Übung zu Betriebssystemtechnik

Paging in StuBSmi

14. Mai 2018

Andreas Ziegler
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Exkurs: aktuelle Entwicklungen in Linux

Systemaufrufe in Linux v4.16/x86_64

Definition des `read` Systemaufrufs:

fs/read_write.c

```
566  SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
567  {
568      struct fd f = fdget_pos(fd);
569      ssize_t ret = -EBADF;
570      ...

577      return ret;
578  }
```

wird expandiert zu:

```
asmlinkage long sys_read(int fd, char __user *buf, size_t count) \
    __attribute__((alias(__stringify(SyS_read))));

asmlinkage long SyS_read(long fd, long buf, long count)
{
    long ret = SYSC_read((int) fd, (char __user *) buf, (size_t) count);
    return ret;
}

static inline SYSC_read(int fd, char __user *buf, size_t count)
/* SYSCALL_DEFINE3() expansion ends here, function body follows */
```

Systemaufrufe in Linux v4.16/x86_64

Funktionspointer in `sys_call_table` zeigt auf

```
asmlinkage long SyS_read(long fd, long buf, long count);
```

Aufruf der Systemaufruf-Implementierung:

`arch/x86/entry/common.c`:

```
284   ...
285   if (likely((nr & __SYSCALL_MASK) < NR_syscalls)) {
286       nr = array_index_nospec(nr & __SYSCALL_MASK, NR_syscalls);
287       regs->ax = sys_call_table[nr](
288               regs->di, regs->si, regs->dx,
289               regs->r10, regs->r8, regs->r9);
290       }
291   ...
```

Wer sieht das “Problem”?

Systemaufrufe in Linux v4.16/x86_64

Funktionspointer in `sys_call_table` zeigt auf

```
asmlinkage long SyS_read(long fd, long buf, long count);
```

Aufruf der Systemaufruf-Implementierung:

`arch/x86/entry/common.c`:

```
284   ...
285   if (likely((nr & __SYSCALL_MASK) < NR_syscalls)) {
286       nr = array_index_nospec(nr & __SYSCALL_MASK, NR_syscalls);
287       regs->ax = sys_call_table[nr](
288               regs->di, regs->si, regs->dx,
289               regs->r10, regs->r8, regs->r9);
290       }
291   ...
```

Wer sieht das “Problem”?

$\rightarrow$ Es werden immer 6 Parameter übergeben, `SyS_read` braucht nur 3

$\rightarrow$ Vom Nutzer angegebene Werte trotzdem auf dem Kernel-Stack!

Neu in Linux v4.17/x86_64

arch/x86/entry/common.c:

```
285  if (likely(nr < NR_syscalls)) {
286      nr = array_index_nospec(nr, NR_syscalls);
287      regs->ax = sys_call_table[nr](regs);
288  }
```

Expandierter Makro-Code:

```
asmlinkage long __x64_sys_read(const struct pg_regs *regs)
{
    return __se_sys_read(regs->di, regs->si, regs->dx);
}

static long __se_sys_read(long fd, long buf, long count)
{
    long ret = __do_sys_read((int) fd, (char __user *) buf, (size_t) count);
    return ret;
}

static inline long __do_sys_read(int fd, char __user *buf, size_t count)
```

Paging in StuBSmI

Ziel dieser Übung

Implementieren sie ein Betriebssystem mit Speicherschutz.

Ziel dieser Übung

Implementieren sie ein Betriebssystem mit Speicherschutz.

(ein unbekannter Manager, echte Welt)

Was ist zu tun?

