Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

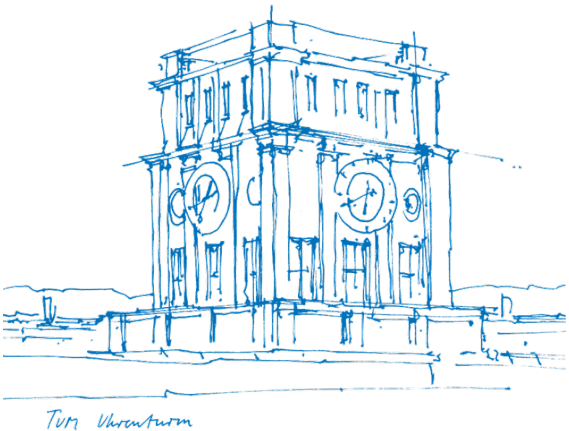
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

Mario Delic

Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

Übungswoche 11



Dateien

• **Genaue Implementierung von Dateimechanismen kann variieren**,z.B.:
↪ Windows: name.endung; interpretierte Endungen (.exe, .txt)
↪ UNIX: freie Benennung; Endungen per default nicht interpretiert

• **Strukturierung**:
- Unstrukturiert: Folge von Bytes (Windows, UNIX)
- Sequenzen von Einträgen: Einträge fester Größe und Struktur (i.d.R. nicht mehr genutzt)
- sortierte Baumstruktur: Einträge variabler Größe; für Verwaltung großer Datenmengen (BS in Großrechnern)

• **Einige unterstütze Typen:**
Directories (d), Files (-: files, hard links)(l: symbolic links), Character Special Files (c), Block Special Files (b)

in Klammern: Symbole neben Permissions bei Ausgabe mit „ls -la"

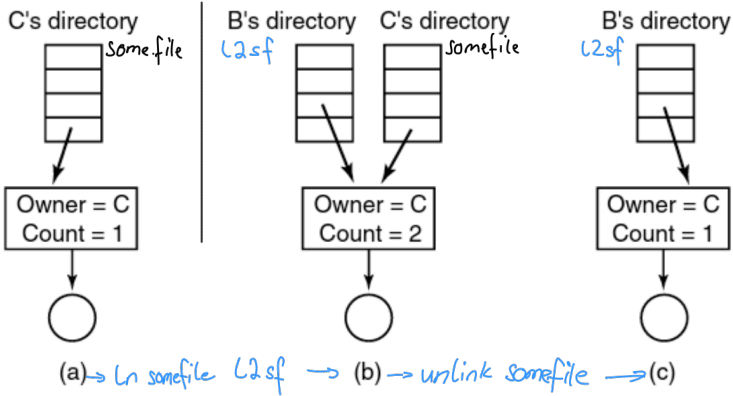
Dateien

•Hard & symbolic links:

- Hard link: verweist direkt auf die selbe i-node wie link target
- Symbolic link: verweist per Dateipfad auf einen Dateieintrag im FS

ln target linkname

ln -s target linkname ⇒ -s = symbolic

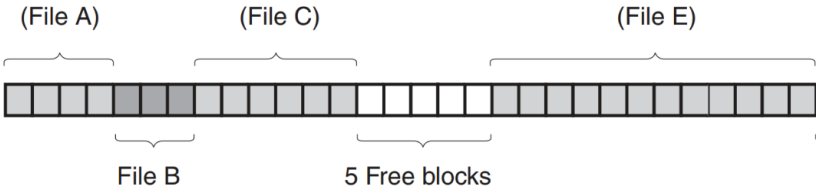


↳ unlink in UNIX wird (u.a) zum Löschen genutzt

Implementierung

• **Contiguous Allocation**:

- Datei wird als zusammenhängende Folge von Blöcken auf der Festplatte verwaltet



- Vorteil: Sehr simpel; Nachteil: Fragmentierung (Löcher im Speicher)
- zu finden bei ROMs (CDs, DVDs, etc.)

