
V.0.01.00 # Kernel in Asm und C # stand: 18.01.2016 # adrian stoop #
System Runtime: 0 h 3 min 13 sek
Mein eigenes Operating System in Asm und C
Adrian Stoop
22.01.2016

Übersicht

- Warum ein Eigenes OS?
- Funktion/Aufbau/Ablauf eines OS
- Was kann mein OS schon
- Probleme, Schwierigkeiten
- Vorführung
- Fragen?

Warum ein Eigenes OS?

- Neugierig
- „nie endendes Projekt“
- Programmierkenntnisse Verbessern
- Einzelarbeit

Funktion/Aufbau/Ablauf eines OS

- Funktion eines OS:
  - Abstraktion von Geräten und Diensten(Schnittstellen)
  - Application Programming Interface (API)
  - Die Unabhängigkeit von jeweiliger Hardware
  - Verwaltung von Ressourcen wie Zeit, Speicher, Interrupts etc.

Funktion/Aufbau/Ablauf eines OS

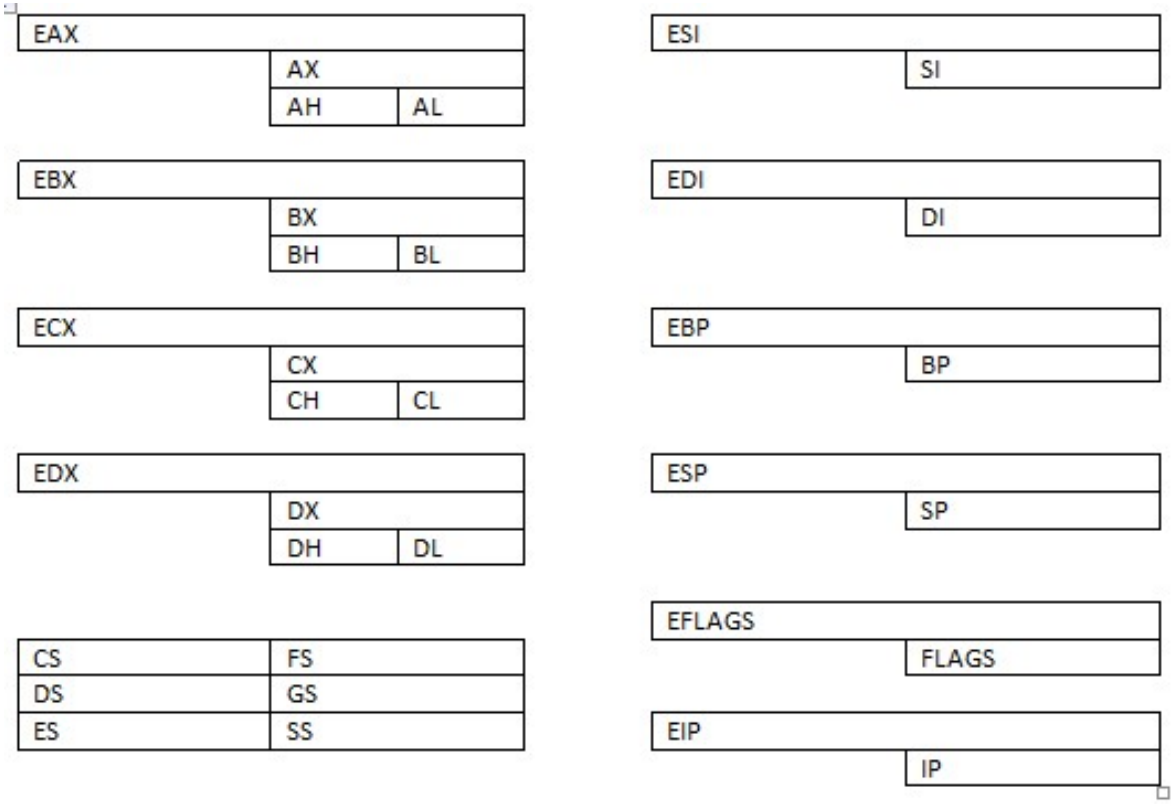
- Aufbau eines OS:

