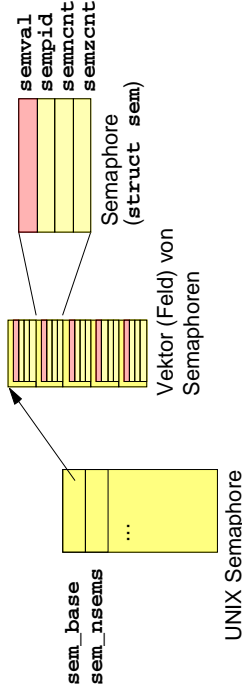


D.7 UNIX Semaphore

- Vektor von Semaphoren (erweitertes Vektoradditionssystem)



- ◆ Gleichzeitige und atomare Operationen auf mehreren Semaphoren im Vektor möglich

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

D-ProcId: 1997-12-08 13:49

D.100

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede für Zustimmung des Autors.

1 Erzeugen einer UNIX Semaphore

- UNIX Semaphore haben systemweit eindeutige Identifikation (Key)

- ◆ Erzeugen und Aufnehmen der Verbindung zu einer Semaphore

```
int semget( key_t key, int nsems, int semflg );
```

Identifikation
– neue für Erzeugung
– bestehende für Verbindungsaufnahme

Anzahl d. Semaphoren im Vektor

Zugriffsrechte;
Erzeugung oder
Verbindungsaufnahme

- ◆ Ergebnis ist eine Semaphore ID ähnlich wie ein Filedescriptor
 - Semaphore ID muß bei allen Operationen verwendet werden
- ◆ Zugriffsrechte: Lesen, Verändern
 - einstellbar für Besitzer, Gruppe und alle anderen (ähnlich wie bei Dateien)

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

D-ProcId: 1997-12-08 13:49

D.101

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede für Zustimmung des Autors.

1 Erzeugen einer UNIX Semaphore (2)

- Verwendung des Keys

- ◆ Alle Prozesse, die auf die Semaphore zugreifen wollen, müssen den Key kennen
- ◆ Keys sind eindeutig innerhalb eines (Betriebs-)Systems
- ◆ Ist ein Key bereits vergeben, kann keine Semaphore mit gleichem Key erzeugt werden
- ◆ Ist ein Key bekannt, kann auf die Semaphore zugegriffen werden
 - gesetzte Zugriffsberechtigungen werden allerdings beachtet

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

D-ProcId: 1997-12-08 13:49

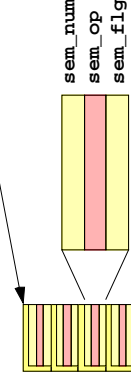
D.102

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede für Zustimmung des Autors.

2 Operationen auf UNIX Semaphoren

- Operationen auf mehreren der Semaphoren im Vektor

```
int semop( int semid, struct sembuf *sops, size_t nsops );
```



- ◆ Operationen
 - **sem_num**: Nummer der Semaphore im Vektor
 - **sem_op** < 0: ähnlich P-Operation – Herunterzählen der Semaphore (blockierend oder mit Fehlerstatus, je nach **sem_flg**)
 - **sem_op** > 0: ähnlich V-Operation – Hochzählen der Semaphore (blockierend oder mit Fehlerstatus, je nach **sem_flg**)
 - **sem_op** == 0: Test auf 0

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

D-ProcId: 1997-12-08 13:49

D.103

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede für Zustimmung des Autors.

2 Operationen auf UNIX Semaphoren (2)

- Kontrolloperationen

```
int semctl( int semid, int semnum, int cmd,  
           [ union semun arg ] );
```

 - ◆ explizites Setzen von Werten (einen, alle)
 - ◆ Abfragen von Werten (einen, alle)
 - ◆ Abfragen von Zusatzinformationen
 - welcher Prozeß hat letzte Operation erfolgreich durchgeführt
 - wann wurde letzte Operation durchgeführt
 - Zugriffsrechte
 - Anzahl der blockierten Prozesse

SPI

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-ProcDoc: 1987:12-08 13-49

D.104

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art der Zustimmung des Autors.

3 Beispiel: Philosophenproblem

- Eine UNIX Semaphore mit fünf Elementen (entsprechen Gabeln)
- ◆ Deklarationen

```
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/sem.h>  
  
int i;  
int j;  
int semid;  
struct sembuf pbuf[2], vbuf[2]; /* operation buffer */  
  
union semun {  
    int val;  
    struct semid_ds *buf;  
    ushort *array;  
} arg;  
  
...  
  
/* number of philosopher */  
/* semaphore ID */  
/* operation buffer */  
  
/* UNION for semctl */
```

SPI

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-ProcDoc: 1987:12-08 13-49

D.105

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art der Zustimmung des Autors.

3 Beispiel: Philosophenproblem (2)

- ◆ Erzeuge Semaphore

```
...  
semid= semget( IPC_PRIVATE, 5, IPC_CREAT | SEM_A | SEM_R );  
if( semid < 0 ) { ... /* error */ }  
  
for( j= 0; j < 5; j++ ) { /* set all values to 1 */  
    arg.val= 1;  
    if( semctl( semid, j, SETVAL, arg ) < 0 ) {  
        ... /* error */  
    }  
}  
  
...
```

SPI

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-ProcDoc: 1987:12-08 13-49

D.106

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art der Zustimmung des Autors.

3 Beispiel: Philosophenproblem (3)

- ◆ Erzeugen der Prozesse

```
...  
for( i=0; i<=3; i++ ) { /* start children i= 0...3; */  
    pid_t pid= fork();  
  
    if( pid < (pid_t)0 ) { ... /* error */ }  
    if( pid ==(pid_t)0 ) {  
        /* child */  
        break;  
    }  
}  
  
...  
/* parent: i= 4; */
```

SPI

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-ProcDoc: 1987:12-08 13-49

D.107

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art der Zustimmung des Autors.

3 Beispiel: Philosophenproblem (4)

◆ Initialisierungen

```
... /* we are philosopher i */
/* initialize buffer for P operation */
pbuf[0].sem_num= i; pbuf[1].sem_num= (i+1)%5;
pbu[0].sem_op= pbuf[1].sem_op= -1;
pbu[0].sem_flg= pbuf[1].sem_flg= 0;
/* initialize buffer for V operation */
vbuf[0].sem_num= i; vbuf[1].sem_num= (i+1)%5;
vbv[0].sem_op= vbuf[1].sem_op= 1;
vbv[0].sem_flg= vbuf[1].sem_flg= 0;
...
```

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-Proc.de: 1987-12-08 13:49

D.108

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jeder der Zustimmung des Autors.

