Praktikum Informatik 2:
Betriebssysteme und Rechnernetze

Thema: **4. Speicherverwaltung**
Datum: 19.03.2008

vorgelegt von: Antje Stoppa
Carsten Erdmann
André Hartwig
Ulrike Saretzki

Inhaltsverzeichnis

1 Motivation

- Programme wachsen schneller als verfügbarer Speicher ⇒ **Parkinsonsche Gesetz** (besagt, dass sich Arbeit ausdehnt, um verfügbare Zeit auszufüllen)
- Notwendigkeit der Steigerung der Effizienz der Speicherausnutzung und dadurch Senkung der Kosten
- im Allgemeinen lässt sich der Speicher in verschiedene Speicherklassen unterteilen, die eine Hierarchie bilden (Abbildung Speicherhierarchie)

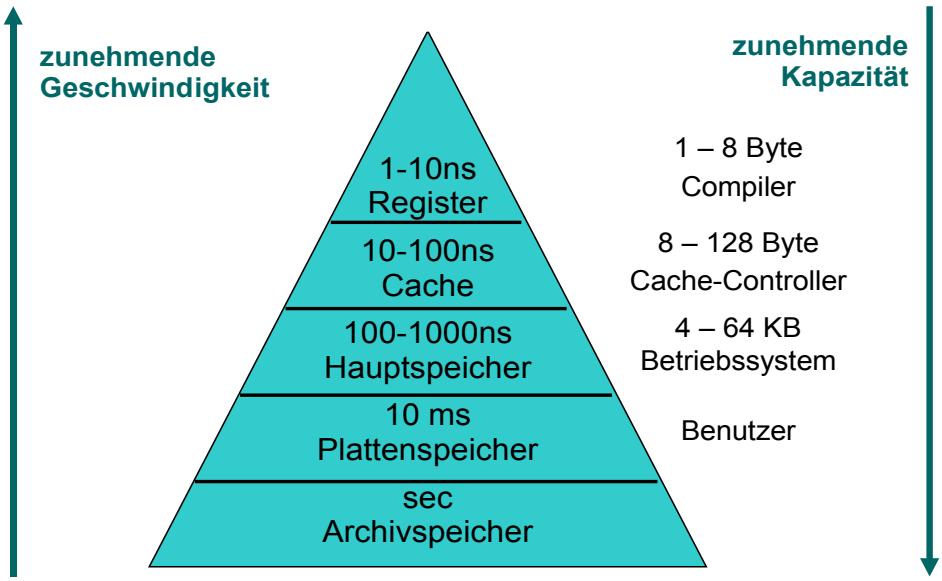

Abbildung 1: Speicherhierarchie

- der Teil des Betriebssystemes, der die Speicherhierarchie verwaltet heißt **Speicherverwaltung**

1.1 Aufgaben der Speicherverwaltung

- Zuteilung von Prozessen (siehe Thema 2) zu den Speicherbereichen
- Übersicht über genutzte Speicherbereiche
- Verwaltung der Auslagerung von Dateien

Speicherverwaltungssysteme können generell in 2 Klassen unterteilt werden

1. Systeme ohne Verschiebung von Dateien
2. Systeme mit Verschiebung von Dateien zwischen Hauptspeicher und Festplatte

2 Speicherverwaltungsstrategien ohne Verschiebung

2.1 Monoprogrammierung ohne Auslagerung oder Seitenverwaltung

- es ist immer nur maximal 1 Programm gleichzeitig aktiv, dieses wird abgearbeitet und erst dann kann ein neues Programm ausgeführt werden
- Einsatz heutzutage nur noch in Palmtops und eingebetteten Systemen, daher für den normalen Programmierer eher unwichtig

2.2 Multiprogrammierung

- Einteilung des Speichers in n (verschieden große) feste Partitionen
- Vorteil gegenüber der Monoprogrammierung ist die bessere CPU-Auslastung, beispielsweise, wenn ein Programm auf das Ende einer E/A wartet, kann ein andereres die CPU benutzen
- zur Implementierung gibt es verschiedene Möglichkeiten (Abbildung feste Partitionen)
  (a) jede Partition bekommt eine eigene Warteschlange (jeder Prozess geht an die Warteschlange der kleinsten Partition, die den Auftrag ausführen kann, Problem → **Verstopfung**)
  (b) alle Partitionen bekommen eine gemeinsame Warteschlange (Prozess geht in irgendeine Partition, die groß genug und gerade frei ist, Problem der Ineffizienz)
- Lösung: mehrere kleine Partitionen einrichten, da in der Regel viele kleinere Prozesse ausgeführt werden müssen
- Anwendung vor allem bei OS/360 und IBM Mainframes

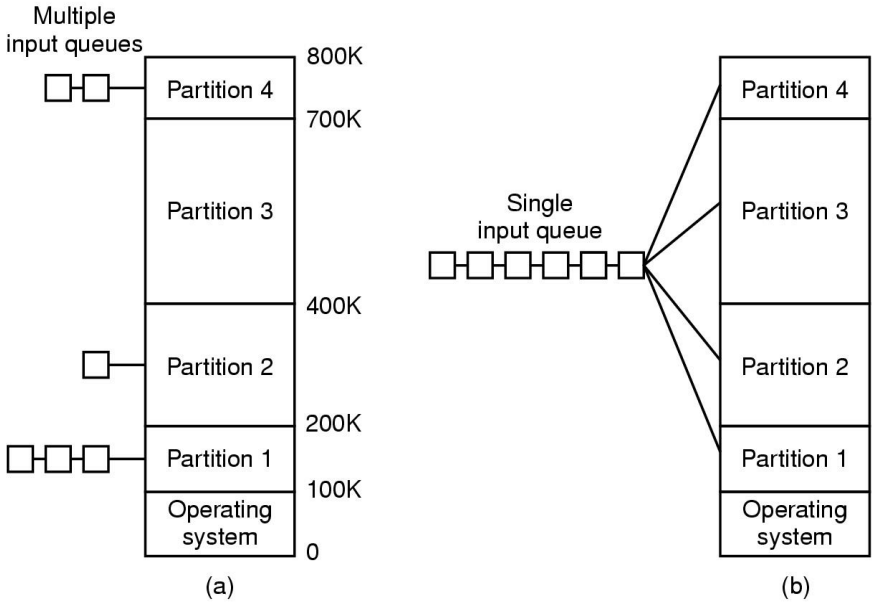

Abbildung 2: feste Partitionen

2.2.1 Modellierung der Multiprogrammierung

- Annahmen:
  - ein Prozess benötigt einen Anteil p seiner Zeit, um auf das Ende einer E/A-Operation zu warten
  - n Prozesse im Speicher
  - Prozesse laufen unabhängig voneinander
  - Vernachlässigung des Betriebssystems Overheads
- die Wahrscheinlichkeit, dass alle gleichzeitig auf E/A warten ist offensichtlich p^n
- die CPU-Auslastung demzufolge $A = 1 - p^n$

2.2.2 Relokation und Speicherschutz

Durch Multiprogrammierung treten 2 grundlegende Probleme auf

1. Relokation
   - besteht darin, dass zum Zeitpunkt des Bindens des Programms nicht bekannt ist, an welchen Speicherplatz das Programm zur Ausführungszeit geladen wird
   - der Adressbereich muss daher zu Beginn der Ausführung verschoben (relokiert) werden
2. Speicherschutz
   - Speicher ist so aufzuteilen, dass laufende Programme so voneinander zu trennen sind, dass sich ein Programmierfehler/Absturz eines einzelnen Programms nicht auf die Stabilität anderer Programme oder des Gesamtsystems auswirken

3 Swapping

- ein Prozess wird komplett in den Arbeitsspeicher geladen und dann ausgeführt und anschließend wieder auf die Festplatte ausgelagert
- Anzahl, Größe und Ort der Partitionen sind dynamisch
- bei auftretenden Löchern zwischen den Speicherbereichen könnten diese zusammengefasst werden ⇒ **Speicherverdichtung**
- in der Praxis kommt es jedoch kaum zu Speicherverdichtungen, da diese sehr rechenintensiv sind
- Beispiel: (Abbildung Swapping)
  (a) Prozess A wird gestartet und im Hauptspeicher abgelegt
  (b) Prozess B kommt hinzu
  (c) Prozess C kommt hinzu
  (d) Prozess A wird ausgelagert, Platz im Hauptspeicher wird frei
  (e) Prozess D wird gestartet und belegt nur einen kleineren Teil als A zuvor
  (f) Prozess B wird ausgelagert
  (g) Prozess A wird wieder eingelagert und belegt ehemaligen Speicherbereich von B und Teil von A

