Betriebssystem-Entwicklung

1. Einführung

Michael Schöttner

Vorschau

Heinrich Heine
Universität Düsseldorf

- Organisation
- Lernziele und Nutzen
- Inhalte
- Literatur
- Einordnung der Veranstaltung
- Beispiel: Töne erzeugen

Vorschau

Heinrich Heine
Universität Düsseldorf

1.1 Organisation

1.2 Lernziele und Nutzen

1.3 Inhalte

1.4 Literatur

1.5 Einordnung der Veranstaltung

1.6 Beispiel: Töne erzeugen

1.1 Organisation

Heinrich Heine
Universität Düsseldorf

- Vorlesung: 1 SWS
  - Montag, 8:30–10:00Uhr, 25.12.01.51
- Übung: 3 SWS
  - Montag, 8:30–10:00Uhr, 25.12.01.51
  - Donnerstag,12:30–14:00Uhr,25.12.01.51
- Hauptziel: Jede Person baut ein eigenes kleines 64 Bit Betriebssystem für x86 = hhuTOS (TOS = Teaching Operating System)
  - Dies entsteht durch die Übungen
  - Für die verschiedenen Komponenten gibt es jeweils Vorgaben, sodass nicht alles selbst programmiert werden muss

1.1 Organisation

Heinrich Heine
Universität Düsseldorf

- Vorlesungs- und Übungsunterlagen: auf unseren Webseiten
- Rocketchat via ILIAS
- Kurs basiert auf einigen Unterlagen von „Betriebssystembau“ von Prof. Spinczyk
- Voraussetzungen:
  - Betriebssystem-Grundkenntnisse empfohlen
  - Spaß and hardwarenaher Programmierung
  - Durchhaltevermögen

Übungen

Heinrich Heine
Universität Düsseldorf

- Ziel: jede Person soll am Ende ein eigenes kleines Betriebssystem geschrieben haben
  → verpflichtende Abgabe
- Das Betriebssystem muss in Qemu laufen!
  - Rechner zum Testen auf echter Hardware werden in der Übung bereitgestellt
- Die Übungsblätter bauen aufeinander auf
- Die Programmierung erfolgt in GNU C++ (und GDB und Qemu);
  punktuell, wenn auch selten NASM
- Alternativ kann das Betriebssystem auch in Rust geschrieben werden

Prüfung

Heinrich Heine
Universität Düsseldorf

- 5 ECTS für den Studiengang Master Informatik
- Mündliche Prüfung nach Ende der Vorlesungszeit
- Die Prüfung geht „nur“ über das eigene Betriebssystem.
  - In der Prüfung muss eine Demo vorgeführt werden
  - Der Quelltext muss mindestens eine Woche vor der Prüfung abgegeben werden
  - Für die Note 1.0 müssen alle Funktionen aus allen Übungsblättern funktionieren und vorzeigbar sein.
  - Die Beantwortung der Fragen zum Projekt fließt auch in die Bewertung ein.

Vorschau

Heinrich Heine
Universität Düsseldorf

1.1 Organisation

1.2 Lernziele und Nutzen

1.3 Inhalte

1.4 Literatur

1.5 Einordnung der Veranstaltung

1.6 Beispiel: Töne erzeugen

1.2 Lernziele und Nutzen

Heinrich Heine
Universität Düsseldorf

- Vertiefung des Wissens von Betriebssystem-Techniken/-Konzepten
  - Funktionsweise
  - Struktur
  - Implementierung
- Grundlegende BS-Funktionen in C++ oder Rust implementieren
- Der Weg ist das Ziel: hhuTOSc / hhuTOSr
  - Entwicklung eines eigenen Betriebssystems von der Pike auf
  - PC-Technologie besser verstehen

1.2 Lernziele und Nutzen

Heinrich Heine
Universität Düsseldorf

- Kenntnisse nützlich für
  - die Entwicklung von Middleware und Anwendungen
  - Entwicklung von Kernel-Komponenten und Treibern
- Zusätzliche Möglichkeiten auf der Hardware-Ebene
- Bei Bedarf Leistungsreserven der Hardware voll nutzbar

- „If you want to travel around the world and be invited to speak at a lot of different places, just write a UNIX operating system“ - Linus Torvalds