3 Beispiel: Philosophenproblem (5)

◆ Philosoph

```
...
while( 1 ) {
... /* thinking */
if( semop( semid, pbuf, 2 ) < 0 ) { ... /* error */ }
... /* eating */
if( semop( semid, vbuf, 2 ) < 0 ) { ... /* error */ }
}
```

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-Proc.de: 1987-12-08 13:49

D.109

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jeder der Zustimmung des Autors.

D.8 Zusammenfassung

- Programmiermodell: Prozeß
- ◆ Zerlegung von Anwendungen in Prozesse oder Threads
- ◆ Ausnutzen von Wartezeiten; Time sharing-Betrieb
- ◆ Prozeß hat verschiedene Zustände: laufend, bereit, wartend etc.
- Auswahlstrategien für Prozesse
- ◆ FCFS, SJF, PSJF, RR, MLFB
- Prozeßkommunikation
- ◆ Pipes, Queues, Signals, Sockets, Shared memory, RPC
- Koordinierung von Prozessen
- ◆ Einschränkung der gleichzeitigen Abarbeitung von Befehlsfolgen in nebenläufigen Prozessen/Aktivitätsträgern

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

D-Proc.de: 1987-12-08 13:49

D.110

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jeder der Zustimmung des Autors.

D.8 Zusammenfassung (2)

- Gegenseitiger Ausschluss mit Spinlocks
- Klassische Koordinierungsprobleme und deren Lösung mit Semaphoren
- ◆ Gegenseitiger Ausschluss
- ◆ Bounded buffers
- ◆ Leser-Schreiber-Probleme
- ◆ Philosophenproblem
- ◆ Schlafende Friseur
- UNIX Systemaufrufe
- ◆ fork, exec, wait, nice
- ◆ pipe
- ◆ msgget, msgsnd, msgrcv
- ◆ signal, kill
- ◆ semget, semop, semctl

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

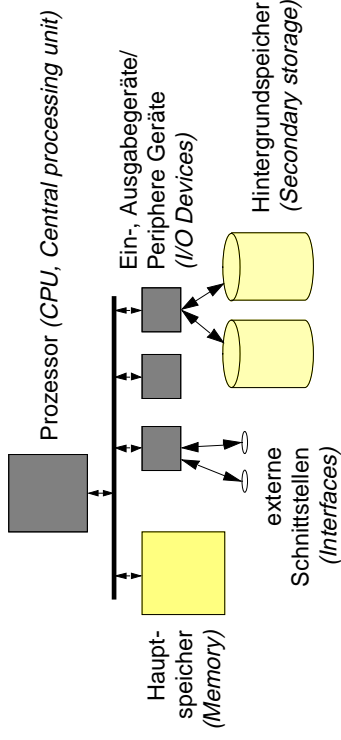
D-Proc.de: 1987-12-08 13:49

D.111

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jeder der Zustimmung des Autors.

E Speicherverwaltung

- Betriebsmittel



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

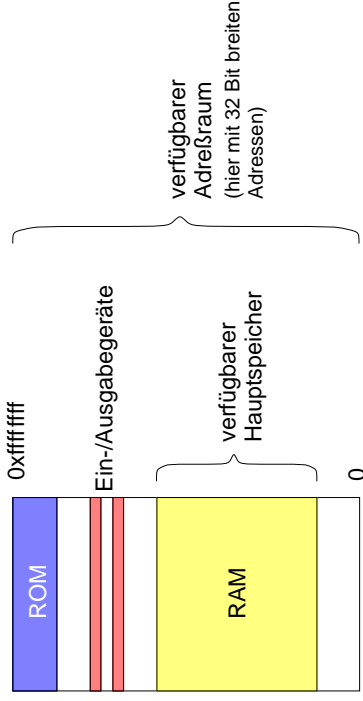
E.1

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art Zustimmung des Autors.

E.1 Speichervergabe

1 Problemstellung

- Verfügbarer Speicher



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.2

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art Zustimmung des Autors.

1 Problemstellung (2)

- Belegung des verfügbaren Hauptspeichers durch
 - ◆ Benutzerprogramme
 - Programmbefehle (Code, Binary)
 - Programmdateien
 - ◆ Betriebssystem
 - Betriebssystemcode
 - Puffer
 - Systemvariablen
- ★ Zuteilung des Speichers nötig

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.3

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art Zustimmung des Autors.

2 Statische Speicherzuteilung

- Feste Bereiche für Betriebssystem und Benutzerprogramm
- ▲ Probleme:
 - ◆ Begrenzung anderer Ressourcen (z.B. Bandbreite bei Ein-/Ausgabe wg. zu kleiner Systempuffer)
 - ◆ Ungenutzter Speicher des Betriebssystems kann von Anwendungsprogramm nicht genutzt werden und umgekehrt
- ★ Dynamische Speicherzuteilung einsetzen

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.4

Reproduktion jeder Art oder Verwendung ohne Erlaubnis, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist strafbar. Jede Art Zustimmung des Autors.

3 Dynamische Speicherzuteilung

- Segmente
 - ◆ zusammenhängender Speicherbereich (Bereich mit aufeinanderfolgenden Adressen)
- Allokation (Anforderung) und Freigabe von Segmenten
- Ein Anwendungsprogramm besitzt üblicherweise folgende Segmente:
 - ◆ Codesegment
 - ◆ Datenssegment
 - ◆ Stacksegment (für Verwaltungsinformationen, z.B. bei Funktionsaufrufen)
- ▲ Suche nach geeigneten Speicherbereichen zur Zuteilung
- ★ Speicherzuteilungsstrategien nötig

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

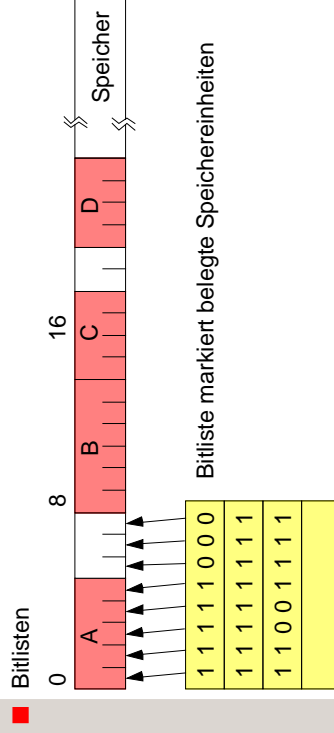
E-Memory.doc (1997-12-10 10:21)

Reproduktion jeder Art ohne Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne Zustimmung des Autors.

