Isolation und Schutz in Betriebssystemen

3. Systemaufrufe bei x86-64

Michael Schöttner

3.1 Ziel

Heinrich Heine
Universität Düsseldorf

- Bisher können unsere User-Mode Threads (Ring 3) einfach alle Funktionen des Kernels direkt aufrufen.
  - Jedoch wirft der Prozessor eine General Protection Fault (GPF), falls in den aufgerufenen Kern-Funktionen eine privilegierte Instruktion verwendet wird
  - Normalerweise werden die Funktionen des Kerns über das Paging vor direkten Aufrufen aus dem Ring 3 geschützt.

- Wir wollen nun Systemaufrufe realisieren, wodurch nur noch bestimmte Kernel-Funktionen indirekt in kontrollierter Weise aufgerufen werden

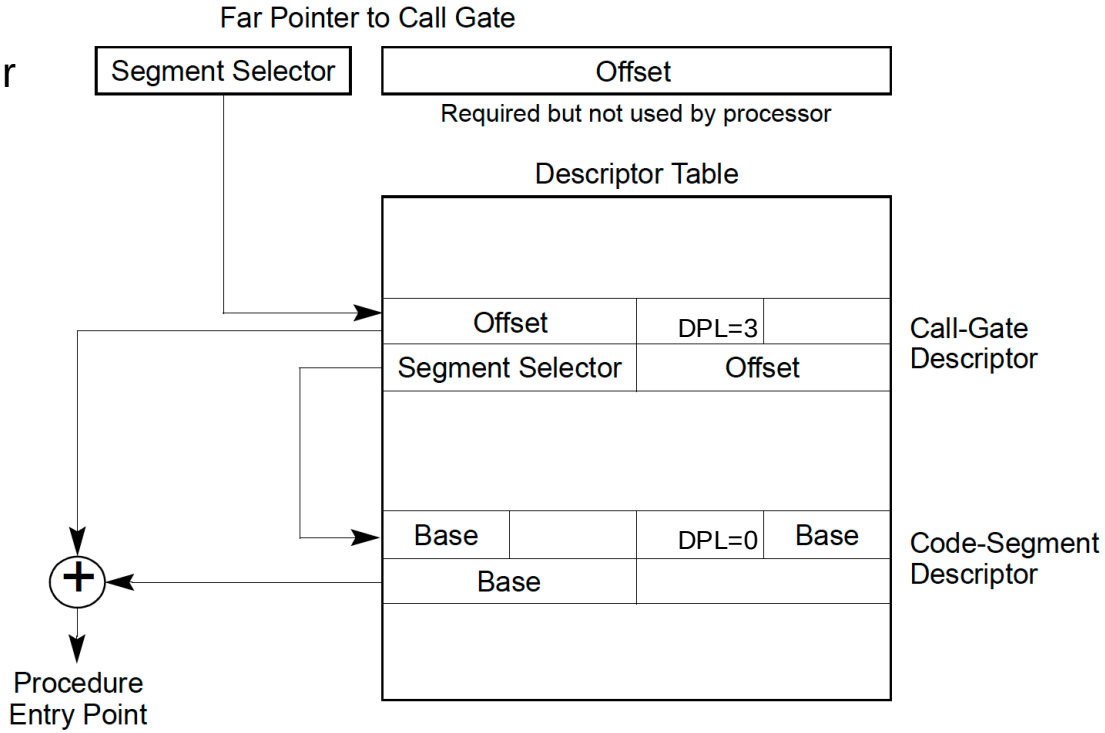
3.2 Mechanismen für Systemaufrufe bei x86

