Betriebssystemtechnik

Übung 2 - Den Speicher beseiten

Daniel Danner
Christian Dietrich
Gabor Drescher

May 19, 2015

Ziel dieser Übung

Ziel dieser Übung

Implementieren sie ein Betriebssystem mit Speicherschutz.

– Ein unbekannter Manager, Echte Welt

Was ist zu tun?

Anwendungen auslagern

- Applikationen als eigene Anwendung
- Erstellen eines Initialen Speicherabbildes

Was ist zu tun?

Anwendungen auslagern

- Applikationen als eigene Anwendung
- Erstellen eines Initialen Speicherabbildes

Was ist zu tun?

Speicher verwalten

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

Anwendungen in einenen Binärdateien

- Bisher:
  - Anwendungscode + Daten sind mit Kernelobjekten vermischt
  - Alles in einer großen system ELF Datei

Anwendungen in einenen Binärdateien

- Bisher:
  - Anwendungscode + Daten sind mit Kernelobjekten vermischt
  - Alles in einer großen system ELF Datei

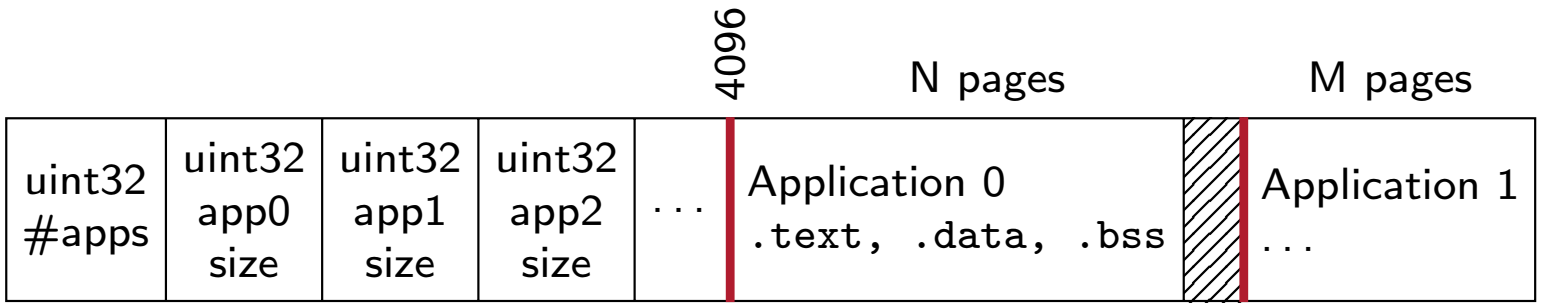
- New Hotness: Initiales Speicherabbild (Initrd)
  Aufteilung des Codes in unterschiedliche Verzeichnisse
  - `kernel/` liefert weiterhin system ELF
  - `libsys/` enthält Funktionen zur Anwendungsunterstützung (Systemaufrufstümpfe, `write()` Wrapper)
  - `user/` Enthält *mehrere* Anwendungen; Jede Anwendung wird gegen libsys gelinkt und zu eigenem ELF kompiliert

Initiales Speicherabbild erstellen

- Die Applikation besteht aus mehreren Objektdateien
- Binden der Applikation mit `init.o` `[sections.ld]`
- Extrahieren von Text und Datensegement `[objcopy]`

Initiales Speicherabbild erstellen

- Die Applikation besteht aus mehreren Objektdateien
- Binden der Applikation mit `init.o` `[sections.ld]`
- 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 Multibootstandard auslesen! Super! `[mmap_addr]`
- Der Booloader gibt uns eine Liste mit freien und belegten Speicherbereichen `[%eax beim Startup]`