Adressenaufteilung einer x86 CPU

Start Adresse	End Adresse	Grösse	Type	Beschreibung
0x00000000	0x000003FF	1 KB	RAM, besetzt	Interrupt Vektor Table
0x00000400	0x000004FF	256 Byte	RAM, besetzt	BIOS Data Area
0x00000500	0x00007BFF	~30 KB	RAM, verfügbar	Free for use
0x00007C00	0x00007DFF	512 Byte	RAM, verfügbar	Boot-Sector
0x00007E00	0x0007FFFF	480.5 KB	RAM, verfügbar	Free for use
0x00080000	0x0009FBFF	120 KB	RAM, verfügbar	Ffu wenn vorhanden
0x0009FC00	0x0009FFFF	1 KB	RAM, besetzt	Extended BIOS Data Area
0x000A0000	0x000FFFFF	384 KB	verschieden	Video Memory

Funktion/Aufbau/Ablauf eines OS

- Aufbau eines OS:



A = Akkumulator
B = Basis register
C = Counter
D = Datenregister

CS = Code Segment
DS = Daten Segment
ES = Extra Segment

SI = Source Index
DI = Destination Index
BP = Base Pointer
SP = Stack Pointer

Flags = Flag Register
IP = Instruction Pointer

Funktion/Aufbau/Ablauf eines OS

- Ablauf eines OS:
  - CS = 0xFFFF (Code Segment Register)
    - BIOS (**B**asic **I**nput **O**utput **S**ystem)
      - Sind alle Medien Angeschlosssen?
  - BIOS sucht nach Signatur(0x55AA)
    - Gefunden? >> lade diesen Sektor(512Byte) nach 0x7C00
  - Kernel Nachladen

Was kann mein OS schon

```c
//video.c
extern void k_clear_screen();
extern void settextcolor(UCHAR forecolor, UCHAR backcolor);
extern void putch(UCHAR c);
extern void puts(UCHAR* text);
extern void scroll();
extern void k_printf(UCHAR* message, UINT line, UCHAR attribute);
extern void set_cursor(UCHAR x, UCHAR y);
extern void update_cursor();
extern void move_cursor_right();
extern void move_cursor_left();
extern void move_cursor_home();
extern void move_cursor_end();
extern void save_cursor();
extern void restore_cursor();
extern void printformat (char *args, ...);

//timer.c
extern void timer_handler(struct regs* r);
extern void timer_wait (ULONG ticks);
extern void sleepSeconds (ULONG seconds);
extern void sleepMilliSeconds (ULONG seconds);
extern void timer_install();
extern void timer_uninstall();

//keyboard.c
extern void keyboard_install();
extern void keyboard_init();
extern UCHAR k_getch();
extern UCHAR FetchAndAnalyzeScancode();
extern void keyboard_handler(struct regs* r);

//util.c
extern void outportb(UINT port, UINT val);
inline UINT inportb(UINT port);
extern void* k_memset(void* dest, char val, size_t count);
extern USHORT* k_memsetw(USHORT* dest, USHORT val, size_t count);
extern void* k_memcpy(void* dest, const void* src, size_t count);
extern size_t k_strlen(const char* str);
extern void reboot();
extern void cli();
extern void sti();
extern void k_itoa(int value, char* valuestring);
extern void k_i2hex(UINT val, UCHAR* dest, int len);
extern void float2string(float value, int decimal, char* valuestring);

//gdt.c
extern void gdt_set_gate(int num, unsigned long base, unsigned long limit, unsigned char access, unsigned char gran);
extern void gdt_install();

//idt.c
extern void idt_set_gate(unsigned char num, unsigned long base, unsigned short sel, unsigned char flags);
extern void idt_install();

//irq.c and isrs.c
extern void isrs_install();
extern void irq_install();
extern void fault_handler(struct regs* r);
extern void irq_install_handler(int irq, void (*handler)(struct regs* r));
extern void irq_uninstall_handler(int irq);
extern void irq_remap(void);
extern void irq_handler(struct regs* r);
```

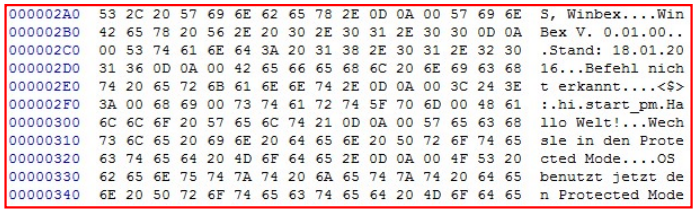
```c
void printformat (char *args, ...)
{
    va_list ap;
    va_start (ap, args);
    int index = 0, d;
    UINT u;
    char c, *s;
    char buffer[100];

    while (args[index])
    {
        switch (args[index])
        {
        case '%':
            ++index;
            switch (args[index])
            {
            case 'u':
                u = va_arg (ap, UINT);
                k_itoa(u, buffer);
                puts(buffer);
                break;
            case 'd':
            case 'i':
                d = va_arg (ap, int);
                k_itoa(d, buffer);
                puts(buffer);
                break;
            case 'X':
            case 'x':
                d = va_arg (ap, int);
                k_i2hex(d, buffer,8);
                puts(buffer);
                break;
            case 's':
                s = va_arg (ap, char*);
                puts(s);
                break;
            case 'c':
                c = (char) va_arg (ap, int);
                putch(c);
                break;
            default:
                putch('%');
                putch('%');
                break;
            }
            break;

        default:
            putch(args[index]); //wenn kein \xyz dan wird der charakter einfach ausgegeben.
            break;
        }
        ++index;
    }
}
```