Heinrich Heine
Universität Düsseldorf

- Call Gate
  - Deskriptor in der GDT
  - Zugriff mit einer `call` oder `jmp` Instruktion
  - Stack wird mithilfe des TSS umgeschaltet
- Interrupt Trap Gate
  - Deskriptor in der IDT
  - Zugriff mit einer `int` Instruktion
  - Stack wird mithilfe des TSS umgeschaltet
- Schnelle Systemaufrufe
  - Mithilfe MSR (Model Specific Register)
  - Zugriff mit einer `syscall/sysret`
  - Stack muss in Software umgeschaltet werden

3.3 Call Gate

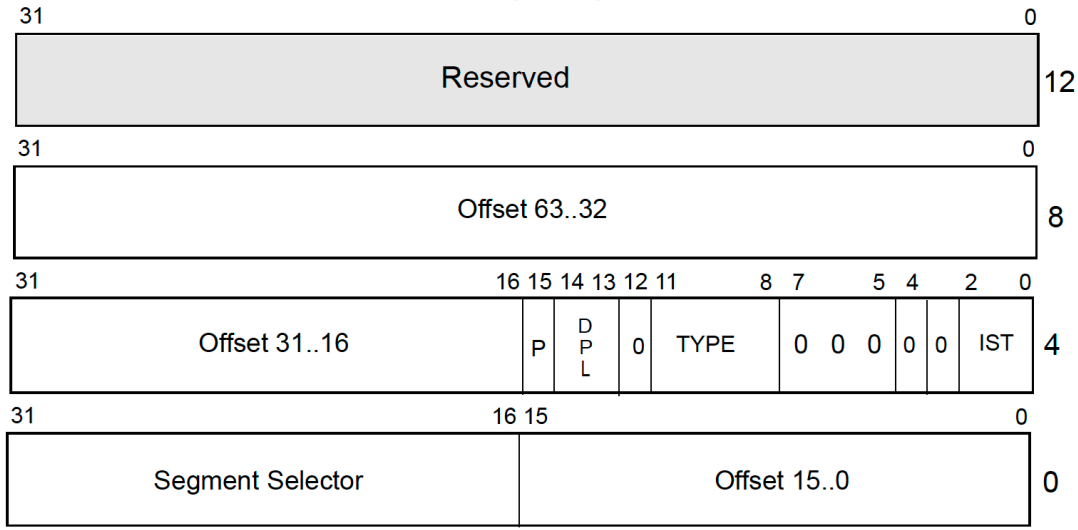
- Offset im Register sowie im Code Segment Deskriptor werden ignoriert



3.4 Interrupt/Trap Gate

- Nutzung durch die Instruktion
  - Beispiel `int 0x80`
  - Referenziert ein Trap Gate mit Index 0x80 in der IDT
- Offset: Adresse des Handlers
- Segment Selector wählt ein Segment Deskriptor in der GDT

Interrupt/Trap Gate



DPL	Descriptor Privilege Level
Offset	Offset to procedure entry point
P	Segment Present flag
Selector	Segment Selector for destination code segment
IST	Interrupt Stack Table

3.4 Interrupt/Trap Gate

Heinrich Heine
Universität Düsseldorf

- Interrupt Gate
  - Interrupt Enable Flag wird gelöscht
  - Verwendet für Interrupt Handler → sequentielle Abarbeitung (auf Single Core)

- Trap Gate
  - Interrupt Enable Flag wird nicht gelöscht
  - Verwendet für Systemaufrufe

3.5 Interrupts

- Externe- oder Hardware-Interrupts
  - von einem Gerät, z.B. Timer-Interrupt
  - Kommen von außerhalb, aus Sicht der CPU

- Interne- oder Software- Interrupts:
  - Kommen von der CPU selbst
    - Exceptions (siehe nächste Seite)
    - Oder durch die Assemblerinstruktion `INT <nr>`
      - Verwendet für Systemaufrufe (Linux, Windows NT, MacOS, MSDOS)

