Betriebssystem-Entwicklung mit Literate Programming

Foliensatz 6: Frame- und Seitenverwaltung

Hans-Georg Eßer
TH Nürnberg

v1.1, 11.11.2014

Chunk: ⟨global variables⟩ (1)

- **Nachtrag zu Foliensatz 5**
- Wir wollen auch auf den Videospeicher zugreifen
- Phys. RAM: 0xb8000-0xb9000 (= 4 KB)
  - eine Page Table reicht
  - wir reservieren den Speicher für diese Tabelle vorab; aligned!
  - Einrichtung erst, nachdem Paging schon aktiv ist und wir das direkte Identity Mapping der ersten 4 MB nicht mehr brauchen

⟨*global variables*⟩≡
```
  page_table video_pt __attribute__ ((aligned (409↩
…6)));  // must be aligned!
```

Chunk: ⟨initialize system⟩ (1)

- **Füllen der Tabelle:**
  - erst mal alles auf 0
  - dann im Eintrag 0xB8 die Adresse 0xB8000 eintragen
  - Im PD Eintrag 0 (für die untersten 4 MB) auf diese Videotabelle zeigen lassen
  - warum "-0xC0000000"?
  - Ergebnis: Zugriff auf virtuelle Adressen 0xB8000..0xB9000 über gleiche phys. Adressen (also auch *Identity Mapping*)

⟨*initialize system*⟩≡
```
  for (int i=0; i<1024; i++) {
    // null entries:
    fill_page_desc ( &(video_pt.pds[i]), false,fal↩
…se,false,false,0 );
  };

  KMAP ( &(video_pt.pds[0xB8]), 0xB8*4096 );   // ↩
…one page of video RAM

  // enter new table in page directory
  KMAPD ( &(current_pd->ptds[0]), (unsigned int) (↩
…&video_pt) - 0xC0000000 );

  gdt_flush();
```

Chunk: ⟨remove video mapping⟩ (1)

- Später werden wir noch generischen Zugriff auf den phys. Speicher freischalten (über 0xD0000000...)
- Wenn das läuft, ist Zugriff auf Videospeicher über 0xD00B8000... möglich
- Dann können wir dieses Mapping wieder löschen

⟨*remove video mapping*⟩≡
```
  // null entry:
  fill_page_desc ( &(video_pt.pds[0xB8]), false,fa↩
…lse,false,false,0 );
  gdt_flush();
```

Chunk: ⟨constants⟩ (1)

Verwalten der Page Frames (phys. Speicher)

- VM hat 64 MB RAM
- Seitengröße = Rahmengröße = 4 KB
- Es gibt also 64 M / 4 K = 16 K Rahmen im RAM
  → `NUMBER_OF_FRAMES`

⟨*constants*⟩≡
```
  #define MEM_SIZE 1024*1024*64         // 64 MByte
  #define MAX_ADDRESS MEM_SIZE-1        // last val↩
…id physical address
  #define PAGE_SIZE 4096                // Intel: 4↩
…K pages
  #define NUMBER_OF_FRAMES MEM_SIZE/PAGE_SIZE
```

Chunk: ⟨global variables⟩ (2)

- In `free_frames` die freien Rahmen merken

⟨*global variables*⟩+≡
```
  unsigned int free_frames = NUMBER_OF_FRAMES;
```

Chunk: ⟨global variables⟩ (3)

- Außerdem legen wir eine Rahmentabelle an
- Die enthält für jeden Rahmen ein Bit:
  - 0 = frei
  - 1 = belegt

⟨*global variables*⟩+≡
```
  char place_for_ftable[NUMBER_OF_FRAMES/8];
  unsigned int* ftable = (unsigned int*)(&place_fo↩
…r_ftable);
```

Chunk: ⟨initialize system⟩ (2)

- Rahmentabelle initialisieren
  = alle Bits auf 0 setzen (alles frei)

⟨*initialize system*⟩+≡
```
  memset (ftable, 0, NUMBER_OF_FRAMES/8);  // all ↩
…frames are free
```

Chunk: ⟨initialize system⟩ (3)

- Die ersten 4 MB RAM enthalten u. a. den Kernel
- Weiteren Speicher dort nicht nutzen → diese 4 MB als belegt eintragen
- wie viele Bits setzen?
  - 4 MB belegen 1024 Rahmen
  - also die ersten 1024 Bits auf 1 setzen
  - 8 Bit pro Byte → 1024 / 8 = 128 Bytes
- 0xFF = 11111111 (bin)

