Vorlesung: Rechnerstrukturen, Teil 2 (Modul IP7)

J. Zhang

zhang@informatik.uni-hamburg.de

Universität Hamburg

Fachbereich Informatik

AB Technische Aspekte Multimodaler Systeme

Inhaltsverzeichnis

Ausnahmebehandlungen und Prozesse

(Exceptional Control Flow)

Themen:

Ausnahmebehandlungen und Prozesse

(Exceptional Control Flow)

Themen:

- Ausnahmen

Ausnahmebehandlungen und Prozesse

(Exceptional Control Flow)

Themen:

- Ausnahmen
- Process Context Switch

Ausnahmebehandlungen und Prozesse

(Exceptional Control Flow)

Themen:

- Ausnahmen
- Process Context Switch
- Erzeugen und Löschen von Prozessen

Kontrollfluss

Computer machen nur eines:

Kontrollfluss

Computer machen nur eines:

- Vom Hoch- bis zum Runterfahren wird jeweils nur eine Anweisungssequenz von der CPU gelesen und ausgeführt - und zwar Anweisung für Anweisung ("Befehlszyklus").

Kontrollfluss

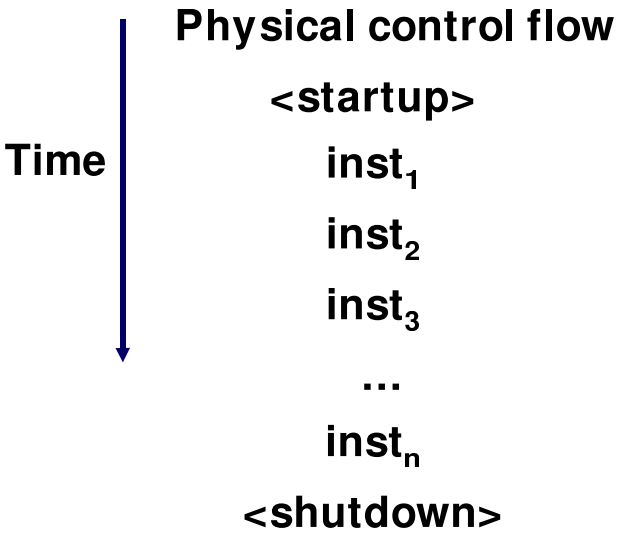
Computer machen nur eines:

- Vom Hoch- bis zum Runterfahren wird jeweils nur eine Anweisungssequenz von der CPU gelesen und ausgeführt - und zwar Anweisung für Anweisung ("Befehlszyklus").
- Diese Sequenz stellt den physikalischen Kontrollfluss des Systems dar.

Kontrollfluss

Computer machen nur eines:

- Vom Hoch- bis zum Runterfahren wird jeweils nur eine Anweisungssequenz von der CPU gelesen und ausgeführt - und zwar Anweisung für Anweisung ("Befehlszyklus").
- Diese Sequenz stellt den physikalischen Kontrollfluss des Systems dar.



Änderung des Kontrollflusses

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

- Sprünge und Verzweigungen (Jumps and Branches)
- Aufrufe und Rücksprünge unter Verwendung des Stacks

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

- Sprünge und Verzweigungen (Jumps and Branches)
- Aufrufe und Rücksprünge unter Verwendung des Stacks
- Beide reagieren auf Änderungen im Programmzustand

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

- Sprünge und Verzweigungen (Jumps and Branches)
- Aufrufe und Rücksprünge unter Verwendung des Stacks
- Beide reagieren auf Änderungen im Programmzustand

Unzureichend für ein sinnvolles System

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

- Sprünge und Verzweigungen (Jumps and Branches)
- Aufrufe und Rücksprünge unter Verwendung des Stacks
- Beide reagieren auf Änderungen im Programmzustand

Unzureichend für ein sinnvolles System

- Schwierig für die CPU, auf Änderungen im Systemzustand zu reagieren

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

- Sprünge und Verzweigungen (Jumps and Branches)
- Aufrufe und Rücksprünge unter Verwendung des Stacks
- Beide reagieren auf Änderungen im Programmzustand

Unzureichend für ein sinnvolles System

- Schwierig für die CPU, auf Änderungen im Systemzustand zu reagieren
  - Daten kommen von einer Festplatte oder einer Netzwerkkarte an
  - Division durch Null
  - Benutzereingabe von ctrl-c über die Tastatur
  - Systemuhr läuft ab

Änderung des Kontrollflusses

Bisher lediglich zwei Mechanismen, um den Kontrollfluss zu ändern

- Sprünge und Verzweigungen (Jumps and Branches)
- Aufrufe und Rücksprünge unter Verwendung des Stacks
- Beide reagieren auf Änderungen im Programmzustand

Unzureichend für ein sinnvolles System

- Schwierig für die CPU, auf Änderungen im Systemzustand zu reagieren
  - Daten kommen von einer Festplatte oder einer Netzwerkkarte an
  - Division durch Null
  - Benutzereingabe von ctrl-c über die Tastatur
  - Systemuhr läuft ab

System braucht Mechanismen für Ausnahmebehandlungen (Exceptional Control Flow)

Ausnahmebehandlungen

(Exceptional Control Flow)

Auf allen Ebenen eines Computersystems existieren Mechanismen für Ausnahmebehandlungen.

Ausnahmebehandlungen

(Exceptional Control Flow)

Auf allen Ebenen eines Computersystems existieren Mechanismen für Ausnahmebehandlungen.

Mechanismus auf niederen Ebenen

Ausnahmebehandlungen

(Exceptional Control Flow)

Auf allen Ebenen eines Computersystems existieren Mechanismen für Ausnahmebehandlungen.

Mechanismus auf niederen Ebenen

- Exceptions
  - Änderung des Kontrollflusses als Reaktion auf ein Systemereignis (also Änderung des Systemzustands)

Ausnahmebehandlungen

(Exceptional Control Flow)

Auf allen Ebenen eines Computersystems existieren Mechanismen für Ausnahmebehandlungen.

Mechanismus auf niederen Ebenen

- Exceptions
  - ✦ Änderung des Kontrollflusses als Reaktion auf ein Systemereignis (also Änderung des Systemzustands)
- Kombination von Hardware und OS-Software

Ausnahmebehandlungen (Forts.)

(Exceptional Control Flow)

Mechanismen auf höheren Ebenen

Ausnahmebehandlungen (Forts.)

(Exceptional Control Flow)

Mechanismen auf höheren Ebenen

- Prozessumschaltung

Ausnahmebehandlungen (Forts.)

(Exceptional Control Flow)

Mechanismen auf höheren Ebenen

- Prozessumschaltung
- Signale

Ausnahmebehandlungen (Forts.)

(Exceptional Control Flow)

Mechanismen auf höheren Ebenen

- Prozessumschaltung
- Signale
- Nichtlokale Sprünge (nonlocal Jumps)

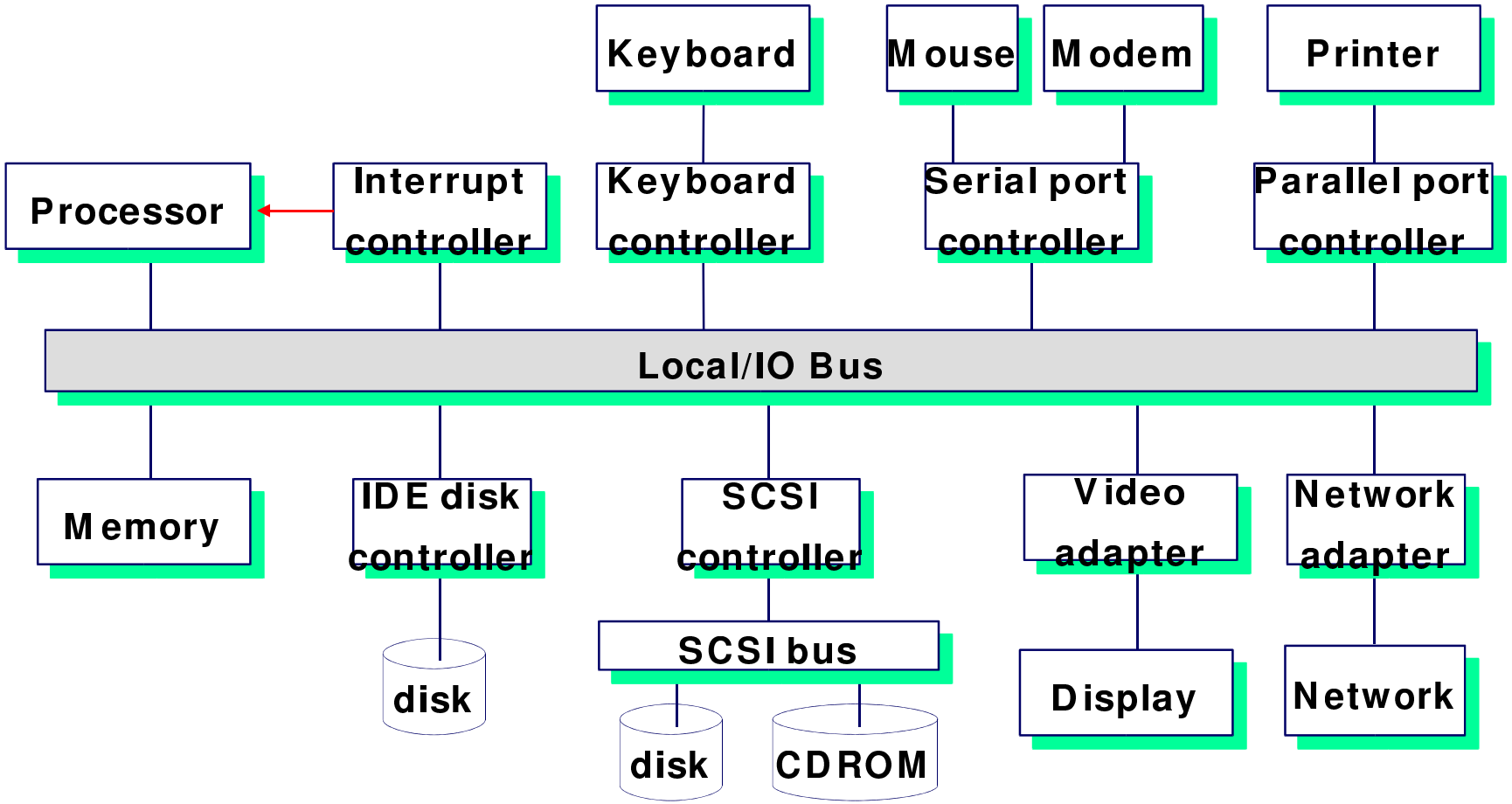
Ausnahmebehandlungen (Forts.)