Vorschau

Heinrich Heine
Universität Düsseldorf

1.1 Organisation

1.2 Lernziele und Nutzen

1.3 Inhalte

1.4 Literatur

1.5 Einordnung der Veranstaltung

1.6 Beispiel: Töne erzeugen

Inhalte der Vorlesung

Heinrich Heine
Universität Düsseldorf

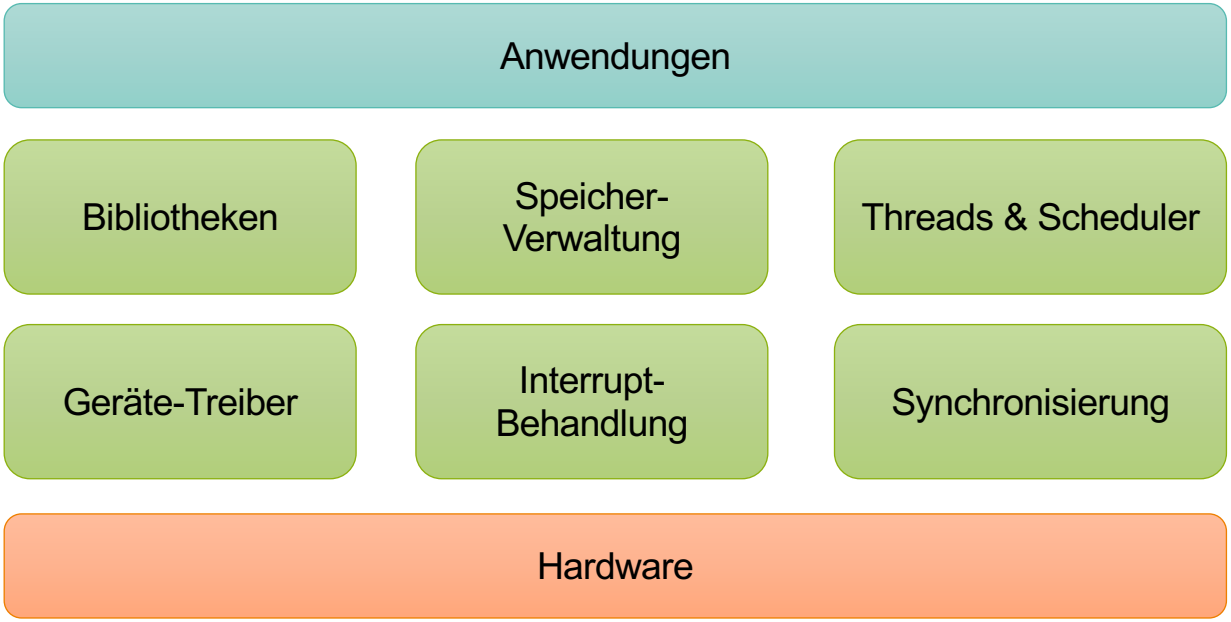
1. Einleitung (mit PC-Speaker)
2. BS-Entwicklung (Bootvorgang & Debugging)
3. x86-64 Architecture
4. Interrupts (PIC & APIC)
5. Koroutinen & Threads & Scheduling
6. Synchronisierung
7. PC Bus Systeme
8. Gerätetreiber (Linux, Windows)