⟨*initialize system*⟩+≡
```
  memset (ftable, 0xff, 128);
  free_frames -= 1024;
```

Chunk: ⟨macro definitions⟩ (1)

Abfragen / Ändern der Bits in ftable

- Wir haben `ftable` als `unsigned int*` deklariert
  → Zeiger auf 32-Bit-Integer
- Die 1. 32 Bits erreichen wir über `ftable[0]`
- Die 2. 32 Bits erreichen wir über `ftable[1]` usw.
- Makro `INDEX_FROM_BIT()` wählt richtigen 32-Bit-Eintrag in ftable,
- Makro `OFFSET_FROM_BIT()` wählt darin die richtige Bitnummer

⟨*macro definitions*⟩≡
```
  #define INDEX_FROM_BIT(b) (b/32)   // 32 bits in↩
… an unsigned int
  #define OFFSET_FROM_BIT(b) (b%32)
```

Chunk: ⟨function implementations⟩ (1)

- Damit ist es leicht, einzelne Bits anzusprechen:
- Bit setzen = mit (1 << offset) "odern"
- Bit entfernen = mit ~(1 << offset) "unden"
- z.B.: offset = 3,
  ...111111 & ~(1<<3)
  = ...111111 & ~(...001000)
  = ...111111 & ...110111
  = ...110111

⟨*function implementations*⟩≡

```
  static void set_frame (unsigned int frame_addr) ↩
…{
    unsigned int frame = frame_addr / PAGE_SIZE;
    unsigned int index = INDEX_FROM_BIT (frame);
    unsigned int offset = OFFSET_FROM_BIT (frame);
    ftable[index] |= (1 << offset);
  }

  static void clear_frame (unsigned int frame_addr↩
…) {
    unsigned int frame = frame_addr / PAGE_SIZE;
    unsigned int index = INDEX_FROM_BIT (frame);
    unsigned int offset = OFFSET_FROM_BIT (frame);
    ftable[index] &= ~(1 << offset);
  }
```

Chunk: ⟨function implementations⟩ (2)

- Abfragen eines Bits durch Und-Verknüpfung und Rechts-Shift
- Beispiel: offset = 4
- 1. Schritt:
  abc**d**efgh & (1<<4)=
  abc**d**efgh & (10000)=
  000**d**0000
- 2. Schritt:
  000**d**0000 >> 4=
  0000000**d** = **d**

⟨*function implementations*⟩+≡

```
  static unsigned int test_frame (unsigned int fra↩
…me_addr) {
    // returns true if frame is in use (false if f↩
…rame is free)
    unsigned int frame = frame_addr / PAGE_SIZE;
    unsigned int index = INDEX_FROM_BIT (frame);
    unsigned int offset = OFFSET_FROM_BIT (frame);
    return ((ftable[index] & (1 << offset)) >> offset);
  }
```

Chunk: ⟨example call of test_frame⟩ (1)

⟨*example call of test_frame*⟩≡

```
  if ( test_frame (address) ) {
    // result non-0 (true); frame is not available
  } else {
    // result 0 (false);    frame is available
  }
```

Chunk: ⟨global variables⟩ (4)

- Zugriff auf phys. Speicher über virtuelle Adressen 0xD0000000...0xD3FFFFFF einrichten
- Für 64 MB RAM brauchen wir 64 M / 4 M = 16 Seitentabellen, die 16 K Seiten verwalten
- vorab reservieren (aligned!)

⟨*global variables*⟩+≡

```
  page_table kernel_pt_ram[16] __attribute__ ((ali↩
…gned (4096)));
```

Chunk: ⟨initialize system⟩ (4)

- Einrichten des Mappings: Schleife von 0 bis 16K-1
- virtuelle Adresse `0xD0000000 + fid * 4K`
  → Frame `fid`

⟨*initialize system*⟩+≡

```
  unsigned int fid;
  for (fid=0; fid<NUMBER_OF_FRAMES; fid++) {
    ⟨map page starting at 0xD0000000 + PAGE_SIZE*fid to frame fid⟩
  }
```

Chunk: ⟨map page starting at 0xD0000000 + PAGE_SIZE*fid to frame fid⟩ (1)

- Wir nutzen das KMAP-Makro (Zugriff nur für Kernel erlaubt)
- Je 1024 Einträge in einer Tabelle, darum:
  - Tabelle fid/1024 (ergibt Werte in 0..15)
  - in der Tabelle Position fid%1024
- Alternativ auch einfacherer Zugriff auf `kernel_pt_ram[0].pds[fid]` möglich (böser Hack)

