Universität Bielefeld – Technische Fakultät
AG Rechnernetze und verteilte Systeme

Vorlesung 5: Interrupts

Wintersemester 2001/2002

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

Kommunikation über den Bus

- CPU läuft zu einer Taktfrequenz
- I/O Geräte laufen zu 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 zu einem Multiplikator der Busfrequenz
- PCI läuft zu 33MHz
- PCIX läuft zu 66MHz
- Prozessoren laufen bis zu 1.8GHz (= 1.800MHz)

Wie kommt I/O ein/aus?

- 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
- Der Charakter wird zwischengespeichert
- Und nochmal, bis alle Charakter gelesen worden sind
- Das "Wort" (Array von Character) wird von der CPU als Datenstruktur im Speicher gehandelt

I/O per Tastatur

- Wie weiss die CPU, wenn ein Charakter zum Lesen gibt?
- Polling: sie 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

- Aber dies ist sehr ineffizient
- CPU-Taktfrequenz ist, sagen wir, 100MHz
- Fingerfrequenz ist, sagen wir 5Hz (schnell!)
- Loop wird 2 Sekunden laufen
- D.h., 200M Schleife-Ausführung
- Fur nur 10 Bytes!

Eingabe allgemein

- Es ist nicht anders wie bei der Tastatur
- Festplatteneingabe ist schneller
- Sowie Diskettenlaufwerkeingabe
- Alle sind sehr langsam in Vergleich mit der CPU Taktfrequenz
- Die Schleife heisst **Busy Waiting**
- **Busy Waiting:** die CPU tut was, obwohl es nichts zu tun gibt

Das Vermeiden von Busy Waiting

- Wird "Textfield (10)" als Befehl gegeben
- Der Programzustand wird gespeichert
- Ein anderes Programm wird eingeholt
- (Alles 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 tut etwas mit der 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

"Bekanntgegeben"

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

"Bekanntgegeben"

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

Alternativen

- Es könnte eine Reihe von Interrupt Bits geben
- Oder es könnte ein (oder wenige) Interrupt Bit(s) geben, plus ein **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

- Da es unterschiedliche Interrupts gibt, gibt es auch unterschiedliche Verfahren
- z.B., das Verfahren für ein "*invalid opcode*" (ungültiges Befehl), das von dem Dekodier-Steuerwerk beschlossen wird, besteht aus einer Sonderfehlermeldung
- *z.B., das Verfahren für eine Tastatureingabe ist, das entsprechende Programm hereinholen*

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 wird* maskable**, d.h., er wird zu bestimmten Zeiten (Zyklen) ignoriert
- *"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
  - Ein Lock-Sensor Eingabe-Leseroutine bei ABS
  - Wenn ein **Mutex**-Verfahren läuft ("Mutual Exclusion")

Nonmaskable Interrupts

- Nonmaskable Interrupts sind die Interrupts, ohne deren Handlung 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