(Exceptional Control Flow)

Mechanismen auf höheren Ebenen

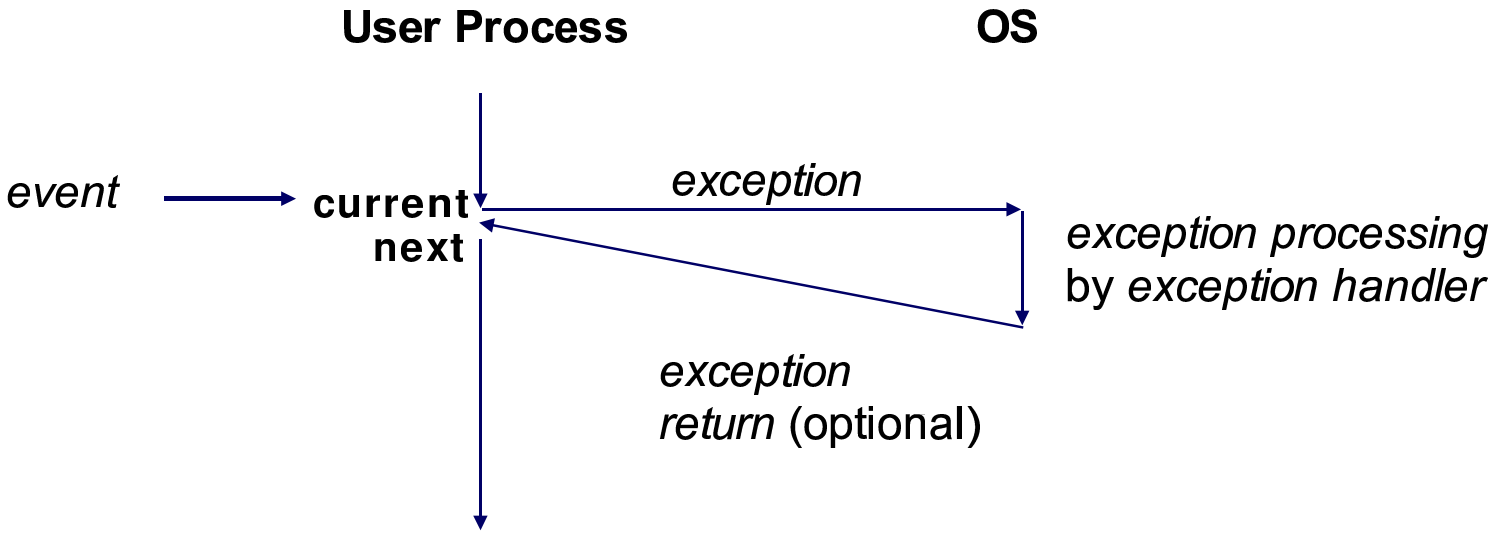
- Prozessumschaltung
- Signale
- Nichtlokale Sprünge (nonlocal Jumps)
- Implementiert durch
  - OS-Software (Context Switch und Signals)
  - oder C Runtime Library: nichtlokale Sprünge

Systemkontext für Exceptions

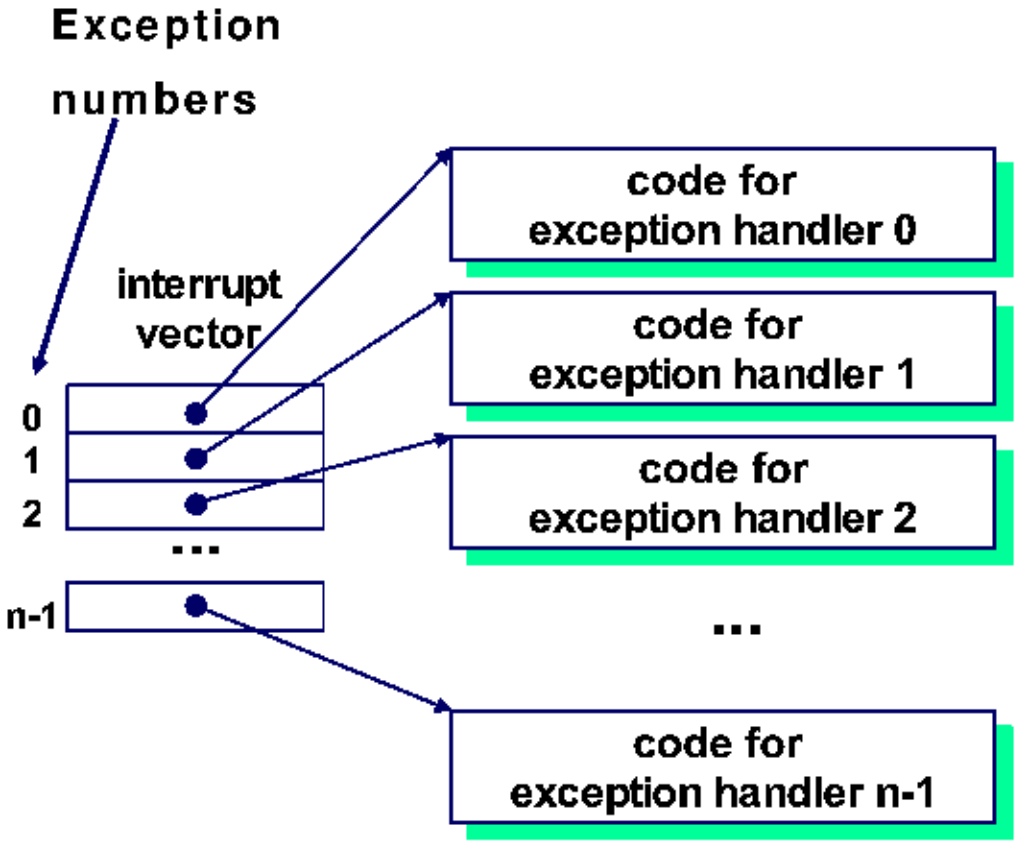


Exceptions

Eine Exception ist die Übergabe der Kontrolle an das OS als Reaktion auf ein Ereignis (d.h. Änderung des Prozessorzustandes)



Interrupt Vectors



- Each type of event has a unique exception number k
- Index into jump table (a.k.a., interrupt vector)
- Jump table entry k points to a function (exception handler).
- Handler k is called each time exception k occurs.

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

- werden durch Setzen der Prozessor-Interruptleitung angezeigt

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

- werden durch Setzen der Prozessor-Interruptleitung angezeigt
- Interruptroutine lässt Prozessor bei der nächsten Anweisung der unterbrochenen Befehlsfolge weitermachen.

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

- werden durch Setzen der Prozessor-Interruptleitung angezeigt
- Interruptroutine lässt Prozessor bei der nächsten Anweisung der unterbrochenen Befehlsfolge weitermachen.

Beispiele

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

- werden durch Setzen der Prozessor-Interruptleitung angezeigt
- Interruptroutine lässt Prozessor bei der nächsten Anweisung der unterbrochenen Befehlsfolge weitermachen.

Beispiele

- I/O Interrupts
  - ✦ Eingabe von ctrl-c über die Tastatur
  - ✦ Eintreffen eines Pakets über das Netzwerk
  - ✦ Eintreffen eines Datensektors von der Festplatte

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

- werden durch Setzen der Prozessor-Interruptleitung angezeigt
- Interruptroutine lässt Prozessor bei der nächsten Anweisung der unterbrochenen Befehlsfolge weitermachen.

Beispiele

- I/O Interrupts
  - Eingabe von ctrl-c über die Tastatur
  - Eintreffen eines Pakets über das Netzwerk
  - Eintreffen eines Datensektors von der Festplatte
- Hard Reset Interrupt
  - Drücken des Resetknopfes

Asynchrone Exceptions (Interrupts)

Verursacht durch Ereignisse außerhalb des Prozessors

- werden durch Setzen der Prozessor-Interruptleitung angezeigt
- Interruptroutine lässt Prozessor bei der nächsten Anweisung der unterbrochenen Befehlsfolge weitermachen.

