Heinrich Heine
Universität Düsseldorf

UNIVERSITÄT DÜSSELDORF
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Prof. Dr. Michael Schöttner
Fabian Ruhland

2. Aufgabenblatt zum Modul „Isolation und Schutz in Betriebssystemen“

Alle Materialien finden Sie in ILIAS

Lernziel
Verstehen wie Systemaufrufe funktionieren damit bei einem Aufruf vom Ring 3 in den Ring 0 geschaltet wird und wir damit in kontrollierter Weise Funktionen des Kerns aufrufen, die wiederum privilegierte Instruktionen verwenden dürfen.

1. Aufgabe: Interrupt Descriptor Table (IDT)

Ein sehr grundlegender Ansatz besteht darin, einen Software-Interrupt auszulösen (wie beispielsweise der Vektor 0x80 in Linux), um in den Ring 0 zu wechseln. Hierzu benötigen wir einen neuen Eintrag in der IDT.
Unsere IDT ist komplett (alle 256 Einträge) mit Interrupt-Gates befüllt, siehe `interrupts.asm` und alle Einträge haben DPL = 0. Damit wir einen Systemaufruf mithilfe eines Software-Interrupts realisieren können benötigen wir in der IDT ein Trap-Gate mit DPL=3 (damit dieses aus dem User-Mode heraus zugegriffen werden darf). Wir überschreiben hierzu einfach den existierenden Eintrag an der Index-Position 0x80 (= Vektornummer).

Das Einrichten des Trap-Gates soll in der Datei `syscalls.asm` erfolgen und darin in der Assembler-Funktion `_init_syscalls`. Damit diese Funktion ausgeführt wird, muss in `kmain` in `starup.rs` die Funktion `syscall_dispatcher::init` aufgerufen werden, wichtig, nach dem Aufruf von `interrupts::init` in `kmain`! Für das Erstellen des Trap-Gates können Sie sich bezüglich des Assembler-Codes an Aufgabe 1 von Übungsblatt 1 orientieren. Als Offset im Trap-Gate wird die Adresse der Funktion `_syscall_handler` eingetragen.

Wichtige Information für diese Aufgabe finden Sie in Intel Software Developer’s Manual Volume 3 in Kapitel 6.11 IDT Descriptors.

2. Aufgabe: Syscall-Handler

Als nächstes muss der Syscall-Handler in `syscalls.asm` programmiert werden. Hierzu kann man sich etwas am `wrapper` in `interrupts.asm` orientieren. Zudem sind in der Vorgabe Kommentare zu finden. Im Syscall-Handler wird dann die Rust-Funktion `syscall_disp` in `syscall_dispatcher.rs` aufgerufen. In den Vorgaben ist schon ein fertiger Systemaufruf HelloWorld ohne Parameter enthalten. Die API für alle Systemaufrufe für den User-Mode befindet

sich in der Datei `user_api.rs`. Jede API-Funktion soll mit dem Präfix `usr_` beginnen und diese rufen die eigentliche Kernel-Funktion mithilfe eines Software-Interrupts (Vektor 0x80, um unser Trap-Gate aus Aufgabe 1 zu verwenden); die Funktionsnummer wird im Register `rax` übergeben.

Die eigentliche Implementierung zu einem Systemaufruf sind Funktionen mit dem Präfix `sys_` und diese befinden sich jeweils in einer separaten Datei im Ordner `kfuncs`. Für den Hello-World-Systemaufruf gibt es in der Datei `user_api.rs` die Funktion `usr_hello_world` und das Gegenstück, mit der eigentlichen Implementierung `sys_hello_world` in der Datei `sys_hello_world.rs`.

Schreiben Sie einen Test-Thread, welcher in `kmain` im Scheduler registriert wird. Dieser Test-Thread soll im User-Mode laufen und den Systemaufruf `hello_world` aufrufen. Wenn alles funktioniert sollte die Textausgabe der Funktion `sys_hello_world` auf der seriellen Schnittstelle zu sehen sein. Wichtig, hierfür werden Portbefehle verwendet, die im User-Mode nicht erlaubt sind. Durch den Systemaufruf wechseln wir jedoch in den Ring 0 können dann den Code ausführen.

3. Aufgabe: Weitere Systemaufrufe (mit Parametern)

Nachdem der `hello_world`-Systemaufruf erfolgreich getestet wurde sollen die nachfolgenden Systemaufrufe implementiert und getestet werden:

- `get_lastkey() -> u64`
- `sys_gettid() -> i64`
- `sys_read(buff: *mut u8, len: u64) -> i64`
- `sys_write(buff: *const u8, len: u64) -> i64`

Änderungen sind in folgenden Dateien notwendig:

- `usr_api.rs`:
  - `NO_SYSCALLS` muss angepasst werden
  - Sowie Funktionsnummern für die neuen Aufrufe definiert werden
  - Dann die Funktionen für jeden Systemaufruf `usr_xxx`
  - Sowie weitere Systemaufrufe für folgende Parameteranzahlen: eins und zwei. Dazu kann man sich an `syscall0` orientieren
- `syscall.asm`: die Variable `NO_SYSCALLS` muss angepasst werden
- `syscall_dispatcher.rs`: hier muss die `SyscallFuncTable` um die neuen Aufrufe erweitert werden
- Im Ordner `kfuncs`: muss für jeden Systemaufruf eine Datei angelegt werden und dort die jeweilige Kernelfunktion `sys_xxx` implementiert werden

Wichtige Information für die Parameterübergabe in Registern finden Sie in System V Application Binary Interface AMD64 Architecture Processor Supplement in Kapitel 3.2.3 Parameter Passing.

4. Aufgabe: Schnelle Systemaufrufe (optional)

Systemaufrufe über ein Trap-Gate sind nicht sehr schnell. Daher wurden schnellere Varianten eingeführt: `sysenter/sysexit` (Intel) und `syscall/sysret` (AMD) eingeführt. Aus Gründen der Kompatibilität verwenden x64-Systeme letztere Variante. Um diese schnellere Variante zu nutzen muss die GDT dem geforderten Aufbau entsprechen.

Argumente können in den Registern übergeben werden, wie beim Interrupt-basierten Systemaufruf, aber es ist zu beachten, dass das Register RCX (4. Parameter) auch von der `syscall`-Instruktion genutzt wird. Als Work-Around kann der 4. Parameter in einem anderen Register übergeben werden.

Als Präfix für die schnellen Systemaufrufe bietet sich beispielsweise `fast_` an.

Wichtige Information für diese Aufgabe finden sich im Intel Software Developer's Manual Volume 3 in Kapitel 5.8.8 Fast System Calls in 64-Bit Mode.

Mögliches Testszenario

Ein mögliches Testszenario für dieses Übungsblatt könnte wie folgt aussehen.