgeschaltet wird)

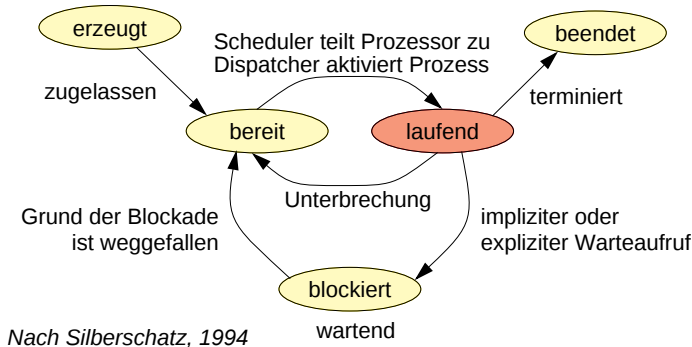


- Ein Prozess befindet sich in einem der folgenden Zustände:
 - **Erzeugt** (*New*)
Prozess wurde erzeugt, besitzt aber noch nicht alle nötigen Betriebsmittel
 - **Bereit** (*Ready*)
Prozess besitzt alle nötigen Betriebsmittel und ist bereit zum Laufen
 - **Laufend** (*Running*)
Prozess wird vom realen Prozessor ausgeführt
 - **Blockiert** (*Blocked/Waiting*)
Prozess wartet auf ein Ereignis (z.B. Fertigstellung einer Ein- oder Ausgabeoperation, Zuteilung eines Betriebsmittels, Empfang einer Nachricht); zum Warten wird er blockiert
 - **Beendet** (*Terminated*)
Prozess ist beendet; einige Betriebsmittel sind jedoch noch nicht freigegeben oder Prozess muss aus anderen Gründen im System verbleiben



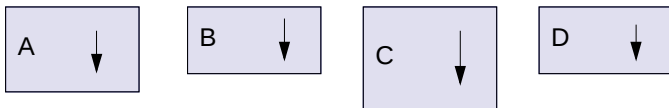
Prozesszustände (2)

■ Zustandsdiagramm



Prozesswechsel

■ Konzeptionelles Modell



vier Prozesse mit eigenständigen Befehlszählern

■ Umschaltung (*Context Switch*)

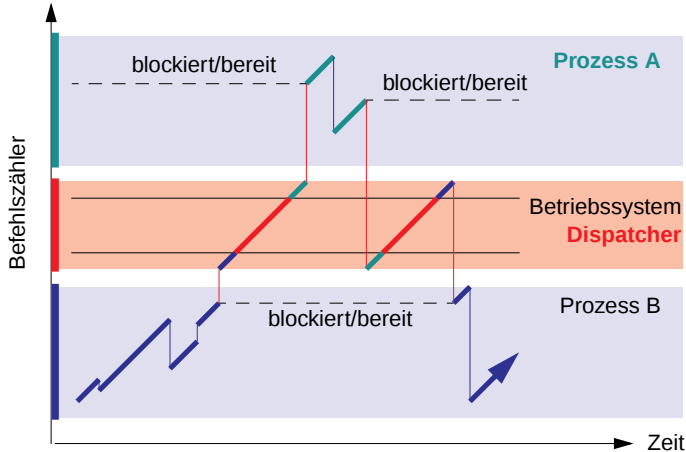
- Sichern der Register des laufenden Prozesses inkl. Programmzähler (Kontext),
- Auswahl des neuen Prozesses,
- Ablaufumgebung des neuen Prozesses herstellen (z.B. Speicherabbildung, etc.),
- gesicherte Register des neuen Prozesses laden und
- Prozessor aufsetzen.



Prozesswechsel (2)



Umschaltung



■ Prozesskontrollblock (*Process Control Block; PCB*)

- Datenstruktur des Betriebssystems, die alle nötigen Daten für einen Prozess hält.

Beispielsweise in UNIX:

- Prozessnummer (*PID*)
- verbrauchte Rechenzeit
- Erzeugungszeitpunkt
- Kontext (Register etc.)
- Speicherabbildung
- Eigentümer (*UID, GID*)
- Wurzelkatalog, aktueller Katalog
- offene Dateien
- ...



Prozesserzeugung (UNIX)

- Erzeugen eines neuen UNIX-Prozesses
- Duplizieren des gerade laufenden Prozesses

`pid_t fork( void );`

```
pid_t p;                Vater
...
p= fork();
if( p == (pid_t)0 ) {
    /* child */
    ...
} else if( p!=(pid_t)-1 ) {
    /* parent */
    ...
} else {
    /* error */
}
```



Prozesserzeugung (UNIX)

- Erzeugen eines neuen UNIX-Prozesses
- Duplizieren des gerade laufenden Prozesses

```
pid_t fork( void );
```

```
pid_t p;                                Vater
...
p= fork();
if( p == (pid_t)0 ) {
    /* child */
    ...
} else if( p!=(pid_t)-1 ) {
    /* parent */
    ...
} else {
    /* error */
}
```

```
pid_t p;                                Kind
...
p= fork();
if( p == (pid_t)0 ) {
    /* child */
    ...
} else if( p!=(pid_t)-1 ) {
    /* parent */
    ...
} else {
    /* error */
}
```



Prozesserzeugung (2)

- Der Kind-Prozess ist eine perfekte **Kopie** des Vaters
 - gleiches Programm
 - gleiche Daten (gleiche Werte in Variablen)
 - gleicher Programmzähler (nach der Kopie)
 - gleicher Eigentümer
 - gleiches aktuelles Verzeichnis
 - gleiche Dateien geöffnet (selbst Schreib-, Lesezeiger ist gemeinsam)
 - ...
- Unterschiede:
 - Verschiedene PIDs
 - **fork()** liefert verschiedene Werte als Ergebnis für Vater und Kind