Beispiele

- I/O Interrupts
  - Eingabe von ctrl-c über die Tastatur
  - Eintreffen eines Pakets über das Netzwerk
  - Eintreffen eines Datensektors von der Festplatte
- Hard Reset Interrupt
  - Drücken des Resetknopfes
- Soft Reset Interrupt
  - Eingabe von ctrl-alt-del über die Tastatur

Synchrone Exceptions

Verursacht durch Ereignisse, die als Ergebnis der Ausführung einer Anweisung auftreten

Synchrone Exceptions

Verursacht durch Ereignisse, die als Ergebnis der Ausführung einer Anweisung auftreten

- Traps
  - Beabsichtigt
  - Beispiele: Systemaufrufe, Breakpoint Traps und spezielle Instruktionen
  - Danach wird an der nächsten Anweisung der unterbrochenen Befehlsfolge weitergearbeitet

Synchrone Exceptions

Verursacht durch Ereignisse, die als Ergebnis der Ausführung einer Anweisung auftreten

- Traps
  - Beabsichtigt
  - Beispiele: Systemaufrufe, Breakpoint Traps und spezielle Instruktionen
  - Danach wird an der nächsten Anweisung der unterbrochenen Befehlsfolge weitergearbeitet
- Faults
  - Unbeabsichtig, aber möglicherweise behebbar
  - Beispiele: Seitenfehler (behebbar), Schutzverletzung (nicht behebbar)
  - Entweder neue Ausführung der fehlerverursachenden (aktuellen) Anweisung oder Abbruch

Synchrone Exceptions

Verursacht durch Ereignisse, die als Ergebnis der Ausführung einer Anweisung auftreten

- Traps
  - Beabsichtigt
  - Beispiele: Systemaufrufe, Breakpoint Traps und spezielle Instruktionen
  - Danach wird an der nächsten Anweisung der unterbrochenen Befehlsfolge weitergearbeitet
- Faults
  - Unbeabsichtig, aber möglicherweise behebbar
  - Beispiele: Seitenfehler (behebbar), Schutzverletzung (nicht behebbar)
  - Entweder neue Ausführung der fehlerverursachenden (aktuellen) Anweisung oder Abbruch
- Aborts
  - Unbeabsichtigt und nicht behebbar
  - Beispiele: Parity Fehler, Hardwarefehler
  - Bricht aktuelles Programm ab

Trap Beispiel

Öffnen einer Datei:

Trap Beispiel

Öffnen einer Datei:

- Benutzer ruft *open(filename, options)* auf
  - ✦ Funktion open führt den Assemblerbefehl *int* aus, der zu einem Systemaufruf führt

Trap Beispiel

Öffnen einer Datei:

- Benutzer ruft *open(filename, options)* auf
  - ✦ Funktion open führt den Assemblerbefehl *int* aus, der zu einem Systemaufruf führt
- OS muss Datei finden oder erzeugen und für Schreiben und Lesen vorbereiten

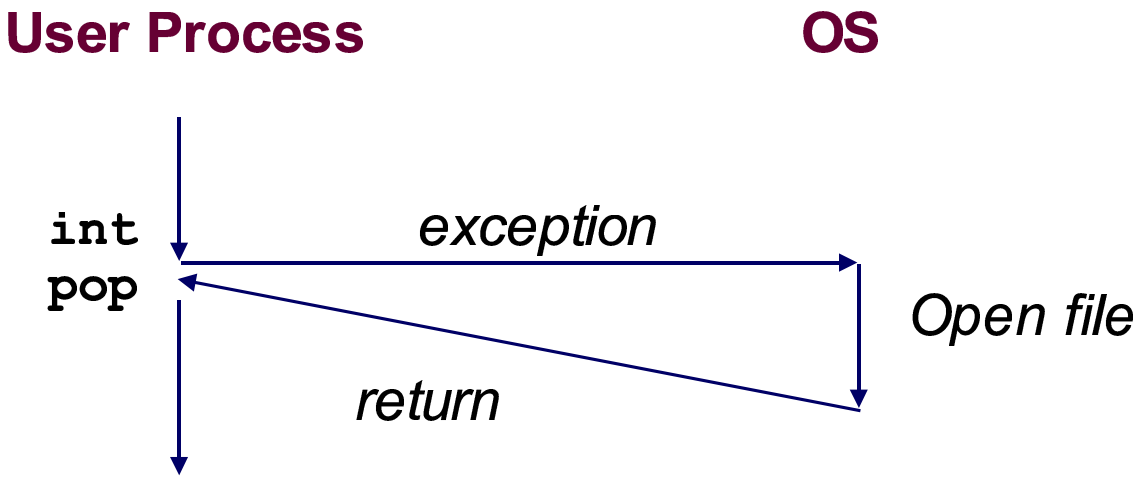
Trap Beispiel

Öffnen einer Datei:

- Benutzer ruft *open(filename, options)* auf
  - ✦ Funktion open führt den Assemblerbefehl *int* aus, der zu einem Systemaufruf führt
- OS muss Datei finden oder erzeugen und für Schreiben und Lesen vorbereiten
- Gibt Integer File Descriptor zurück

Trap Beispiel (Forts.)

```
0804d070 <__libc_open>:
 . . .
 804d082:      cd 80                    int    $0x80
 804d084:      5b                       pop    %ebx
 . . .
```



Fault Beispiel 1

Speicherzugriff:

Fault Beispiel 1

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse

Fault Beispiel 1

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Diese Seite des Benutzerspeichers ist auf die Festplatte ausgelagert

Fault Beispiel 1

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Diese Seite des Benutzerspeichers ist auf die Festplatte ausgelagert
- Seiten-Handler muss die Seite in den physikalischen Speicher laden

Fault Beispiel 1

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Diese Seite des Benutzerspeichers ist auf die Festplatte ausgelagert
- Seiten-Handler muss die Seite in den physikalischen Speicher laden
- Kehrt zur fehlerverursachenden Anweisung zurück

Fault Beispiel 1

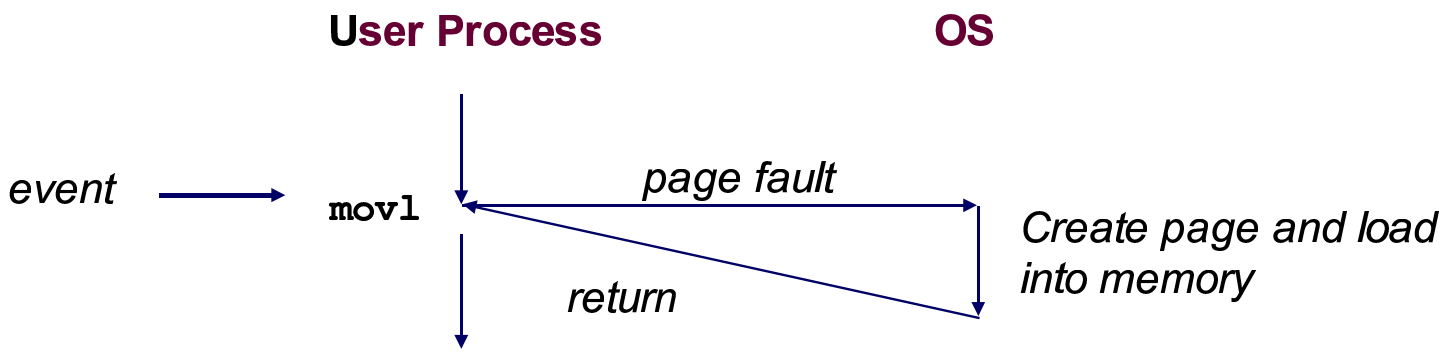
Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Diese Seite des Benutzerspeichers ist auf die Festplatte ausgelagert
- Seiten-Handler muss die Seite in den physikalischen Speicher laden
- Kehrt zur fehlerverursachenden Anweisung zurück
- Erfolgreicher zweiter Versuch

Fault Beispiel 1 (Forts.)

```
int a[1000];
main ()
{
    a[500] = 13;
}
```

```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



Fault Beispiel 2

Speicherzugriff:

Fault Beispiel 2

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse

Fault Beispiel 2

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Adresse ist ungültig

Fault Beispiel 2

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Adresse ist ungültig
- Seiten-Handler entdeckt ungültige Adresse

Fault Beispiel 2

Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Adresse ist ungültig
- Seiten-Handler entdeckt ungültige Adresse
- Sendet *SIGSEG*-Signal zum Benutzerprozess

Fault Beispiel 2

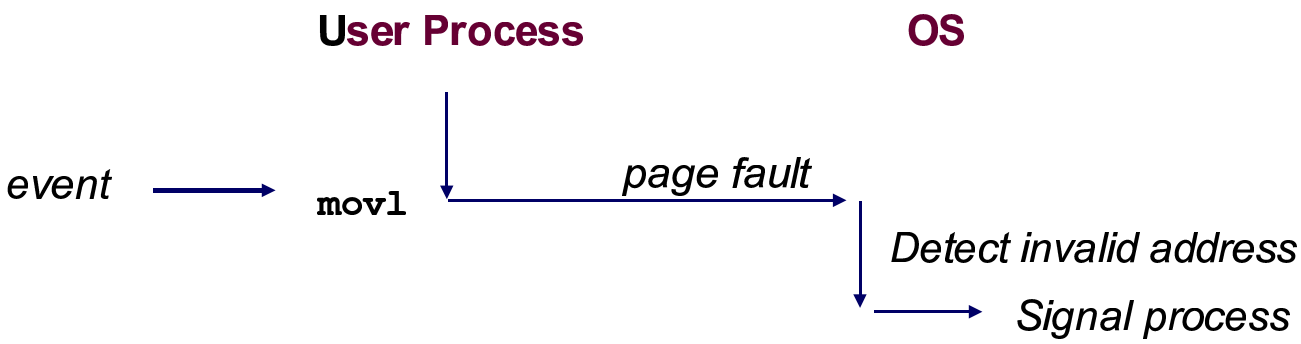
Speicherzugriff:

- Benutzer schreibt in eine Speicheradresse
- Adresse ist ungültig
- Seiten-Handler entdeckt ungültige Adresse
- Sendet *SIGSEG*-Signal zum Benutzerprozess
- Benutzerprozess wird mit "Segmentation Fault" beendet

Fault Beispiel 2 (Forts.)