E.5

4 Freispeicherverwaltung

- Freie (evtl. auch belegte) Segmente des Speichers müssen repräsentiert werden



Speichereinheiten gleicher Größe (z.B. 1 Byte, 64 Byte, 1024 Byte)

SP I

Systemprogrammierung I

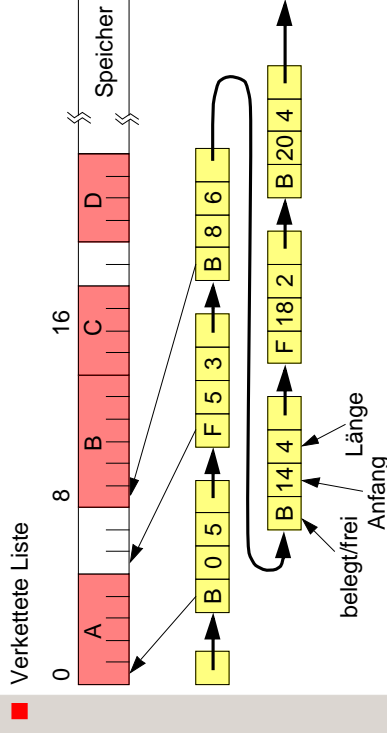
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc (1997-12-10 10:21)

Reproduktion jeder Art ohne Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne Zustimmung des Autors.

E.6

4 Freispeicherverwaltung (2)



Repräsentation auch von belegten Segmenten

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

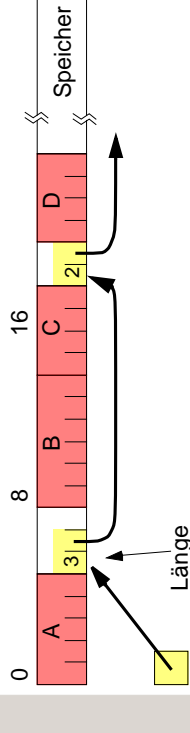
E-Memory.doc (1997-12-10 10:21)

Reproduktion jeder Art ohne Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne Zustimmung des Autors.

E.7

4 Freispeicherverwaltung (3)

- Verkettete Liste in dem freien Speicher



Mindestlückengröße muß garantiert werden

- Zur Effizienzsteigerung eventuell Rückwärtsverkettung nötig
- Repräsentation letztlich auch von der Vergabestrategie abhängig

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

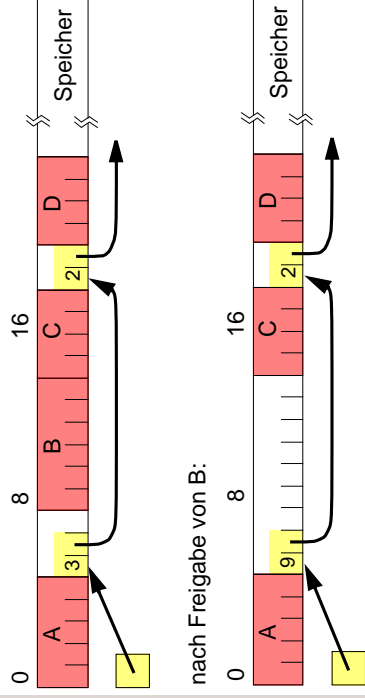
E-Memory.doc (1997-12-10 10:21)

Reproduktion jeder Art ohne Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne Zustimmung des Autors.

E.8

5 Speicherfreigabe

- Verschmelzung von Lücken



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.9

Reproduktion ist für den persönlichen Gebrauch gestattet. Die Weitergabe ist untersagt.

6 Vergabestrategien

- First Fit
 - ◆ erste passende Lücke wird verwendet
- Rotating First Fit / Next Fit
 - ◆ wie First Fit aber Start bei der zuletzt zugewiesenen Lücke
- Best Fit
 - ◆ kleinste passende Lücke wird gesucht
- Worst Fit
 - ◆ größte passende Lücke wird gesucht
- ▲ Probleme:
 - ◆ Speicherverschmitt
 - ◆ zu kleine Lücken

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

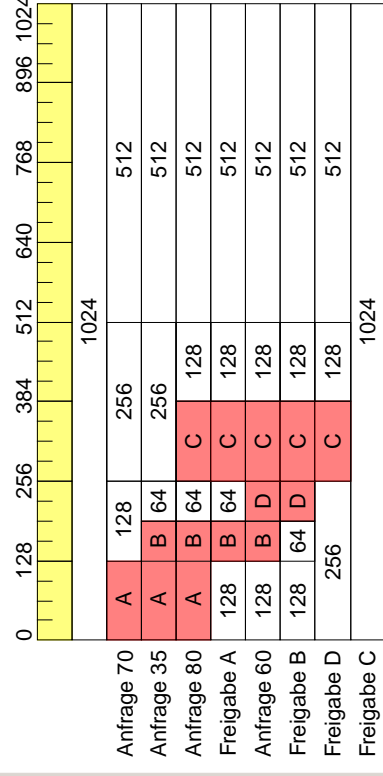
E-Memory.doc: 1987-12-10 10:21

E.10

Reproduktion ist für den persönlichen Gebrauch gestattet. Die Weitergabe ist untersagt.

7 Buddy Systeme

- Einteilung in dynamische Bereiche der Größe 2^n



Effiziente Repräsentation der Lücken und effiziente Algorithmen

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.11

Reproduktion ist für den persönlichen Gebrauch gestattet. Die Weitergabe ist untersagt.

8 Einsatz der Verfahren

- Einsatz im Betriebssystem
 - ◆ Verwaltung des Systemspeichers
 - ◆ Zuteilung von Speicher an Prozesse und Betriebssystem
- Einsatz innerhalb eines Prozesses
 - ◆ Verwaltung des Haldenspeichers (*Heap*)
 - ◆ erlaubt dynamische Allokation von Speicherbereichen durch den Prozess (`malloc` und `free`)
- Einsatz für Bereiche des Sekundärspeichers
 - ◆ Verwaltung bestimmter Abschnitte des Sekundärspeichers
z.B. Speicherbereich für Prozessauslagerungen (*Swap space*)

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

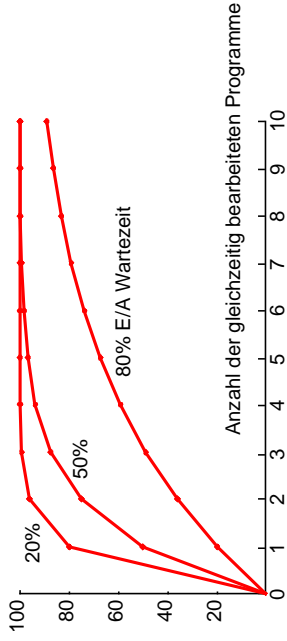
E.12

Reproduktion ist für den persönlichen Gebrauch gestattet. Die Weitergabe ist untersagt.