Speicher erkennen

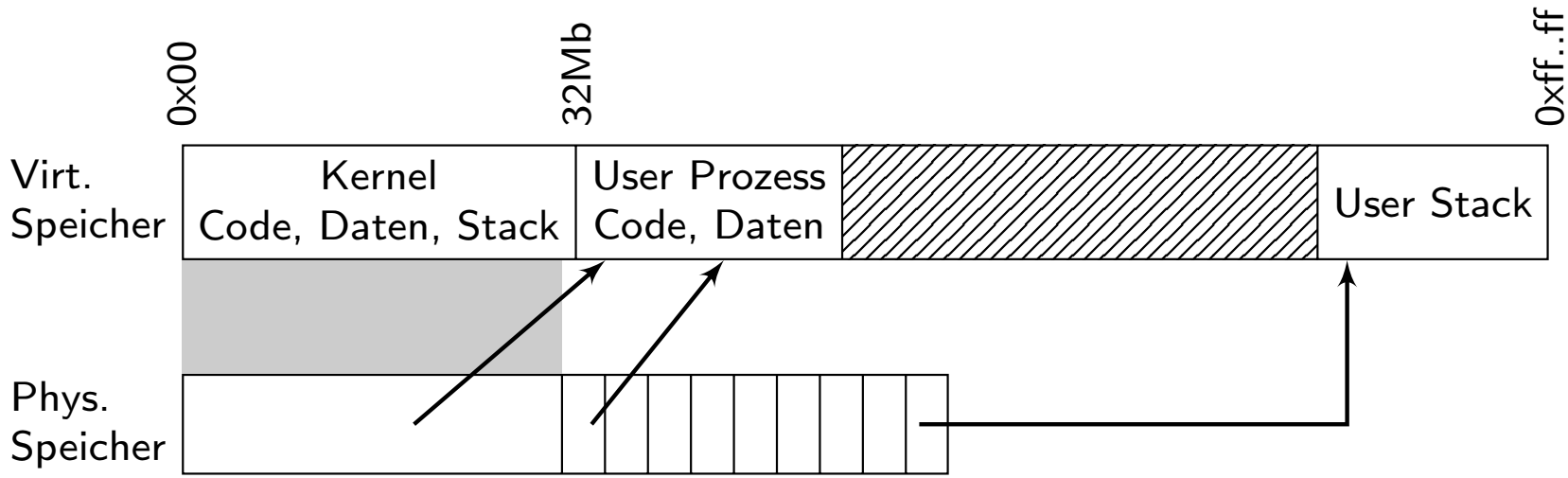
- Wieviel Speicher hat unser Betriebssystem eigentlich zur Verfügung?
- Können wir aus dem Multibootstandard auslesen! Super! `[mmap_addr]`
- Der Booloader gibt uns eine Liste mit freien und belegten Speicherbereichen `[%eax beim Startup]`
- ... aber:
  - Bereiche können sich überlappen
  - Bereiche können sich wiedersprechen (belegt vs. frei)

Speicher erkennen

- Wieviel Speicher hat unser Betriebssystem eigentlich zur Verfügung?
- Können wir aus dem Multibootstandard auslesen! Super! `[mmap_addr]`
- Der Booloader gibt uns eine Liste mit freien und belegten Speicherbereichen `[%eax beim Startup]`
- ... aber:
  - Bereiche können sich überlappen
  - Bereiche können sich wiedersprechen (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 in den untersten 32 Mb `[lower-half kernel]`
- Identity-Mapping: Physikalische Addresse ist gleich der Virtuellen
- Einen Allokator für Kernelpages, einen für Userpages