```
int a[1000];
main ()
{
    a[5000] = 13;
}
```

```
80483b7:    c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

- eine der Grundideen der Informatik

Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

- eine der Grundideen der Informatik
- nicht das Gleiche wie “Programm” oder “Prozessor”

Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

- eine der Grundideen der Informatik
- nicht das Gleiche wie “Programm” oder “Prozessor”

Prozessbegriff stellt zwei wesentliche Abstraktionen zur Verfügung

Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

- eine der Grundideen der Informatik
- nicht das Gleiche wie “Programm” oder “Prozessor”

Prozessbegriff stellt zwei wesentliche Abstraktionen zur Verfügung

- Logischer Kontrollfluss
  - ✦ jedes Programm hat scheinbar exklusiven Zugriff auf die CPU

Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

- eine der Grundideen der Informatik
- nicht das Gleiche wie “Programm” oder “Prozessor”

Prozessbegriff stellt zwei wesentliche Abstraktionen zur Verfügung

- Logischer Kontrollfluss
  - ✦ jedes Programm hat scheinbar exklusiven Zugriff auf die CPU
- privater Adressraum
  - ✦ jedes Programm hat scheinbar exklusiven Zugriff auf den Hauptspeicher

Prozesse

Def.: Ein Prozess ist die Instanz eines laufenden Programms

- eine der Grundideen der Informatik
- nicht das Gleiche wie "Programm" oder "Prozessor"

Prozessbegriff stellt zwei wesentliche Abstraktionen zur Verfügung

- Logischer Kontrollfluss
  - ✦ jedes Programm hat scheinbar exklusiven Zugriff auf die CPU
- privater Adressraum
  - ✦ jedes Programm hat scheinbar exklusiven Zugriff auf den Hauptspeicher

Wie werden diese Illusionen aufrechterhalten?

- Prozessausführungen werden verschränkt (Multitasking)
- Adressraum wird durch das virtuelle Speichersystem verwaltet

Multitasking

Mehrere Prozesse laufen konkurrent auf dem System

Multitasking

Mehrere Prozesse laufen konkurrent auf dem System

- Prozess: Ausführen eines Programms
  - ✦ Zustand besteht aus Speicherbelegung, Registerwerten und Programmzähler

Multitasking

Mehrere Prozesse laufen konkurrent auf dem System

- Prozess: Ausführen eines Programms
  - ✦ Zustand besteht aus Speicherbelegung, Registerwerten und Programmzähler
- Wechselt ständig von einem Prozess zum anderen
  - ✦ Prozess wird suspendiert, wenn er auf I/O wartet oder Timer Event auftritt
  - ✦ Prozess wird entsprechend der Scheduling Strategie wiederaufgesetzt wenn I/O Operation abgeschlossen ist

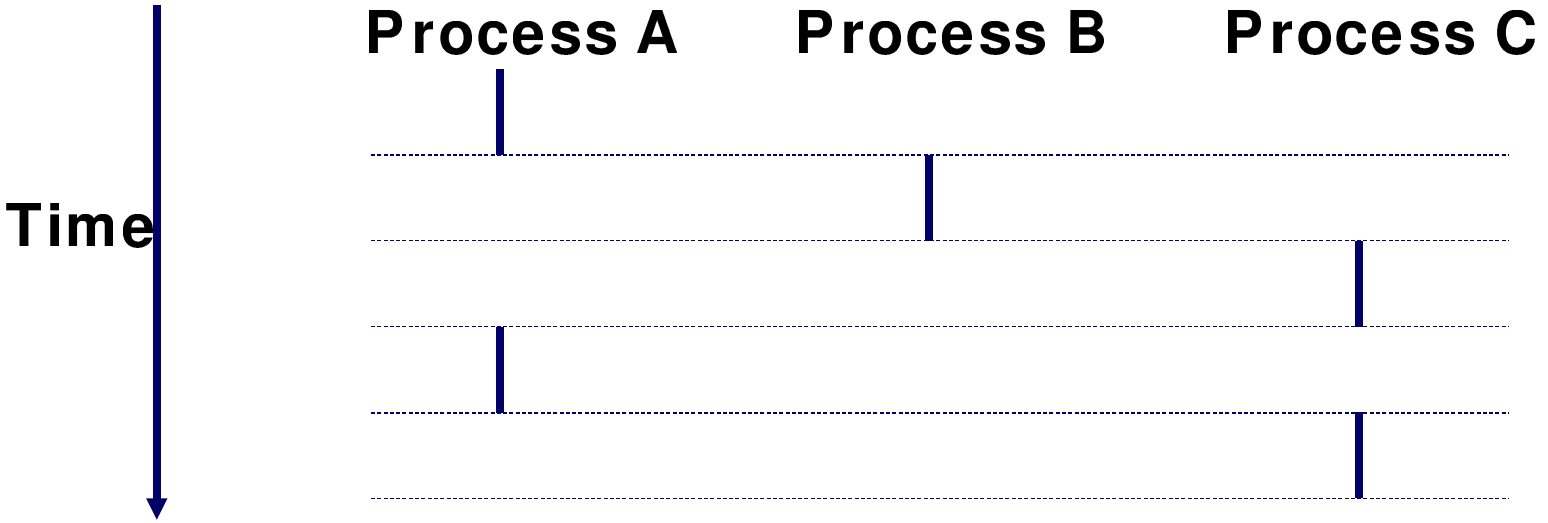
Multitasking

Mehrere Prozesse laufen konkurrent auf dem System

- Prozess: Ausführen eines Programms
  - ✦ Zustand besteht aus Speicherbelegung, Registerwerten und Programmzähler
- Wechselt ständig von einem Prozess zum anderen
  - ✦ Prozess wird suspendiert, wenn er auf I/O wartet oder Timer Event auftritt
  - ✦ Prozess wird entsprechend der Scheduling Strategie wiederaufgesetzt wenn I/O Operation abgeschlossen ist
- Stellt sich dem Benutzer dar, als würden alle Prozesse gleichzeitig ausgeführt
  - ✦ obwohl die meisten Syteme nur einen Prozess zur Zeit ausführen können
  - ✦ allerdings ist die Verarbeitungsleistung bedingt durch den Overhead geringer als würden sie wirklich alleine laufen

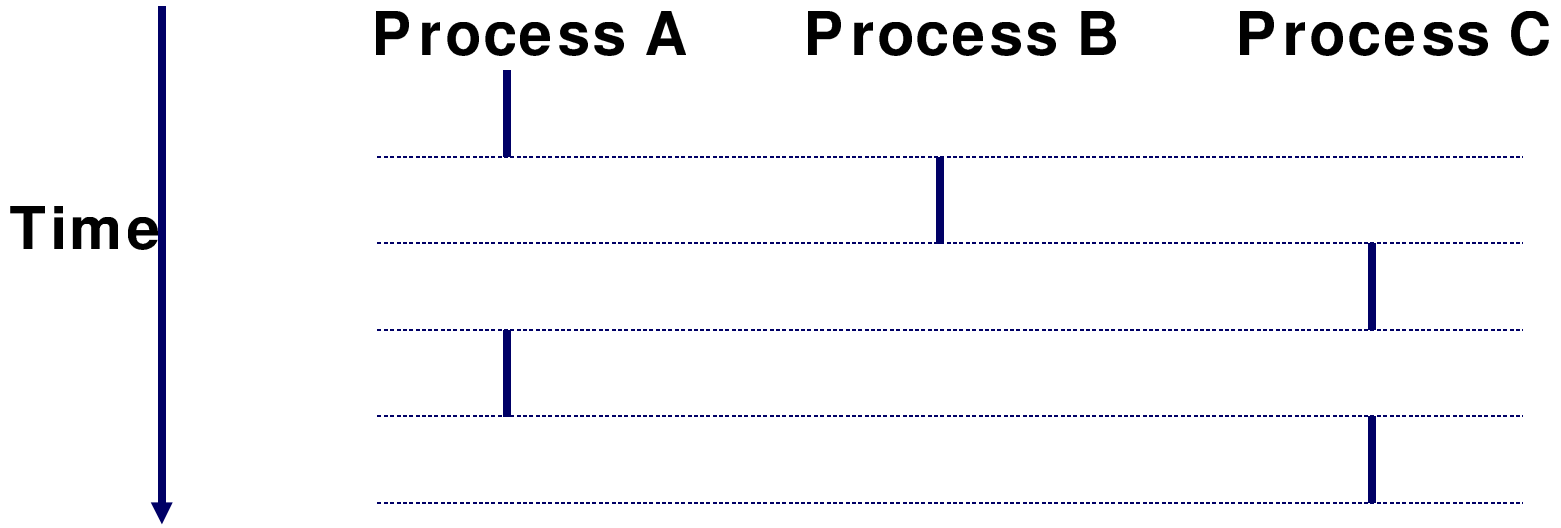
Logische Kontrollflüsse

Jeder Prozess hat seinen eigenen logischen Kontrollfluss.



Konkurrente Prozesse

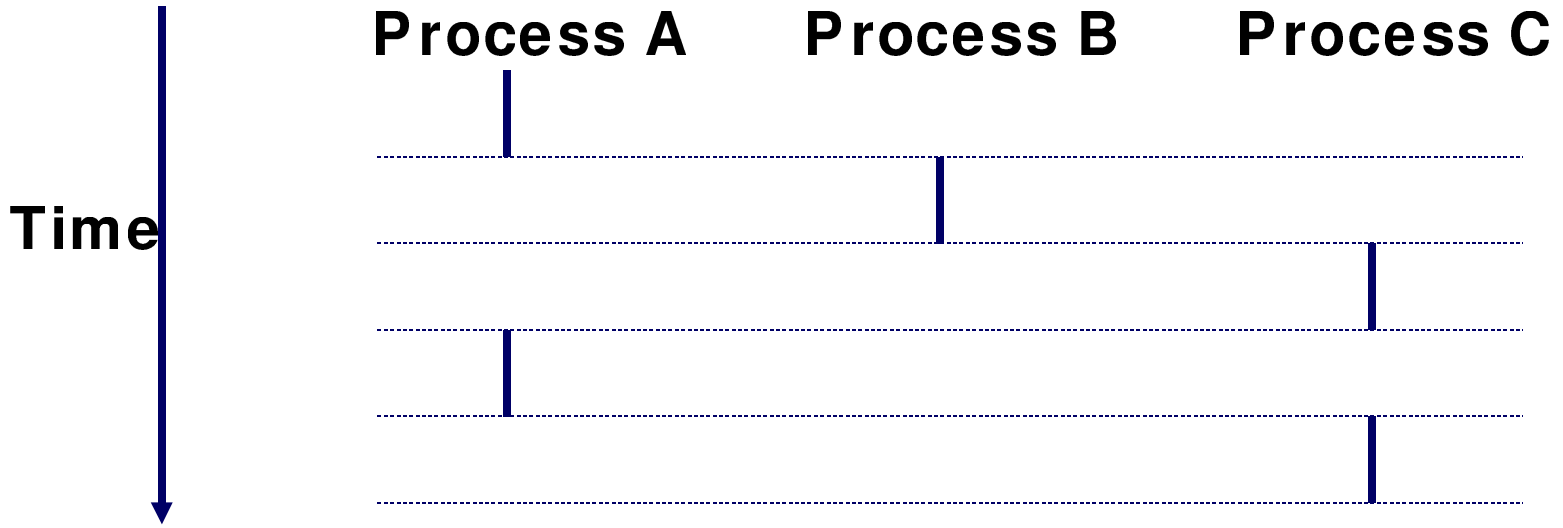
Zwei Prozesse laufen konkurrent, wenn ihre Kontrollflüsse sich zeitlich überlappen.