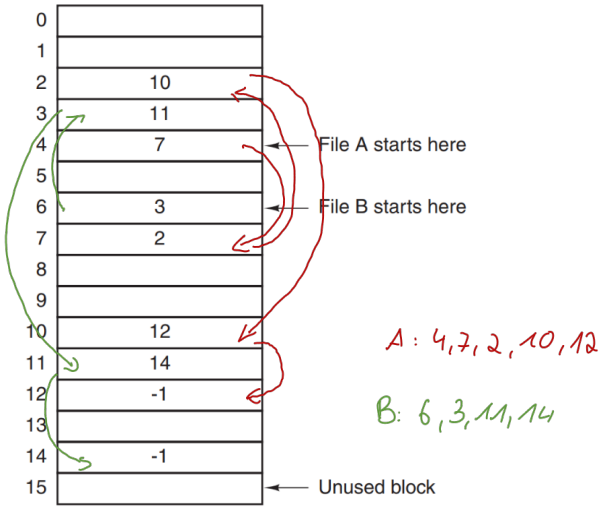
Implementierung

• **Linked List Allocation**:
- Die durch die Datei belegten Blöcke werden in einer verketteten Liste verwaltet
- Das erste Wort innerhalb des Blocks wird als Zeiger auf den nächsten Block verwendet

- Vorteil: Nahezu keine Fragmentierung (lediglich letzer Block); Nachteil: langsam, Blöcklgröße =|= 2er-Potenz (wegen Anfangspointer))
- Lösung: Seperate Tabelle speichert die Allokation der Blöcke

↪ FAT (File Allocation Table)

Implementierung



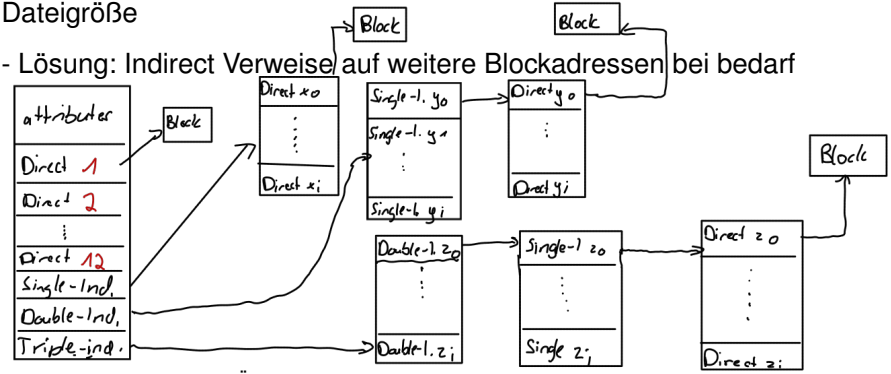
Quelle: Tanenbaum/Bos

Implementierung

• **index-nodes (i-node)**:
- Jede Datei wird repräsentiert durch eine i-node
- Enthält Attribute und Belegte Blöcke

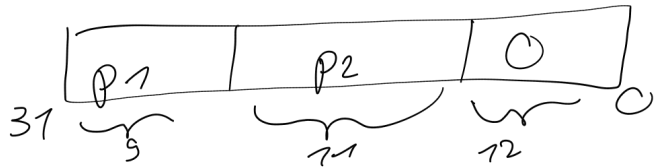
- Vorteil: Muss nur für geöffnete Dateien geladen werden; Nachteil: theoretisch begrenzte Dateigröße

- Lösung: Indirect Verweise auf weitere Blockadressen bei bedarf



Aufgabe 2

a) Ein Computer mit **32-bit breiten virtuellen Adressen** benutzt eine zweistufige Seitentabelle zur Adressübersetzung.
Eine virtuelle Adresse bestehe aus **9 Bits für die erste Stufe**, **11 Bits für die zweite Stufe** sowie einem Offset. Wie sieht die Adresse aus?



b) Wie groß sind die Seiten?