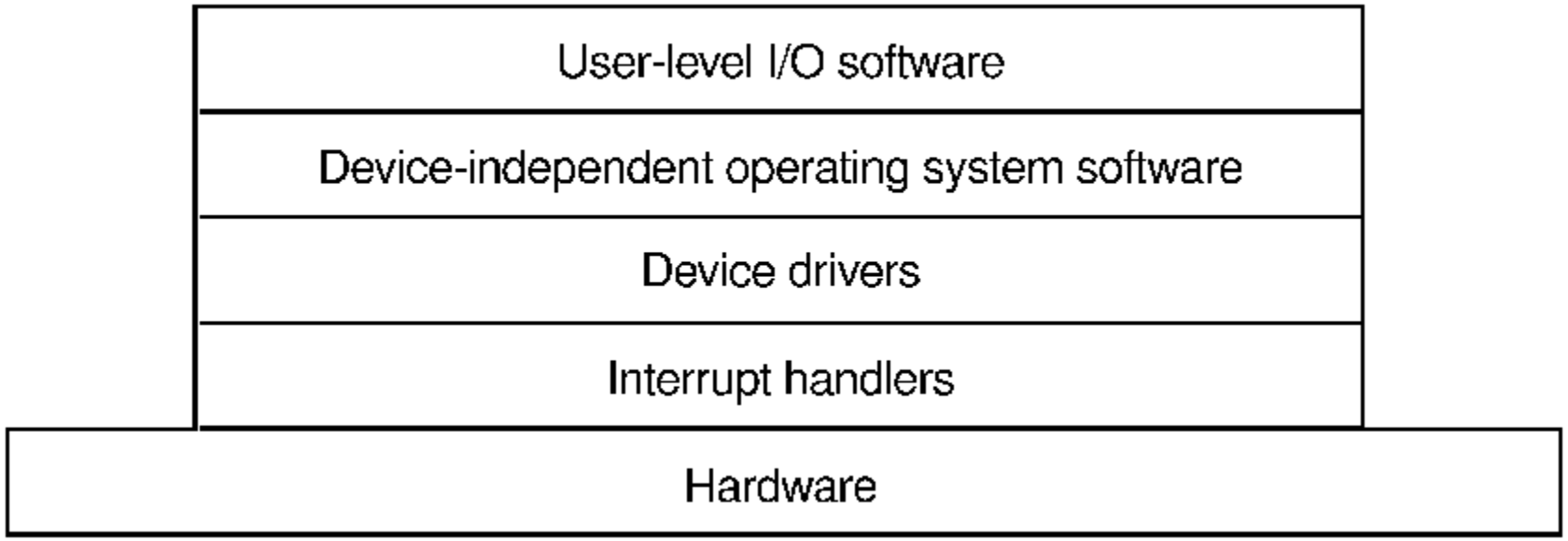
Allgemeine Ausgabe

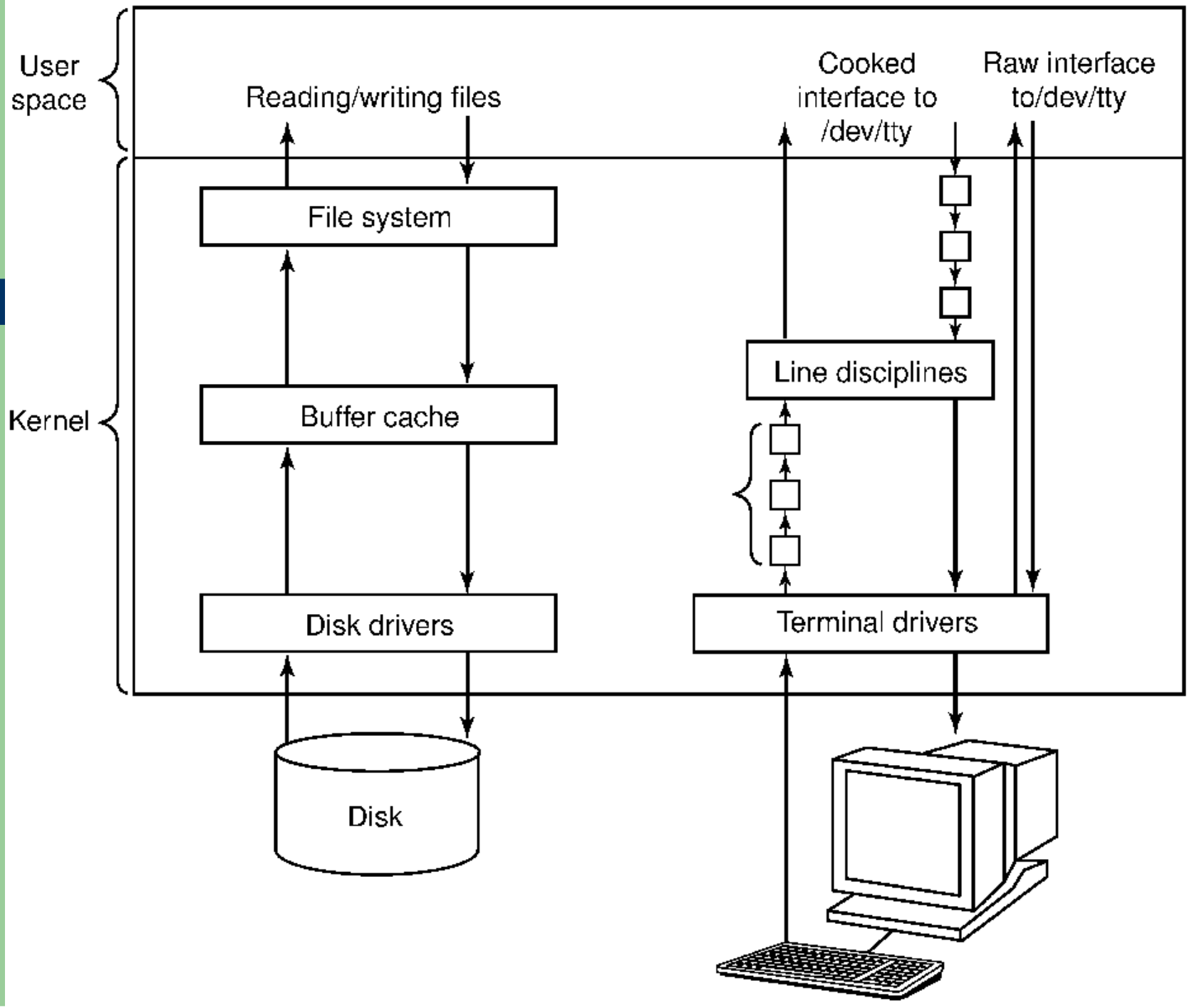
- Die CPU möchte was ausgeben
  - "print" usw: geben die Character zum Monitor
- Ausgabe wird zwischengespeichert
- **Device Driver** Software wird hereingeholt
- **Device Driver berechnet die entsprechende Parameter für das Geräte-Controller**
- **..und gibt die Parameter an das Gerät weiter**