Anwendungen auslagern

- Applikationen als eigene Anwendungen
- Erstellen eines initialen Speicherabbildes

Speicher verwalten

- Speicher erkennen
- Freien (Kern-/Anwendungs-)speicher verwalten
- Adressräume anlegen

Anwendungen in eigenen Binärdateien

Bisher:

- Anwendungscode und Daten sind mit Kernelobjekten vermischt
- Alles in einer großen System-ELF-Datei

Anwendungen in eigenen Binärdateien

Bisher:

- Anwendungscode und Daten sind mit Kernelobjekten vermischt
- Alles in einer großen System-ELF-Datei

Ab heute: Initiales Speicherabbild (Initrd)
Aufteilung des Codes in unterschiedliche Verzeichnisse

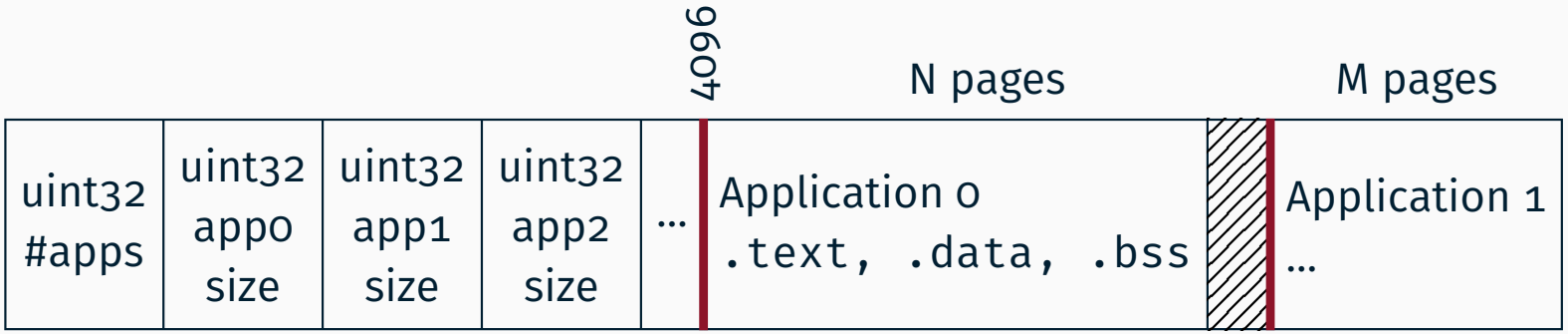
- `kernel/` liefert weiterhin System-ELF
- `libsys/` enthält Funktionen zur Anwendungsunterstützung (Systemaufrufstümpfe, `write()` Wrapper)
- `user/` Enthält mehrere Anwendungen; Jede Anwendung wird gegen libsys gelinkt und zu eigenem ELF kompiliert

Initiales Speicherabbild erstellen

- Die Applikation besteht aus mehreren Objektdateien
  - Keine Nutzung/Abh. von Kernel-Objekten (z.B. `Thread`)!
  - Binden der Applikation mit `init.o` `[sections.ld]`
  - Einsprungspunkt aus `init.o`: `main` (statt `action`)
- Extrahieren von Text und Datensegement `[objcopy]`

Initiales Speicherabbild erstellen

- Die Applikation besteht aus mehreren Objektdateien
  - Keine Nutzung/Abh. von Kernel-Objekten (z.B. `Thread`)!
  - Binden der Applikation mit `init.o` `[sections.ld]`
  - Einsprungspunkt aus `init.o`: `main` (statt `action`)
- Extrahieren von Text und Datensegement `[objcopy]`
- `./imgbuilder app0.img app1.img app2.img > initrd.img`



Speicher erkennen

Wieviel Speicher hat unser Betriebssystem eigentlich zur Verfügung?

Speicher erkennen

Wieviel Speicher hat unser Betriebssystem eigentlich zur Verfügung?

- Können wir aus dem Multiboot-Info auslesen `[mmap_addr]`
- Bootloader gibt uns u.a. eine Liste mit freien und belegten Speicherbereichen, Anzahl geladener Module, …

  `[Adresse der Multiboot-Struktur in %ebx beim Startup]`

Speicher erkennen

Wieviel Speicher hat unser Betriebssystem eigentlich zur Verfügung?

- Können wir aus dem Multiboot-Info auslesen `[mmap_addr]`
- Bootloader gibt uns u.a. eine Liste mit freien und belegten Speicherbereichen, Anzahl geladener Module, …

  `[Adresse der Multiboot-Struktur in %ebx beim Startup]`

… aber:

- Bereiche können sich überlappen
- Bereiche können sich widersprechen (belegt vs. frei)

Speicher erkennen

Wieviel Speicher hat unser Betriebssystem eigentlich zur Verfügung?