⟨*map page starting at 0xD0000000 + PAGE_SIZE*fid to frame fid*⟩≡

```
  KMAP ( &(kernel_pt_ram[fid/1024].pds[fid%1024]),↩
… fid*PAGE_SIZE );
```

Chunk: ⟨initialize system⟩ (5)

- Jetzt müssen wir die 16 neuen Seitentabellen noch in das Page Directory eintragen
- Welche Einträge?
  - Jeder Eintrag im Page Directory ist für eine Seitentabelle, also 1024 Seiten = 4 MB zuständig
  - 0xD0000000 / 4M = 832
  - also 832 i (i=0,1,...,15)
- Vorsicht: Adressen wie `&(kernel_pt_ram[i])` sind >= 0xC0000000, also abziehen

⟨*initialize system*⟩+≡

```
  unsigned int physaddr;
  for (int i=0; i<16; i++) {
    // get physical address of kernel_pt_ram[i]
    physaddr = (unsigned int)(&(kernel_pt_ram[i]))↩
… - 0xc0000000;
    KMAPD ( &(current_pd->ptds[832+i]), physaddr )↩
…;
  };
  kputs ("Physical RAM (64 MB) mapped to 0xD000000↩
…0-0xD3FFFFFF.\n");
```

Chunk: ⟨macro definitions⟩ (2)

- Makro für den Schnellzugriff

⟨*macro definitions*⟩+≡

```
  #define PHYSICAL(x) ((x)+0xd0000000)
```

Chunk: ⟨initialize system⟩ (6)

- Jetzt den Zugriff auf den Videospeicher umstellen von 0xB8000 auf 0xD00B8000
- Danach kann das Identity Mapping für den Videospeicher weg

⟨*initialize system*⟩+≡

```
  VIDEORAM=0xD00B8000;
  textmemptr = (unsigned short *)VIDEORAM;
  ⟨remove video mapping⟩
```

Chunk: ⟨function prototypes⟩ (1)

- Manchmal Umwandlung Seitennummer → Rahmennummer nützlich
- (das macht die MMU bei der Adressübersetzung automatisch)
- Wir simulieren das Verhalten der MMU (→ nächste Folie)

⟨*function prototypes*⟩≡

```
  unsigned int pageno_to_frameno (unsigned int pag↩
…eno);
```

Chunk: ⟨function implementations⟩ (3)

⟨*function implementations*⟩+≡
```
  unsigned int pageno_to_frameno (unsigned int pageno) {
    unsigned int pdindex = pageno/1024;
    unsigned int ptindex = pageno%1024;
    if ( ! current_pd->ptds[pdindex].present ) {
      return -1;       // we don't have that page table
    } else {
      // get the page table
      page_table* pt = (page_table*)
                       ( PHYSICAL(current_pd->ptds[pdindex].frame_addr << 12) );
      // note: frame_addr holds a physical address. luckily we have a mapping
      // of physical address space to 0xd000.0000 and above...
      if ( pt->pds[ptindex].present ) {
        return pt->pds[ptindex].frame_addr;
      } else {
        return -1;     // we don't have that page
      };
    };
  };
```

Chunk: ⟨macro definitions⟩ (3)

⟨*macro definitions*⟩+≡
```
  /* Peek and Poke for virtual addresses */
  #define PEEK(addr)      (*(unsigned char *)(addr))
  #define POKE(addr, b)   (*(unsigned char *)(addr) = (b))
  /* Peek and Poke for physical addresses 0..64 MB */
  #define PEEKPH(addr)    (*(unsigned char *)(PHYSICAL(addr)))
  #define POKEPH(addr, b) (*(unsigned char *)(PHYSICAL(addr)) = (b))
  #define PEEK_UINT(addr)    (*(unsigned int *)(addr))
  #define POKE_UINT(addr, b) (*(unsigned int *)(addr) = (b))
  #define PEEKPH_UINT(addr)    (*(unsigned int *)(PHYSICAL(addr)))
  #define POKEPH_UINT(addr, b) (*(unsigned int *)(PHYSICAL(addr)) = (b))
```

Chunk: ⟨peek and poke example⟩ (1)

⟨*peek and poke example*⟩≡
```
  unsigned int testvar;
  unsigned int address = (unsigned int)&testvar;
  POKE (address, 0x12);
  POKE (address+1, 0x34);
  POKE (address+2, 0x56);
  POKE (address+3, 0x78);
  printf ("32-bit value: %x\n", PEEK_UINT (address));
```

