Vorlesung 5

# Interrupts

**Peter B. Ladkin**
ladkin@rvs.uni-bielefeld.de

# Kommunikation über den Bus

- CPU läuft mit einer Taktfrequenz
- I/O Geräte laufen mit anderen Frequenzen
- Die Propagation der Signale über den Bus ist allgemein langsamer als die CPU Taktfrequenz
- Ausnahme: die ersten Pentiums liefen mit der gleichen Frequenz wie der Bus (60MHz bzw. 66MHz)
- 1 Hz = 1 Taktschlag pro Sekunde

# Kommunikation über den Bus

- Die Pentiums laufen immer mit einem Multiplikator der Busfrequenz
- Der PCI-Bus läuft mit 33MHz
- PCIX läuft mit 66MHz
- Prozessoren laufen mit Frequenzen über 2 GHz (= 2.000MHz)

# Wie kommt I/O rein/raus?

- Auf der CPU läuft ein Programm
- Das Programm enthält einen Tastatur-Eingabebefehl
- z.B. "getc" (1 Byte); "input", "Textfield" (quasi-String, d.h. ein Array von Bytes)

# I/O per Tastatur

- Die CPU liest die Tastatur-Eingaberegister und leert danach die Register
- Das eingelesene Zeichen wird gespeichert
- Wird wiederholt, bis alle Zeichen gelesen worden sind
- Das "Wort" (Feld von Zeichen) wird von der CPU als Datenstruktur im Speicher abgelegt

# I/O per Tastatur

- Wie weiß die CPU, wenn es ein Zeichen zum Lesen gibt?
- Polling: CPU liest ständig die Tastatur-Eingaberegister: z.B.

```
i=10;
while (i > 0) do
    read(TEG-R);
    if char(TEG-R)then Store;
    decrement(i);
weitermachen
```

# I/O per Tastatur

- Sehr ineffizient!
- CPU-Taktfrequenz ist z.B. 100MHz
- Fingerfrequenz ist z.B. 5Hz (schnell!)
- Schleife wird 2 Sekunden laufen
- D.h., 200M Schleifen-Ausführungen
- Für nur 10 Bytes!

# Eingabe allgemein

- Nicht anders als bei der Tastatur
- Eingabe von der Festplatte ist schneller
- Sowie Eingabe vom Diskettenlaufwerk
- Alle sind sehr langsam in Vergleich zur Taktfrequenz der CPU
- Die Schleife heißt **Busy Waiting**
- **Busy Waiting:** Die CPU tut etwas, obwohl es nichts zu tun gibt

# Das Vermeiden von Busy Waiting

- "Textfield (10)" wird als Befehl gegeben
- Der Programzustand wird gespeichert
- Ein anderes Programm wird eingeholt
- (Macht der Scheduler, wie wir später sehen werden)
- Dieses andere Programm wird ausgeführt, bis es eine Tastatureingabe gibt

# Das Vermeiden von Busy Waiting

- Tastatureingabe in TEG-R wird "bekanntgegeben"
- Der Zustand vom zweiten Programm wird gespeichert
- Das erste Programm(-zustand) wird wieder hereingeholt
- Das Programm verarbeitet die Tastatureingabe

# Das Vermeiden von Busy Waiting

- Die Schleife heißt jetzt

```
i <- 10;
while i > 0 do
      read (TEG-R);
      Store;
      i <- i - 1;
weitermachen
```

- Die Schleife läuft nur 10 Mal durch

# "Bekanntgeben"

- Wie wird "bekanntgegeben"?
- Es gibt in der CPU eine Reihe von Bits, die **Interrupt Bits**
- z.B. ein Bit für die Tastatur
- Wenn irgendetwas in der TEG-R liegt, wird gleichzeitig das entsprechende Bit "gesetzt"
- Alles auf der Hardware-Ebene