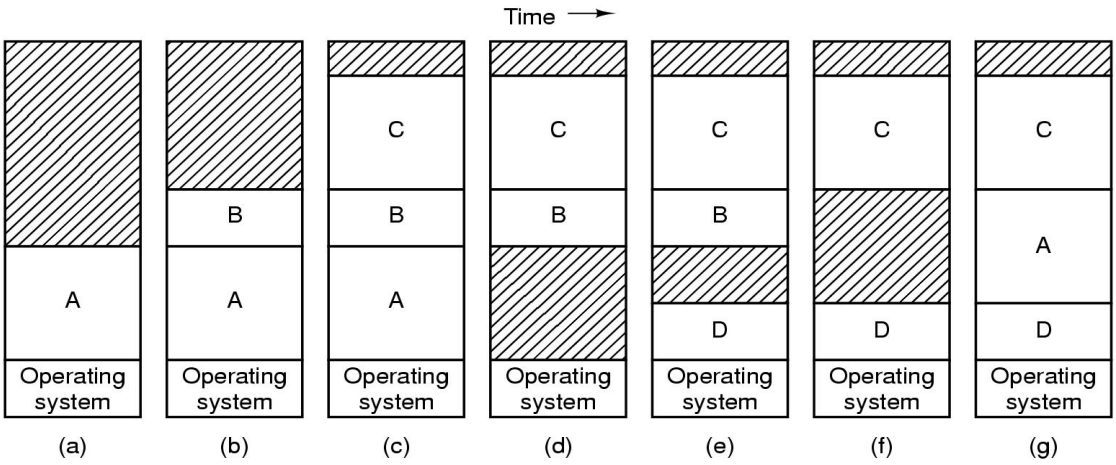

Abbildung 3: Swapping

- Problem: wieviel Speicher braucht ein Programm (moderne Programme sind dynamisch programmiert)?

3.1 Speicherverwaltung mit Bitmaps

- Einteilung des Speichers in Allokationseinheiten
- Frage: Wie groß sollen diese Einheiten seien, je kleiner, desto größer die Bitmap; je größer, desto mehr ungenutzter Speicher
- Speicher sollte also so aufgeteilt werden, dass Prozesse, falls sie mehr Speicher benötigen, sich diesen von einem großen Speicherbereich abschneiden
- Beispiel: (Abbildung Speicherverwaltung mittels Bitmap)
  (a) ein Teil des Speichers mit 5 Prozessen und 3 Löchern, die Teilstriche markieren jeweils die Grenzen der Allokationseinheiten, die schraffierten Bereiche sind frei
  (b) Darstellung als Bitmap
  (c) dasselbe nochmal als Liste

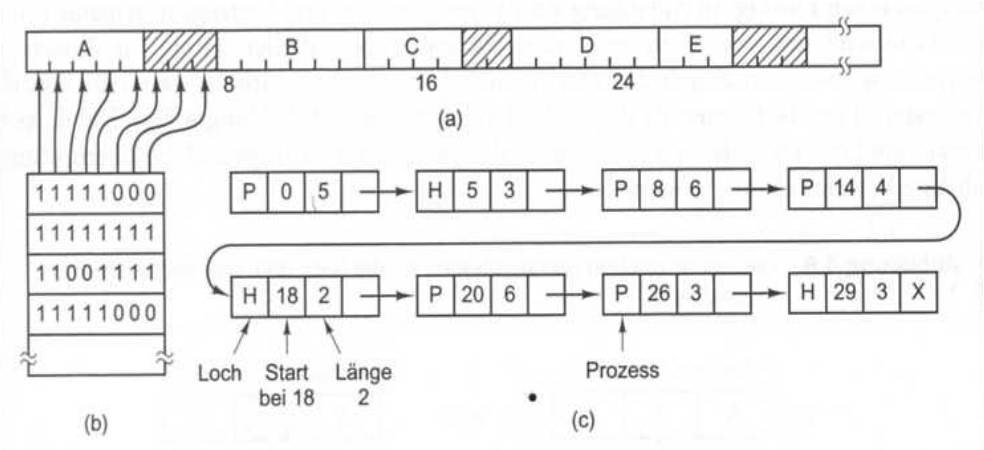

Abbildung 4: Speicherverwaltung mittels Bitmap

3.2 Speicherverwaltung mit verketteten Listen

- Liste enthält als Eintrag Loch oder Prozess
- Speicherverwaltung geht Liste durch, bis ein ausreichend groß es Loch gefunden wurde
- Methoden der Speicherverwaltung
  - **First Fit** (Liste wird durchgegangen bis auf ein ausreichend großes Loch gestoßen wurde $\Rightarrow$ schnell, jedoch wenig effizient)
  - **Next Fit** (wie First Fit, jedoch beginnt die nächste Suche dort, wo zuletzt ein passendes Loch gefunden wurde $\Rightarrow$ schnell, jedoch wenig effizient)
  - **Best Fit** (durchsucht die gesamte Liste und sucht das am besten passende Loch aus $\Rightarrow$ langsam, großer Overhead, effizienter)
  - **Worst Fit** (durchsucht die gesamte Liste und wählt das größte verfügbare Loch aus $\Rightarrow$ langsam, großer Overhead, effizienter)
  - **Quick Fit** (verschiedene Listen mit üblichen Größen, es wird die Liste der jeweiligen Größe durchgegangen $\Rightarrow$ extrem schnell)

4 virtueller Speicher

- der virtuelle Speicher bezeichnet den vom tatsächlich vorhandenen Arbeitsspeicher unabhängigen Adressraum, der einem Prozess für Daten und das Programm vom Betriebssystem zur Verfügung gestellt wird
- es werden **nur** die Daten eines Programmes in den Arbeitsspeicher geladen, die gerade benötigt werden, der **Rest verbleibt** auf der Festplatte

4.1 Paging

- Programme benötigen Speicheradressen, damit im Programm generierte Daten gespeichert werden können, z.B. MOV REG, 1000 (Verschiebe den Inhalt der Speicheradresse 1000 in das Register)
- diese müssen jedoch mit dem Speicherbereich, die dem System als Ganzes zur Verfügung steht, abgestimmt werden
- die von den Programmen generierten Speicheradressen werden als **virtuelle Adressen** bezeichnet und zusammen bilden sie so einen **virtuellen Adressraum**
- bei Rechnern ohne virtuellem Speicher wird die Adresse direkt auf den Speicherbus gelegt und das physische Speicherwort direkt übernommen
- bei Rechnern mit virtuellem Speicher geht die Adresse an die **MMU(Memory Management Unit)** und wird von dieser auf die physische Adresse abgebildet (Abbildung Ort und Funktion der MMU)

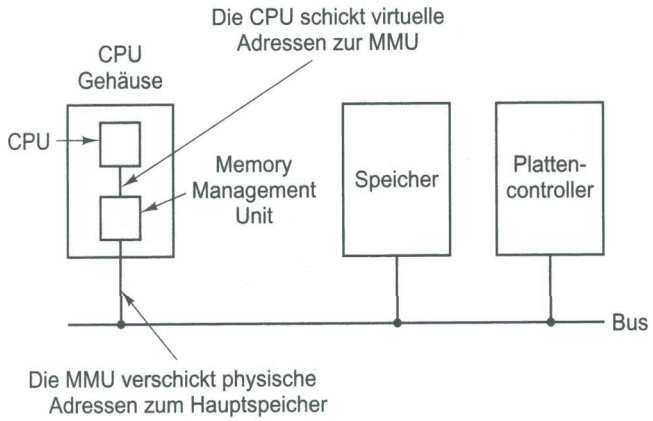

Abbildung 5: Ort und Funktion der Memory Management Unit

- der virtuelle Adressraum ist in Einheiten unterteilt, den **Seiten**
- die zugehörigen Einheiten im physischen Speicher heißen **Seitenrahmen**
- Seite und Seitenrahmen sind logischerweise gleich groß
- eine **Seitentabelle** legt die Beziehungen zwischen virtuellem und physischem Speicher fest, also welche virtuelle Seite ist überhaupt physisch im Speicher vorhanden und welche Adresse hat dieser Speicherbereich dann

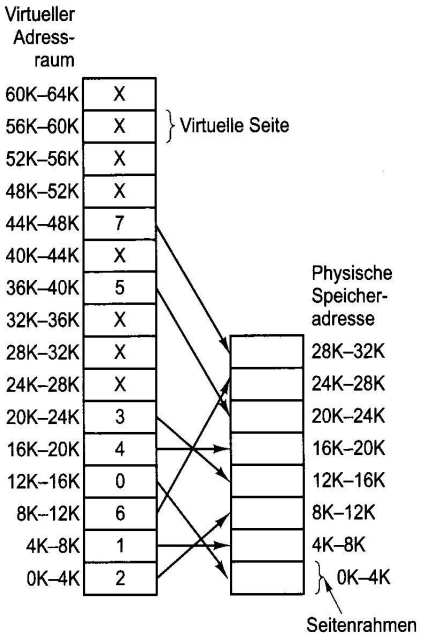

Abbildung 6: Seitentabelle

- Beispiel: (Abbildung Seitentabelle)
  - Programm will auf die Adresse 0 zugreifen, also Befehl: MOV REG, 0
  - virtuelle Adresse 0 wird an MMU geschickt
  - MMU stellt fest, dass die virtuelle Adresse zur Seite 0 gehört, was laut Abbildung dem Seitenrahmen 2 entspricht
  - die virtuelle Adress 0 wird also auf die physische Adresse 8192 abgebildet (8192 weil laut Abbildung auf physikalische Adresse 8K-12K und $8 \cdot 1024 = 8192$.
- Frage: Wie löst man dann das eigentliche Problem, dass der virtuelle Speicher größer ist als der physische?
  - durch Einführung eines present/absent-Bit, dieses gibt an, welche Seiten auf physischen Speicher abgebildet sind und welche nicht
- falls auf eine Seite zugegriffen wird, die nicht im Speicher liegt, kommt es zu einem **Seitenfehler**
- wenn ein Seitenfehler auftritt, muss eine vorhandene Seite aus der Seitentabelle ausgelagert werden, mittels einem Seitenersetzungsalgorithmus