Chunk: ⟨function implementations⟩ (4)

Verwaltung der Rahmen

- Neuen Rahmen anfordern

⟨*function implementations*⟩+≡
```
  int request_new_frame () {
    ⟨find a free frame und reserve it⟩
  };
```

Chunk: ⟨find a free frame und reserve it⟩ (1)

- Suche mit `test_frame` nach freiem Rahmen

⟨*find a free frame und reserve it*⟩≡
```
  unsigned int frameid;
  boolean found=false;
  for (frameid = 0; frameid < NUMBER_OF_FRAMES; fr↩
…ameid++) {
    if ( !test_frame (frameid*4096) ) {
      found=true;
      break;   // frame found
    };
  }
```

Chunk: ⟨find a free frame und reserve it⟩ (2)

- Wenn wir einen freien finden:
  - als belegt markieren
  - `free_frames` anpassen
  - Nummer zurückgeben

⟨*find a free frame und reserve it*⟩+≡
```
  if (found) {
    set_frame (frameid*4096);
    free_frames--;
    return frameid;
  } else {
    return -1;
  }
```

Chunk: ⟨function implementations⟩ (5)

- Rahmen wieder freigeben:
  - Bit wieder auf 0 setzen
  - `free_frames` anpassen

⟨*function implementations*⟩+≡
```
  void release_frame (unsigned int frameaddr) {
    if ( test_frame (frameaddr) ) {
      // only do work if frame is marked as used
      clear_frame (frameaddr);
      free_frames++;
    };
  };
```

Chunk: ⟨constants⟩ (2)

- (Wir haben keine Standardbibliothek; irgendwo muss NULL mal definiert werden)

⟨*constants*⟩+≡
```
  #define NULL ((void*) 0)
```

Chunk: ⟨function implementations⟩ (6)

Verwaltung der Seiten (virtueller Speicher)

- neue Seite anfordern:
- zuerst einen Rahmen organisieren

⟨*function implementations*⟩+≡
```
  unsigned int* request_new_page (int need_more_pa↩
…ges) {
    unsigned int newframeid = request_new_frame ()↩
…;
    if (newframeid == -1)  return NULL;   // exit ↩
…if no frame was found
    ⟨enter frame in page table⟩
  };
```

Chunk: ⟨enter frame in page table⟩ (1)

- dann den Rahmen in die Seitentabelle eintragen
- dazu nächste freie Seitennummer suchen
- `mmu_p()` ist im Prinzip `pageno_to_frameno()`, nur mit Adressräumen (→ später)
- `pageno` enthält Seitennummer

⟨*enter frame in page table*⟩≡
```
  unsigned int pageno = -1;
  for (unsigned int i=0xc0000; i<1024*1024; i++) {
    if ( mmu_p (current_as, i) == -1 ) {
      // mmu_p() is similar to pageno_to_frameno()
      pageno = i;
      break;       // end loop, unmapped page was ↩
…found
    };
  };

  if ( pageno == -1 ) {
    return NULL;   // we found no page -- whole 4 ↩
…GB are mapped???
  };
```

Chunk: ⟨enter frame in page table⟩ (2)

- Jetzt das Mapping `pageno` → `frameno` einrichten
- Welche Seitentabelle? `pdindex`
- Und darin welche Seite? `ptindex`
- Evtl. erst neue Seitentabelle einrichten (verwendet dann den frischen Rahmen, und wir brauchen einen neuen)
- Eintragung mit KMAP (übernächste Folie)

⟨*enter frame in page table*⟩+≡
```
  unsigned int pdindex = pageno/1024;
  unsigned int ptindex = pageno%1024;
  page_table* pt;

  if (ptindex == 0) {
    // last entry!!
    ⟨create new page table⟩
    newframeid = request_new_frame ();  // get yet↩
… another frame
    if (newframeid == -1) {
      return NULL;                      // exit if↩
… no frame was found
      // note: we're not removing the new page tab↩
…le since we assume
      // it will be used soon anyway
    }
  };
```

Chunk: ⟨global variables⟩ (5)

- (ignorieren)

⟨*global variables*⟩+≡
```
  short int DEBUG=0;
```

Chunk: ⟨enter frame in page table⟩ (3)