12-bit → 2^{12} Byte → 4 KiB

c) Aus wie vielen Seiten besteht der virtuelle Adressraum?

$\frac{2^{32}}{2^{12}} = 2^{20}$ | $2^9 \cdot 2^{11} = 2^{20}$

Auszug aus dem Intel-Manual:

32-bit Paging mit 4 KiB Pages ≈ „Gas" Paging

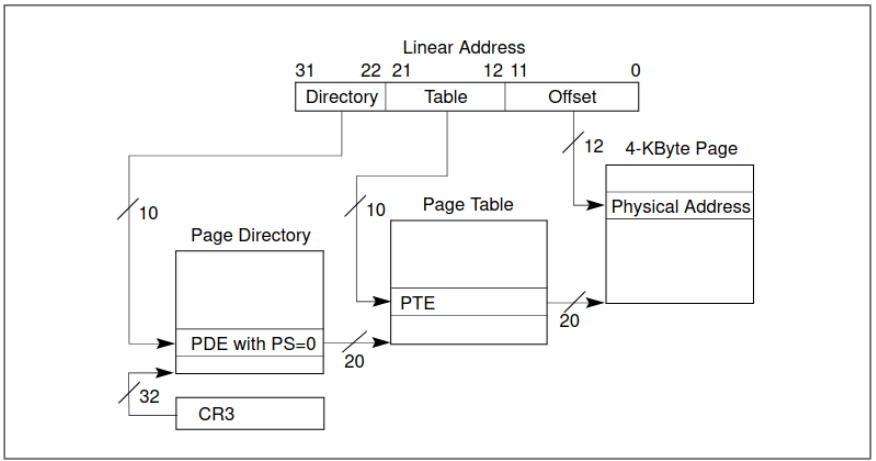


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Auszug aus dem Intel-Manual:
32-bit Paging mit 4 MiB Pages

