Proseminar
**Konzepte von Betriebssystem-Komponenten (KVBK)**

# Vortrag zum Thema: Speicheraddressierung, Segmentierung, Paging

**Von Christian Hubert**

## 1.: Speicherung und Adressierung von Daten

Bei der Speicheradressierung muss man sich zwei Dinge fragen:

### 1.1.: Wie sind die Daten auf dem Speicher angeordnet???

#### 1.1.1.: Lineare Speicherung

Die Daten werden linear auf dem Speicher abgelegt. Bei Löschen oder Änderung der Größe wird daher evtl. eine Neusortierung des gesamten Speichers nötig, da die neue Datei nicht mehr an ihren bisherigen Platz passt oder weil man „Löcher“ im Speicher berhindern will.

#### 1.1.2.: Segmentierte Speicherung

Die Daten werden in Segmenten verschiedener Größe auf dem Speicher abgelegt. Dies kann zu enormem Verwaltungsaufwand führen, wenn man eine zu kleine Segmentgröße wählt. Die Reihenfolge der Segmente ist nicht zwingend vorgegeben, d.h. bei Änderungen muss evtl. nur dem entsprechenden Segment ein neuer Speicherplatz zugeordnet werden. Verschwendung von Speicherplatz tritt nur bei ineffizienter Sortierung der Segmente auf.

#### 1.1.3.: Gekachelte Speicherung

Der Speicher wird in festgesetzte gleiche Teile partitioniert (Kacheln). Die Daten werden als Seiten mit identischer Größe (4 KB) in den Kacheln auf dem Speicher abgelegt. Die Reihenfolge der Kacheln ist linear, allerdings erfordert eine Änderung der Daten keine Umsortierung des Speichers, da die Kacheln ja immer identische Größe besitzen. Lediglich die Verwaltungstabellen müssen aktualisiert werden. Eine Verschwendung von Speicherplatz und der Verwaltungsaufwand sind abhängig von der Größe der Kacheln
Wie aber kann ich nun bei den jeweiligen Methoden effizient auf die Daten zugreifen? Bei der linearen Speicherung geschieht das durch Lineare Adressierung, bei segmentierter Speicherung durch Segmentierung und bei gekachelter Speicherung durch Seitenadressierung (Paging).

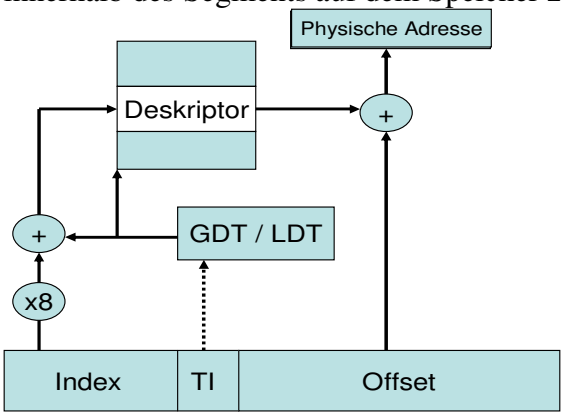
### 1.2.: Wie kann ich effizient auf Die Daten zugreifen??

#### 1.2.1.: Lineare Adressierung

Bei der Linearen Adressierung wird für jede Datei eine 32-bit Integer genutzt, die die Adresse der ersten Speicherzelle enthält (bis zu 4 GB Speicher ansprechbar). Diese wird üblicherweise in hexadezimaler Schreibweise dargestellt. Die Verwaltung der Adressen in Tabellen bringt einen enormen Verwaltungsaufwand mit sich (durch die ständigen Umsortierungen) und kann sehr unübersichtlich werden (bei zu vielen kleinen Dateien).

#### 1.2.2.: Segmentierung

Bei der Segmentierung werden die Daten über logische Adressen (Segmentselektoren) verwaltet. Diese werden in speziellen Registern verwaltet und enthalten Adresse des Segmentdeskriptors und die Lage der Daten innerhalb des Segments (Offset). Die Identifikation und Adressierung des jeweiligen Segments geschieht mittels des Segmentdeskriptors, die wiederum werden in speziellen Tabellen verwaltet und dienen zusätzlich auch der Kontrolle der Zugriffsrechte.
Der Speicherzugriff wird von Segmentierungseinheit durchgeführt. Das TI-Feld des Segmentselektors gibt an in welcher Tabelle der Deskriptor liegt.Mit der Basisadresse von der GDT bzw. LDT und dem Index vom Segmentselektor wird die Lage vom Deskriptor innerhalb der Tabelle ermittelt. Nun wird die Adresse aus dem BASE-Feld des Deskriptors mit dem Offset des Selektors addiert um die physische Adresse der Daten innerhalb des Segments auf dem Speicher zu ermitteln.