PageTable.translate :: VirtualAddress -> PhysicalAddress

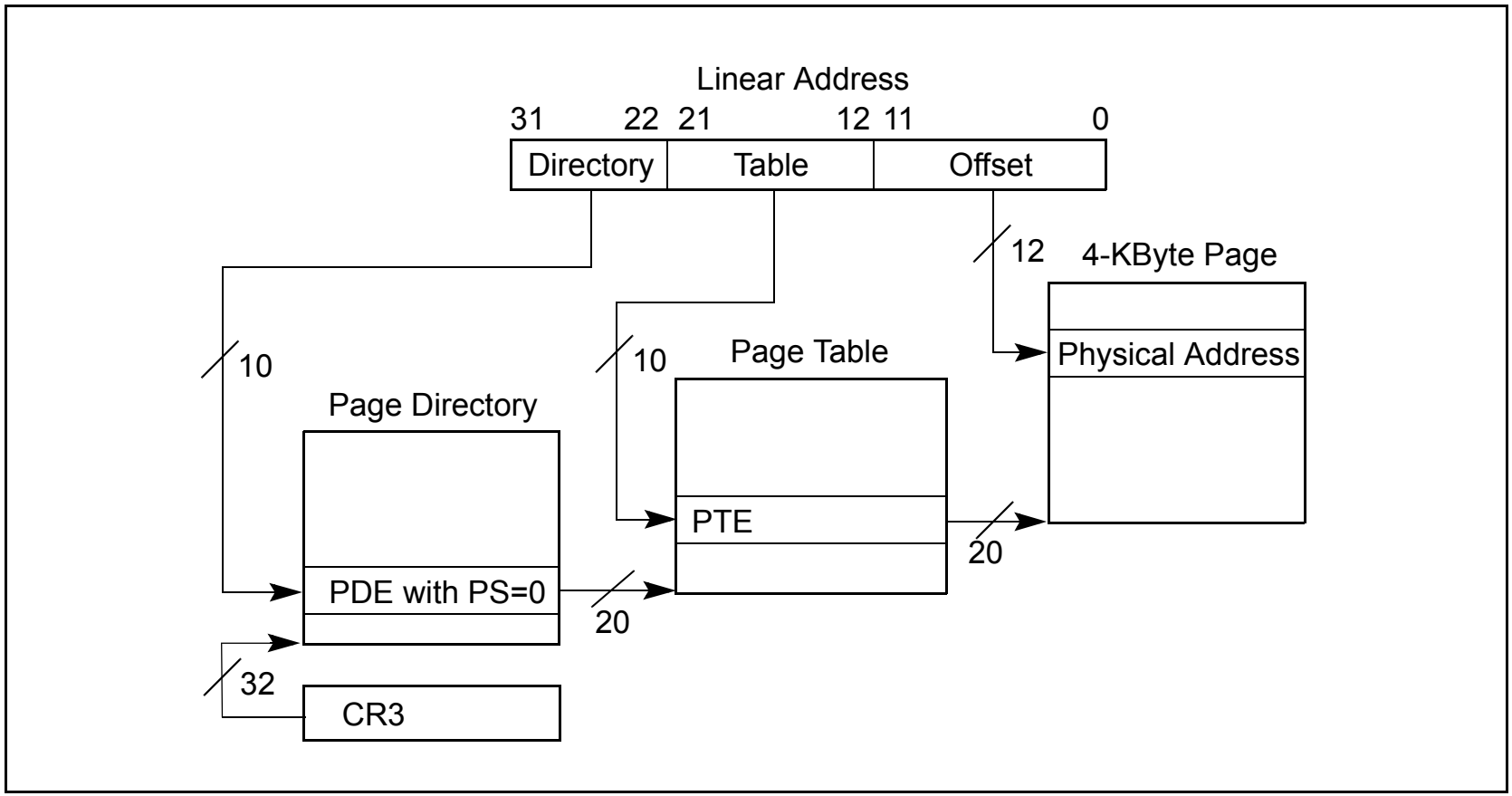


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

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: preset
- 1: read/write
- 2: user-mode
- 3,4: caching (page write thru, page cache disabled) $\rightarrow$ 0
- 5: accessed $\rightarrow$ unwichtig für uns
- 6: dirty *rightarrow* unwichtig für uns
- 7: 4k paging $\rightarrow$ 0
- 8: ignore global $\rightarrow$ unwichtig
- 9..11: zur freien Verfügung
- 12-31: Physikalische Page Addresse

Aktivierung und weitere Details

- Aktivierung des MMU
  - Addresse des Page Directories muss in `%cr3`
  - Bit 31 muss in `%cr0` gesetz werden
  - Hinweis: Mit schreiben von `%cr3` wird TLB geflusht.

Aktivierung und weitere Details

- Aktivierung des MMU
  - Addresse des Page Directories muss in `%cr3`
  - Bit 31 muss in `%cr0` gesetz werden
  - Hinweis: Mit schreiben von `%cr3` wird TLB geflusht.
- Anpassungen am Rest des Kernels
  - Mapping beim `dispatch()` und `go()` ändern
  - Einstiegspunkt jedes Tasks ist gleich und fest (z.b. 0x2000000 = 32Mbyte)
  - Stackpointer?

Aktivierung und weitere Details

- Aktivierung des MMU
  - Addresse des Page Directories muss in `%cr3`
  - Bit 31 muss in `%cr0` gesetz werden
  - Hinweis: Mit schreiben von `%cr3` wird TLB geflusht.
- Anpassungen am Rest des Kernels
  - Mapping beim `dispatch()` und `go()` ändern
  - Einstiegspunkt jedes Tasks ist gleich und fest (z.b. 0x2000000 = 32Mbyte)
  - Stackpointer?
- Quiz:
  - Ist eine verkettete Liste von Userpages sinnvoll?