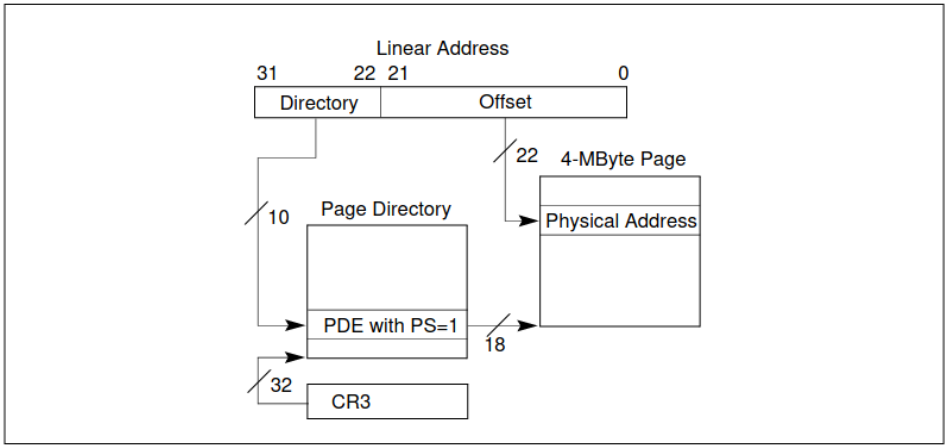


Figure 4-3. Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

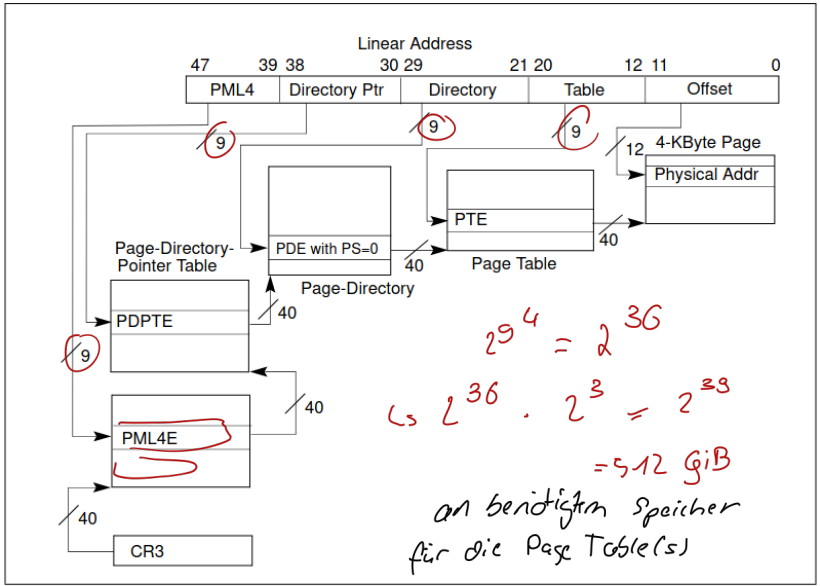


Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

Aufgabe 2

d) Wie groß sind die Seitentabellen jeweils, wenn für die Größe eines Eintrags vereinfachend 8 Byte angenommen werden?

e) Nehmen wir an, wir verwenden statt der zweistufigen Übersetzung eine einstufige, bei der die erste Stufe 20-bit breite Seitennummern verwendet. Wie viel Speicher kann adressiert werden? Kann mittels der zweistufigen Übersetzung mehr Speicher adressiert werden?

Aufgabe 2

d) Wie groß sind die Seitentabellen jeweils, wenn für die Größe eines Eintrags vereinfachend 8 (2^3) Byte angenommen werden?

Tabelle der Ersten Stufe: 2^9 Einträge $* 2^3 \frac{\text{Bytes}}{\text{Eintrag}} = 2^{12}$ Bytes = 4 KiB

Tabelle der Zweiten Stufe: 2^{11} Einträge $* 2^3 \frac{\text{Bytes}}{\text{Eintrag}} = 2^{14}$ Bytes = 16 KiB (pro Tabelle!)

Summe der Tabellen der Zweiten Stufe: $2^{14} \frac{\text{Bytes}}{\text{Tabelle}} * 2^9$ Tabellen = 2^{23} Bytes = 8 MiB

Insgesamt also: 2^{23} Bytes + 2^{13} Bytes = 4 KiB + 8 MiB

e) Nehmen wir eine einstufige Tabelle, die 20-bit breite Seitennummern verwendet. Kann mittels der zweistufigen (9+11) Übersetzung mehr Speicher adressiert werden?

Aufgabe 2

d) Wie groß sind die Seitentabellen jeweils, wenn für die Größe eines Eintrags vereinfachend 8 (2^3) Byte angenommen werden?

Tabelle der Ersten Stufe: 2^9 Einträge $* 2^3 \frac{\text{Bytes}}{\text{Eintrag}} = 2^{12}$ Bytes = 4 KiB

Tabelle der Zweiten Stufe: 2^{11} Einträge $* 2^3 \frac{\text{Bytes}}{\text{Eintrag}} = 2^{14}$ Bytes = 16 KiB (pro Tabelle!)

Summer der Tabellen der Zweiten Stufe: $2^{14} \frac{\text{Bytes}}{\text{Tabelle}} * 2^9$ Tabellen = 2^{23} Bytes = 8 MiB

Insgesamt also: 2^{23} Bytes $+ 2^{13}$ Bytes = 4 KiB + 8 MiB

e) Nehmen wir eine einstufige Tabelle, die 20-bit breite Seitennummern verwendet. Kann mittels der zweistufigen (9+11) Übersetzung mehr Speicher adressiert werden?