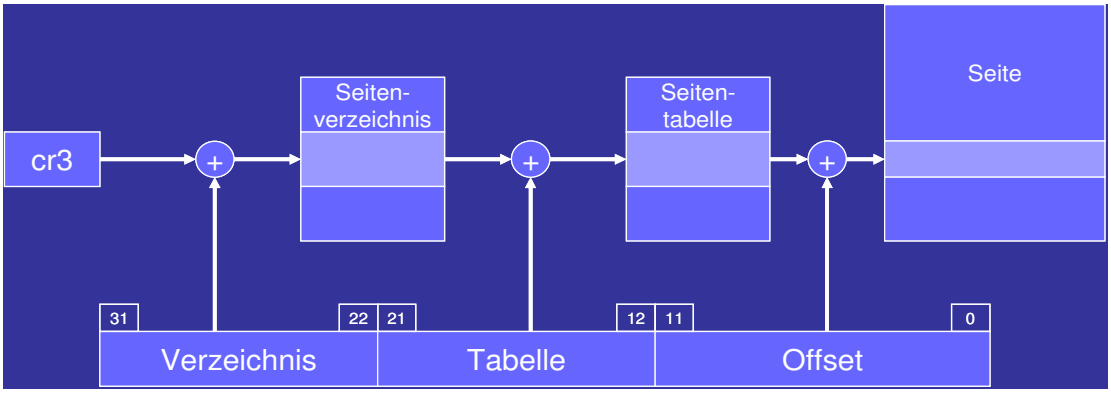

### 1.2.3.: Seitenadressierung

Bei der Seitenadressierung (Paging) werden die Daten auch über logische Adressen verwaltet. Der Zugriff auf die Seitenadressierungsstrukturen erfolgt über Register. Die jeweiligen Einträge in Verzeichnissen und Tabellen sind identisch aufgebaut.
Es gibt drei Arten von Seitenadressierung:

- Reguläre Seitenadressierung
- Erweiterte Seitenadressierung
- Drei-Stufen-Seitenadressierung
-

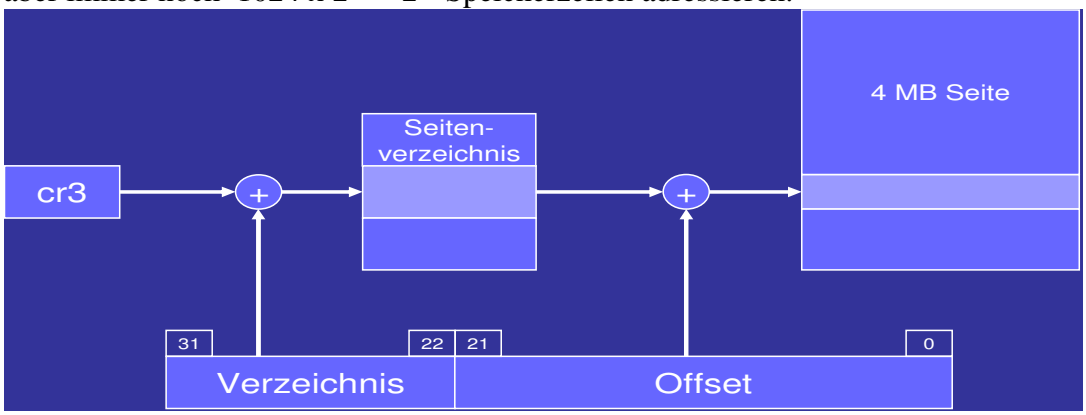
#### *1.2.3.1.: Reguläre Seitenadressierung*

Bei der Regulären Seitenadressierung sind die Seiten bzw. Kacheln 4 KB groß. Die Logische Adresse ist in 3 Felder unterteilt, eins für die Adresse des Verzeichnis (10-Bit, verwaltet die Seitentabellen), eins für die Adresse der Tabelle (10-Bit, verwaltet die Seiten), und eins für das Offset (12-Bit, adressiert die Daten innerhalb der Seite). Ein Seitenverzeichnis kann somit 1024 x 1024 x 4096 = $2^{32}$ Speicherzellen adressieren.