- Können wir aus dem Multiboot-Info auslesen `[mmap_addr]`
- Bootloader gibt uns u.a. eine Liste mit freien und belegten Speicherbereichen, Anzahl geladener Module, …

  `[Adresse der Multiboot-Struktur in %ebx beim Startup]`

… aber:

- Bereiche können sich überlappen
- Bereiche können sich widersprechen (belegt vs. frei)

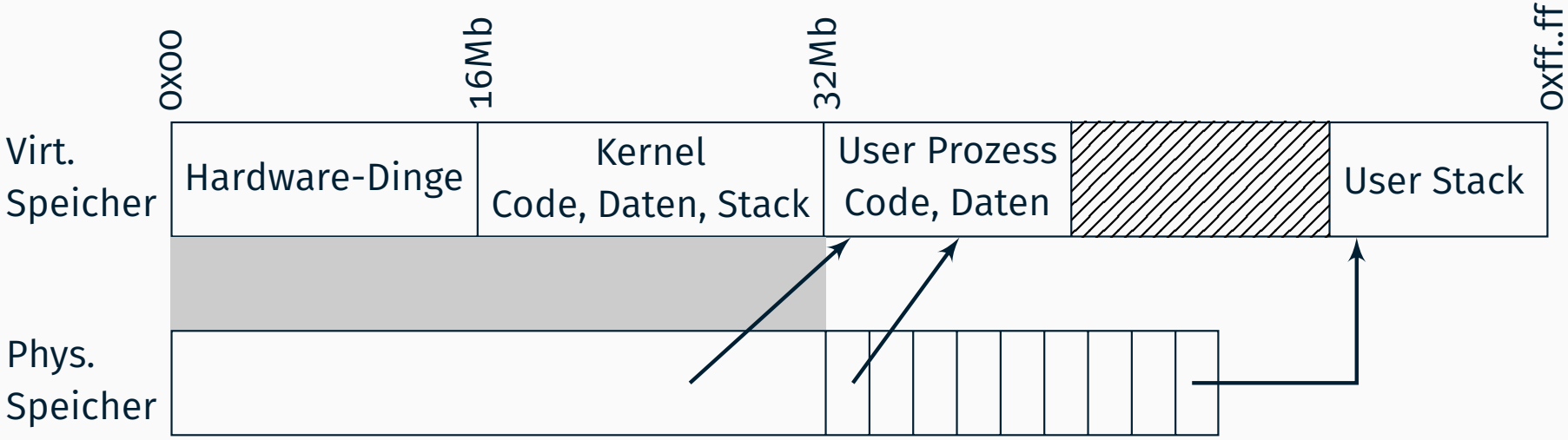
mögliche Lösung:

- Ein Bit pro Page in einer Bitmap
- Alle verfügbaren Pages markieren
- Alle belegten Pages wieder ausknipsen

Wie soll unser Kernel Layout aussehen?

- Der Kernel zwischen 16 und 32 MiB `[lower-half kernel]`
- Identity-Mapping: Physikalische == Virtuelle Adresse

  $\rightarrow$ Für alles zwischen 0 und 32 MiB!
- Möglichkeit Kernelpages oder Userpages zu allokieren