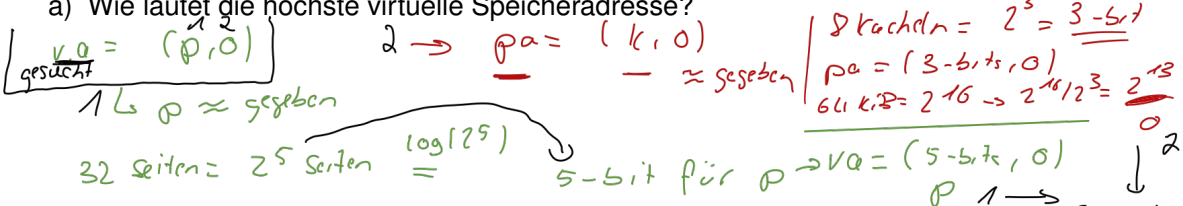
- Einstufig = $2^{20} * 2^{12} = 2^{32}$
- Zweistufig = $2^9 * 2^{11} * 2^{12} = 2^{32}$

Es kann gleich viel Speicher adressiert werden!

Aufgabe 3

Hauptspeichergröße: 64 KiB; 32 Seiten; 8 Kacheln.

a) Wie lautet die höchste virtuelle Speicheradresse?



b) Wie viele Bit sind für die virtuelle und physische Adresse jeweils breit?

Aufgabe 3

Hauptspeichergröße: 64 KiB; 32 Seiten; 8 Kacheln.

a) Wie lautet die höchste virtuelle Speicheradresse?

2^{16} Bytes / 2^3 Kacheln = 2^{13} Bytes pro Kachel → 2^{13} Bytes pro Seite

2^{13} Bytes pro Seite $*$ 2^5 Seiten = 2^{18} Bytes (im virtueller Speicher)

↪ Höchstes Offset ist immer Bytes-1, da man bei Byte 0x00 startet: $2^{18} - 1 = 262.144 - 1 = 262.143$

↳ Speicher „length": Analogie Array of Bytes

b) Wie viele Bit sind für die virtuelle und physische Adresse jeweils breit? Also höchster Index = length − 1 !

va → 18-bit

pa → 16-bit

Aufgabe 3

Hauptspeichergröße: 64 KiB; 32 Seiten; 8 Kacheln.

a) Wie lautet die höchste virtuelle Speicheradresse?

2^{16} Bytes / 2^3 Kacheln = 2^{13} Bytes pro Kachel $\rightarrow$ 2^{13} Bytes pro Seite

2^{13} Bytes pro Seite $*$ 2^5 Seiten = 2^{18} Bytes (im virtueller Speicher)

$\hookrightarrow$ Höchstes Offset ist immer Bytes-1, da man bei Byte 0x00 startet: $2^{18} - 1 = 262.144 - 1 = 262.143$

b) Wie viele Bit sind für die virtuelle und physische Adresse jeweils breit?

Physische Adresse: 64 KiB = 2^{16} adressierbare Bytes = 16 bit.

Alternativ: 2^3 Kacheln = 3 bits Kachel; 2^{13} Bytes pro Kachel/Seite = 13 bits Offset

$\hookrightarrow$ 16 bit Adresse

Virtuelle Adresse: 2^5 Seiten = 5 bits Seite; 2^{13} Bytes pro Kachel/Seite = 13 bits Offset

$\hookrightarrow$ 18 bit Adresse

Aufgabe 3

Hauptspeichergröße: 64 KiB; 32 Seiten (2^5); 8 Kacheln (2^3).

c) Ermitteln Sie die jeweils angesprochene physische Adresse. Benutzen Sie die Pagetable aus Abbildung 1.

	Virtuelle Adresse (P | O)	Seitennummer	Offset	Kachelnummer	Physische Adresse
0x00559 =	00 0000 | 0101 0101 1001	0	0x559	0	0x0 + 0x559 = 0x559
0x1208c =	01 0010 | 0000 1000 1100	9	0x08c	7	0x7 · 0x2000 + 0x08c = 0xe08c
0x16001 =	01 0110 | 0000 0000 0001	11 = B	0x1	4	(0x4 << 13) + 0x1 = 0x8001
0x0a777 =	00 1010 | 0111 0111 0111	5	0x777	PF	
0x13992 =	01 0011 | 1001 1001 0010	9	0x1992	7	(0x7 << 13) + 0x1992 = 0xf992

Aufgabe 3

Hauptspeichergröße: 64 KiB; 32 Seiten (2^5); 8 Kacheln (2^3).

d) Ermitteln Sie die jeweils angesprochene virtuelle Adresse. Benutzen Sie die Pagetable aus Abbildung 1.