⟨*enter frame in page table*⟩+≡
```
  if ( ! current_pd->ptds[pdindex].present ) {
    // we don't have that page table -- this should not happen!
    kputs ("FAIL! No page table entry\n");
    return NULL;
  } else {
    // get the page table
    pt = (page_table*)( PHYSICAL(current_pd->ptds[pdindex].frame_addr << 12) );
    // finally: enter the frame address
    KMAP ( &(pt->pds[ptindex]), newframeid * PAGE_SIZE );

    // invalidate cache entry
    asm volatile ("invlpg %0" : : "m"(*(char*)(pageno<<12)) );
  };
```

Chunk: ⟨enter frame in page table⟩ (4)

- Seite ist jetzt eingetragen
- Inhalt initialisieren (alles auf 0 setzen)
- und Seitennummer zurückgeben

⟨*enter frame in page table*⟩+≡
```
memset ((unsigned int*) (pageno*4096), 0, 4096);
return ((unsigned int*) (pageno*4096));
```

Chunk: ⟨create new page table⟩ (1)

- Es bleibt noch das Erzeugen einer neuen Seitentabelle (falls die "letzte" voll ist)
- wir verwenden dafür den Rahmen, der eigentlich für die neue Seite gedacht war

⟨*create new page table*⟩≡
```
// create a new page table in there
page_table* pt = (page_table*) PHYSICAL(newframeid<<12);
memset (pt, 0, PAGE_SIZE);
```

Chunk: ⟨create new page table⟩ (2)

- Im ursprünglichen Code (ohne Prozesse) ging es so
- später kommen aber noch Adressräume hinzu (→ mehrere Page Directories, eines pro Prozess)
- wir müssen die neue Seitentabelle in *allen* Page Directories eintragen (→ nächste Folie)

⟨*create new page table*⟩+≡
```
// KMAPD ( &(current_pd->
   ptds[pdindex]), newframeid << 12 );
```

Chunk: ⟨create new page table⟩ (3)

⟨*create new page table*⟩+≡
```
int asid;
page_directory* tmp_pd;
for (asid=0; asid<1024; asid++) {
  if (!address_spaces[asid].free   // is this address space in use?
      && asid != create_as)        // do not modify an address space which is cur-
                                   // rently created via create_new_address_space
  {
    tmp_pd = address_spaces[asid].pd;
    KMAPD ( &(tmp_pd->ptds[pdindex]), newframeid << 12 );
  }
}
```

Chunk: ⟨function implementations⟩ (7)

Freigeben einer Seite ist leichter:

- Mapping Seite → Rahmen entfernen
- zugehörigen Rahmen freigeben

⟨*function implementations*⟩+≡
```
void release_page (unsigned int pageno) {
  ⟨remove page to frame mapping from page table⟩
  ⟨release corresponding frame⟩
};
```

Chunk: ⟨remove page to frame mapping from page table⟩ (1)

- Rahmennummer aus Seitennummer berechnen
- braucht wieder `mmu_p` statt `pageno_to_frameno` (wegen Adressräumen, s. o.)

⟨*remove page to frame mapping from page table*⟩≡

```
  // int frameno = pageno_to_frameno (pageno);  //↩
… we will need this later
  int frameno = mmu_p (current_as, pageno);  // we↩
… will need this later
  if ( frameno == -1 )  return;              // ex↩
…it if no such page
```

Chunk: ⟨remove page to frame mapping from page table⟩ (2)

- Richtige Seitentabelle und darin richtige Position suchen (wie beim Eintragen, über `pdindex` und `ptindex`)

⟨*remove page to frame mapping from page table*⟩+≡

```
  unsigned int pdindex = pageno/1024;
  unsigned int ptindex = pageno%1024;
  page_table* pt;
  pt = (page_table*)( PHYSICAL(current_pd->ptds
     [pdindex].frame_addr << 12) );
  // write null page descriptor
  fill_page_desc ( &(pt->pds[ptindex]), false, fal↩
…se, false, false, 0 );
```

Chunk: ⟨release corresponding frame⟩ (1)

- Schließlich Rahmen freigeben
- `frameno<<12`, weil `release_frame` Adresse erwartet

⟨*release corresponding frame*⟩≡

```
  release_frame (frameno<<12);   // expects an address, not an ID
  // note: release_frame increases free_frames
```

Chunk: ⟨function implementations⟩ (8)

- Service-Funktion: Ganzen Bereich freigeben
- und das war's auch schon.

⟨*function implementations*⟩+≡

```
  void release_page_range (unsigned int start_page↩
…no, unsigned int end_pageno) {
    for (int i = start_pageno; i < end_pageno+1; i↩
…++ )  release_page (i);
  };
```