Exceptions

- **Fault** (dt. Störung):
  - kann behoben werden, z.B. Page Fault
  - CPU-Zustand wird gesichert & Adresse der Instruktion, die Fault ausgelöst hat
- **Trap** (~ dt. Falle):
  - ausgelöst durch speziellen Befehl, z.B. INT 3 (Breakpoint)
  - Programm kann fortgeführt werden
- **Abort** (dt. Abbruch):
  - bei schwerem Fehler
  - Auslöser oft nicht genau lokalisierbar
  - führt zum Restart (z.B. Double Fault)

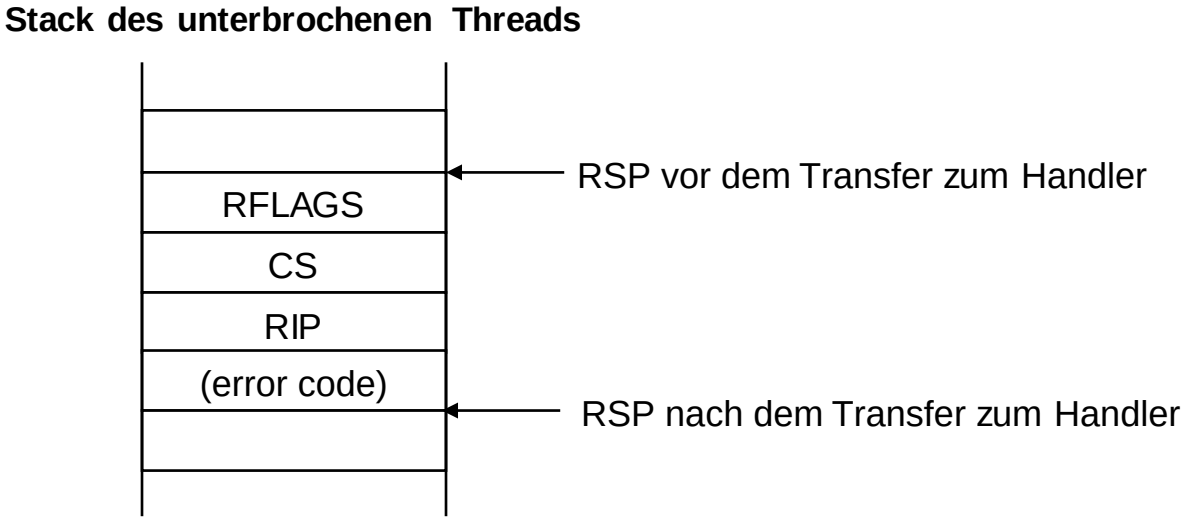
Vectors

- Interrupts und Exceptions werden durch eine Vektornummer identifiziert
- 0 – 31 ist reserviert für x86 Exceptions
- 32 – 255 steht zur freien Verfügung
- Vektoren und ihre Bedeutung (Auszug)

Vektor	Bedeutung	Vektor	Bedeutung
0	Division by 0	11	Segment fault
1	Debug	12	Stack overflow
2	NMI	13	General protection fault
3	Break	14	Page fault
4	Overflow	16	-
5	Bounds range exceeded	18	Machine check
6	Illegal instruction	…	. . .
8	Double fault		