E.2 Mehrprogrammbetrieb

1 Problemstellung

- Mehrere Prozesse laufen gleichzeitig
 - ◆ Wartezeiten von Ein-/Ausgabeoperationen ausnutzen
 - ◆ CPU Auslastung verbessern
- CPU-Nutzung in Prozent, abhängig von der Anzahl der Prozesse



SP I

Systemprogrammierung I

© Franz J. Haack, Universität Erlangen-Nürnberg, IMMD IV, 1987

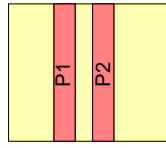
E-Memory.doc (1997-12-10 10:21)

E.13

Reproduktionen jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, sind der Zustimmung des Autors.

1 Problemstellung (2)

- ▲ Mehrere Prozesse benötigen Hauptspeicher
 - ◆ Prozesse an verschiedenen Stellen im Hauptspeicher liegen
 - ◆ Speicher reicht eventuell nicht für alle Prozesse
 - ◆ Schutzbedürfnis des Betriebssystems und der Prozesse untereinander



zwei Prozesse und deren Codesegmente im Speicher

- ★ Relokation von Programmbefehlen (Binaries)
- ★ Ein- und Auslagern von Prozessen
- ★ Hardwareunterstützung

SP I

Systemprogrammierung I

© Franz J. Haack, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc (1997-12-10 10:21)

E.14

Reproduktionen jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, sind der Zustimmung des Autors.

2 Relokation

- Festlegung absoluter Adressen in den Programmbefehlen
 - ◆ z.B. ein Sprungbefehl in ein Unterprogramm oder ein Ladebefehl für eine Variable aus dem Datenssegment
- Absolutes Binden (*Compile time*)
 - ◆ Adressen stehen fest
 - ◆ Programm kann nur an bestimmter Speicherstelle korrekt ablaufen
- Statisches Binden (*Load time*)
 - ◆ Beim Laden (Starten) des Programms werden die absoluten Adressen angepasst (reloziert)
 - ◆ Relokationsinformation nötig, die vom Compiler oder Assembler geliefert wird

SP I

Systemprogrammierung I

© Franz J. Haack, Universität Erlangen-Nürnberg, IMMD IV, 1987

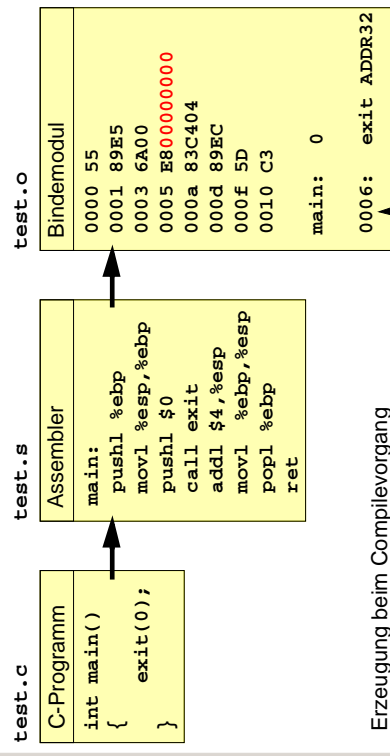
E-Memory.doc (1997-12-10 10:21)

E.15

Reproduktionen jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, sind der Zustimmung des Autors.

2 Relokation (2)

- Compilervorgang (Erzeugung der Relokationsinformation)



Erzeugung beim Compilervorgang

ersetze Adresse von exit

SP I

Systemprogrammierung I

© Franz J. Haack, Universität Erlangen-Nürnberg, IMMD IV, 1987

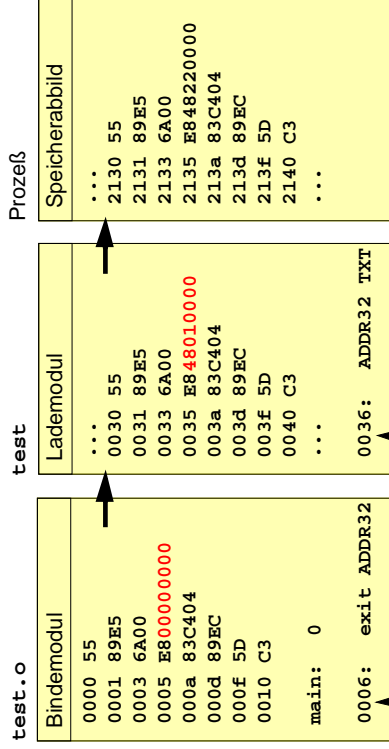
E-Memory.doc (1997-12-10 10:21)

E.16

Reproduktionen jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, sind der Zustimmung des Autors.

2 Relokation (3)

- Binde- und Ladevorgang



ersetze Adresse von exit

ersetze Adresse relativ zum Befehlssegment

Adresse ist nun absolut
Befehlssegment: 2100

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne schriftliche Genehmigung des Autors.

E.17

2 Relokation (4)

- Relokationsinformation im Bindemodul
 - ◆ erlaubt das Binden von Modulen in beliebige Programme
- Relokationsinformation im Lademodul
 - ◆ erlaubt das Laden des Programms an beliebige Speicherstellen
 - ◆ absolute Adressen werden erst beim Laden generiert
- ▲ Alternative
 - ◆ Programm benutzt keine absoluten Adressen und kann daher immer an beliebige Speicherstellen geladen werden

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

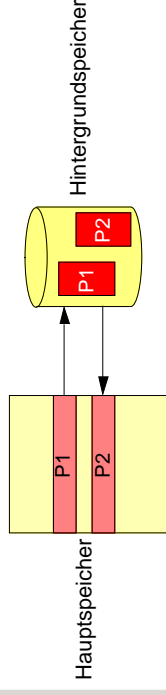
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne schriftliche Genehmigung des Autors.

E.18

3 Ein-, Auslagerung (Swapping)

- Segmente eines Prozesses werden auf Hintergrundspeicher ausgelagert und im Hauptspeicher freigegeben
 - ◆ z.B. zur Überbrückung von Wartezeiten bei E/A oder Round-Robin Schedulingstrategie

- Einlagern der Segmente in den Hauptspeicher am Ende der Wartezeit



- ▲ Aus-, Einlagerzeit ist hoch

- ◆ Latenzzeit der Festplatte
- ◆ Übertragungszeit

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

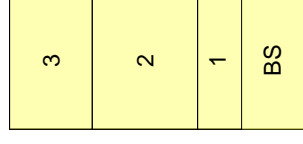
E-Memory.doc: 1987-12-10 10:21

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne schriftliche Genehmigung des Autors.