Inhalte der Übungen

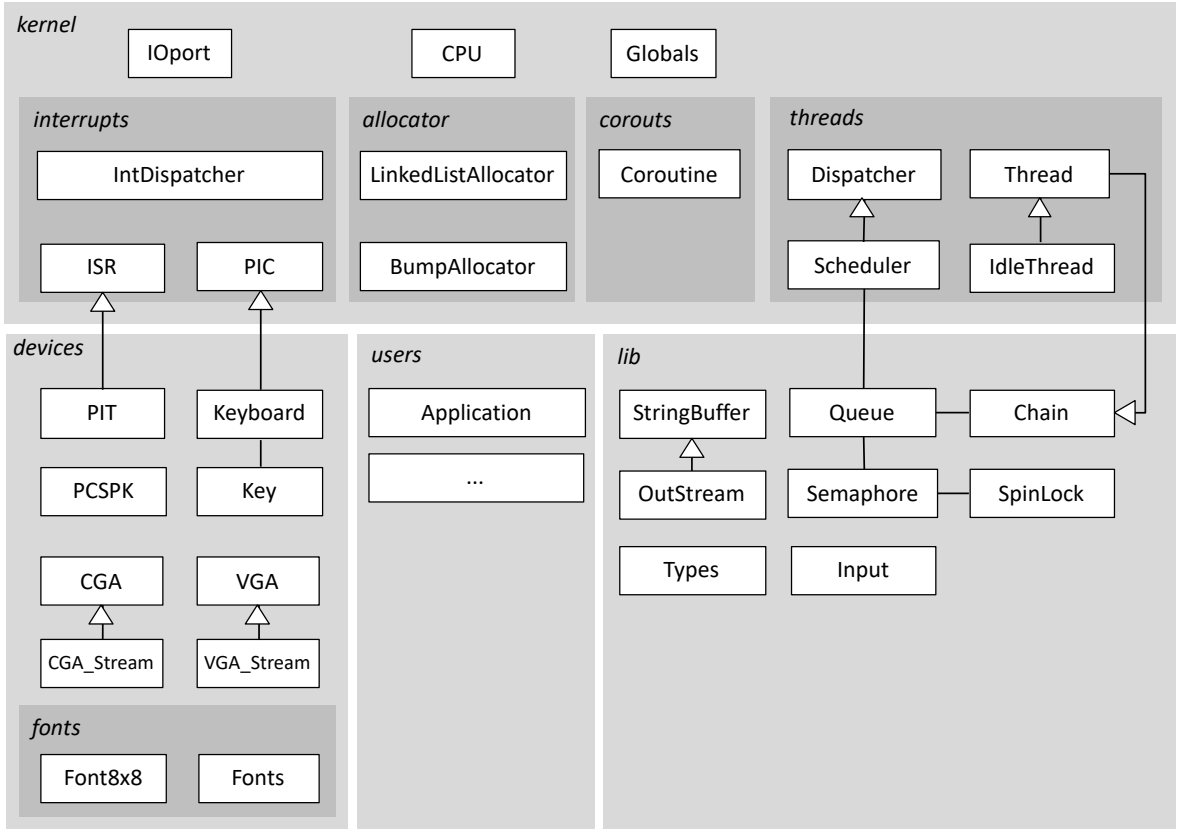
Heinrich Heine
Universität Düsseldorf

1. Ausgabe: CGA & PC-Speaker
2. Speicher: Heap-Verwaltung
3. Eingabe & Interrupts: PIC & Tastatur
4. Nebenläufigkeit: Koroutinen, kooperative Threads, Scheduler
5. Preemptives Multi-Threading
6. Synchronisierung: Semaphore
7. Eine eigene Anwendung

Komponenten von hhuTOS



Klassendiagramm von hhuTOS



Vorschau

Heinrich Heine
Universität Düsseldorf

1.1 Organisation

1.2 Lernziele und Nutzen

1.3 Inhalte

1.4 Literatur

1.5 Einordnung der Veranstaltung

1.6 Beispiel: Töne erzeugen

1.4 Literatur

Heinrich Heine
Universität Düsseldorf

- https://OSDev.org
- Manuals von Intel und AMD
- Hans Peter Messmer: PC - Hardwarebuch, Addison-Wesley 2003
- Manuals von Intel und AMD

Vorschau

Heinrich Heine
Universität Düsseldorf

1.1 Organisation

1.2 Lernziele und Nutzen

1.3 Inhalte

1.4 Literatur

1.5 Einordnung der Veranstaltung

1.6 Beispiel: Töne erzeugen

1.5 Einordnung der Veranstaltung

Heinrich Heine
Universität Düsseldorf

Bachelor

Betriebssysteme und Systemprogrammierung, 4V+2Ü, 10 LP

Grundlagen Verteilter Systeme, 2V+2Ü, 5 LP

Master

Betriebssystem-Entwicklung, 1V+3Ü, 5 CP

Betriebssystem-Entwicklung in Rust Seminar, 5CP

Isolation und Schutz in Betriebssystemen, 2V+2Ü, 5 CP

System-Software für Big-Data-Computing, 2S, 5 CP

Vorschau

Heinrich Heine
Universität Düsseldorf

1.1 Organisation

1.2 Lernziele und Nutzen

1.3 Inhalte

1.4 Literatur

1.5 Einordnung der Veranstaltung

1.6 Beispiel: Töne erzeugen

1.6 Beispiel: Töne erzeugen

- Lautsprecher des PCs wurde über einen Zeitgeber-Baustein angesteuert, welcher Impulse erzeugen kann