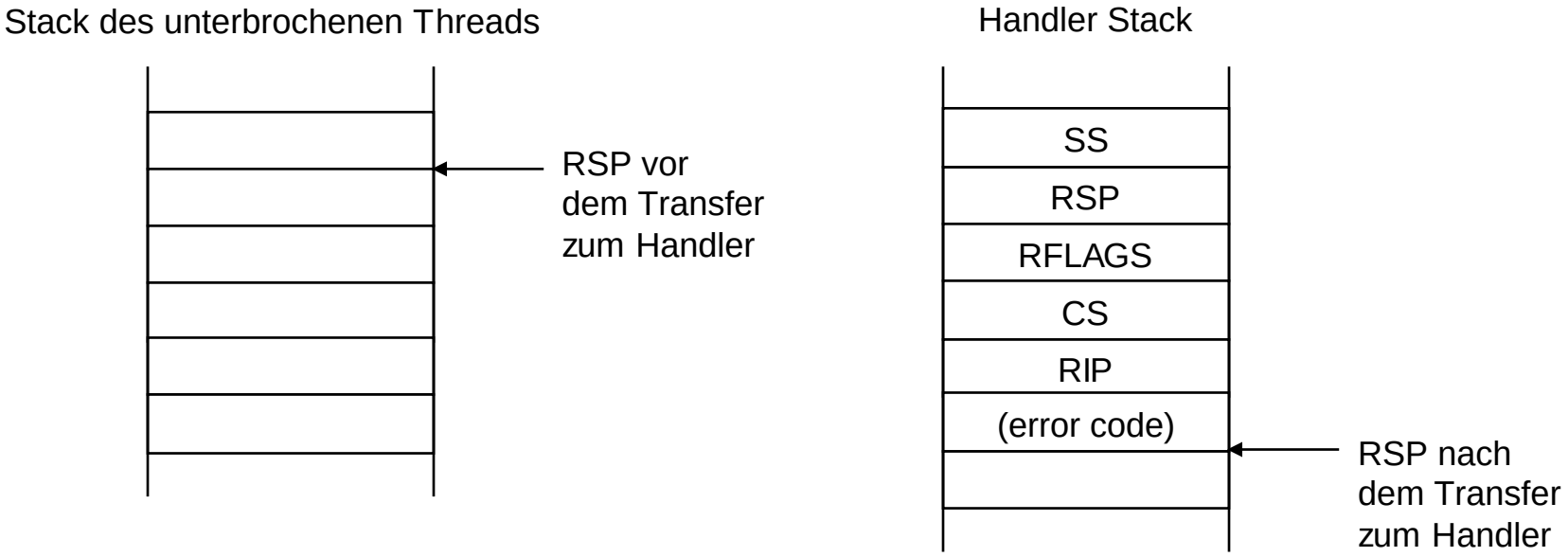
Stackaufbau ohne Ringwechsel

- Falls keine Privilegstufe gewechselt wird, so wird auch der Stack nicht umgeschaltet
- D.h. der Interrupt-Handler verwendet den Stack des unterbrochenen Threads
- Dies entspricht dem Ablauf bei hhuTOS (unser gesamter Code läuft im Ring 0)
- Einen error code gibt es nur bei manchen Exceptions



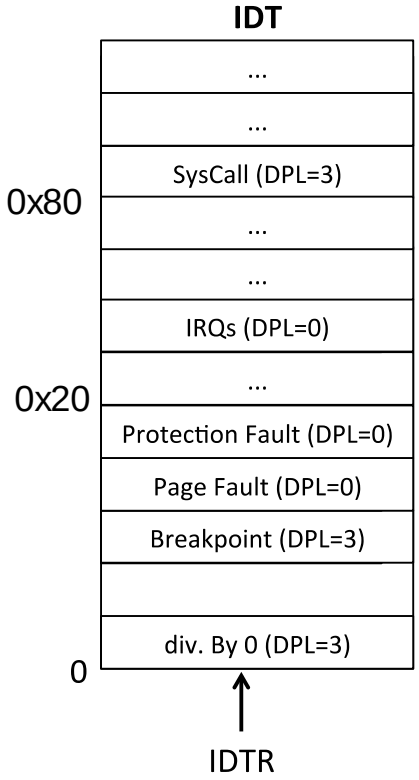
Stackaufbau bei einem Ringwechsel

- Neuer Stack wird aus dem Task State Segment ermittelt (siehe letztes Kapitel)