E.19

3 Ein-, Auslagerung (2)

- ▲ Prozeß ist statisch gebunden
 - ◆ kann nur an gleiche Stelle im Hauptspeicher wieder eingelagert werden
 - ◆ Kollisionen mit eventuell neu im Hauptspeicher befindlichen Segmenten
- Mögliche Lösung: Partitionierung des Hauptspeichers



- ◆ In jeder Partition läuft nur ein Prozeß
- ◆ Einlagerung erfolgt wieder in die gleiche Partition
- ◆ Speicher kann nicht optimal genutzt werden

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

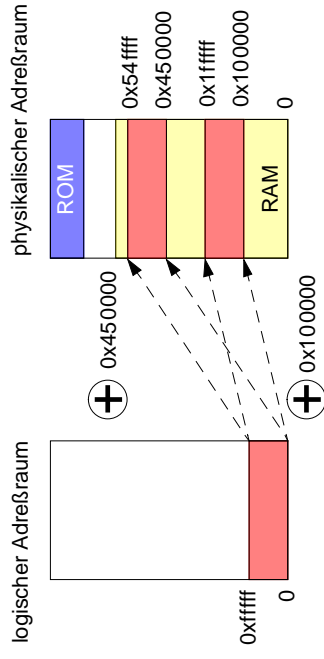
E-Memory.doc: 1987-12-10 10:21

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer in Lehrzwecken an der Universität Erlangen-Nürnberg, ist ohne schriftliche Genehmigung des Autors.

E.20

4 Segmentierung

- Hardwareunterstützung: Umsetzung logischer in physikalische Adressen
- Prozesse erhalten einen logischen Adressraum



Das Segment im logischen Adressraum kann an jeder beliebige Stelle im physikalischen Adressraum liegen.

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

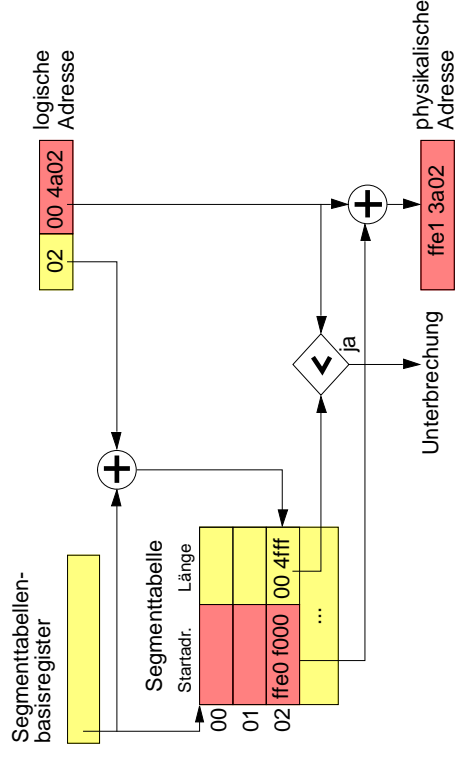
E-Memory.doc: 1987-12-10 10:21

E.21

Reproduzieren Sie die Abbildung ohne Änderung, oder im Lehrwerk in der Universität Erlangen-Nürnberg, Institut für Zusammenfassung des Autors.

4 Segmentierung (2)

- Realisierung mit Übersetzungstabelle



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.22

Reproduzieren Sie die Abbildung ohne Änderung, oder im Lehrwerk in der Universität Erlangen-Nürnberg, Institut für Zusammenfassung des Autors.

4 Segmentierung (3)

- Hardware wird MMU (*Memory management unit*) genannt
- Schutz vor Segmentübertretung
 - Unterbrechung zeigt Speicherverletzung an
 - Programme und Betriebssystem voneinander geschützt
- Prozessumschaltung durch Austausch der Segmentbasis
 - jeder Prozess hat eigene Übersetzungstabelle
- Ein- und Auslagerung vereinfacht
 - nach Einlagerung an beliebige Stelle muß lediglich die Übersetzungstabelle angepaßt werden
- Gemeinsame Segmente möglich
 - Befehlssegmente
 - Datensegmente (*Shared memory*)

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.23

Reproduzieren Sie die Abbildung ohne Änderung, oder im Lehrwerk in der Universität Erlangen-Nürnberg, Institut für Zusammenfassung des Autors.

4 Segmentierung (4)

- Zugriffsschutz einfach integrierbar
 - z.B. Rechte zum Lesen, Schreiben und Ausführen von Befehlen, die von der MMU geprüft werden
- Fragmentierung des Speichers durch häufiges Ein- und Auslagern
 - es entstehen kleine, nicht nutzbare Lücken
- Kompaktifizieren
 - Segmente werden verschoben, um Lücken zu schließen; Segmenttabelle wird jeweils angepaßt
- lange E/A Zeiten für Ein- und Auslagerung
 - nicht alle Teile eines Segments werden gleich häufig genutzt

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

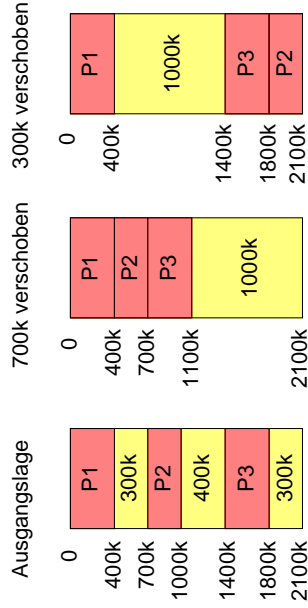
E-Memory.doc: 1987-12-10 10:21

E.24

Reproduzieren Sie die Abbildung ohne Änderung, oder im Lehrwerk in der Universität Erlangen-Nürnberg, Institut für Zusammenfassung des Autors.

5 Kompaktifizieren

- Verschieben von Segmenten
- ◆ Erzeugen von weniger aber größeren Lücken
- ◆ Verringern des Verschnitts
- ◆ aufwendige Operation, abhängig von der Größe der verschobenen Segmente



SP I

Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

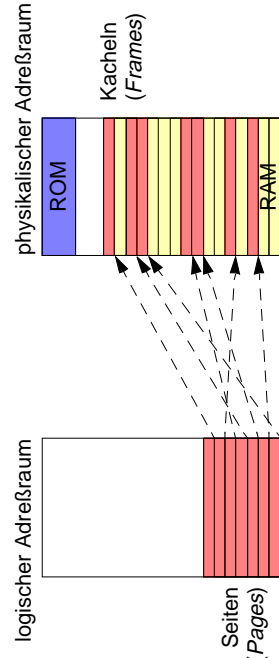
E-Memory.doc: 1997-12-10 10:21

E.25

Reproduzieren Sie für alle Verwendungszwecke, unter der Lizenzierung der Universität Erlangen-Nürnberg, vorbehaltlich der Zustimmung des Autors.