# "Bekanntgeben"

- Am Ende eines Befehls-Zyklus.....
  (Wir erinnern uns, dass ein Maschinenbefehl aus einer Reihe von Mikrocodebefehlen besteht)
- .....werden die Interrupt Bits gelesen.
- Die CPU bemerkt, dass das Tastatur-I-Bit gesetzt worden ist
- Das entsprechende Programm wird hereingeholt

# Alternativen

- Es kann eine Reihe von Interrupt Bits geben
- Oder es kann ein (oder wenige) Interrupt Bit(s) geben, plus einem **Interrupt Vector** (ein Register, das eine Nummer enthält), der sagt, welches Gerät sich gemeldet hat
- Geräte-IV-Nummern sind 32-255 bei PCs
- Also 1-Byte-Register

# Interrupt Handler

- Für unterschiedliche Interrupts gibt es unterschiedliche Behandlungen
- z.B. bei einem vom Dekodierwerk ausgelösten "*invalid opcode*" (ungültiger Befehl) wird eine Sonderfehlermeldung ausgegeben
- z.B. bei einer Tastatureingabe wird das entsprechende Programm hereingeholt

# Interrupt Handler

- Es wird also beim Interrupt (gesetzt) eine Sondersoftware hereingeholt, die **Interrupt Handler** heißt
- Der entsprechende Interrupt Handler wird hereingeholt und ausgeführt
- Unterschiedliche Interrupts haben unterschiedliche Handler

# (Non)Maskable Interrupts

- Ein Interrupt wie "*invalid opcode*" ist **nonmaskable**, d.h. er wird immer gelesen und beachtet (der entsprechende Handler wird hereingeholt)
- Ein I/O Interrupt ist **maskable**, d.h. er kann zu bestimmten Zeiten (Zyklen) ignoriert werden
- ***"****Mask Interrupts*", *"Unmask Interrupts*" sind Sonderbefehle

# Maskable Interrupts

- Interrupts wie Tastatureingabe werden gemasked, wenn etwas sehr wichtiges läuft, das nie unterbrochen werden soll
- z.B.
  - ein Kernel-Panic bei Unix/Linux
  - Eine Lock-Sensor Eingabe-Leseroutine bei ABS
  - Wenn ein **Mutex**-Verfahren läuft ("Mutual Exclusion")

# Nonmaskable Interrupts

- Nonmaskable Interrupts sind die Interrupts, ohne deren Behandlung die CPU nicht richtig weiter laufen kann (oder weiter laufen soll)
- Für den Pentium
  - 0 divide error
  - 1 debug exception
  - 2 null interrupt
  - 3 breakpoint
  - 4 INTO-detected overflow

# Pentium Interrupts

- 5 bound range exception
- 6 invalid opcode
- 7 device not available
- 8 double fault
- 9 coprocessor segment overrun (reserved)
- 10 invalid task state segment
- 11 segment not present
- 12 stack fault
- 13 general protection

# Pentium Interrupts

- 14 page fault
- 15 (Intel reserved: do not use)
- 16 floating point error
- 17 alignment check
- 18 machine check
- 19-31 (Intel reserved; do not use)
- 32-255 maskable interrupts