### *1.2.3.2.: Erweiterte Seitenadressierung*

Bei der Erweiterten Seitenadressierung sind die Seiten bzw. Kacheln 4 MB groß, damit man größere, linear zusammenhängende Datenfelder mit weniger Verwaltungsaufwand auch zusammenhängend auf Speicher legen kann. Die logische Adresse ist in diesem Fall nur in 2 Felder unterteilt. Das Verzeichnisfeld (10-Bit, verwaltet die Seiten(!)) und das Offsetfeld (22-Bit, adressiert die Daten innerhalb der Seite). Ein Seitenverzeichnis kann aber immer noch 1024 x $2^{22} = 2^{32}$ Speicherzellen adressieren.




## 1.3.: Hardwareerweiterungen

Probleme, die durch die schnelle CPU und dem verhältnismäßig langsameren Arbeitsspeicher entstehen, werden durch Hardwareerweiterungen weitestgehend gelöst.

### 1.3.1.: Hardware Cache

Eine Möglichkeit wäre der Hardware Cache. Das ist ein kleiner, schneller Speicher, der zwischen die CPU-Register und den Arbeitspeicherzellen geschaltet wird. In dem Cache werden häufig genutzte Daten zeilenorientiert verwaltet. Zwar muss der Cache durch ein Register in der Seitenadressierungseinheit global aktiviert werden, allerdings ist trotzdem eine individuelle Nutzung des Cache für jede Seite durch bestimmte Deskriptorfunktionen (PCD/PWT-Flags) der Seitenverzeichnisse bzw. -tabellen möglich.

### 1.3.2. Translation-Lookaside-Buffer

Eine Alternative wäre der Translation-Lookaside-Buffer (TLB), der im Prinzip wie der Cache funktioniert, allerdings keine Daten sondern physische Adressen speichert. Dadurch werden keine wiederholten Berechnungen ein und derselben physischen Adresse nötig. Evtl. muss die automatische Aktualisierung des TLB durch die Seitenadressierungseinheit über das System unterbunden werden (z.B. bei Wechsel von Prozessen die auf die gleichen Tabellen zugreifen).

# 2. Speicherung und Adressierung von Daten in Linux

## 2.1. Segmentierung in Linux

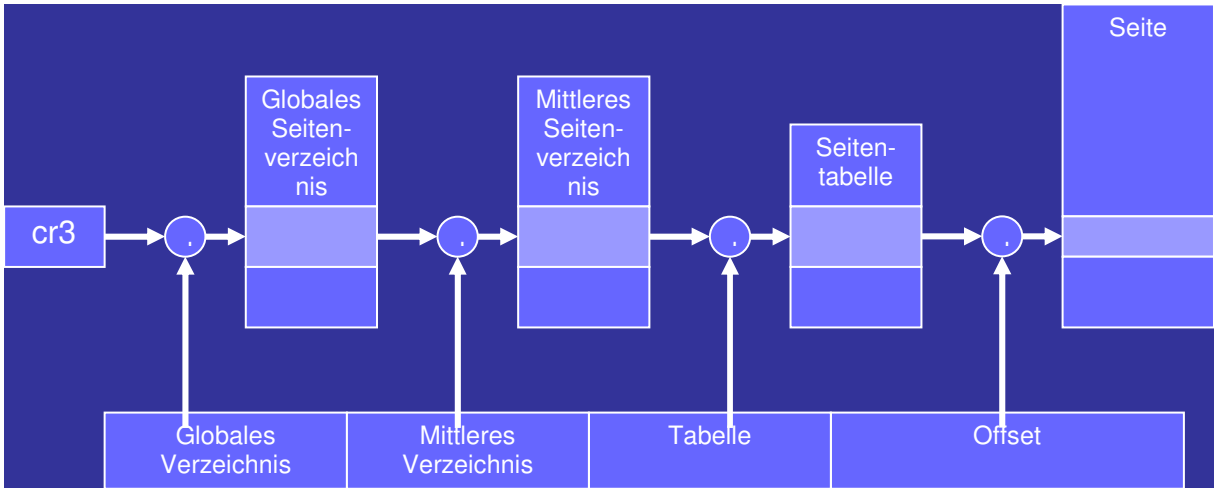
In Linux wird die Segmentierung nur begrenzt angewendet, aufgrund der Überlegenheit von der Seitenadressierung und weil Linux eine große Bandbreite an Architekturen bedienen will.
Bei der Segmentierung in Linux nutzen alle Prozesse logische Adressen, d.h. die Gesamtanzahl an Segmenten ist begrenzt. Alle Segment-Deskriptoren können in der GDT gespeichert werden, allerdings können LDTs trotzdem von Prozessen erzeugt werden
Die Einträge des GDT in Linux bestehen aus dem Kernel Code Segment, einem Kernel Daten Segment sowie je einem User Code Segment und einem User Daten Segment.