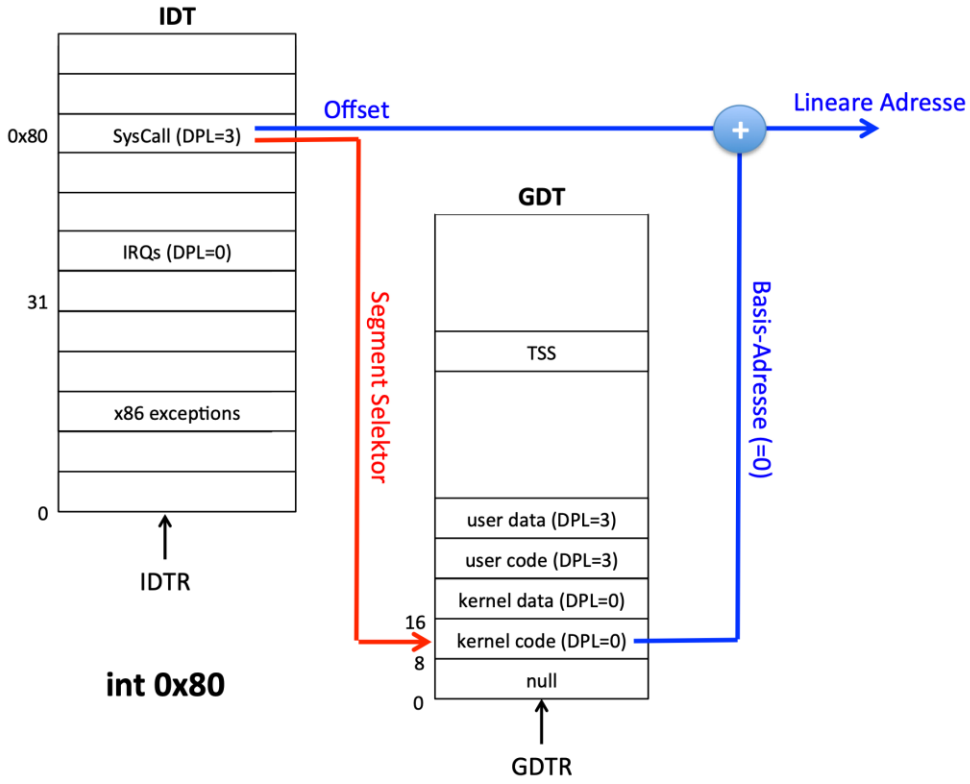


Beispiel unserer IDT

- Einträge 0-31 sind durch x86 reserviert für Exceptions
  - Interrupt Gates, alle DPL = 0
- Einträge 32 – 47 für externe Interrupts
  - Interrupt Gates, alle DPL = 0
  - IRQ0 != Vektor0
- Eintrag 0x80 für System-Aufrufe
  - Trap-Gate
  - DPL=3