4.2 Seitentabellen

- einfachster Fall:
  - virtuelle Adresse wird in eine virtuelle Seitennummer und einen Offset unterteilt
  - die virtuelle Seitennummer wird als Index benutzt, um in der Seitentabelle den Eintrag für diese Seite zu finden

  - der Seitentabelleneintrag enthält die Nummer des entsprechenden Seitenrahmens
  - die Seitenrahmennummer wird an das höherwertige Ende des Offsets angehängt und ersetzt die virtuelle Seitennummer
  - Seitenrahmennummer und Offset ergeben die physische Adresse

- 2 grundlegende Probleme
  1. die Seitentabelle kann extrem groß werden (die virtuellen Adressen moderner Computer sind mindestens 64 Bit lang)
  2. die Umrechnung muss dennoch extrem schnell sein (folgt aus der Notwendigkeit, dass für jeden Zugriff auf den physischen Speicher eine Umrechnung erforderlich ist)

- Lösungsansätze
  1. eine einzige Seitentabelle (Aber: Problem der enormen Größe)
  2. Seitentabelle komplett in den Hauptspeicher (Problem der ineffizienten Nutzung von schnellem Speicher)
  3. mehrstufige Seitentabellen
     - die Seitentabellen, die nicht mehr gebraucht werden, werden ausgelagert
     - dabei enthält die oberste Tabelle Einträge mit Verweisen auf andere untergeordnete Seitentabellen, die wiederum Einträge auf Seitentabellen oder auch nur Seiten haben können

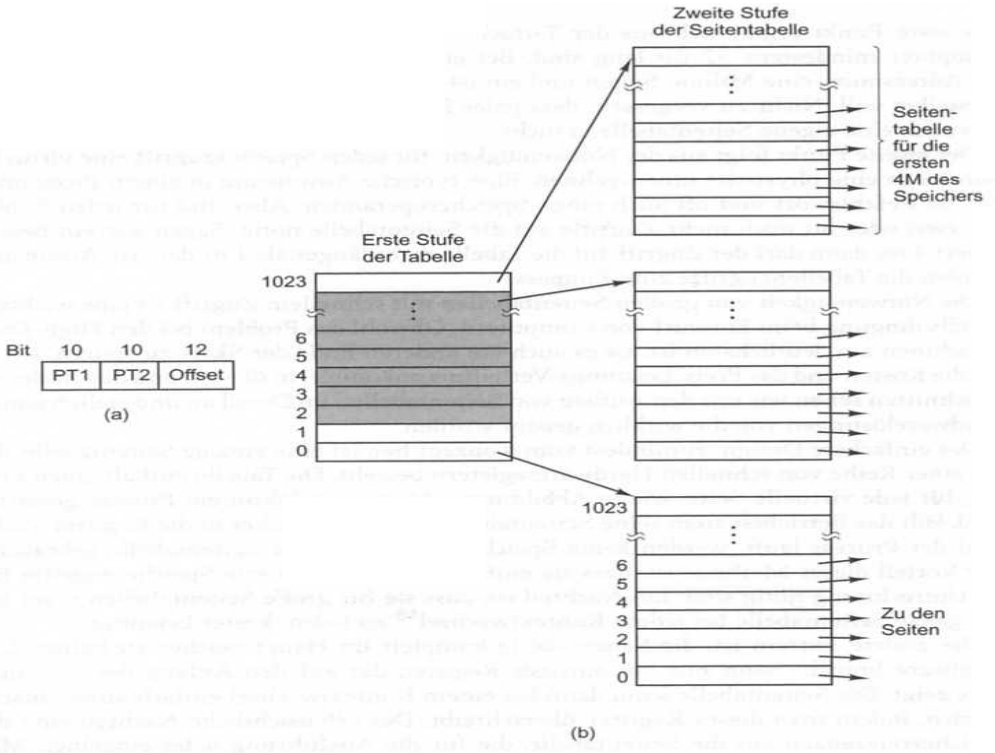

Abbildung 7: Mehrstufige Seitentabelle

4.3 Aufbau eines Seitentabelleneintrages

ein Seitentabelleneintrag enthält

- Seitenrahmennummer
- present/absent Bit (gibt an, ob Seite im Hauptspeicher)
- Protection-Bits (Schutz der Seite vor überschreiben)
- Modified-Bit / M-Bit (wird gesetzt, wenn auf eine Seite geschrieben wurde)
- Referenced-Bit / R-Bit(wird bei jedem Zugriff auf eine Seite gesetzt)
- Caching-Bit (regelt das Caching für eine Seite)

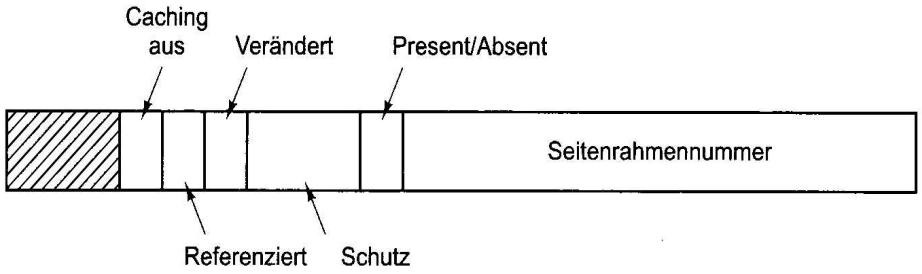

Abbildung 8: Aufbau Seitenrahmen

4.4 TLB - Translation Lookaside Buffer

In der Praxis tritt das Problem auf, dass Programme dazu neigen, auf wenige Seiten sehr häufig zuzugreifen. Daher werden diese virtuellen Adressen ohne Umweg mittels Hardware auf die physische Adresse abgebildet. Diese Hardware wird mit TLB bezeichnet, in der Regel Teil der MMU. Wird eine virtuelle Adresse an die MMU geschickt, werden erst parallel alle Einträge der TLB überprüft. Wenn ein Eintrag noch nicht in der TLB steht, wird dieser normal aus der Seitentabelle geholt und der älteste Eintrag der TLB wird durch diesen neuen überschrieben. Der TLB hat den Vorteil, dass er extrem schnell ist.

4.5 Invertierte Seitentabellen

- Seitentabellen benötigen nach der bisher beschriebenen Methode einen Eintrag pro Seite, weil auf sie über die virtuelle Seitennummer zugegriffen wird (Abbildung Invertierte Seitentabelle)
- dies würde für moderne 64-Bit-Rechner jedoch bedeuten, dass die Seitentabelle 2^{52} Einträge hätte, also bei 8 Byte pro Eintrag wären das 30 Millionen GB
- daher wird für jeden physischen Seitenrahmen ein Eintrag in der Seitentabelle gespeichert, anstatt für jede Seite im virtuellem Adressraum
- für ein 64-Bit System mit 2048MB und 8KB großen Seiten wären nur 262144 Einträge nötig (Abbildung Invertierte Seitentabelle)
- dadurch sparen invertierte Seitentabellen zwar enorm viel Speicherplatz, dafür ist es aber wesentlich schwieriger eine virtuelle Seite auf eine physische abzubilden
- Lösung vor allem durch den TLB und Hash-Tabelle
- in der Hash-Tabelle befinden sich virtuelle Adressen als Hash-Werte, alle virtuellen Adressen mit denselben Hash-Wert sind verkettet (Abbildung: invertierte Seitentabelle) (Abbildung Invertierte Seitentabelle)
- hat die Hash-Tabelle so viele Einträge, wie das System physische Seiten, so wird die Suche extrem beschleunigt
- ist die Seitenrahmennummer gefunden, wird das neue Paar aus virtueller Seite und physischem Seitenrahmen in die TLB eingetragen

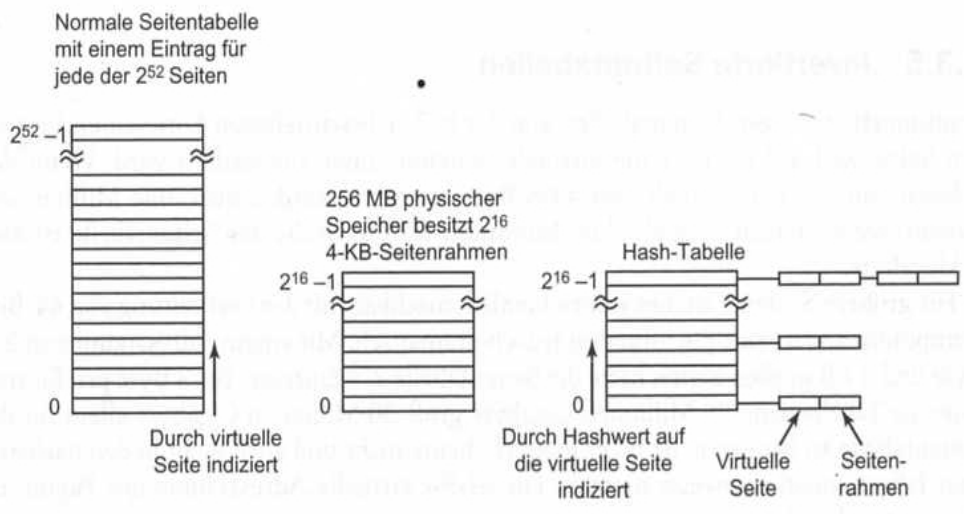

Abbildung 9: Invertierte Seitentabelle

5 Seitenersetzungsalgorithmen

Welche Seite soll bei einem auftretenden Seitenfehler entfernt werden, wenn die Seitentabelle voll ist?