Konkurrente Prozesse

Zwei Prozesse laufen konkurrent, wenn ihre Kontrollflüsse sich zeitlich überlappen.

Ansonsten sind sie sequenziell.

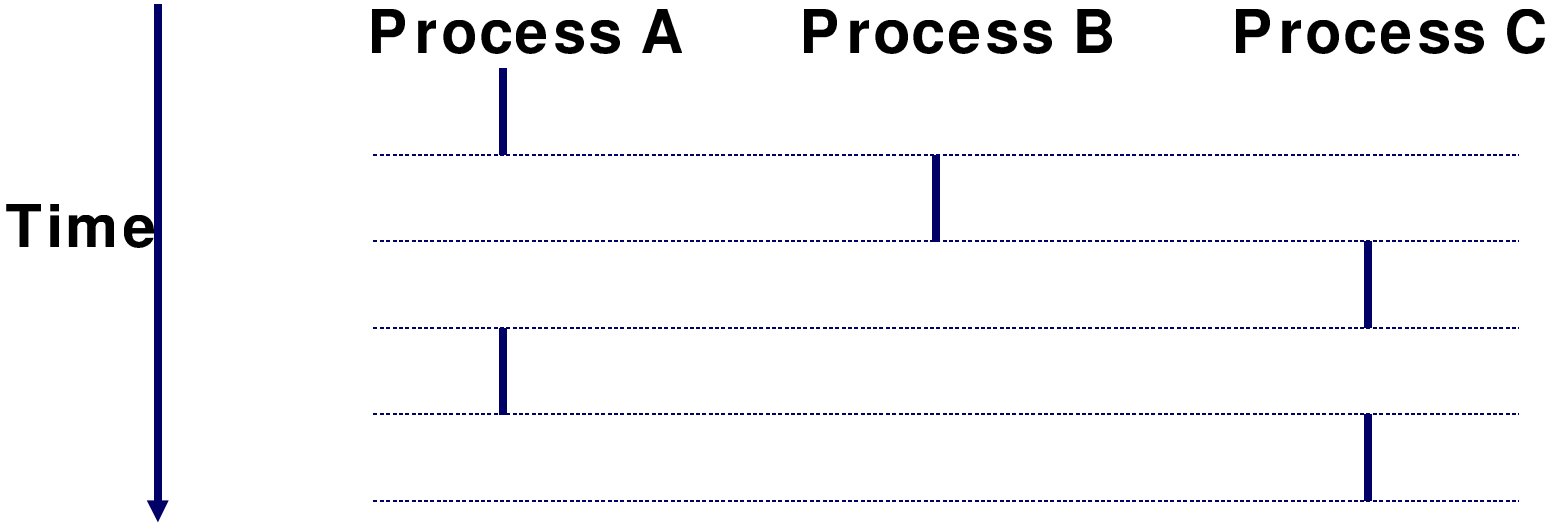


Konkurrente Prozesse

Zwei Prozesse laufen konkurrent, wenn ihre Kontrollflüsse sich zeitlich überlappen.

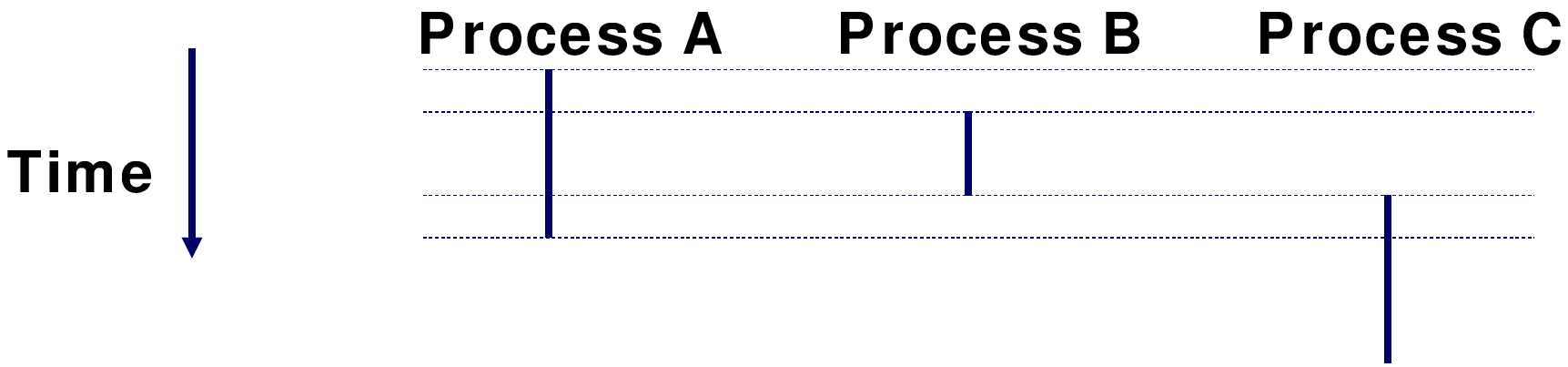
Ansonsten sind sie sequenziell.

Beispiele:
- Konkurrent: A & B, A & C
- Sequenziell: B & C



Benutzersicht auf konkurrente Prozesse

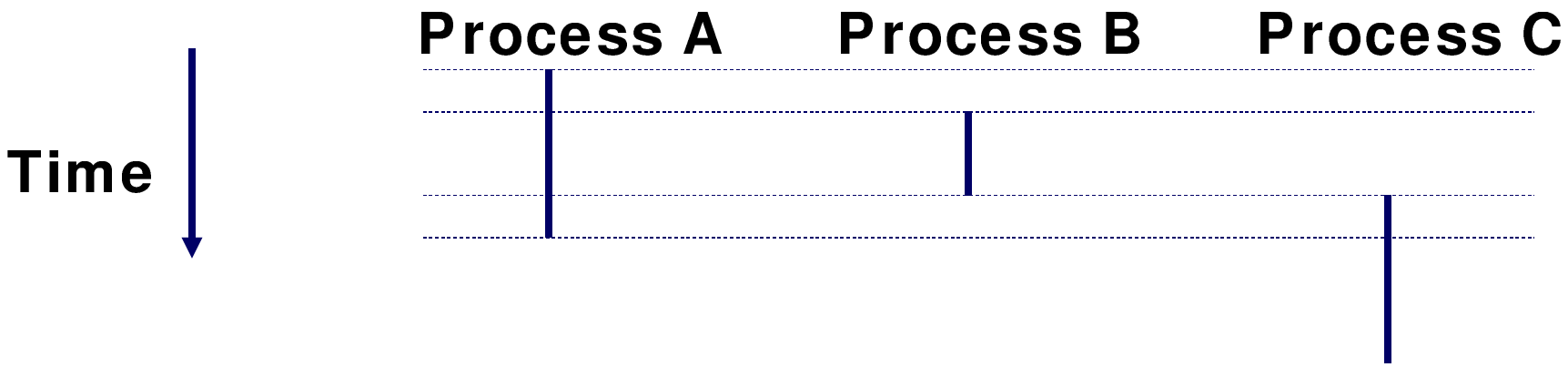
Kontrollflüsse für konkurrente Prozesse sind physikalisch getrennt in der Zeit.



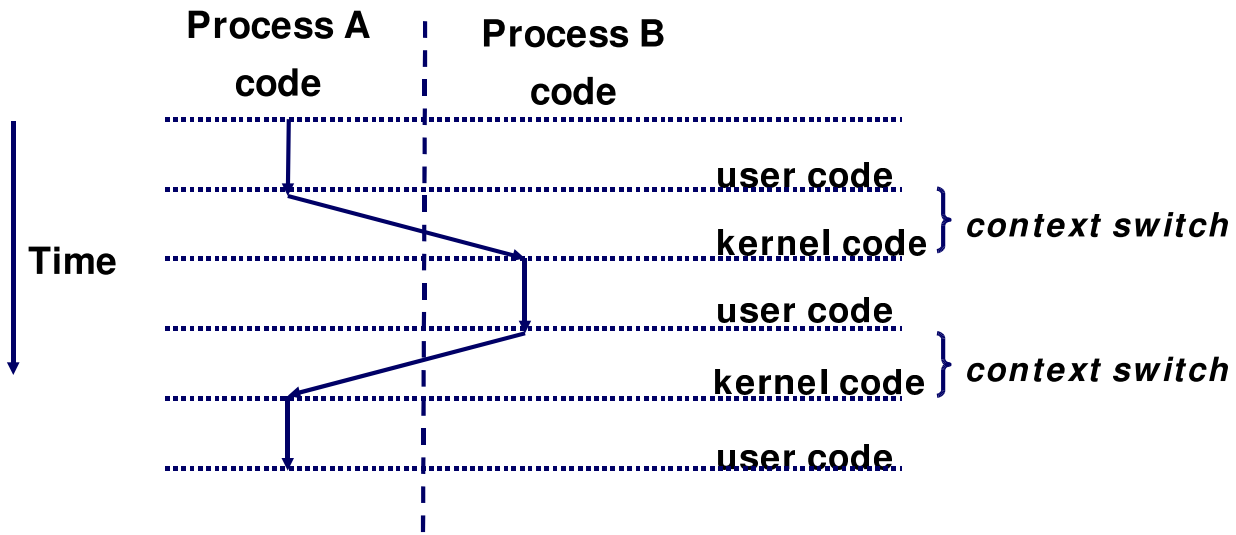
Benutzersicht auf konkurrente Prozesse

Kontrollflüsse für konkurrente Prozesse sind physikalisch getrennt in der Zeit.