- Zeitgeber = Programmable Interval Timer (PIT)
  - Mittlerweile sind die Funktionen des PIT im Chipsatz des Mainboards integriert

- Früher erzeugt die Firmware des Mainboards mithilfe des PITs Beep-Codes bei einem Hardwarefehler

- Heute ist häufig kein Lautsprecher mehr verbaut, sondern stattdessen eine Hex-Anzeige auf dem Mainboard, um Fehlercodes so anzuzeigen →

Programmable Interval Timer

Heinrich Heine
Universität Düsseldorf

- PIT hat als Basis-Taktfrequenz 1,19318 MHz → Warum?
  - 1,19318 MHz = ¼ von 4,77 MHz = Taktfrequenz des IBM PC XT
  - Warum hatte der XT genau 4,77 MHz?
  - 4,77MHz = 1/3 von 14,31816 MHz
  - 14,31816 MHz ist die Grundfrequenz, die für NTSC-Fernsehen benötigt
    → dafür gab es billige Quarze (Massenproduktion)

- Generell haben alle Computer einen Quarz für einen Grundtakt
  - Dieser Takt wird mittels Teilern angepasst

- Heute ist der PIT immer noch für den Entzug der CPU bei preemptiven Multitasking notwendig und zur Realisierung der Systemzeit

PIT: Aufbau

- Besitzt drei Zähleinheiten
  - Diese werden dekrementiert bei der abfallenden Flanke des Grundtakt-Signals
- Verwendung:
  - Zähler 0: periodische Unterbrechungen → Signal OUT0 an IRQ-Controller
  - Zähler 1: DRAM Refresh → Signal OUT1 an DMA-Controller (nicht mehr relevant)
  - Zähler 2: Tonerzeugung für PC-Lautsprecher → OUT2 über anderen Baustein an Verstärker
- Programmierung erfolgt über sogenannte Ports (alle 8 Bit breit)

Port	Register	Zugriffsarten
0x40	Zähler 0	schreiben/lesen
0x41	Zähler 1	schreiben/lesen
0x42	Zähler 2	schreiben/lesen
0x43	Steuerregister	nur schreiben

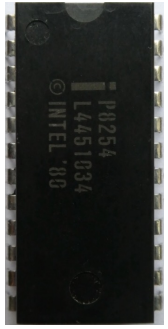
PIT: Aufbau des Steuerregisters (Port 0x43)

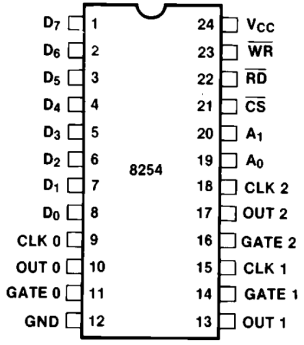
- Durch das Schreiben eines 8-Bit Steuerworts wird dem PIT gesagt werden, was er als Nächstes tun soll:
- Funktionsmodus bestimmt, wie der Zähler arbeitet und ob er mit Hilfe seiner OUT-Leitungen externe Ereignisse auslöst