5.1 Optimaler Seitenersetzungsalgorithmus

- jede Seite wird mit der Anzahl der Befehle markiert, die bis zum nächsten Zugriff auf diese Seite ausgeführt werden
- die Seite mit der höchsten Anzahl von Befehlen wird entfernt
- Problem: Wann wird auf eine Seite das nächste Mal zugegriffen?
- nicht implementierbar, jedoch Möglichkeit des mathematischen Vergleichs bestehender Algorithmen mit dem Bestmöglichsten

5.2 Not Recently Used Algorithmus

- Aufteilung der Seiten in 4 Kategorien, abhängig von dem M- und R-Bit
  - Klasse 0: nicht referenziert (R-Bit ist 0), nicht modifiziert (M-Bit ist 0)
  - Klasse 1: nicht referenziert, modifiziert
  - Klasse 2: referenziert, nicht modifiziert
  - Klasse 3: referenziert, modifiziert
- es wird zufällig eine Seite aus der niedrigsten nicht leeren Klasse entfernt
- Vorteile: leichte Implementierung, leicht verständlich
- Nachteile: Leistung nicht optimal, aber in vielen Fällen ausreichend

5.3 First In First Out (FIFO)

- es wird diejenige Seite entfernt, deren Einlagerung am weitesten zurück liegt
- Vor- und Nachteile wie bei NRU

5.4 Second-Chance-Algorithmus

- Variante von FIFO
- überprüft, ob die älteste Seite referenziert wurde, wenn nein, dann Löschung, ansonsten wird Referenzierung aufgehoben und Seite so behandelt, als wäre sie gerade eingelagert worden, ihr wird also noch eine *zweite Chance* gegeben
- Beispiel: (Abbildung: Second-Chance Algorithmus)

(a) Seiten wurden in FIFO-Ordnung sortiert (die Zahlen sind Ladezeiten)

(b) angenommen, bei A war das R-Bit gesetzt, dann wird es auf 0 gesetzt und wieder behandelt wie als wäre gerade darauf zugegriffen worden

- Vor- und Nachteile wie bei NRU

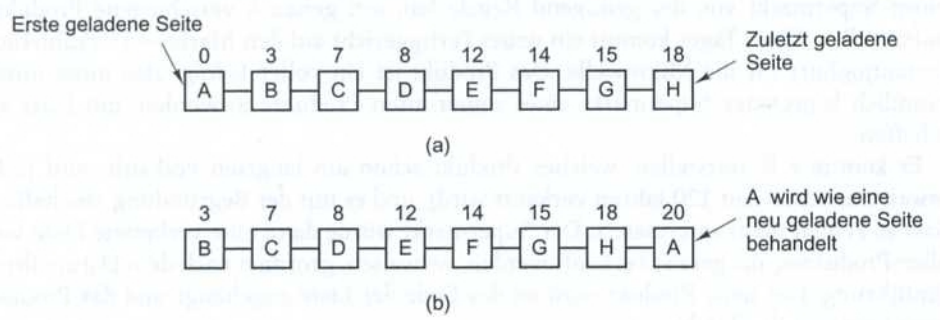

Abbildung 10: Second-Chance Algorithmus

5.5 Clock-Algorithmus

- der gleiche Algorithmus wie Second-Chance, jedoch keine Implementierung mittels Liste, sondern per Zeiger

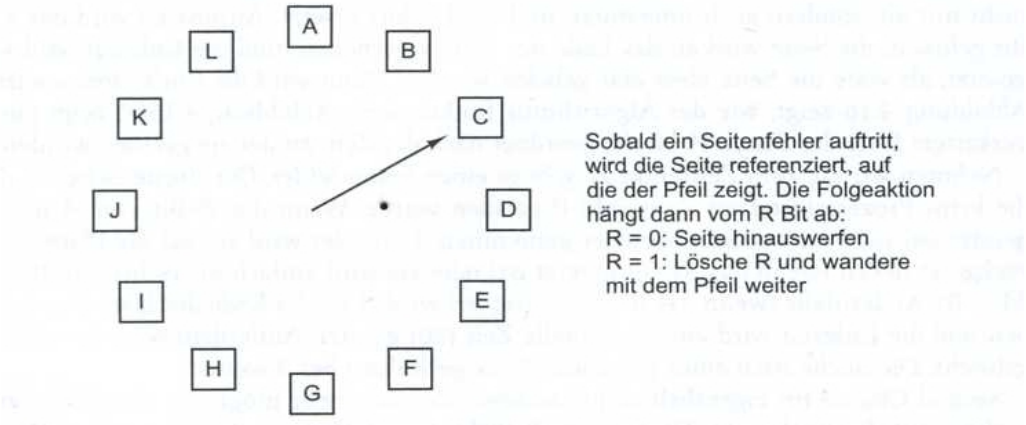

Abbildung 11: Second-Chance Algorithmus mittels Clock

5.6 Least Recently Used Algorithmus

- entfernt die Seite, die am längsten unbenutzt ist
- zur Realisierung ist eine verkettete Liste von allen Seiten im Speicher nötig (zuletzt Benutzte am Anfang und am längsten Unbenutzte am Ende)

- problematisch ist die Aktualisierung, da Liste jedes mal neu durchgegangen werden muss
- Lösung durch Hardware:
  1. Hardware enthält einen 64-Bit Zähler
     - jede Seite ist mit eigenem Zähler ausgestattet
     - dieser enthält den aktuellen (globalen) Zählerstand des letzten Zugriffs auf die Seite
     - es wird diejenige Seite mit dem niedrigsten Zählerstand entfernt
  2. LRU-Hardware enthält eine $n \times n$ Matrix (für n Seitenrahmen) (Abbildung Least Recently Used)
     - zum Anfang sind alle Einträge 0 gesetzt
     - wenn auf Seitenrahmen k zugegriffen wird, werden alle Bits der Zeile k auf 1 und die der k-ten Spalte auf 0 gesetzt
     - zu jedem Zeitpunkt ist die Zeile mit dem niedrigsten Binärwert die am längsten nicht benutzte
- Vorteil: sehr effizient
- Nachteil: aufwendige Implemtentierung

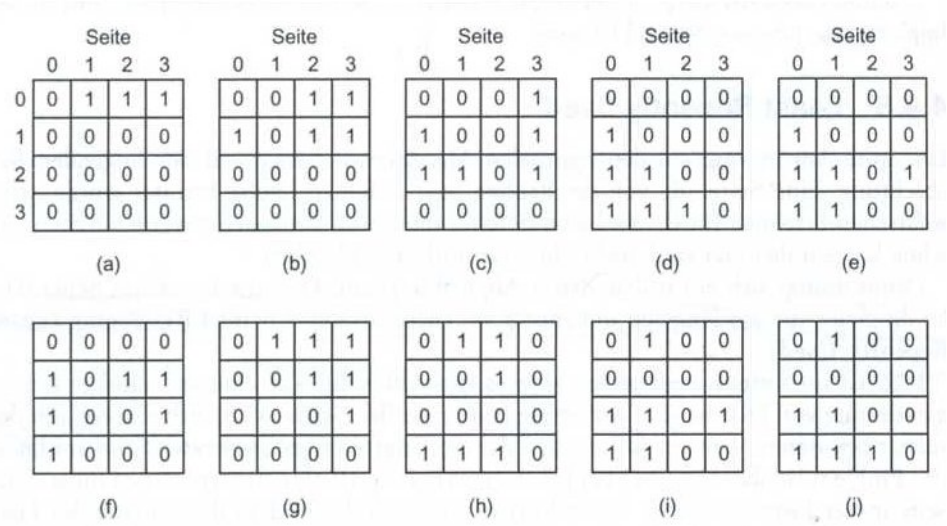

Abbildung 12: Least Recently Used

5.7 Realisierung des LRU-Algorithmusses durch Software - Aging Algorithmus

- jede Seite wird mit einem Softwarezähler ausgestattet, dieser ist zu Beginn auf 0 gesetzt
- bei einer Unterbrechung wird zum ganz linken (dem höchstwertigsten) Bit des Zählers der jeweilige Wert des R-Bits addiert
- zuvor wird jedoch der Zähler um ein Bit nach rechts verschoben