- Bootbarer Minikernel
- Bootloader + nachgeladener Kernel
- Auf Instruktionen im RM reagieren (dank BIOS einfache Handhabung)
- A20-Gate und PM aktivieren, GDT/GDTR
- Sprung vom ASM- zum C-Kernel (Lesbarkeit, Module)
- Scansonde und ASCII Tabelle einlesen (Tastatur- Treiber)
- Rudimentäre Textausgabe auf Bildschirm (Video RAM 0xB8000)
- Rudimentäre Texteingabe mit Tastatur ("Polling")
- Auf Interrupt Request (IRQ) soll das OS reagieren. (z.b. System Cloack).
- Eine Interrupt Description Table anlegen (IDT)
- IRQ handler um Master PIC und Slave PIC zu setzten und zurücksetzten.
- OS soll eine System Uhr haben die stimmt beim starten(System Clock/ Programable Interval Timer)
- Rudimentäre Texteingabe mit Tastatur durch interrupts (IRQ)
- Key Queue und Key handlen (Tastenschläge >> Warteschlange(register) bis Programm Tasten braucht)
- Systemfrequenz verändern.

Probleme, Schwierigkeiten

```
150   print_string:
151     mov ah, 0x0E
152
153   .loop:
154     lodsb
155     test al, al
156     jz .done
157     int 0x10
158     jmp .loop
159
160   .done:
161     ret
```

- Asm befehle (AAA,AAD,AAM,…,FYL2XP1)
- Englisch > Deutsch
- Binär Code Auslessen
- Zeit im Internet statt Programmieren
- Compilervorgang debugen/Optimieren
- Backgroundwissen
- Doku



```
000002A0  53 2C 20 57 69 6E 62 65 78 2E 0D 0A 00 57 69 6E  S, Winbex....Win
000002B0  42 65 78 20 56 2E 20 30 2E 30 31 2E 30 30 0D 0A  Bex V. 0.01.00..
000002C0  00 53 74 61 6E 64 3A 20 31 38 2E 30 31 2E 32 30  .Stand: 18.01.20
000002D0  31 36 0D 0A 00 42 65 66 65 68 6C 20 6E 69 63 68  16...Befehl nich
000002E0  74 20 65 72 6B 61 6E 6E 74 2E 0D 0A 00 3C 24 3E  t erkannt....<$>
000002F0  3A 00 68 69 00 73 74 61 72 74 5F 70 6D 00 48 61  :.hi.start_pm.Ha
00000300  6C 6C 6F 20 57 65 6C 74 21 0D 0A 00 57 65 63 68  llo Welt!...Wech
00000310  73 6C 65 20 69 6E 20 64 65 6E 20 50 72 6F 74 65  sle in den Prote
00000320  63 74 65 64 20 4D 6F 64 65 2E 0D 0A 00 4F 53 20  cted Mode....OS
00000330  62 65 6E 75 74 7A 74 20 6A 65 74 7A 74 20 64 65  benutzt jetzt de
00000340  6E 20 50 72 6F 74 65 63 74 65 64 20 4D 6F 64 65  n Protected Mode
```

```
000000be <_outportb>:
  be:   55                      push   %ebp
  bf:   89 e5                   mov    %esp,%ebp
  c1:   83 ec 04                sub    $0x4,%esp
  c4:   8b 45 0c                mov    0xc(%ebp),%eax
  c7:   88 45 ff                mov    %al,0xffffffff(%ebp)
  ca:   8b 55 08                mov    0x8(%ebp),%edx
  cd:   8a 45 ff                mov    0xffffffff(%ebp),%al
  d0:   ee                      out    %al,(%dx)
  d1:   c9                      leave
  d2:   c3                      ret
```

Vorführung

Fragen?