Trotzdem kann man konkurrente Prozesse als parallel laufend bezeichnen.

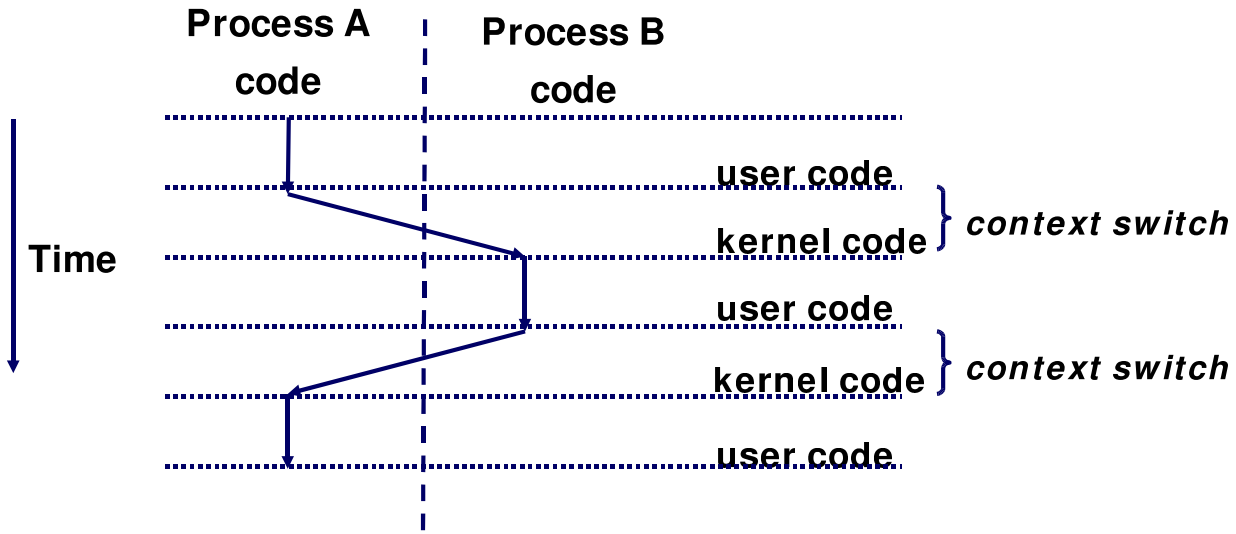


Context Switching



Context Switching

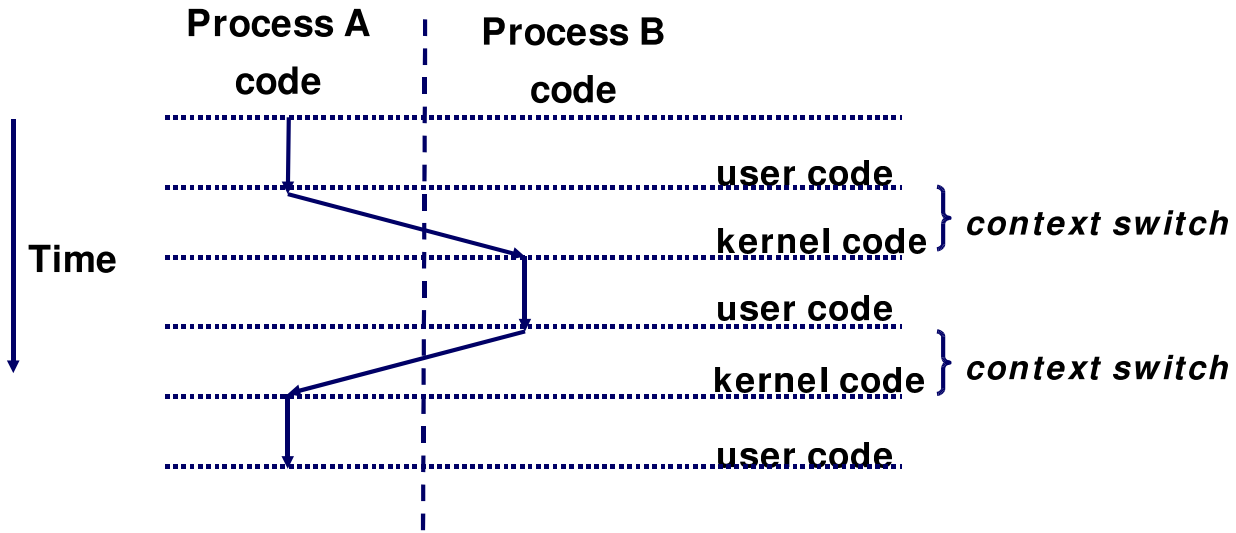
Prozesse werden vom *Kernel* verwaltet, einem gemeinsam genutzten OS Code-Block.



Context Switching

Prozesse werden vom *Kernel* verwaltet, einem gemeinsam genutzten OS Code-Block.

- Wichtig: Der Kernel ist kein eigenständiger Prozess, sondern läuft als Teil eines Benutzerprozesses.

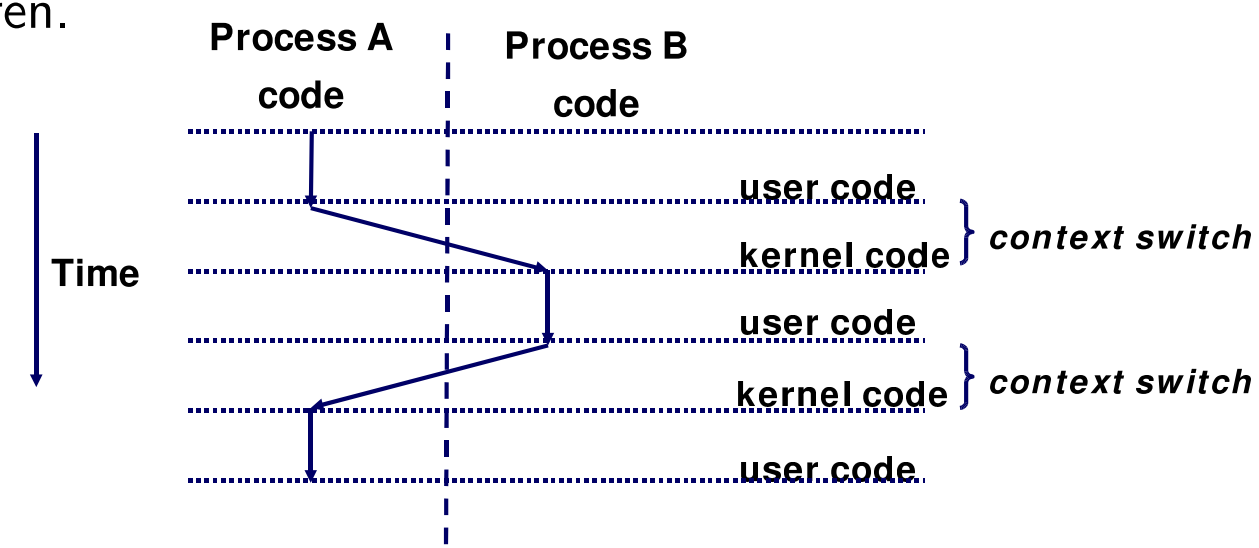


Context Switching

Prozesse werden vom *Kernel* verwaltet, einem gemeinsam genutzten OS Code-Block.