- Beispiel: (Abbildung Aging Algorithmus)
  (a) auf Seite 0,2,4,5 wurde zugegriffen
  (b) auf Seite 0,1,4 wurde zugegriffen, der Zähler verschiebt sich entsprechend nach rechts
  (c) auf Seite 0,1,3,5 wurde zugegriffen
  (d) ...
- Probleme:
  1. der Zähler ist endlich, man erkennt nicht ob auf eine Seite vor 9 oder vor 1000 Intervallen zugegriffen wurde
  2. im Intervall erkennt man nicht auf welche Seite vorher zugegriffen wurde
- letztendlich wird in diesen Fällen eine zufällige, der in Frage kommenden Seiten gelöscht

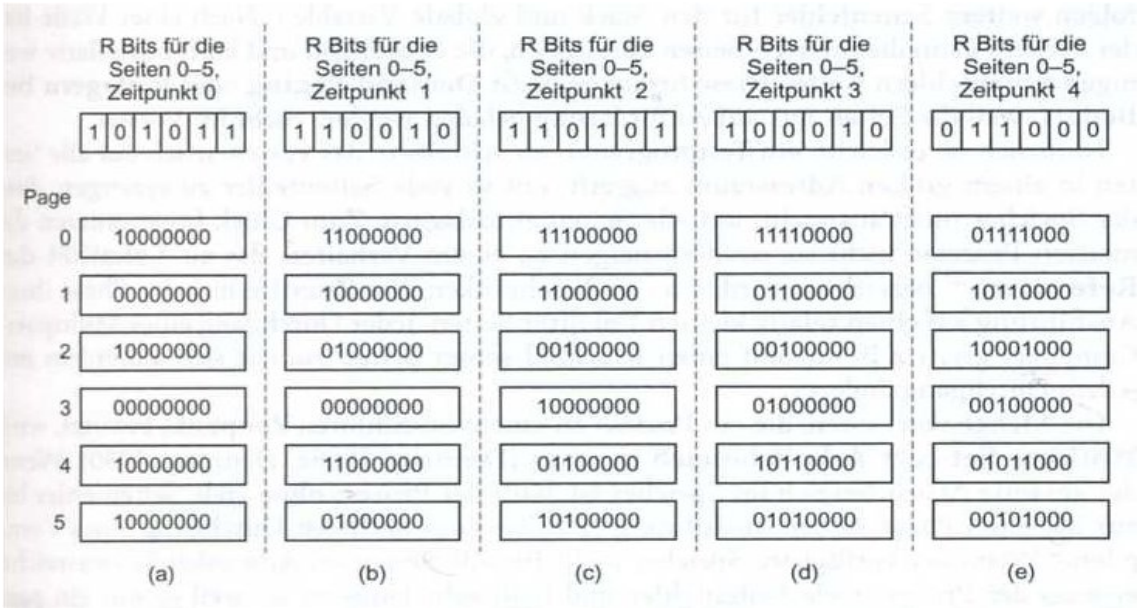

Abbildung 13: Aging Algorithmus

5.8 Working-Set-Algorithmus

- beim reinen Paging werden Seiten erst bei Bedarf eingelagert $\Rightarrow$ Seitenfehler beim Initialisieren
- unter einem **working set** versteht man die Menge von Seiten, die ein Prozess zu einem Zeitpunkt benutzt
- wird ein Prozess ausgelagert z.B. bei Multiprogrammierung, und wird später wieder eingelagert, so entstehen beim Einlagern der einzelnen Seiten solange Seitenfehler, bis der gesamte Arbeitsbereich wieder hergestellt wurde $\Rightarrow$ ineffizient
- das **Working-Set Modell** sieht nun vor, dass der Arbeitsbereich erst vollständig geladen wird, bevor der Prozess gestartet wird; wenn ein Seitenfehler auftritt, wird die Seite ausgelagert, die nicht zum Arbeitsbereich gehört

- dazu muss das BS den Arbeitsbereich eines Programmes exakt kennen
- der Arbeitsbereich umfasst all diejenigen Seiten, die in den letzten k Speicherzugriffen benutzt wurden
- in der Realität wird der Arbeitsbereich eher so definiert, dass er sämtliche Seiten umfasst, auf denen in den letzten 100 ms zugegriffen worden ist
- Beispiel: (Abbildung Working-Set Algorithmus)
  - zuerst wird das R-Bit untersucht, ist es gesetzt, so wird die aktuelle virtuelle Zeit in das Feld für die *Zeit des letzten Zugriffs* eingetragen
  - ist es nicht gesetzt, so wird die Zeitspanne zwischen dem letzten Zugriff und der virtuellen Zeit berechnet und mit einer Vorgabezeit τ verglichen
  - ist diese Zeitspanne größer als τ so wird sie ausgelagert
  - der Rest der Tabelle wird durchlaufen und der Zeitstempel aktualisiert
  - ist keine Seite mit einer Zeitspanne kleiner τ gefunden, so wird die Seite ausgelagert, die am ältesten ist

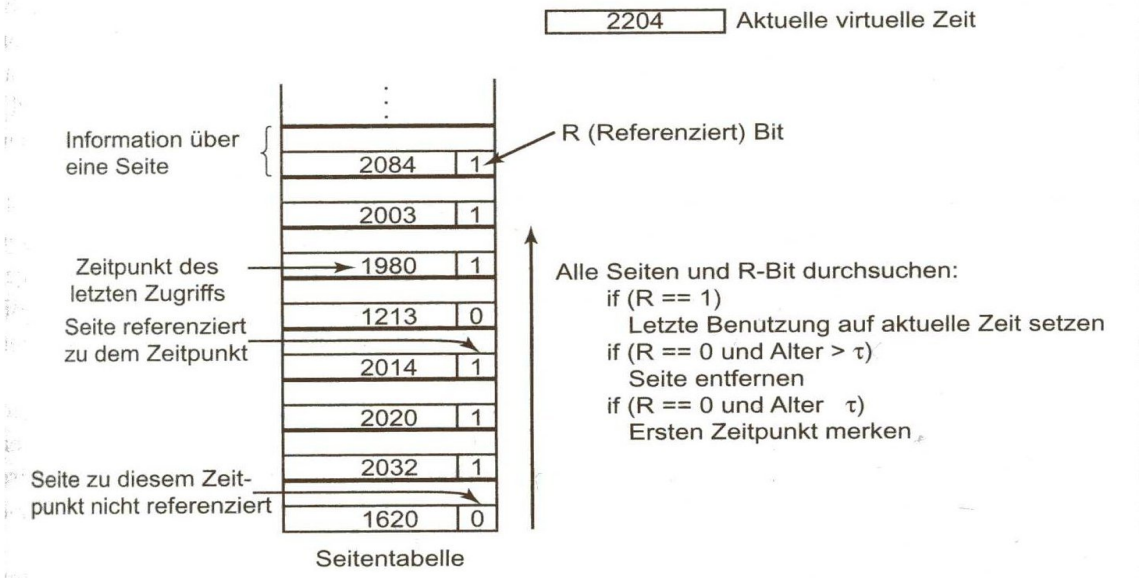

Abbildung 14: Working-Set Algorithmus

5.9 WSClock - Algorithmus

- der normale WS-Algorithmus ist umständlich, weil bei jedem Seitenfehler die gesamte Seitentabelle durchlaufen werden muss
- der WSClock benutzt eine ringförmige Liste von Seitenrahmen
- zum Anfang ist die Liste leer
- wird eine Seite geladen, dann wird sie in die Liste eingefügt, solange bis Liste voll ist
- jeder Listeneintrag hat ein R-Bit, ein M-Bit und ein Feld für die *Zeit des letzten Zugriffs*
- Beispiel: (Abbildung Funktionsweise WSClock-Algorithmus)
  - ist das R-Bit gesetzt, wird es auf 0 gesetzt und der Zeiger rückt auf die nächste Seite vor (Abbildung (a),(b))
  - ist das R-Bit 0 und das Alter der Seite größer als τ wird diese ausgelagert und durch eine neue Seite ersetzt, der Zeiger rückt auf die nächste Seite vor (Abbildung (c),(d))
  - wurde keine passende Seite gefunden, dann wird die älteste ausgelagert