Weiterhin wird der Zustand eines jeden Prozesses in einem Segment gespeichert (TSS) und der GDT beinhaltet auch ein Standard-LDT-Segment, das lediglich einen NULL-Segement-Deskriptor enthält, aber auf Anforderung eines Prozesses wird ein 4096 Byte Segment erstellt und das Standard-Segment wird spezifiziert. Desweiteren gibt es noch vier Segment-Deskriptoren für die erweiterte Energieverwaltung.
Zudem enthält der GDT für jeden existierenden Prozess je einen Segment-Deskriptor für das TSS-Segment und das LDT-Segment des jeweiligen Prozesses. Der GDT kann 4090 dieser Prozesseinträge verwalten. Für jeden Prozess existiert ein Prozess-Deskriptor im Kernel-Data-Segment, der sein eigenes TSS und einen Zeiger zu seinem, ebenfalls im KDS enthaltenen LDT-Segment

## 2.2. Seitenadressierung in Linux

### 2.2.1.: Drei-Stufen-Adressierung

Die Seitenadressierung hingegen wird bevorzugt von Linux angewendet aufgrund der angestrebten Lauffähigkeit von Linux auf verschiedenen Architekturen. Die Drei-Stufen-Seitenadressierung auf 64-Bit-Basis ist zwar noch kein Standard, wird aber aufgrund der immer Leistungsfähigeren Rechnersysteme vorangetrieben. Bei dieser art der Seitenadressierung werden lediglich 43 der 64 Bits genutzt. Diese teilen sich ein in ein 13-Bit Offset-Feld und drei 10-Bit-Felder (Globales Seitenverzeichnis, Mittleres Seitenverzeichnis, Seitentabelle). Insgesamt sind dadurch mehr und größere Seiten adressierbar und das findet gern Verwendung bei der Multiprozessverwaltung.




### 2.2.2.: Multiprozessverwaltung

Bei der Multiprozessverwaltung besitzt jeder Prozess ein eigenes Globales Seitenverzeichnis zur Verwaltung prozesseigener Daten und Codes. Das stellt eine Trennung prozesseigener Adressräume sicher und verhindert das Mischen mit „fremden“ Daten. Zudem ermöglicht es eine einfache Ein- und Auslagerung aller wichtigen Elemente eines Prozesses, da diese in einem Adressblock enthalten sind und eine einfache und schnelle Prozessumschaltung ohne Umordnung von geteiltem Speicher.
Die Verwaltung von verschiedenen Prozessen wird durch die Aufteilung des Adressraumes des Arbeitsspeichers (max. 4 GB) durch Prozess-Seitentabellen in einen User-Bereich und einen Kernel-Bereich, begrenzt auf 1 GB durch Linux, gesichert.
Falls der Arbeitsspeicher nicht ausreicht ermöglicht die Implementierung der Physischen Adresserweiterung (PAE) in die Seitenadressierungseinheit, das 64 GB adressierbar sind (durch Adresspinerweiterung von 32 auf 36 Pins).

### 2.2.3.: Kernel-Seitentabellen

Die Kernel Seitentabellen werden während Initialisierung des Systems erstellt. Dabei wird die CPU analyisiert und entsprechende Startwerde gesetzt, z.B. ist die erweiterte Seitenadressierung oder PAE aktiviert? Sind der TLB oder Hardware Cache aktiviert?Diese Tabellen dienen als Referenz für weitere Verzeichnisse und Tabellen die von ihnen abgeleitet werden

### 2.2.4.: Reservierte Seitenkacheln

Es Existieren reservierte Seitenkacheln die ab dem 2. MB des Arbeitsspeichers beginnen. Sie enthalten Kernel-Code und –Daten und sind unverschachtelt für einen schnellen Zugriff.In der Regel haben sie eine Gesamtgröße bis zu 2 MB.

Quellen:

Understanding the Linux Kernel (2nd Edition)
Daniel P. Bovet, Marco Cesati, 2002, Sebastopol