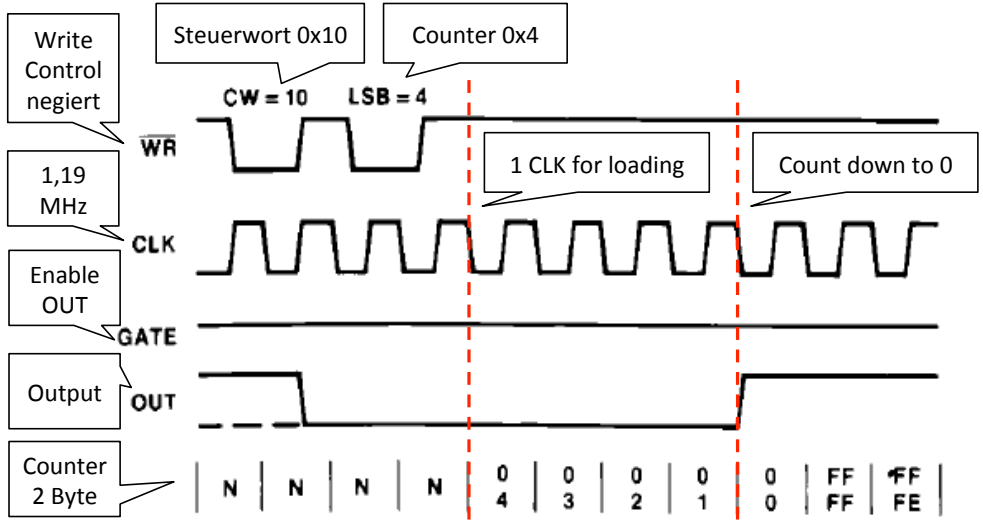
Bits	Wert	Bedeutung
6-7	**Zählerauswahl**	
	00	Zähler 0
	01	Zähler 1
	10	Zähler 2
	11	ungültig
4-5	**Zugriffsmodus auf oben gewählten Zähler**	
	00	Zähler-Latch-Befehl
	01	niederwertiges Zählerbyte
	10	höherwertiges Zählerbyte
	11	niederwertiges, anschließend höherwertiges Zählerbyte
1-3	**Funktionsmodus 0 bis 5**	
0	Zählformat	
	0	binäre Zählung mit 16 Bit
	1	Zählung mit vierstelligen BCD-Zahlen

PIT-Modus 0: Interrupt On Terminal Count

- Alle 838ns eine Dekrementierung (1s / 1193180 Hz = 838ns)
- Wenn der Zählerwert 0 ist, wird die OUTx-Leitung auf „1“ gesetzt und ein Interrupt ausgelöst
  - x = Nummer des verwendeten Zählers
  - Es erfolgt kein automatischer Reset des Zählers