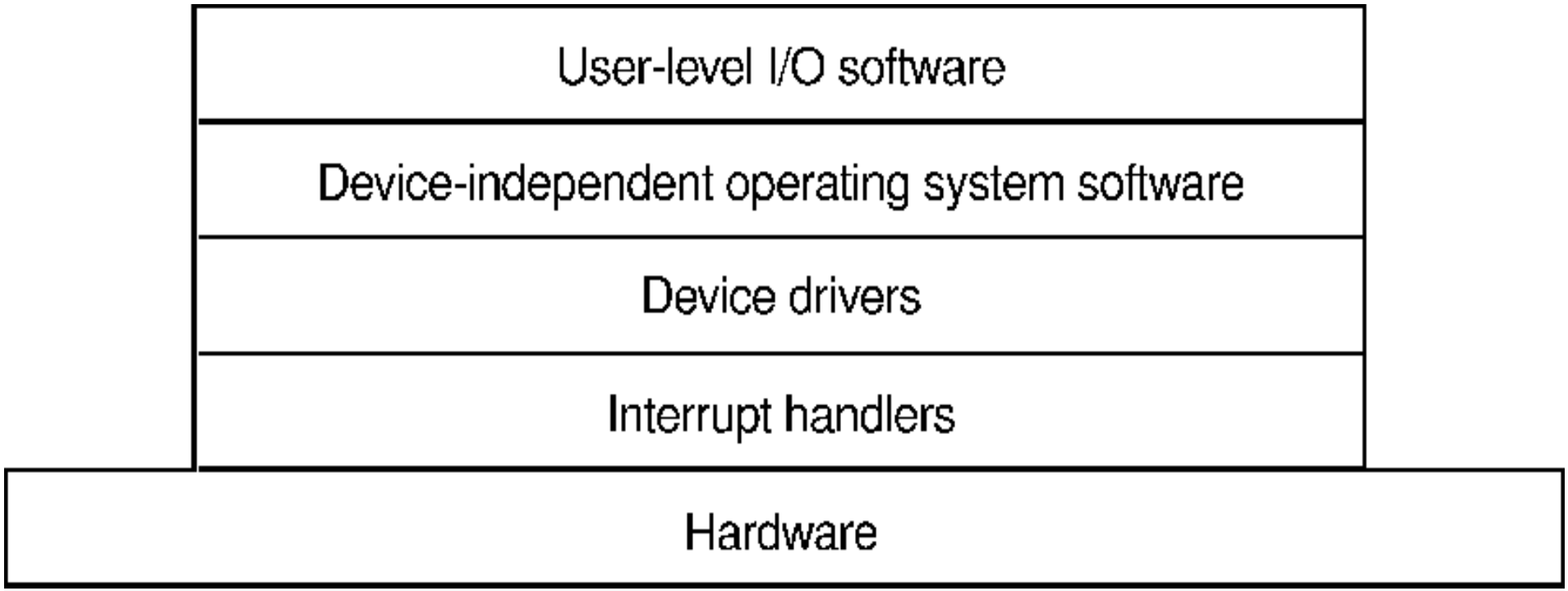
# Allgemeine Ausgabe

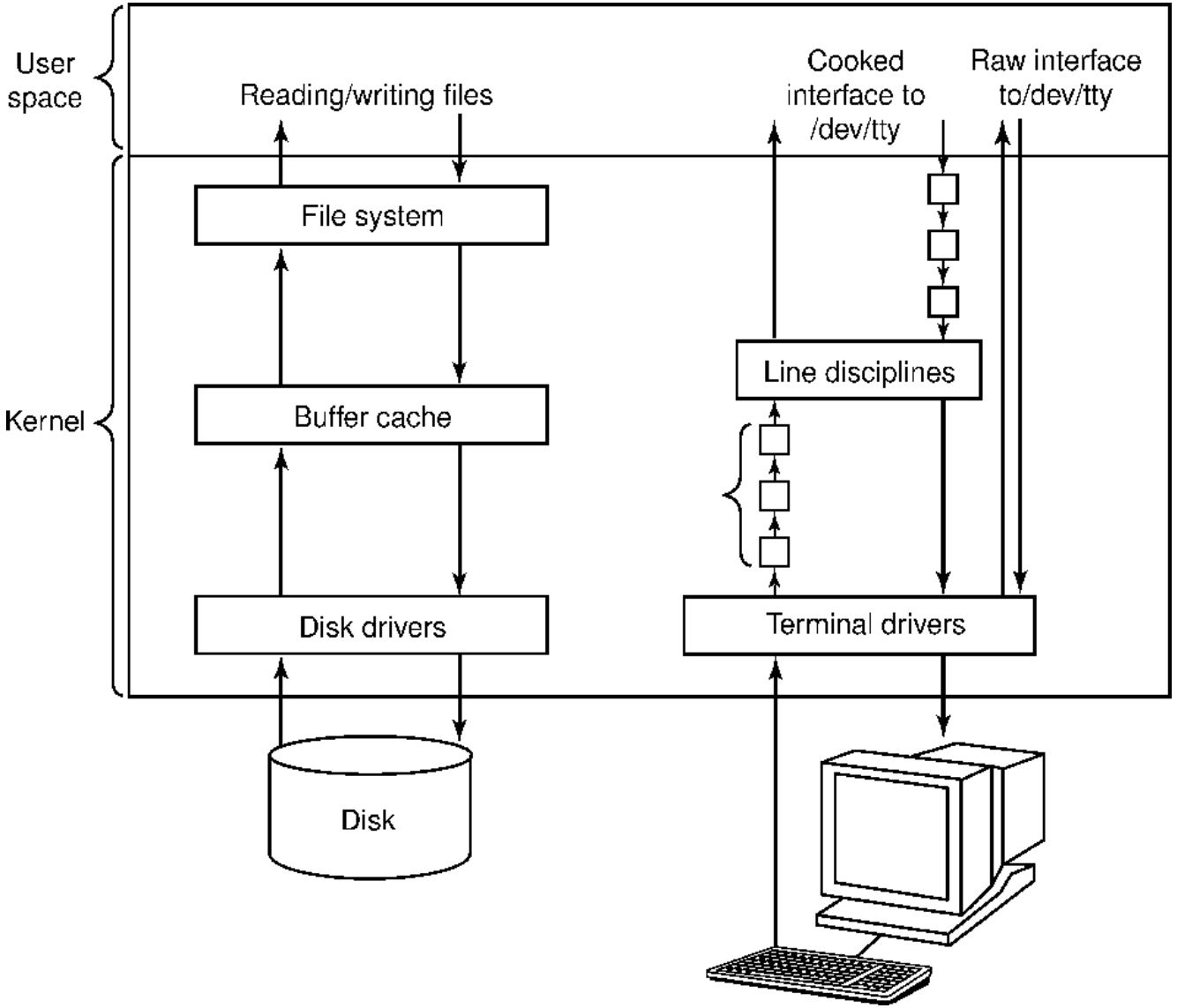
- Die CPU möchte etwas ausgeben
  - "print" usw. geben die Zeichen zum Monitor
- Ausgabe wird zwischengespeichert
- **Device Driver** Software wird hereingeholt
- Device Driver berechnet die entsprechenden Parameter für den Geräte-Controller
- ..und gibt die Parameter an das Gerät weiter