PageTable.translate :: VirtualAddress -> PhysicalAddress

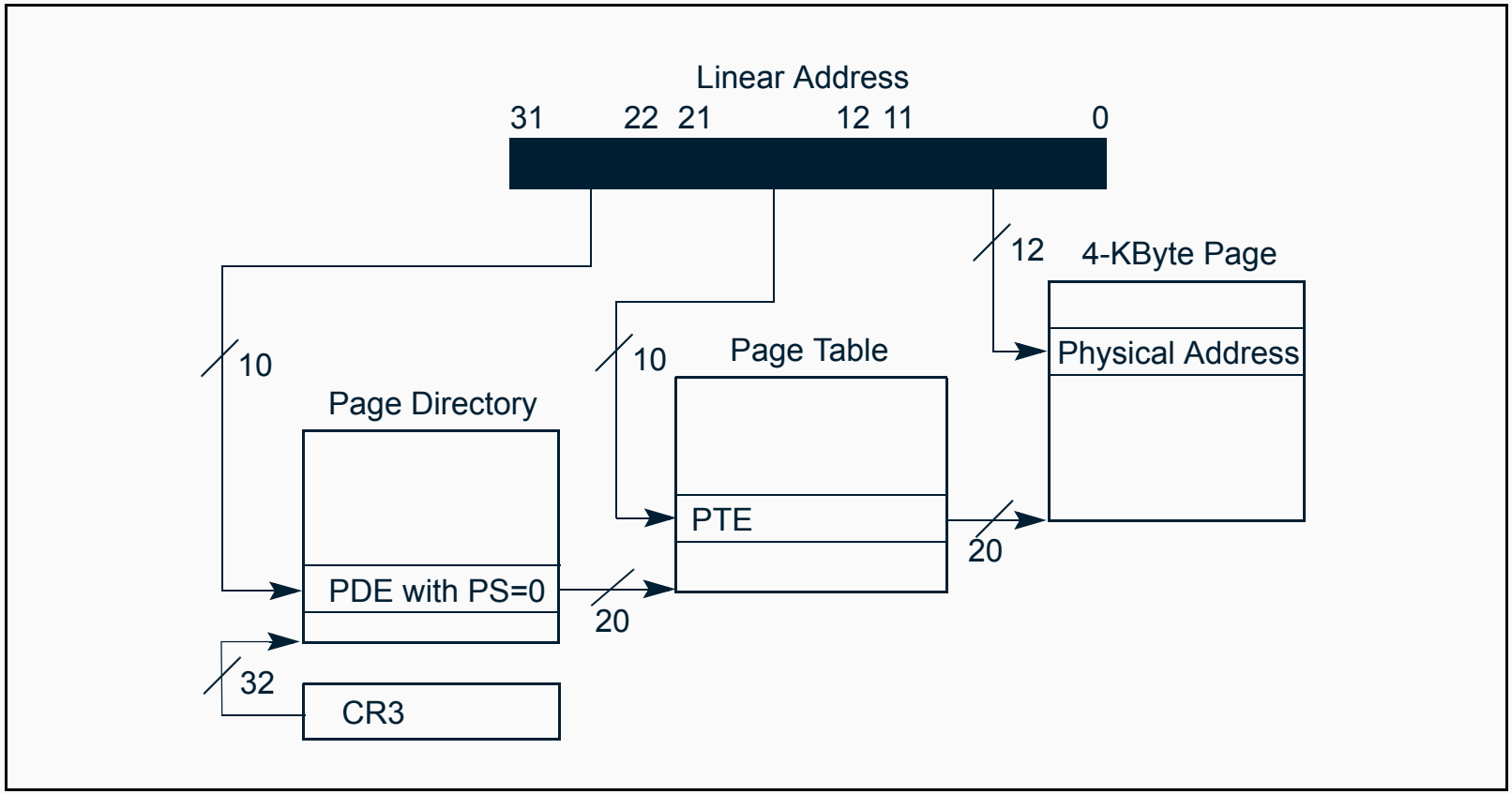


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Ein einzelner Page Table Entry

Address of 4KB page frame	Ignored	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page

0	present
1	read/write
2	user-mode
3,4	caching (page write thru, page cache disabled) $\rightarrow$ 0
5	accessed $\rightarrow$ unwichtig für uns, 0
6	dirty $\rightarrow$ unwichtig für uns, 0
7	Pages der Größe 4kB $\rightarrow$ 0
8	global $\rightarrow$ unwichtig, 0
9..11	zur freien Verfügung
12-31	Physikalische Adresse der Page

Ein einzelner Page Table Entry

Address of 4KB page frame	Ignored	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page