E.3 Seitenadressierung (Paging)

- Einteilung des logischen Adreßraums in gleichgroße Seiten, die an beliebigen Stellen im physikalischen Adreßraum liegen können
- ◆ Lösung des Fragmentierungsproblem
- ◆ keine Kompaktifizierung mehr nötig
- ◆ Vereinfacht Speicherbelegung



SP I

Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

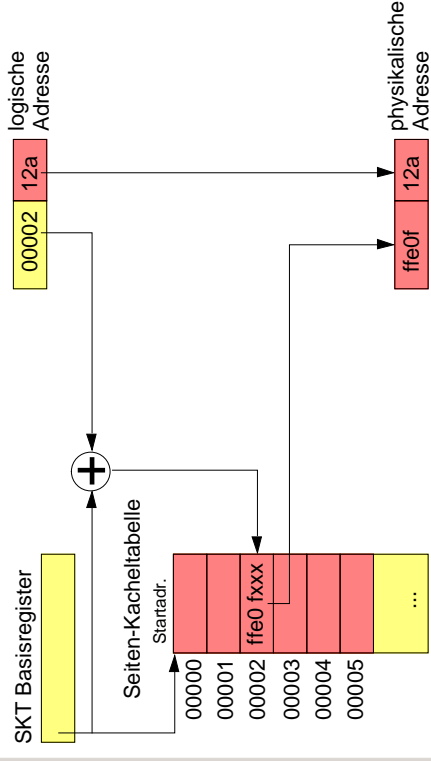
E-Memory.doc: 1997-12-10 10:21

E.26

Reproduzieren Sie für alle Verwendungszwecke, unter der Lizenzierung der Universität Erlangen-Nürnberg, vorbehaltlich der Zustimmung des Autors.

1 MMU mit Seiten-Kachelntabelle

- Tabelle setzt Seiten in Kacheln um



SP I

Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

E-Memory.doc: 1997-12-10 10:21

E.27

Reproduzieren Sie für alle Verwendungszwecke, unter der Lizenzierung der Universität Erlangen-Nürnberg, vorbehaltlich der Zustimmung des Autors.

1 MMU mit Seiten-Kachelntabelle (2)

- ▲ Seitenadressierung erzeugt internen Verschnitt
- ◆ letzte Seite eventuell nicht vollständig genutzt
- Seitengröße
- ◆ kleine Seiten verringern internen Verschnitt, vergrößern aber die Seiten-Kachelntabelle (und umgekehrt)
- ◆ übliche Größen: 512 Bytes — 8192 Bytes
- ▲ große Tabelle, die im Speicher gehalten werden muß
- ▲ viele implizite Speicherzugriffe nötig
- ▲ nur ein „Segment“ pro Kontext
- ★ Kombination mit Segmentierung

SP I

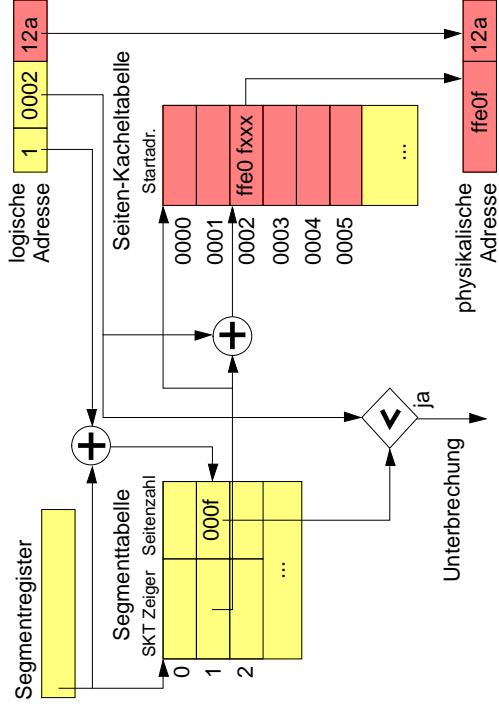
Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1997

E-Memory.doc: 1997-12-10 10:21

E.28

Reproduzieren Sie für alle Verwendungszwecke, unter der Lizenzierung der Universität Erlangen-Nürnberg, vorbehaltlich der Zustimmung des Autors.

2 Segmentierung und Seitenadressierung



SP I

Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

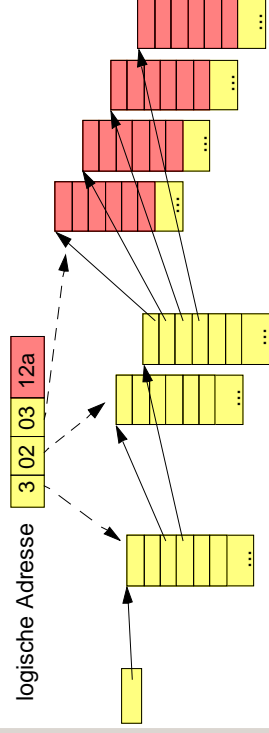
E.29

E-Memory.doc: 1987-12-10 10:21
Reproduktion jeder Art ohne Verwendung dieser Unterlagen ist ein Verstoß gegen die Universität Erlangen-Nürnberg, Institut für Zentrums des Autors.

2 Segmentierung und Seitenadressierung (2)

- ▲ noch mehr implizite Speicherzugriffe
- ▲ große Tabellen im Speicher
- ★ Mehrstufige Seitenadressierung mit Ein- und Auslagerung

3 Mehrstufige Seitenadressierung



SP I

Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E.30

E-Memory.doc: 1987-12-10 10:21
Reproduktion jeder Art ohne Verwendung dieser Unterlagen ist ein Verstoß gegen die Universität Erlangen-Nürnberg, Institut für Zentrums des Autors.

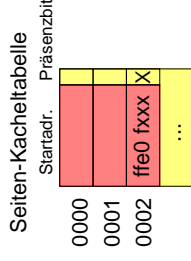
3 Mehrstufige Seitenadressierung (2)

- Ein- und Auslagerung von Seiten

◆ Ist das Präsenzbit gesetzt, bleibt alles wie bisher.

◆ Ist das Präsenzbit gelöscht, wird eine Unterbrechung ausgelöst (*Page fault*).

◆ Die Unterbrechungsbehandlung kann nun für das Laden der Seite vom Hauptspeicher sorgen und den Speicherzugriff danach wiederholen (benötigt HW Support in der CPU).