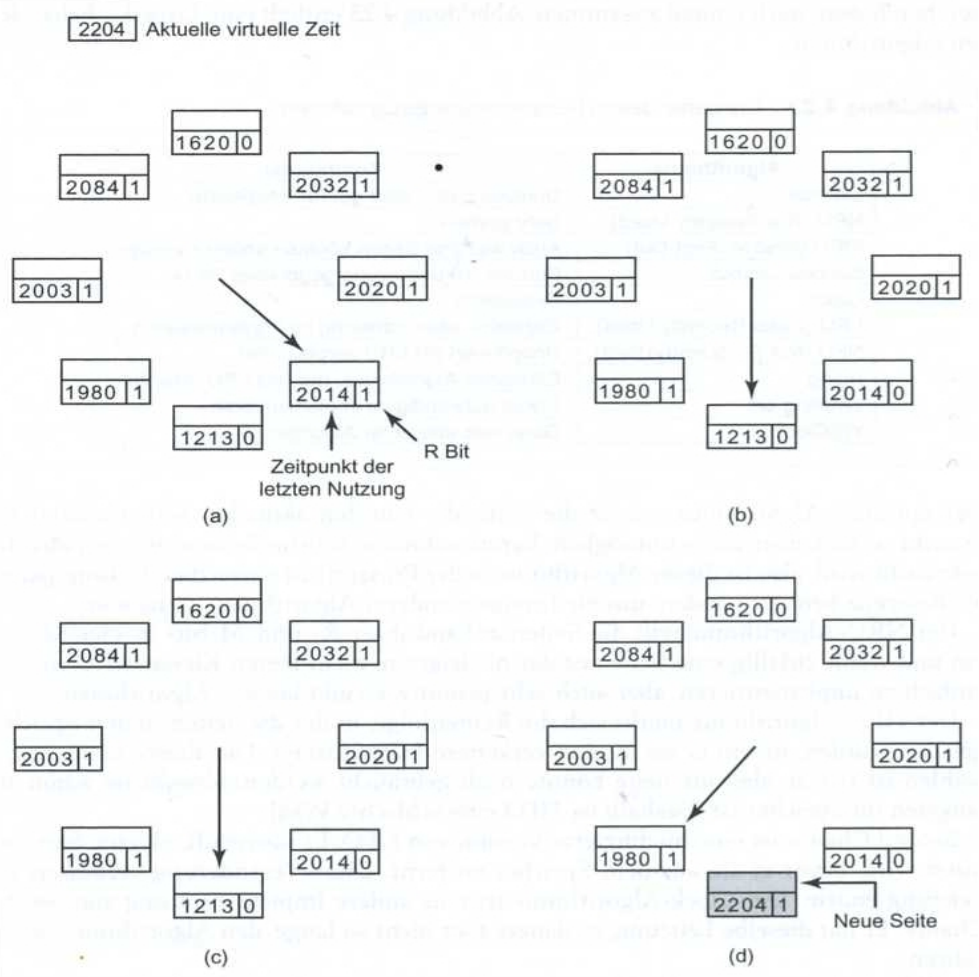

Abbildung 15: Funktionsweise WSClock-Algorithmus

6 Modellierung von Seitenersetzungsalgorithmen

6.1 Beladys Anomalie

- intuitiv sollte ein System mit mehr physischen Seitenrahmen auch weniger Fehler erzeugen $\Rightarrow$ denkste!!
- Gegenbeispiel: (Abbildung Belayds Anomalie)
  - FIFO
  - 5 virtuelle Seiten
  - Zugriff auf Seiten nach Reihenfolge: 0 1 2 3 0 1 4 0 1 2 3 4
  - bei 3 Seitenrahmen ergeben sich 9 Seitenfehler, bei 4 jedoch 10 Seitenfehler

Abbildung 16: Belayds Anomalie

6.2 Keller Algorithmus

- jeder Prozess erzeugt, während er läuft, eine Folge von Speicherzugriffen
- die Speicherzugriffe eines Prozesses kann man sich also als eine geordnete Liste von Seitennummern vorstellen $\Rightarrow$ **Referenzkette**
- ein Pagingsystem wird daher durch 3 Dinge charakterisiert
  1. die Referenzkette des Prozesses der ausgeführt wird
  2. den Seitenersetzungsalgorithmus
  3. die Anzahl $\boldsymbol{m}$ der physischen Seitenrahmen
- nun kann man sich einen abstrakten Interpreter vorstellen
  - dieser enthält eine interne Tabelle $\boldsymbol{M}$, diese repräsentiert den Zustand des Speichers

- die Tabelle enthält $\boldsymbol{n}$ Einträge, eine für jede virtuelle Seite eines Prozesses
- die Tabelle $\boldsymbol{M}$ hat 2 Teile
- der Obere enthält alldiejenigen Einträge, die momentan im Speicher liegen
- der Untere enthält die Seiten auf die schon einmal zugegriffen wurde, die jedoch ausgelagert wurden
- ein Prozess startet und fängt an, die Seiten seiner Referenzkette auszugeben
- für jede Seite wird geprüft, ob diese im Speicher ist, wenn nicht, dann wird ein Seitenfehler erzeugt
- dann wird die Seite an den Anfang der Speichertabelle getan und die anderen Einträge dementsprechend verschoben (Abbildung Speichertabelle)

- Algorithmen mit der Eigenschaft $M(m,r) \subset M(m+1,r)$ werden **Keller Algorithmen** genannt

6.3 Distanzkette

- Darstellung der Referenzkette nicht durch Seitennummern, sondern auf abstrakte Weise durch den Abstand vom oberen Keller bei einem Zugriff
- Seiten auf die noch nicht zugegriffen wurde, haben den Abstand ∞
- die Distanzkette ist also nicht nur von der Referenzkette, sondern auch von dem Seitenersetzungsalgorithmus abhängig
- Beispiel: (Abbildung Speichertabelle)
  - wir haben 8 verschiedene Seiten
  - die Referenzkette besteht aus 24 Seiten: 0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 4 1
  - die oberen 4 Zeilen der Tabelle geben jeweils die Seiten an, die im Speicher liegen
  - wird eine Seite angefordert, die nicht im Speicher liegt, so kommt es zu einem Seitenfehler

Referenzkette	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Seitenfehler	P	P	P	P	P	P	P		P					P			P						P	
Distanzkette	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	2	3	5	3

Abbildung 17: Speichertabelle

6.4 Seitenfehlerrate hervorsagen

- Ziel ist es hervorzusagen, wie viele Seitenfehler bei einem Speicher von $1, 2, 3, .., n$ Seitenrahmen entstehen
- dazu durchläuft der Algorithmus die Distanzkette und zählt, wie oft der Abstand 1, 2, .. vorkommt
- die Seite i kommt C_i mal vor, c_∞ steht für die Anzahl des Vorkommens der Distanzkette mit Abstand ∞
- die Anzahl F_m der Seitenfehler für eine gegebene Distanzkette und vorgegebenen Speicher mit m Seitenrahmen errechnet sich also aus

$$F_m = \sum_{k=m+1}^{n} C_k + c_\infty$$

- Beispiel:(Abbildung Speichertabelle)
  - $m = 4\,,\ n = 24\,,\ c_\infty = 8$
  - $F_5 = 2\,,\ F_6 = 1\,,\ F_7 = 0$
  - $F_m = 2 + 1 + 0 + 8 = 11$

7 Design-Kriterien

7.1 lokale und globale Paging-Strategien

- Frage: Wie soll der Speicher zwischen den konkurrierenden Prozessen aufgeteilt werden
- bei **lokalen Paging-Strategien** wird jedem Prozess ein **fester Speicherbereich** zugeteilt
- bei **globalen Strategien** verteilen sich die Seitenrahmen **dynamisch** unter den lauffähigen Prozessen
- im Allgemeinen ist die globale Strategie zu bervorzugen

7.2 Laststeuerung

Trotz optimierter Seitenersetzungsstrategien kann es vorkommen, dass zu viele Seitenfehler produziert werden z.B. wenn der PPF-Algorithmus anzeigt, dass alle Programme zu wenig Speicher haben.

7.3 Seitengröße

- die optimale Seitengröße ist unter Abwägung mehrerer Faktoren zu bestimmen
  - **interne Fragmentierung** (in der Regel wird ein zufälliges Text-, Datum-, oder Stacksegment keine ganze Zahl von Seiten füllen, dieser Verschnitt wird als interne Fragmentierung bezeichnet)
  - besteht ein Programm aus mehreren Phasen, die z.B. jeweils 4 KB benötigen, dann wird bei zu großen Seiten zu viel Speicher als belegt markiert, obwohl er leer ist
  - andererseits belegen Programme bei kleineren Seiten mehr von diesen, was zu einer längeren Seitentabelle führt
  - bei einigen Rechnern werden beim Prozesswechsel die Seitentabellen in einem Satz geladen, je kleiner die Seiten, desto mehr Zeit wird benötigt

7.4 getrennte Programm- und Datenbereiche

- die meisten Rechner haben einen einzigen Adressraum, der Programme und Daten enthält (Abbildung Adressräume (a))
- dies führt jedoch zu Unannehmlichkeiten, wenn der Adressraum nicht groß genug ist
- eine Lösung besteht in der Trennung des Befehlsraumes von dem Datenraum (Abbildung Adressräume (b))
- dadurch können die Paging Mechanismen der beiden Adressräume unabhängig voneinander operieren, es entstehen keine Komplikationen