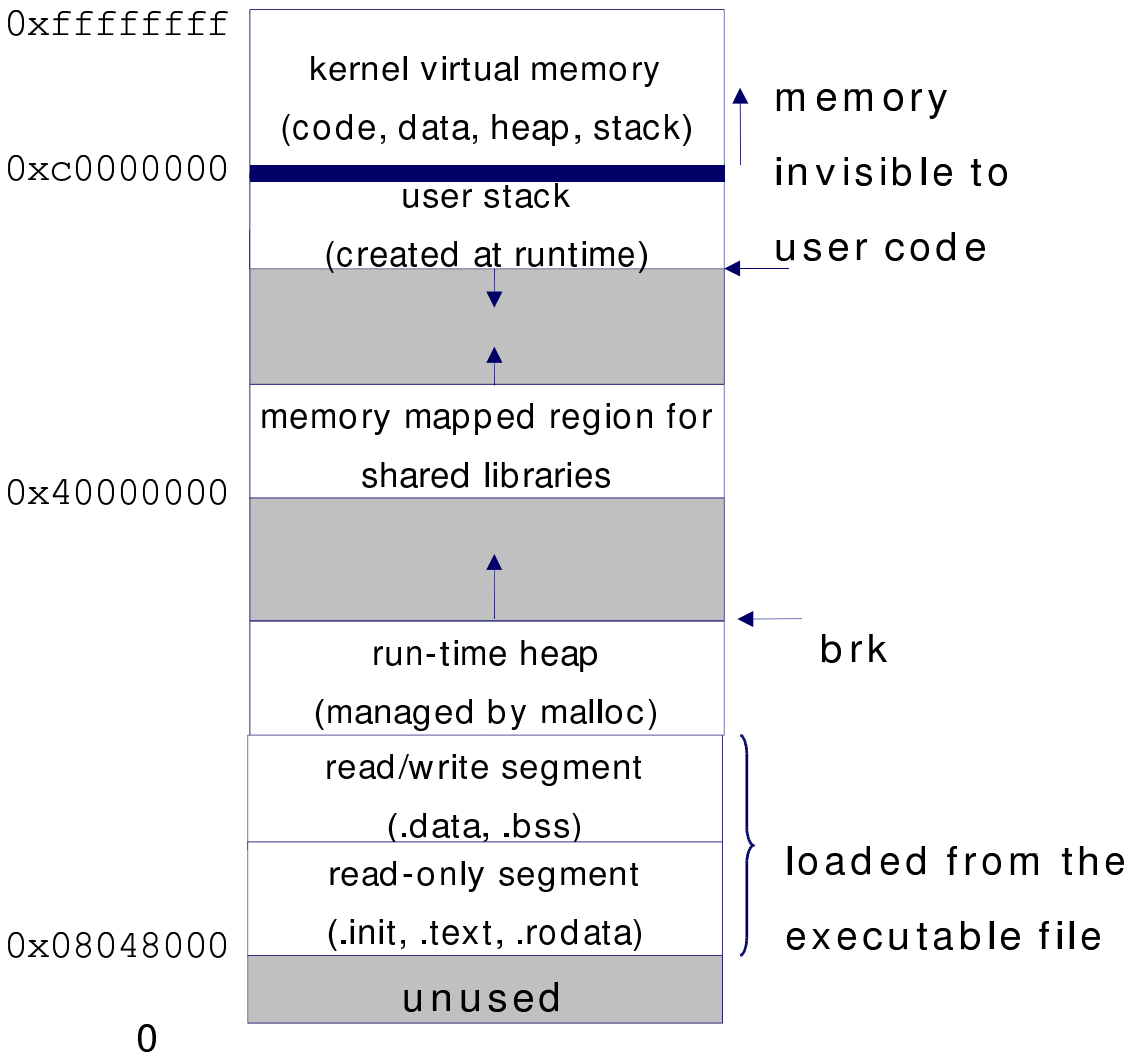
- Wichtig: Der Kernel ist kein eigenständiger Prozess, sondern läuft als Teil eines Benutzerprozesses.

Der Kontrollfluss wechselt über einen Context Switch von einem Prozess zum anderen.



Private Adressräume

Jeder Prozess
hat seinen eigenen
privaten Adressraum:



Zusammenfassung

Exceptions

Zusammenfassung

Exceptions

- Ereignisse, die einen abweichenden Kontrollfluss erfordern

Zusammenfassung

Exceptions

- Ereignisse, die einen abweichenden Kontrollfluss erfordern
- Generierung extern (Interrupts) oder intern (Traps und Faults)

Zusammenfassung

Exceptions

- Ereignisse, die einen abweichenden Kontrollfluss erfordern
- Generierung extern (Interrupts) oder intern (Traps und Faults)

Prozesse

Zusammenfassung

Exceptions

- Ereignisse, die einen abweichenden Kontrollfluss erfordern
- Generierung extern (Interrupts) oder intern (Traps und Faults)

Prozesse

- Zu jeder gegebenen Zeit sind mehrere Prozesse im System aktiv

Zusammenfassung

Exceptions

- Ereignisse, die einen abweichenden Kontrollfluss erfordern
- Generierung extern (Interrupts) oder intern (Traps und Faults)

Prozesse

- Zu jeder gegebenen Zeit sind mehrere Prozesse im System aktiv
- Nur jeweils einer kann zur Zeit ausgeführt werden

Zusammenfassung

Exceptions

- Ereignisse, die einen abweichenden Kontrollfluss erfordern
- Generierung extern (Interrupts) oder intern (Traps und Faults)

Prozesse

- Zu jeder gegebenen Zeit sind mehrere Prozesse im System aktiv
- Nur jeweils einer kann zur Zeit ausgeführt werden
- Jeder Prozess hat scheinbar volle Kontrolle über Prozessor und privaten Adressraum

Parallelrechner

Ständig steigende Anforderungen an die Rechenleistung, u.a. für:

Parallelrechner

Ständig steigende Anforderungen an die Rechenleistung, u.a. für:

- Wettervorhersage, Geologie
- Astronomie, Kernphysik, Gentechnologie
- Datenbanken, Transaktionssysteme

Parallelrechner

Ständig steigende Anforderungen an die Rechenleistung, u.a. für:

- Wettervorhersage, Geologie
- Astronomie, Kernphysik, Gentechnologie
- Datenbanken, Transaktionssysteme

- Performance eines einzelnen Rechners ist begrenzt
- Verteilen eines Programms auf mehrere Prozessoren

Parallelrechner

Ständig steigende Anforderungen an die Rechenleistung, u.a. für:

- Wettervorhersage, Geologie
- Astronomie, Kernphysik, Gentechnologie
- Datenbanken, Transaktionssysteme

- Performance eines einzelnen Rechners ist begrenzt
- Verteilen eines Programms auf mehrere Prozessoren

Vielfältige Möglichkeiten:

Parallelrechner

Ständig steigende Anforderungen an die Rechenleistung, u.a. für:

- Wettervorhersage, Geologie
- Astronomie, Kernphysik, Gentechnologie
- Datenbanken, Transaktionssysteme

- Performance eines einzelnen Rechners ist begrenzt
- Verteilen eines Programms auf mehrere Prozessoren

Vielfältige Möglichkeiten:

- Wie viele und welche Prozessoren?
- Kommunikation zwischen den Prozessoren?
- Programmierung und Hilfssoftware?

Performance

- Antwortzeit ("wall clock time", "response time", "execution time"): Gesamtzeit zwischen Programmstart und -ende, inkl. I/O

Performance

- Antwortzeit ("wall clock time", "response time", "execution time"):
  Gesamtzeit zwischen Programmstart und -ende, inkl. I/O
- Ausführungszeit (reine CPU-Zeit)

user-time	CPU-Zeit für Benutzerprogramm
system-time	CPU-Zeit für OS-Aktivitäten
`Unix: time make`	`7.950u 2.390s 0:22.98 44.9%`

Performance

- Antwortzeit ("wall clock time", "response time", "execution time"):
  Gesamtzeit zwischen Programmstart und -ende, inkl. I/O
- Ausführungszeit (reine CPU-Zeit)

user-time	CPU-Zeit für Benutzerprogramm
system-time	CPU-Zeit für OS-Aktivitäten
`Unix: time make`	`7.950u 2.390s 0:22.98 44.9%`

- Durchsatz: Anzahl bearbeiteter Programme / Zeit

Performance

- Antwortzeit ("wall clock time", "response time", "execution time"): Gesamtzeit zwischen Programmstart und -ende, inkl. I/O
- Ausführungszeit (reine CPU-Zeit)

user-time	CPU-Zeit für Benutzerprogramm
system-time	CPU-Zeit für OS-Aktivitäten
`Unix: time make`	`7.950u 2.390s 0:22.98 44.9%`

- Durchsatz: Anzahl bearbeiteter Programme / Zeit
- performance $= \frac{1}{execution\ time}$

Performance

- Antwortzeit ("wall clock time", "response time", "execution time"): Gesamtzeit zwischen Programmstart und -ende, inkl. I/O
- Ausführungszeit (reine CPU-Zeit)

user-time	CPU-Zeit für Benutzerprogramm
system-time	CPU-Zeit für OS-Aktivitäten
`Unix: time make`	`7.950u 2.390s 0:22.98 44.9%`

- Durchsatz: Anzahl bearbeiteter Programme / Zeit
- performance $= \frac{1}{execution\ time}$
- speedup $= \frac{performance\ x}{performance\ y} = \frac{execution\ time\ y}{execution\ time\ x}$

Performancegewinn

Ausführungszeit: Anzahl der Befehle * Zeit pro Befehl

Performancegewinn

Ausführungszeit: Anzahl der Befehle * Zeit pro Befehl

• weniger Befehle:	besserer Compiler mächtigere Befehle (CISC)
• weniger Zeit pro Befehl	einfachere Befehle (RISC) bessere Technologie Pipelining Caches
• parallele Ausführung	superskalar, SIMD, MIMD

Amdahls Gesetz:

"Speedup" durch Parallelisierung

Amdahls Gesetz:

"Speedup" durch Parallelisierung

System 1:	berechnet Funktion X, zeitlicher Anteil $0 < F < 1$
System 2:	Funktion X' ist schneller als X mit "speedup" SX: SX = Zeitbedarf(X) / Zeitbedarf(X')

Amdahls Gesetz: $S_{gesamt} = \frac{1}{(1-F)+\frac{F}{SX}}$

Amdahls Gesetz:

"Speedup" durch Parallelisierung

System 1:	berechnet Funktion X, zeitlicher Anteil $0 < F < 1$
System 2:	Funktion X' ist schneller als X mit "speedup" SX: SX = Zeitbedarf(X) / Zeitbedarf(X')

Amdahls Gesetz: $S_{gesamt} = \frac{1}{(1-F)+\frac{F}{SX}}$

$\longrightarrow$ Optimierung lohnt nur für häufige Operationen !!