	Physische Adresse (3 | 13)	Kachel nummer	Offset	Seiten nummer	Virtuelle Adresse
0x2000 =	001 | 0 0000 0000 0000	1	0x0	1	0x1 · 0x2000 = 0x2000
0x8235 =	100 | 0 0010 0011 0101	4	0x735	0x11	0x16735

Aufgabe 4
Segmentierung

pa = 0x base + 0x Addr

↳ pa = Seg + Addr = gs + 0x1000

a) Erweitern Sie das Bild, sodass ein Zugriff auf gs:0x1000 auf einen Zugriff auf 0x40000 übersetzt wird.

0x40000 − 0x1000 = 0x39000

Segmentregister

cs	0x08
ss	0x30
ds	0x28
es	0x10
fs	0x18
gs	0x20

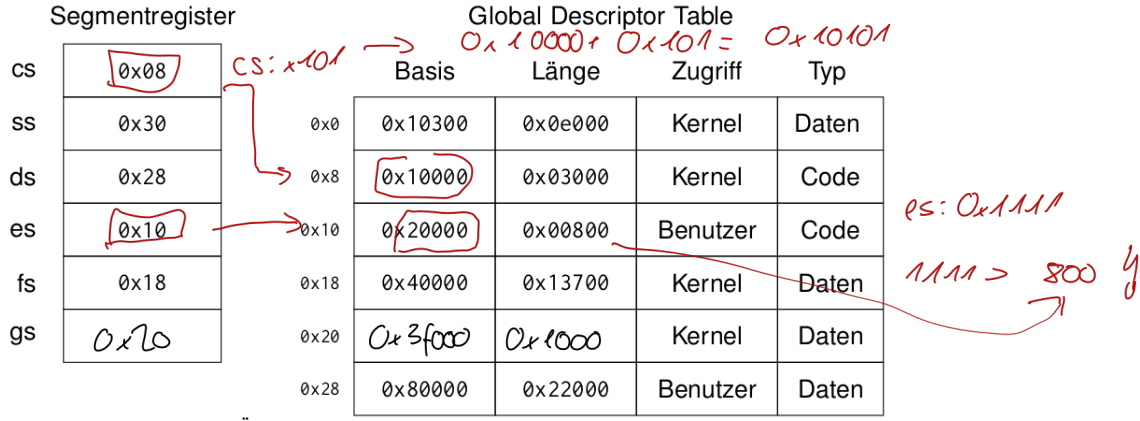
Global Descriptor Table

	Basis	Länge	Zugriff	Typ
0x0	0x10300	0x0e000	Kernel	Daten
0x8	0x10000	0x03000	Kernel	Code
0x10	0x20000	0x00800	Benutzer	Code
0x18	0x40000	0x13700	Kernel	Daten
0x20	0x39000	0x1001	Kernel	Daten
0x28	0x80000	0x22000	Benutzer	Daten

(gs: 0x1001 ↳)

Aufgabe 4
Segmentierung

a) Erweitern Sie das Bild oben, sodass ein Zugriff auf gs:0x1000 auf einen Zugriff auf 0x40000 übersetzt wird.



Aufgabe 4
Segmentierung

b) Lösen Sie die folgenden Speicherzugriffe auf.
Falls nicht anders angegeben erfolgen die Zugriffe lediglich mit Nutzerrechten.

Lesezugriff auf `ss:0`:

Segfault

Lesezugriff mit Kernelrechten auf `cs:0x101`:

0x10101

Schreibzugriff auf `es:0x1111`:

Segfault