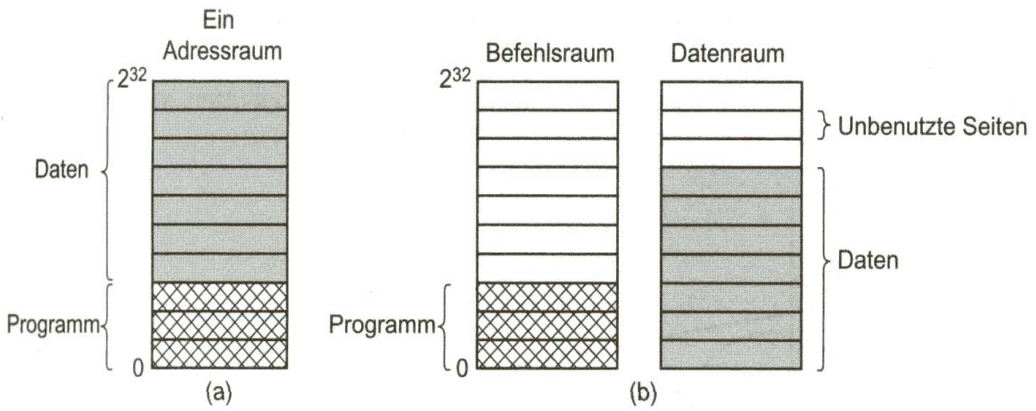

Abbildung 18: Adressräume

7.5 Gemeinsame Seiten

- bei Mehrbenutzersystemen kommt es vor, dass Benutzer zur selben Zeit auf das gleiche Programm zugreifen
- Problem: nicht alle Seiten können gemeinsam genutzt werden, was ist wenn z.B. 2 Prozesse gleichzeitig versuchen auf eine Seite zu schreiben?
- es können nur die Seiten gemeinsam genutzt werden, auf die nur lesend zugegriffen wird, z.B. Quellcodedaten
- sobald ein Prozess jedoch schreibend auf eine Seite zugreifen will, so wird die Seite kopiert und jeder Prozess bekommt seine eigene Kopie $\Rightarrow$ **copy-on-write**

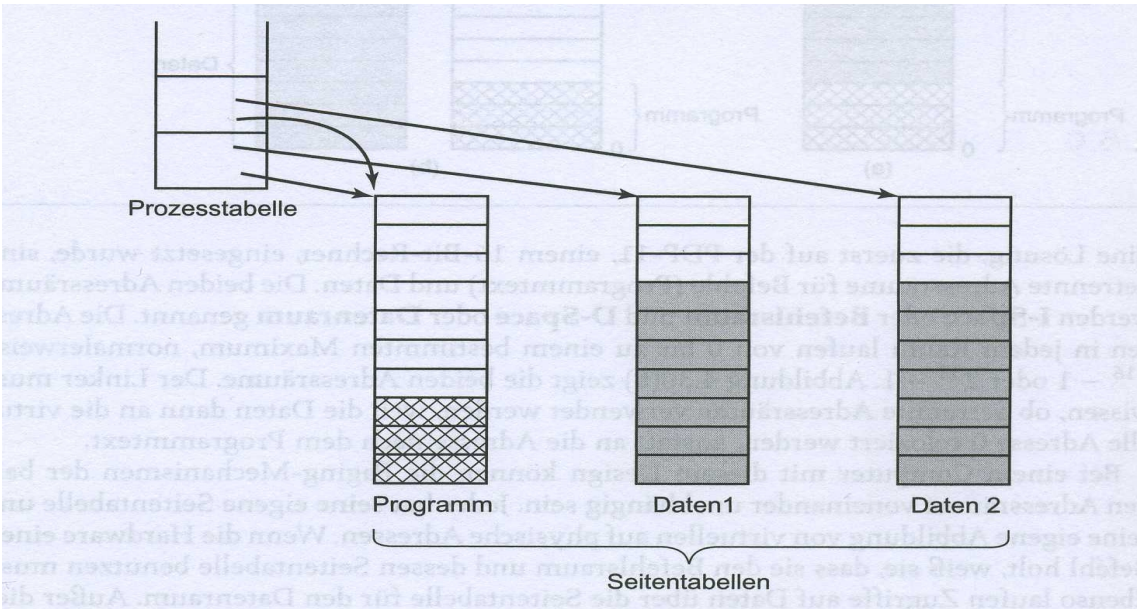

Abbildung 19: Prozesstabelle

7.6 Freigabe Strategien

- Paging funktioniert am besten, wenn jederzeit genügend freie Seitenrahmen zur Verfügung stehen
- um dies sicherzustellen, existiert ein **Paging-Daemon**, dieser prüft in regelmäßigen Abständen den Speicher
- falls nicht genügend Seitenrahmen frei sind, dann werden nach einem Seitenersetzungsalgorithmus Seiten ausgelagert
- die Implementierung benutzt eine Uhr mit 2 Zeigern
- wenn der 1. Zeiger auf eine saubere Seite zeigt, rückt der Zeiger auf die nächste Seite, zeigt er auf eine modifizierte Seite, dann wird der Inhalt auf die Platte geschrieben und der Zeiger rückt weiter vor
- der 2. Zeiger wird wie beim Clock-Algorithmus für die Seitenersetzung verwendet

7.7 Schnittstelle zum virtuellem Speicher

- bisher sind wir davon ausgegangen, dass der Programmierer keine Kontrolle über die Speicherverwaltung hat, bei fortgeschrittenen Systemen ist dies nicht immer der Fall
- dadurch enorme Verbesserung der Programme möglich
- dem Programmierer ist weiterhin möglich, mehrere Prozesse den selben Speicher verwenden zu lassen
- über diesen Speicher könnten die Programme dann mit hoher Bandbreite Daten austauschen
- Beispiel: Hochleistungs-Nachrichtenaustauschsystem
  - normalerweise werden die Daten von einem Adressraum in den anderen kopiert
  - hätten Prozesse jedoch Kontrolle über ihre Speicherabbildungen, könnten Nachrichten ausgetauscht werden indem der Sender die Seiten mit der Nachricht aus seinem Adressraum ausblendet und der Empfänger sie einblendet
  - anstatt der Daten werden nur die Namen der Seiten kopiert $\Rightarrow$ wesentlich schneller und effizienter

8 Implementierung

Bei der Implementierung eines Systems mit virtuellem Speicher ist zuvor eine Auswahl unter den verschiedenen Algorithmen zu treffen.

8.1 Betriebssystemaufgabe beim Paging

- das Betriebssystem wird im Allgemeinen zu 4 Zeitpunkten des Lebenszyklusses eines Prozesses gebraucht
  - Erzeugung des Prozesses (Erzeugung einer Seitentabelle; Speicherzuteilung und Initialisierung der Tabelle; Reservierung des Platzes für ausgelagerte Daten; Festlegung des Swapping-Bereiches)
  - Ausführung des Prozesses (Einstellung der MMU auf den neuen Prozess; Leerung des TLB; aktuelle Seitentabelle auf die des Prozesses setzen)
  - bei einem Seitenfehler (Hardwareregister auslesen, um festzustellen, welche virtuelle Adresse den Fehler erzeugte; benötigte Seite einlesen; Programmzähler auf den Befehl zurücksetzen, der den Seitenfehler verursacht hat)
  - Terminierung des Prozesses (Freigabe von Seitentabelle, Seitenrahmen, Plattenplatz für ausgelagerte Seite)

8.2 Behandlung von Seitenfehlern

Was passiert genau bei Seitenfehlern?

1. Hardware schreibt Programmzähler auf Stack und löst einen Sprung im Kernel aus
2. Assembler-Routine wird gestartet, die Mehrzweckregister und andere flüchtige Informationen speichert
3. Betriebssystem stellt fest, dass ein Seitenfehler erzeugt wurde und versucht herauszufinden, welche virtuelle Adresse gebraucht wird
4. sobald die virtuelle Adresse bekannt ist, wird überprüft, ob diese gültig ist und nicht gegen Schutzvorschriften verstößt
5. falls alles in Ordnung, Suche nach einem freien Seitenrahmen, ansonsten Terminierung des Prozesses
6. falls es keine freien Seitenrahmen gibt, wird der Seitenersetzungsalgorithmus gestartet
7. wenn der gewählte Seitenrahmen modifiziert wurde, wird die Seite auf die Platte geschrieben
8. Kontextwechsel, Prozess wird suspendiert und ein anderer Prozess laufen gelassen bis die Seite zurückgeschrieben wurde
9. sobald der Seitenrahmen sauber ist, wird die benötigte Seite geladen
10. Anpassung der Seitentabelle, virtuelle Seite wird auf den Seitenrahmen abgebildet

11. auslösender Befehl wird in Anfangszustand versetzt und Programmzähler auf Befehl gesetzt
12. Prozess wird neu ausgeführt

8.3 Sicherung des unterbrochenen Befehls

- bei einem auftretenden Fehler wird der Befehl auf 'halbem Wege' gestoppt und es kommt zu einem Sprung ins BS, der Befehl muss aber erhalten bleiben
- zur Lösung sind in den meisten CPU's interne Register eingebaut, in die der Programmzähler jedes mal vor Befehlsausführung reinkopiert wird