Ausführen eines Programms (UNIX)

- Das von einem Prozess gerade ausgeführte Programm kann durch ein neues Programm ersetzt werden

```
int execv( const char *path, char *const argv[]);
```

Prozess A

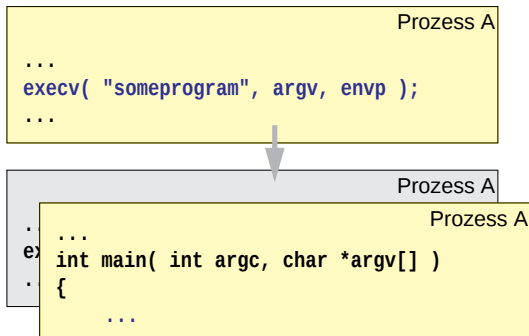
```
...  
execv( "someprogram", argv, envp );  
...
```



Ausführen eines Programms (UNIX)

- Das von einem Prozess gerade ausgeführte Programm kann durch ein neues Programm ersetzt werden

```
int execv( const char *path, char *const argv[]);
```



das zuvor ausgeführte Programm wird dadurch beendet.



Operationen auf Prozessen (UNIX)

■ Prozess beenden

```
void _exit( int status );  
[ void exit( int status ); ]
```

■ Prozessidentifikator

```
pid_t getpid( void );           /* eigene PID */  
pid_t getppid( void );        /* PID des Vaterprozesses */
```

■ Warten auf Beendigung eines Kindprozesses

```
pid_t wait( int *statusp );
```



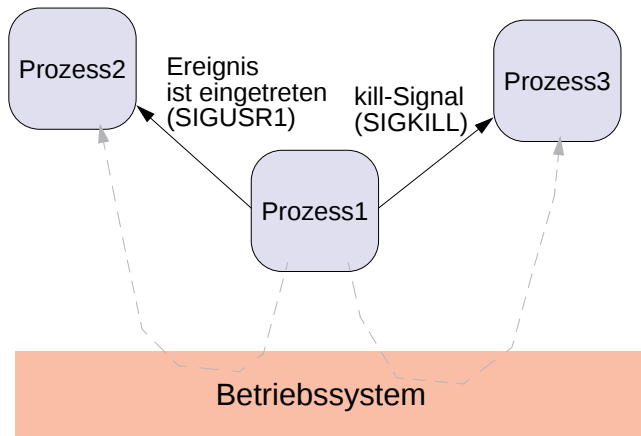
Signalisierung des Systemkerns an einen Prozess

- Software-Implementierung der Hardware-Konzepte
 - **Interrupt:** asynchrones Signal aufgrund eines "externen" Ereignisses
 - CTRL-C auf der Tastatur gedrückt (Interrupt-Signal)
 - Timer abgelaufen
 - Kind-Prozess terminiert
 - ...
 - **Trap:** synchrones Signal, ausgelöst durch die Aktivität des Prozesses
 - Zugriff auf ungültige Speicheradresse
 - Illegaler Maschinenbefehl
 - Division durch NULL
 - Schreiben auf eine geschlossene Kommunikationsverbindung
 - ...



Kommunikation zwischen Prozessen

- ein Prozess will einem anderen ein Ereignis signalisieren



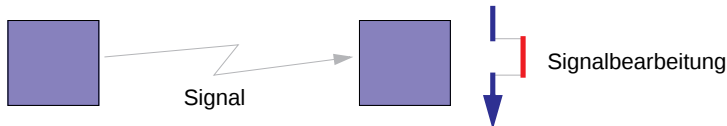
Reaktion auf Signale

- abort
 - erzeugt Core-Dump (Segmente + Registercontext) und beendet Prozess
- exit
 - beendet Prozess, ohne einen Core-Dump zu erzeugen
- ignore
 - ignoriert Signal
- stop
 - stoppt Prozess
- continue
 - setzt gestoppten Prozess fort
- signal handler
 - Aufruf einer Signalbehandlungsfunktion, danach Fortsetzung des Prozesses



POSIX Signalbehandlung

- Betriebssystemschnittstelle zum Umgang mit Signalen
- Signal bewirkt Aufruf einer Funktion (analog ISR)



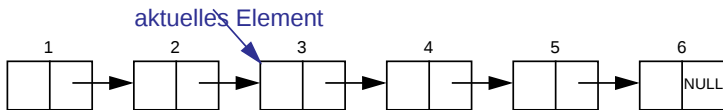
- nach der Behandlung läuft der Prozess an der unterbrochenen Stelle weiter
- Systemschnittstelle
 - sigaction – Anmelden einer Funktion = Einrichten der ISR-Tabelle
 - sigprocmask – Blockieren/Freigeben von Signalen $\approx$ cli() / sei()
 - sigsuspend – Freigeben + passives Warten auf Signal + wieder Blockieren $\approx$ sei() + sleep_cpu() + cli()
- kill – Signal an anderen Prozess verschicken



Signale und Nebenläufigkeit → Race Conditions

- Signale erzeugen Nebenläufigkeit innerhalb des Prozesses
- resultierende Probleme völlig analog zu Nebenläufigkeit bei Interrupts auf einem Mikrocontroller
- Beispiel:

■ main-Funktion läuft durch eine verkettete Liste



■ Prozess erhält Signal; Signalhandler entfernt Elemente 3 und 4 aus der Liste und gibt den Speicher dieser Elemente frei