Allgemeine Ausgabe

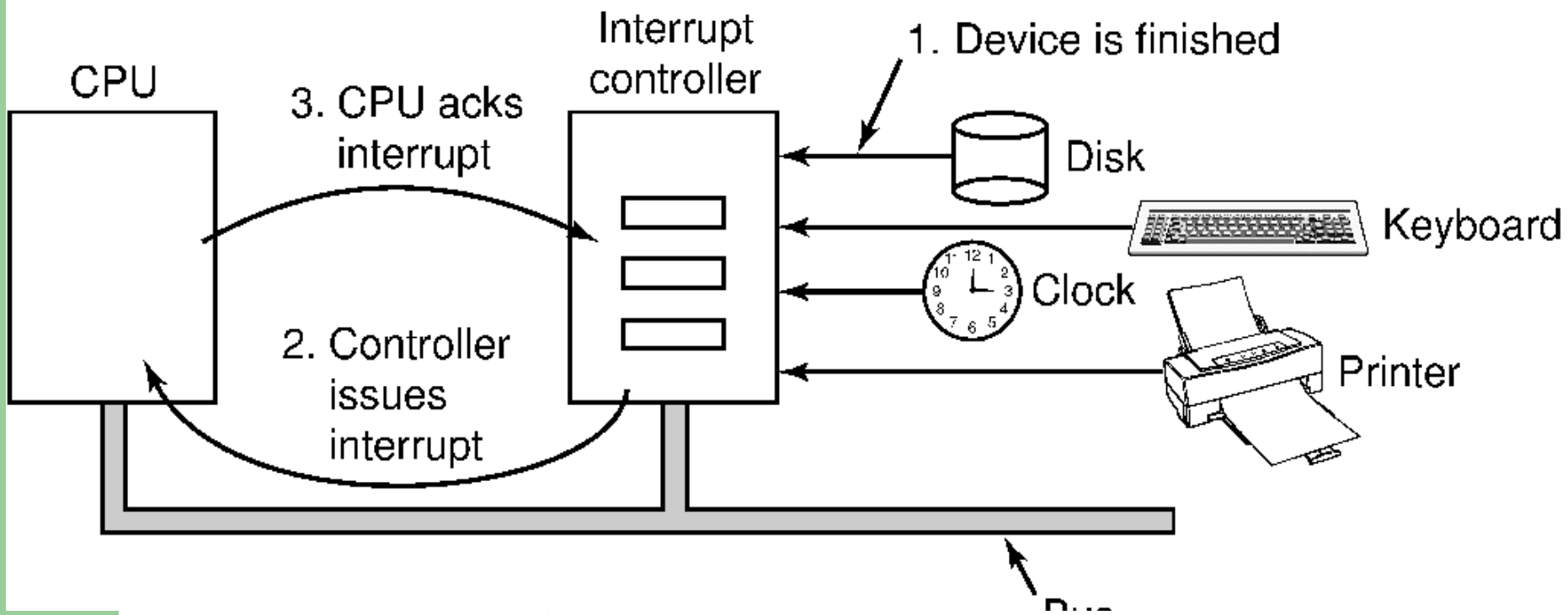
- Die Device Drivers gehören zur Betriebssystem-Software
- Die DDrs werden im "Kernel"-Programm des Betriebssystemes eincompiliert...
- ...oder sie liegen nebenan als priviligierte Software und werden vom Kernel angerufen
- Was "nebenan" bedeutet?