Aktivierung und weitere Details

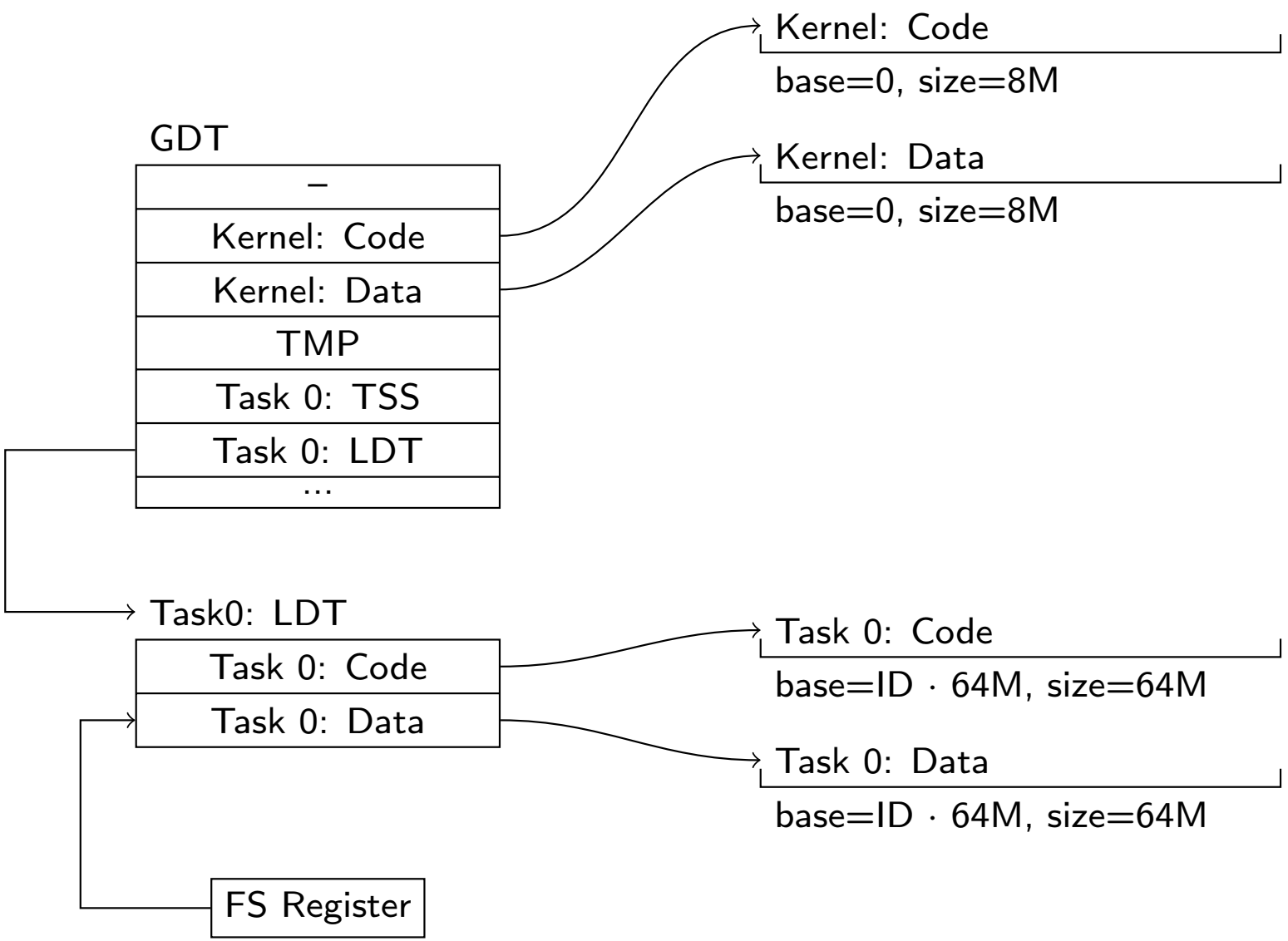
- Aktivierung des MMU
  - Addresse des Page Directories muss in `%cr3`
  - Bit 31 muss in `%cr0` gesetz werden
  - Hinweis: Mit schreiben von `%cr3` wird TLB geflusht.
- Anpassungen am Rest des Kernels
  - Mapping beim `dispatch()` und `go()` ändern
  - Einstiegspunkt jedes Tasks ist gleich und fest (z.b. 0x2000000 = 32Mbyte)
  - Stackpointer?
- Quiz:
  - Ist eine verkettete Liste von Userpages sinnvoll?
  - Bit Sets

Linux 0.01

Linux 0.01

Oder: Wie lebt man mit nur einer Seitenkacheltabelle?

Exkurs: Linux 0.01 – GDT Aufbau



Exkurs: Linux 0.01 - mm/memory.c

```
171  /* This function puts a page in memory at the wanted address.
172   * It returns the physical address of the page gotten, 0 if
173   * out of memory (either when trying to access page-table or
174   * page.) */
175  unsigned long put_page(unsigned long page,unsigned long address)
176  {
177      unsigned long tmp, *page_table;
178   /* NOTE !!! This uses the fact that _pg_dir=0 */
179      if (page < LOW_MEM || page > HIGH_MEMORY)
180          printk("Trying to put page %p at %p\n",page,address);
181      if (mem_map[(page-LOW_MEM)>>12] != 1)
182          printk("mem_map disagrees with %p at %p\n",page,address);
183      page_table = (unsigned long *) ((address>>20) & 0xffc);
184      if ((*page_table)&1)
185          page_table = (unsigned long *) (0xfffff000 & *page_table);
186      else {
187          if (!(tmp=get_free_page()))
188              return 0;
189          *page_table = tmp|7;
190          page_table = (unsigned long *) tmp;
191      }
192      page_table[(address>>12) & 0x3ff] = page | 7;
193      return page;
194  }
```