- Präsenzbit auch für jeden Eintrag in den höheren Stufen

◆ Tabellen sind aus- und einlagerbar

▲ Noch mehr implizite Speicherzugriffe

SP I

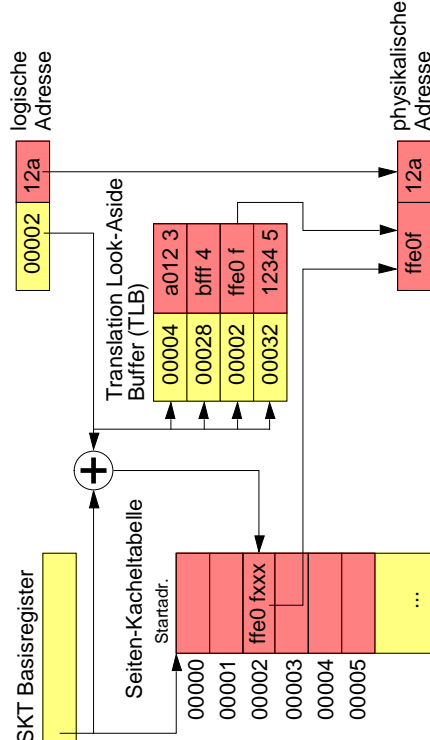
Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E.31

E-Memory.doc: 1987-12-10 10:21
Reproduktion jeder Art ohne Verwendung dieser Unterlagen ist ein Verstoß gegen die Universität Erlangen-Nürnberg, Institut für Zentrums des Autors.

4 Translation Look-Aside Buffer

- Schneller Registersatz wird konsultiert bevor auf die SKT zugegriffen wird



SP I

Systemprogrammierung I
© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E.32

E-Memory.doc: 1987-12-10 10:21
Reproduktion jeder Art ohne Verwendung dieser Unterlagen ist ein Verstoß gegen die Universität Erlangen-Nürnberg, Institut für Zentrums des Autors.

4 Translation Look-Aside Buffer (2)

- Schneller Zugriff auf Seitenabbildung, falls Information im voll-assoziativen Speicher des TLB
- ◆ keine impliziten Speicherzugriffe nötig
- Bei Kontextwechseln muß TLB gelöscht werden (*Flush*)
- Bei Zugriffen auf eine nicht im TLB enthaltene Seite wird die entsprechende Zugriffsinformation in den TLB eingetragen
- ◆ Ein alter Eintrag muß zur Ersetzung ausgesucht werden
- TLB Größe
 - ◆ Pentium: Daten TLB = 64, Code TLB = 32, Seitengröße 4K
 - ◆ Sparc V9: Daten TLB = 64, Code TLB = 64, Seitengröße 8K
 - ◆ Größere TLBs bei den üblichen Taktraten zur Zeit nicht möglich

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

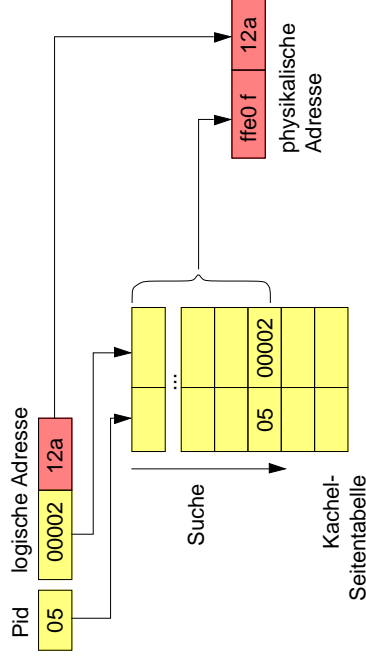
E-Memory.doc: 1997-12-10 10:21

E.33

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrwerken an der Universität Erlangen-Nürnberg, befreit der Zustimmung des Autors.

5 Invertierte Seiten-Kacheltabelle

- Zum Umsetzen der Adressen nur Abbildung der belegten Kacheln nötig
- ◆ eine Tabelle, die zu jeder Kachel die Seitenabbildung hält



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1997-12-10 10:21

E.34

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrwerken an der Universität Erlangen-Nürnberg, befreit der Zustimmung des Autors.

5 Invertierte Seiten-Kacheltabelle (2)

- Vorteile
 - ◆ wenig Platz zur Speicherung der Abbildung notwendig
 - ◆ Tabelle kann immer im Hauptspeicher gehalten werden
- ▲ Nachteile
 - ◆ prozesslokale SKT zusätzlich nötig für Seiten, die ausgelagert sind
 - diese können aber ausgelagert werden
 - ◆ Suche in der KST ist aufwendig
 - Einsatz von Assoziativspeichern und Hashfunktionen

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1997-12-10 10:21

E.35

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrwerken an der Universität Erlangen-Nürnberg, befreit der Zustimmung des Autors.

E.4 Fallstudie: Pentium

- Physikalische Adresse
 - ◆ 32 bit breit
- Segmente
 - ◆ CS – Codesegment: enthält Instruktionen
 - ◆ DS – Datensegment
 - ◆ SS – Stacksegment
 - ◆ ES, FS, GS – zusätzliche Segmente
- Befehle beziehen sich auf eines oder mehrere der Segmente
- Segmentadressierung
 - ◆ Segmentelektor zur Auswahl eines Segments: 16 bit bezeichnen das Segment

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

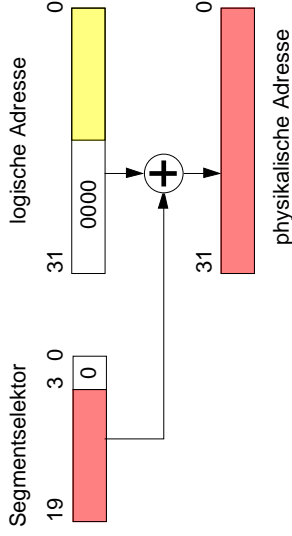
E-Memory.doc: 1997-12-10 10:21

E.36

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrwerken an der Universität Erlangen-Nürnberg, befreit der Zustimmung des Autors.

1 Real Mode Adressierung

- Adreßgenerierung im Real Mode
- ◆ 16 bit breiter Segmentselektor wird als 20 bit breite Adresse interpretiert und auf die logische Adresse addiert



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.37

Reproduzieren Sie für alle Verwendungszwecke. Änderungen sind in der Verantwortung der Universität Erlangen-Nürnberg. Jede Art der Zustimmung des Autors.