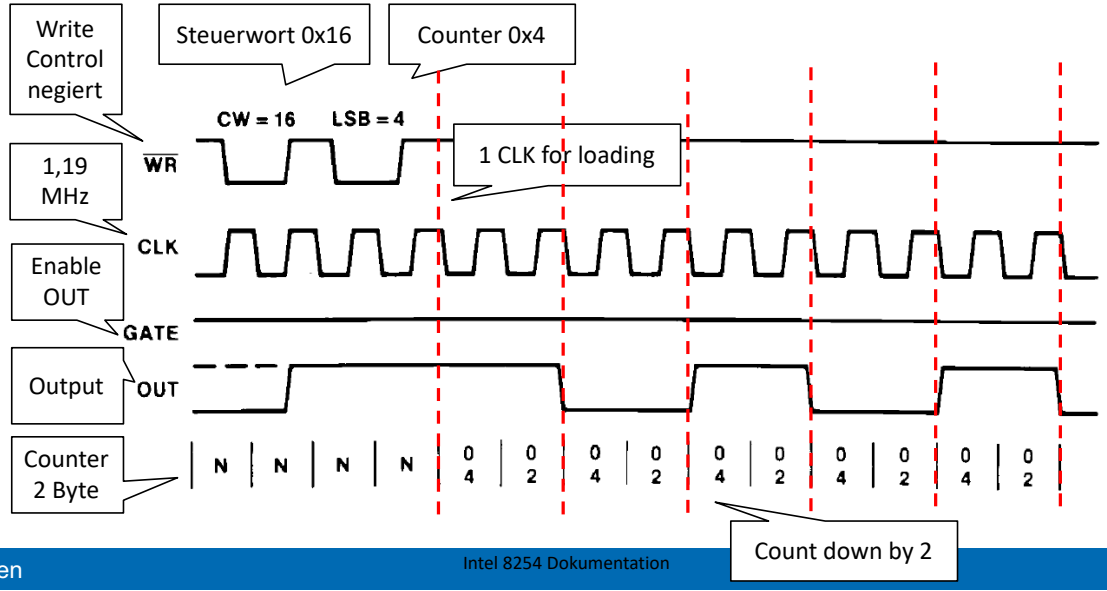




Intel 8254 Dokumentation

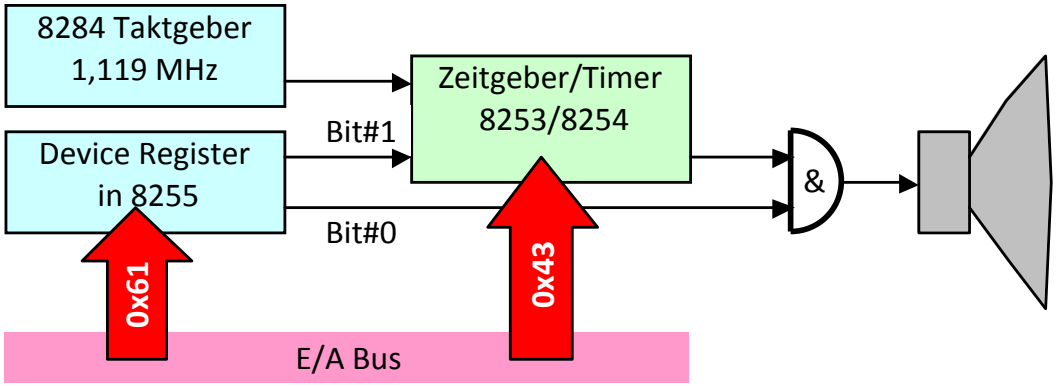
PIT-Modus 3: Square Wave Generator

- Rechtecksignal wird mithilfe einem internen FlipFlop erzeugt
  - Dies verdoppelt das Zeitintervall, weswegen in diesem Modus bei jeder abfallenden Flanke des Grundtaktes 2x dekrementiert wird → damit hat das ausgegebene Signal die richtige Frequenz
- Verwendet für preemptives Multitasking & Systemzeit; aber auch zur Tonerzeugung



Tonerzeugung

- Programmierung erfolgt über Port-Assemblerbefehle: `in`, `out`
- Zuerst muss Zähler 2 über das Steuerregister konfiguriert werden → 0xB6, entspricht binär 1011 0110 und bedeutet:
  - Zähler 2
  - Zugriffsmodus: zuerst „Lowbyte“ dann „Highbyte“
  - Modus Square Wave
  - Binär zählen (nicht BCD)
- Dann wird der Startwert geschrieben
- Schließlich, damit ein Ton erklingt, muss der
- Lautsprecher eingeschaltet werden → erfolgt über das Peripheral Port Interface (PPI 8255, Port 0x61)



C++ Code-Auszug

```
IOport pit_steuerung (0x43);                    // PIT Steuerregister
IOport pit_zaehler2 (0x42);                     // PIT Zaehler 2
IOport ppi (0x61);

void Tonerzeugung (float tonhoehe) {            // tonhoehe in Hz
    uint32_t grundfrequenz = 1193180;           // 1,19 MHz
    uint32_t zaehler       = grundfrequenz / tonhoehe;
    uint8_t  status;

    pit_steuerung.outb (0xB6);                  // Zaehler2 konfigurieren
    pit_zaehler2.outb  (zaehler % 256);         // Zaehler2 laden (Lobyte)
    pit_zaehler2.outb  (zaehler >> 8);          // Zaehler2 laden (Hibyte)

    // Lautsprecher ueber PPI einschalten
    status = ppi.inb();                         // PPI-Status lesen
    ppi.outb ( status|3 );                      // Lautsprecher einschalten
}
```

Rust Code-Auszug

Heinrich Heine
Universität Düsseldorf

```
const PORT_CTRL:u16  = 0x43;
const PORT_DATA0:u16 = 0x40;
const PORT_DATA2:u16 = 0x42;
const PORT_PPI:u16   = 0x61;

pub fn tonerzeugung (tonhoehe: f32) {                // tonhoehe in Hz
    let grundfrequenz = 1193180.0;                   // 1,19 MHz
    let zaehler       = (grundfrequenz / tonhoehe) as u32;
    let status: u8;

    cpu::outb(PORT_CTRL, 0xb6);                      // Zaehler2 konfigurieren
    cpu::outb(PORT_DATA2, (zaehler % 256) as u8);    // Zaehler2 laden (Lobyte)
    cpu::outb(PORT_DATA2, (zaehler >> 8) as u8);     // Zaehler2 laden (Hibyte)

    // Lautsprecher ueber PPI einschalten
    status = cpu::inb(PORT_PPI);                     // PPI-Status lesen
    cpu::outb( status | 3 );                         // Lautsprecher einschalten
}
```