# Allgemeine Ausgabe

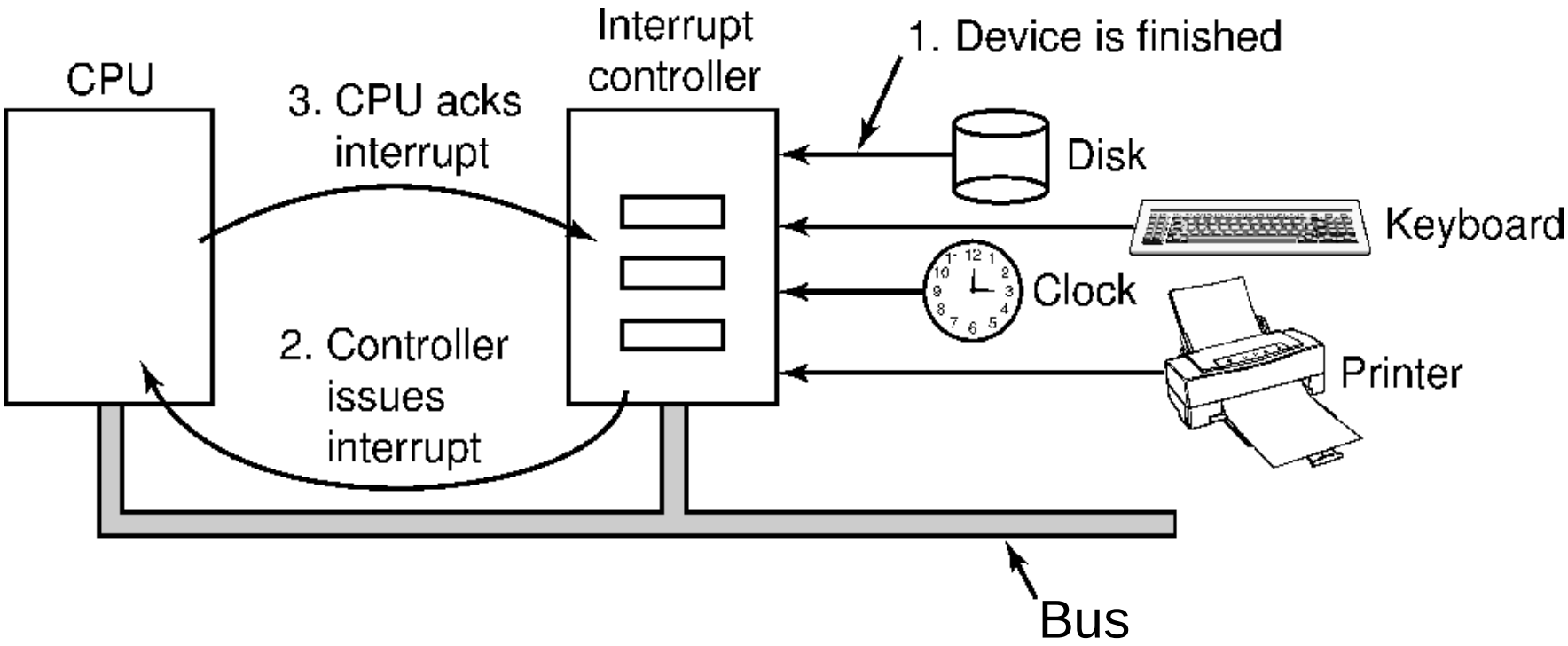
- Die **Device Driver** gehören zur Betriebssystem-Software
- Sie werden im "Kernel"-Programm des Betriebssystemes einkompiliert...
- ...oder sie liegen “nebenan” als priviligierte Software und werden vom Kernel aufgerufen
- Was bedeutet "nebenan"?

# Betriebssystem Software-Schichten




User space
Kernel
Reading/writing files
Cooked interface to /dev/tty
Raw interface to/dev/tty
File system
Buffer cache
Disk drivers
Line disciplines
Terminal drivers
Disk

# Ein Hardware Interrupt-Controller



# Interrupts mit Prioritäten

- I/O Interrupts können Prioritäten haben
- Die Prioritäten geben an, welcher Handler "Vorfahrt" hat, wenn gleichzeitig mehrere Interrupts gesetzt werden

# Interrupts mit Priotitäten

- Wenn mehrere Interrupts gleichzeitig gesetzt werden, wird der Handler des höchstpriorisierten Interrupts aufgerufen.
- Wenn ein Handler schon läuft, und ein Interrupt mit niedrigerer Priorität gesetzt wird, muss dieser Interrupt warten.
- Wird ein Interrupt mit höherer Priorität gesetzt, wird der Zustand des Handlers gespeichert und der Handler vom höchsten aufgerufen

# Interrupt-Verfahren mit Prioritäten

- Ein Drucker-Interrupt, gefolgt von...
- ...einem Serial-Port (RS232) Interrupt, gefolgt von.....
- ...einem Festplatten Interrupt
- Die Interrupts haben Prioriäten
- RS232 (Priorität 5); Festplatte (Priorität 4); danach Drucker (Priorität 2)

# Interrupt-Verfahren mit Prioritäten