2 Protected Mode Adressierung (2)

- Deskriptortabelle
- ◆ enthält bis zu 8192 Segmentdeskriptoren
 - physikalische Basisadresse
 - Längenangabe
 - Granularität (Angaben für Bytes oder Seiten)
 - Präsenzbit
 - Privilegiierungsstufe
- ◆ globale Deskriptortabelle für alle Prozesse zugänglich (Register GDTR)
- ◆ lokale Deskriptortabelle pro Prozess möglich (Register LDTR gehört zum Prozesskontext)

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.39

Reproduzieren Sie für alle Verwendungszwecke. Änderungen sind in der Verantwortung der Universität Erlangen-Nürnberg. Jede Art der Zustimmung des Autors.

2 Protected Mode Adressierung

- Vier Betriebsmodi (Stufen von Privilegien)
 - ◆ Stufe 0: höchste Privilegien (privilegierte Befehle, etc.): BS Kern
 - ◆ Stufe 1: BS Treiber
 - ◆ Stufe 2: BS Erweiterungen
 - ◆ Stufe 3: Benutzerprogramme
- ◆ Speicherverwaltung kann nur in Stufe 0 konfiguriert werden
- Segmentselektoren enthalten Privilegiierungsstufe
- ◆ Stufe des Codesegments entscheidet über Zugriffserlaubnis
- Segmentselektoren werden als Indizes interpretiert
- ◆ Tabellen von Segmentdeskriptoren
 - Globale Deskriptor Tabelle
 - Lokale Deskriptor Tabelle

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

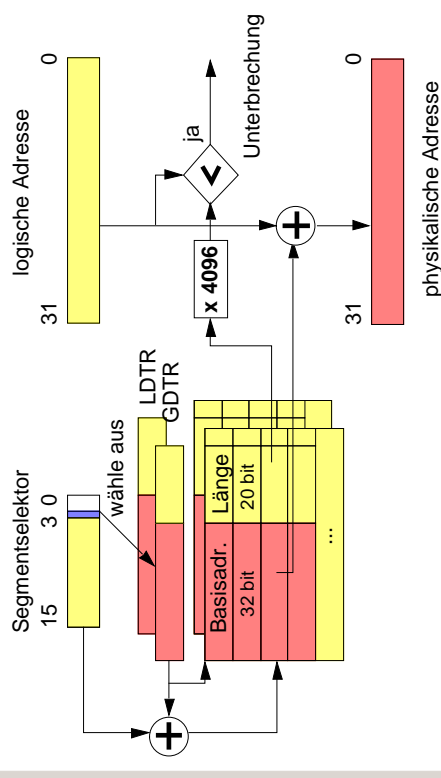
E-Memory.doc: 1987-12-10 10:21

E.38

Reproduzieren Sie für alle Verwendungszwecke. Änderungen sind in der Verantwortung der Universität Erlangen-Nürnberg. Jede Art der Zustimmung des Autors.

3 Adreßberechnung bei Segmentierung

- Verwendung der Protected mode Adressierung



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

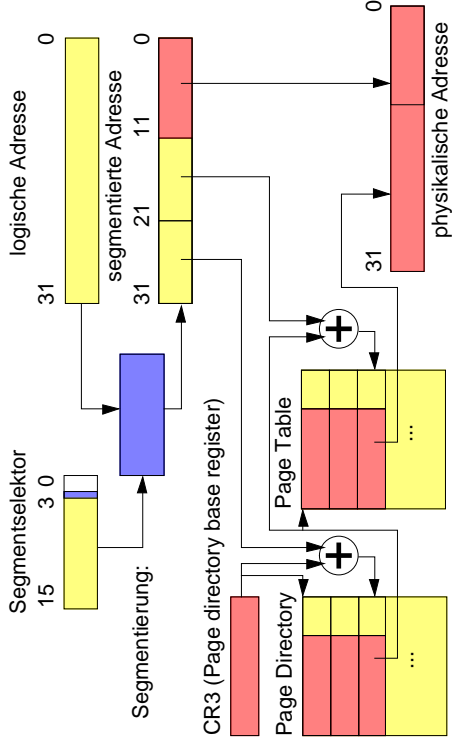
E-Memory.doc: 1987-12-10 10:21

E.40

Reproduzieren Sie für alle Verwendungszwecke. Änderungen sind in der Verantwortung der Universität Erlangen-Nürnberg. Jede Art der Zustimmung des Autors.

4 Adressberechnung bei Paging

- Seitenadressierung wird der Segmentierung nachgeschaltet



SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.41

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrveranstaltungen der Universität Erlangen-Nürnberg, bestraft die Zustimmung des Autors.

4 Adressberechnung bei Paging

- Zweistufige Seitenadressierung
 - Directory — Page table
 - Seitengröße fest auf 4096 Bytes
- Inhalt des Seitendeskriptor
 - Kacheladresse
 - Dirty Bit: Seite wurde beschrieben
 - Access Bit: Seite wurde gelesen oder geschrieben
 - Schreibschutz: Seite nur lesbar
 - Präsenzbit: Seite ausgelagert (31 Bits für BS-Informationen nutzbar)
 - Kontrolle des Processorcaches
- Getrennte TLBs für Codesegment und Datensegmente
 - 64 Einträge für Datenseiten; 32 Einträge für Codeseiten

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.42

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrveranstaltungen der Universität Erlangen-Nürnberg, bestraft die Zustimmung des Autors.

E.5 Gemeinsamer Speicher (Shared Memory)

- Speicher, der mehreren Prozessen zur Verfügung steht
 - gemeinsame Segmente (gleiche Einträge in verschiedenen Segmenttabellen)
 - gemeinsame Seiten (gleiche Einträge in verschiedenen SKTs)
 - gemeinsame Seitenbereiche (gemeinsames Nutzen einer SKT bei mehrstufigen Tabellen)
- Gemeinsamer Speicher wird beispielsweise benutzt für
 - Kommunikation zwischen Prozessen
 - gemeinsame Befehlssegmente

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.43

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrveranstaltungen der Universität Erlangen-Nürnberg, bestraft die Zustimmung des Autors.

E.5 Gemeinsamer Speicher (2)

- Systemaufrufe unter Solaris 2.5
 - Erzeugen bzw. Holen eines gemeinsamen Speichersegments


```
int shmget( key_t key, int size, int shmflg );
```
 - Einblenden und Ausblenden des Segments in den Speicher


```
void *shmat( int shmid, void *shmaddr, int shmflg );
```

```
int shmctl( void *shmaddr );
```
 - Kontrolloperation


```
int shmctl( int shmid, int cmd, struct shmids *buf );
```

SP I

Systemprogrammierung I

© Franz J. Hauck, Universität Erlangen-Nürnberg, IMMD IV, 1987

E-Memory.doc: 1987-12-10 10:21

E.44

Reproduktion ist ohne Verwendung dieser Unterlagen, außer in Lehrveranstaltungen der Universität Erlangen-Nürnberg, bestraft die Zustimmung des Autors.