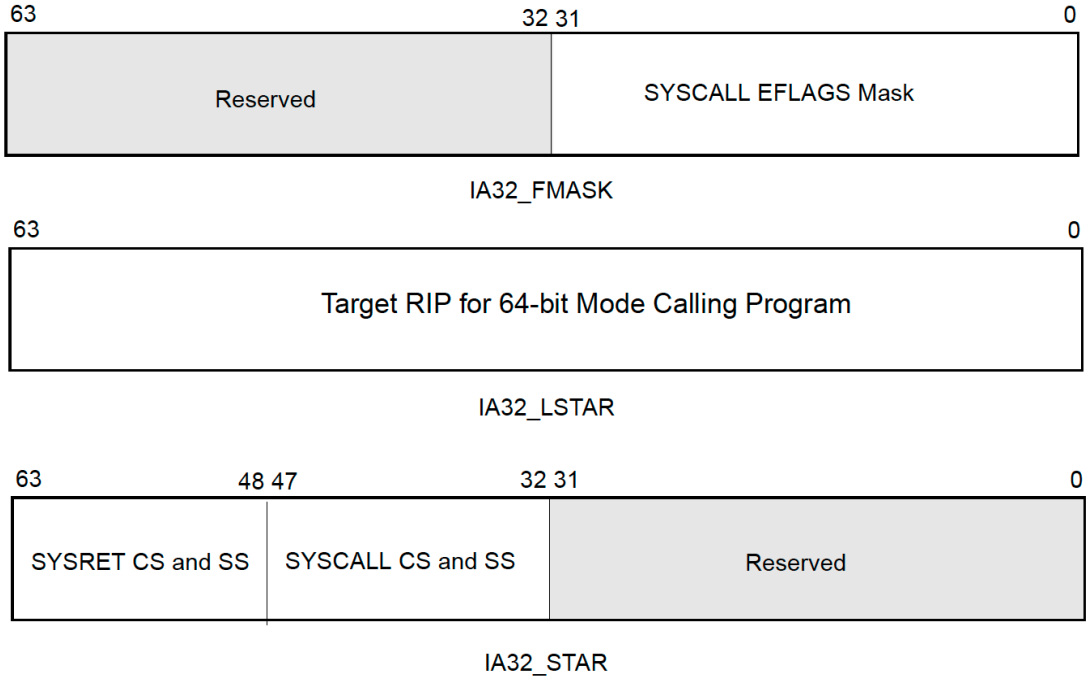
Systemaufruf über ein Trap-Gate



3.6 Syscall/sysret

- Verwendet Model Specific Registers (MSRs) →
- Benötigt bestimmtes Layout in der GDT
- Stack muss händisch im Syscall-Handler umgeschaltet werden

- Verwenden FS und GS
  - Hier gibt es noch eine Basisadresse
  - Typicherweise speichert die Basisadresse in GS den Zeiger auf den Kernel-Stack



3.7 interrupts.asm

```
[SECTION .data]

;
; Interrupt Descriptor Table mit 256 Einträgen
;

idt:
%macro idt_entry 1
  dw  (wrapper_%1 - wrapper_0) & 0xffff ; Offset 0 .. 15
  dw  0x0000 | 0x8 * 2 ; Selector zeigt auf den 64-Bit-Codesegment-Deskriptor der GDT
  dw  0x8e00 ; 8 -> interrupt is present, e -> 80386 64-bit interrupt gate
  dw  ((wrapper_%1 - wrapper_0) & 0xffff0000) >> 16 ; Offset 16 .. 31
  dd  ((wrapper_%1 - wrapper_0) & 0xffffffff00000000) >> 32 ; Offset 32..63
  dd  0x00000000 ; Reserviert
%endmacro

%assign i 0
%rep 256
idt_entry i
%assign i i+1
%endrep


idt_descr:
  dw  256*8 - 1    ; 256 Einträge
  dq idt
```

Template für einen IDT-Eintrag (16 Byte), hier „nur“ Interrupt Gates

256 IDT-Einträge erzeugen

Limit (16 Bit)
Base address (64-Bit)

3.7 interrupts.asm

```
; Spezifischer Kopf der Unterbrechungsbehandlungsroutinen
%macro _wrapper 1
_wrapper_%1:
    ; alle Register sichern
    push   rax
    ...

    ; Error-Codes fuer General Protection Fault (GPF)
    %if %1 == 13
        mov    rdi, [rsp+112] ; error code
        mov    rdx, [rsp+120] ; rip
        mov    rsi, [rsp+128] ; cs
        call   int_gpf
    %else
        ; Vektor als Parameter übergeben
        xor rax, rax
        mov al, %1
        mov    rdi, rax
        call   int_disp
    %endif

    ; Register wiederherstellen
    ...
    pop    rax

    ; Fertig!
    iretq
%endmacro
```

```
; ... wird automatisch erzeugt.
%assign i 0
    %rep 256
    _wrapper i
    %assign i i+1
%endrep
```

Pro IDT-Eintrag eine wrapper-Routine. Diese speichert die Vektor-Nummer in `rax`. Damit kann später die Vektor-Nummer ermittelt werden.