Betriebssystemsoftware-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-Kontroller



Interrupts mit Prioritäten

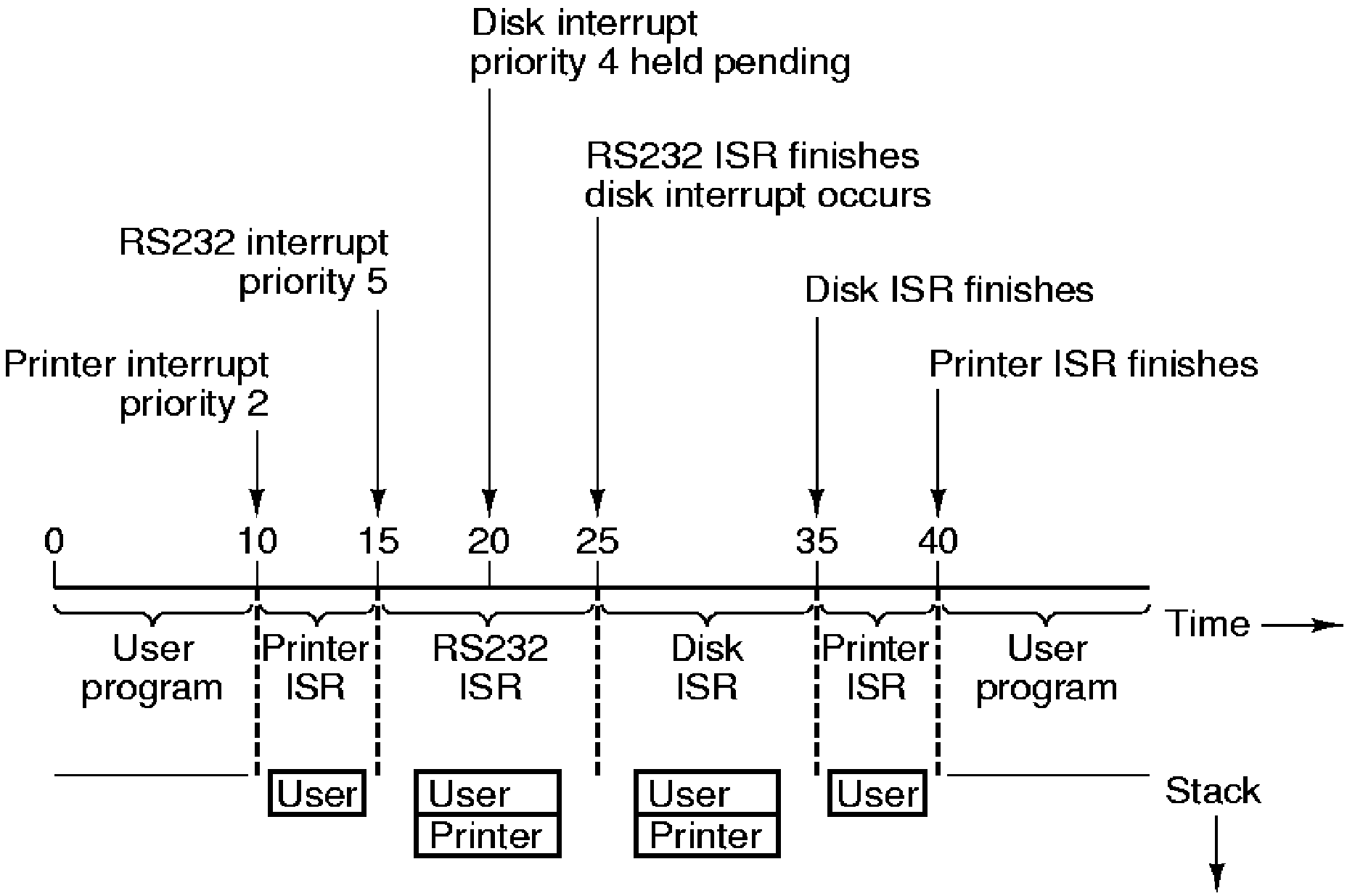
- I/O Interrupts können Prioritäten haben
- Die Prioritäten bedeuten, welcher Handler "Vorfahrt" hat, wenn gleichzeitig mehrere Interrupts gesetzt werden

Interrupts mit Priotitäten

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

Interrupt-Verfahren mit Prioritäten

- Ein Drucker-Interrupt, gerfolgt von...
- ...einem Serial-Port (RS232) Interrupt, gerfolgt von.....
- ...einem Festplatten Interrupt
- Die Interrupts sind prioritisiert
- RS232 (Priorität 5); Festplatte (Priorität 4); danach Drucker (Priorität 2)


Disk interrupt
priority 4 held pending
RS232 ISR finishes
disk interrupt occurs
RS232 interrupt
priority 5
Disk ISR finishes
Printer interrupt
priority 2
Printer ISR finishes
0
10
15
20
25
35
40
User
program
Printer
ISR
RS232
ISR
Disk
ISR
Printer
ISR
User
program
Time
User
User
Printer
User
Printer
User
Stack