0	present
1	read/write
2	user-mode 
3,4	caching (page write thru, page cache disabled) $\rightarrow$ 0
5	accessed $\rightarrow$ unwichtig für uns, 0
6	dirty $\rightarrow$ unwichtig für uns, 0
7	Pages der Größe 4kB $\rightarrow$ 0
8	global $\rightarrow$ unwichtig, 0
9..11	zur freien Verfügung
12-31	Physikalische Adresse der Page

Aktivierung und weitere Details

Aktivierung der MMU

- Adresse des Page Directories muss in `%cr3`
- Bit `31` muss in `%cr0` gesetzt werden (aktiviert Paging)
- Bit `16` in `%cr0`: `0` $\rightarrow$ Kern darf read-only Pages schreiben
- Hinweis: Mit Schreiben von `%cr3` wird TLB geflusht

Aktivierung und weitere Details

Aktivierung der MMU

- Adresse des Page Directories muss in `%cr3`
- Bit `31` muss in `%cr0` gesetzt werden (aktiviert Paging)
- Bit `16` in `%cr0`: `0` $\rightarrow$ Kern darf read-only Pages schreiben
- Hinweis: Mit Schreiben von `%cr3` wird TLB geflusht

Anpassungen am Rest des Kernels

- Mapping bei `dispatch()` und `go()` ändern
- Einstiegspunkt jedes Tasks ist gleich und fest (z.b. 0x2000000 = 32Mbyte)
- Stackpointer?

Fragen?

Meltdown?

Code aus deutscher Wikipedia (→ Link)

```
; rcx = kernel address
; rbx = probe array
retry:
movzx rax, byte [rcx]      ; Lädt den Inhalt der auszulesenden Speicherstelle in Register rax.
                           ; Das führt zu einer Ausnahmebehandlung.
                           ; Der folgende Code wird nur prozessorintern im Voraus ausgeführt.
shl rax, 12                ; Multipliziert den Inhalt des 64-bit Registers rax mit 4096,
                           ; so dass es nun eine Seitenadresse enthält, die vom Inhalt der
                           ; auszulesenden Speicherstelle abhängt.
jz retry                   ; Beginnt von vorn, wenn das Zero-Flag (und damit hier auch rax)
                           ; gleich 0 ist (Auch bei [rcx]=0 keine Endlosschleife, da aufgrund
                           ; der Ausnahmebehandlung die out-of-order Ausführung schließlich
                           ; abgebrochen wird).
mov rbx, qword [rbx + rax];Greift auf eine Speicherstelle auf der Seite zu, deren Adresse in
                           ; rax steht. Dies lädt eine Seite in den Cache, deren Adresse vom
                           ; Inhalt der auszulesenden Speicherstelle abhängt.
```

Meltdown?

Problembehebung?

Meltdown?

Problembehebung?

$\rightarrow$ Kernel in Anwendung erst gar nicht einblenden!

Meltdown?

Problembehebung?

$\rightarrow$ Kernel in Anwendung erst gar nicht einblenden!

Was passiert bei Systemaufrufen/Interrupts?

Meltdown?

Problembehebung?

→ Kernel in Anwendung erst gar nicht einblenden!

Was passiert bei Systemaufrufen/Interrupts?

→ Kern muss eingeblendet werden → zweites Page Directory
→ Minimaler Zustand auch im User-Page-Directory sichtbar
(Landestelle für Interrupts, IDT, (kleiner) Kernel-Stack)

Problem?

Meltdown?

Problembehebung?

→ Kernel in Anwendung erst gar nicht einblenden!

Was passiert bei Systemaufrufen/Interrupts?

→ Kern muss eingeblendet werden → zweites Page Directory
→ Minimaler Zustand auch im User-Page-Directory sichtbar
(Landestelle für Interrupts, IDT, (kleiner) Kernel-Stack)

Problem?

Wechsel des Page Directories führt zu TLB-Leerung → langsam

Abhilfe schafft z.B. PCID-Unterstützung der MMU (Tagging von Seiten im TLB)