Amdahls Gesetz:

"Speedup" durch Parallelisierung

System 1: berechnet Funktion X, zeitlicher Anteil $0 < F < 1$
System 2: Funktion X' ist schneller als X mit "speedup" SX:
SX = Zeitbedarf(X) / Zeitbedarf(X')

Amdahls Gesetz: $S_{gesamt} = \frac{1}{(1-F)+\frac{F}{SX}}$

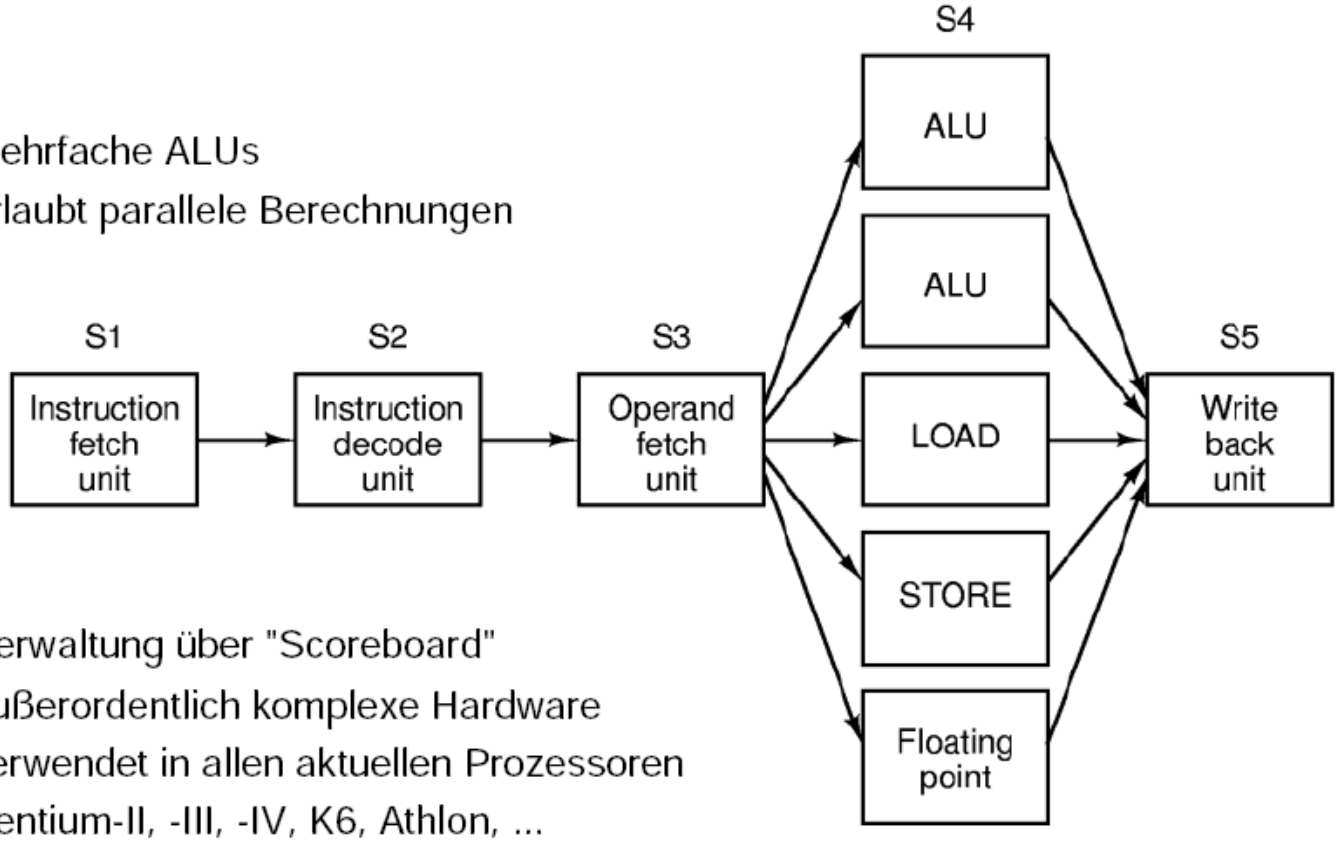
$\longrightarrow$ Optimierung lohnt nur für häufige Operationen !!

Beispiele:

SX = 1.1,	F = 0.98,	Sgesamt = 1/(0.02 + 0.89)	= 1.10
SX = 2,	F = 0.9,	Sgesamt = 1/(0.1 + 0.45)	= 1.82
SX = 2,	F = 0.5,	Sgesamt = 1/(0.5 + 0.25)	= 1.33
SX = 10,	F = 0.1,	Sgesamt = 1/(0.9 + 0.01)	= 1.09

Parallele Ausführung: Superskalarer Prozessor

- mehrfache ALUs
- erlaubt parallele Berechnungen



- Verwaltung über "Scoreboard"
- außerordentlich komplexe Hardware
- verwendet in allen aktuellen Prozessoren
- Pentium-II, -III, -IV, K6, Athlon, ...

Parallelrechner: Motivation

Leistungssteigerung durch schnellere Einzelprozessoren
Grenze: Takte oberhalb von 10 GHz derzeit unrealistisch

$\Rightarrow$ mehrere Prozessoren

- diverse Architekturkonzepte
- shared-memory vs. message-passing
- Overhead durch Kommunikation
- Programmierung ist ungelöstes Problem
- derzeit beliebtester Kompromiss:
  - ✦ bus-basierte SMPs mit 2..16 Prozessoren

SIMD: Flynn-Klassifikation

SIMD "single instruction, single data"
⇒ jeder klassische PC

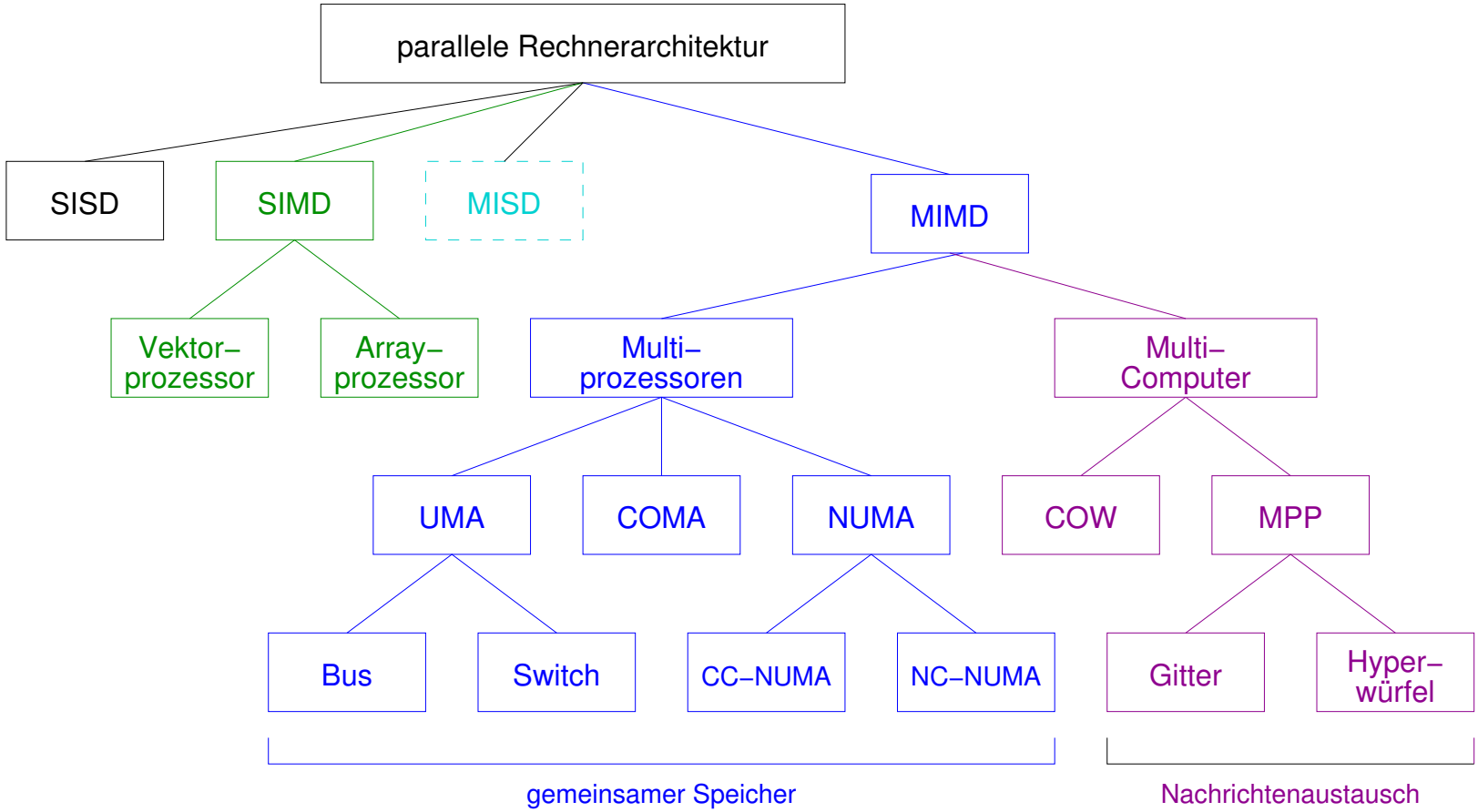
SIMD "single instruction, multiple data"
⇒ Feldrechner/Parallelrechner
⇒ z.B. Connection-Machine 2: 64K Prozessoren
⇒ eingeschränkt: MMX/SSE: 2-8 fach parallel

MIMD "multiple instruction, multiple data"
⇒ Multiprozessormaschinen
⇒ z.B. vierfach PentiumPro-Server

MISD . . .

Parallelrechner: Klassifikation

(Tanenbaum)



Literatur

[1] Randal E. Bryant and David O'Hallaron. *Computer Systems*. Pearson Education, Inc., New Jersey, 2003.

[2] David A. Patterson and John L. Hennessy. *Computer Organization and Design. The Hardware / Software Interface*. Morgan Kaufmann Publishers, Inc., San Francisco, 1998.

[3] Andrew S. Tanenbaum and James Goodman. *Computerarchitektur*. Pearson Studium München, 2001.