8.4 Sperren von Seiten im Speicher

- angenommen ein Prozess läuft, um Daten von einem Gerät oder einer Datei in einem Puffer in den Adressraum einzulesen, dieser wartet auf das Ende von E/A
- ein anderer Prozess bekommt währenddessen die CPU und produziert einen Seitenfehler, nun kann es passieren, dass die Seite, die den Puffer enthält, zur Auslagerung ausgewählt wird
- um dieses Problem zu lösen, muss die Seite gesperrt werden $\Rightarrow$ **Pinning**
- alternativ könnten auch E/A-Daten in einen Kernel-Puffer geschrieben werden und später in der richtigen Adressraum kopiert werden

8.5 Hintergrundspeicher

- Wohin werden die Daten auf die Festplatte geschrieben, wenn sie ausgelagert werden sollen
- die einfachste Art ist wohl die Einrichtung eines SWAP-Bereiches auf der Platte
- dabei ergibt sich aber das Problem, dass Prozesse wachsen können
-
- Beispiel: (Abbildung Hintergrundspeicher)
  (a) statischer SWAP-Bereich
    - Seitentabelle mit 8 Seiten
    - die Seiten 0,3,4,6 liegen im Speicher
    - die Seiten 1,2,5,7 auf der Platte
    - SWAP-Bereich ist genauso groß wie der virtuelle Adressraum
    - jede Seite hat eine feste Adresse auf der Platte

- Seiten im Speicher haben immer eine Hintergrundkopie, die jedoch nicht aktuell sein muss

(b) dynamischer Hintergrundspeicher

- Seiten haben keine feste Adresse auf der Platte
- wird eine Seite ausgelagert, sucht das BS eine leere Seite auf der Platte und trägt die Adresse in die Plattenzuordnungstabelle ein
- die Seiten im Speicher haben keine Festplattenkopie

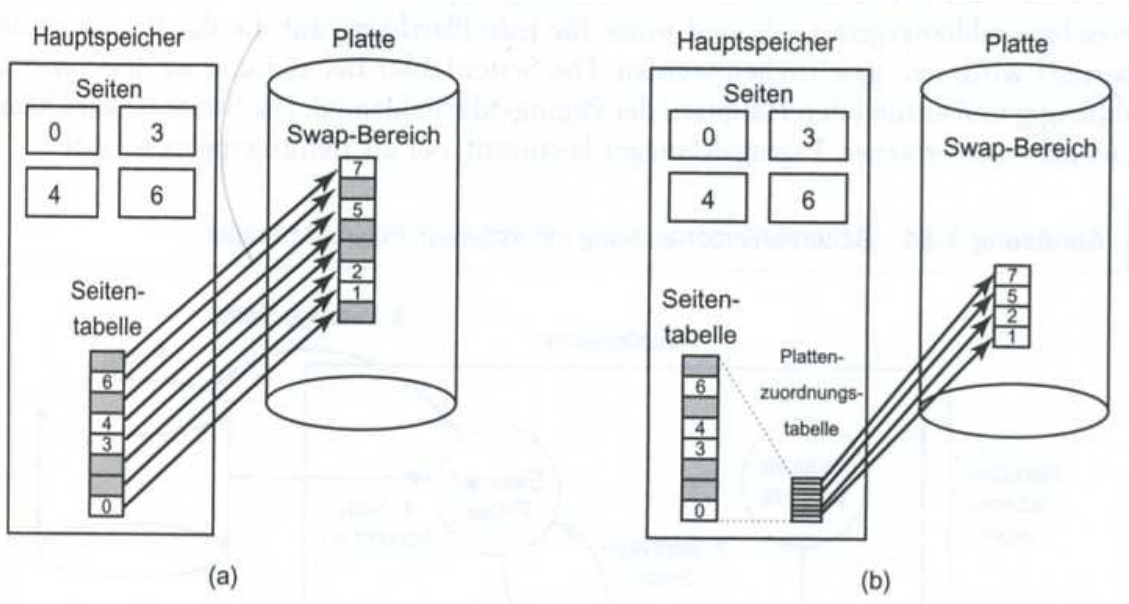

Abbildung 20: Hintergrundspeicher

8.6 Trennung von Strategien und Mechanismen

- um die Komplexität eines Systems zu verbessern, ist es sinnvoll Strategien und Mechanismen voneinander zu trennen
- Beispiel: (Abbildung Seitenfehlerbehebung)
  - die Speicherverwaltung besteht aus 3 Teilen
    1. ein Low-Level MMU-Handler
    2. eine Seitenfehler-Bahandlungsroutine im Kernel
    3. ein externer Paging-Manager im Benutzermodus
  - wird ein neuer Prozess gestartet, wird der externe Paging-Manager benachrichtigt um die Seitentabelle zu initialisieren und wenn nötig Hintergrundspeicher zu reservieren
  - werden neue Speicherobjekte in den Adressraum eingeblendet, wird wieder der externe Paging-Manager benachrichtigt
  - kommt es zu Seitenfehlern findet die Behandlungsroutine heraus, welche Seite benötigt wird und schickt eine Nachricht an den externen Paging-Manager

  - dieser liest die Seite und kopiert sie in seinen Adressraum
  - die Behandlungsroutine holt sich die Seite aus dem Adressraum, ruft den MMU-Handler auf
  - dieser blendet die Seite an die richtige Stelle im Adressraum des Benutzerprozesses ein

- Vorteil: größere Modularität, Flexibilität
- Nachteil: Leistungsverlust durch Kontextwechsel zwischen Kernel- und Benutzermodus

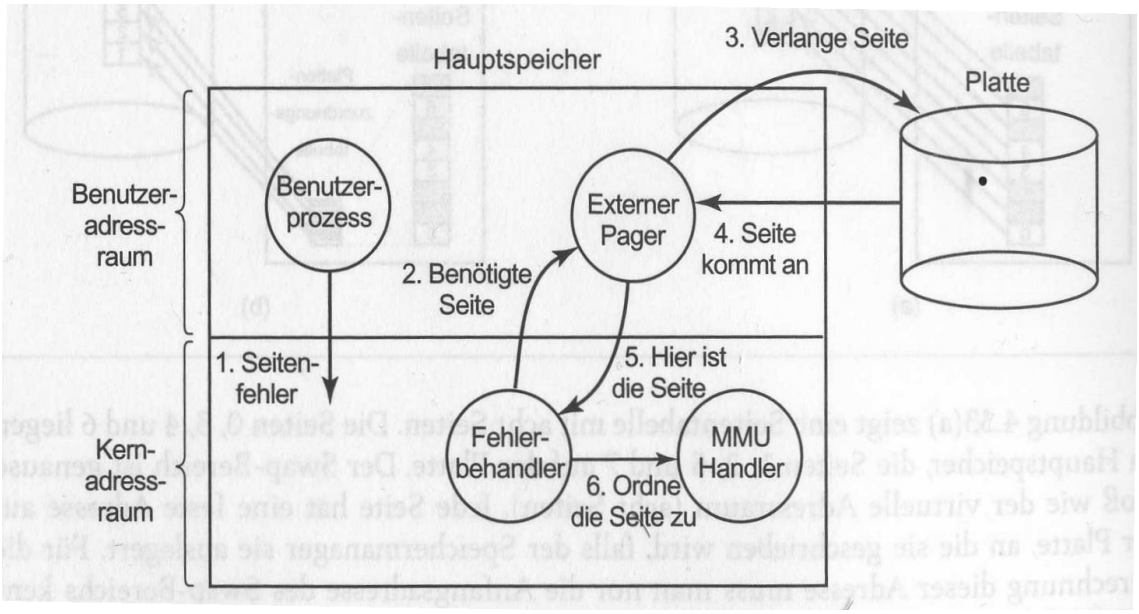

Abbildung 21: Seitenfehlerbehandlung

9 Segmentierung

- der bisher behandelte Speicher war eindimensional
- bei einer Vielzahl von Problemen wäre es aber besser, mehrere Adressräume zu haben
- Ziel ist es dabei, dem Programmierer nicht mit der Verwaltung von wachsenden und schrumpfenden Tabellen zu belasten
- eine Lösung besteht darin, die Maschine mit vielen voneinander unabhängigen Adressräumen auszustatten $\Rightarrow$ **Segmente**
- jedes Segment besteht aus einer linearen Folge von Adressen, startend bei 0 bis hin zu irgendein Maximum
- die verschiedenen Segmente können unterschiedlich groß sein und die Größe eines Segmentes kann sich dynamisch während der Ausführung verändern
- der Adressraum kann beliebig vergrößert werden, weil nichts in dem Adressraum ist, mit dem dieser kollidieren könnte
- weiterhin vereinfacht sich das Verlinken, da jede Prozedur eines Programms auf der Adresse 0 des jeweiligen Segments liegt, beim späterem Linken ist auch keine Veränderung der Anfangsadressen notwendig
- außerdem vereinfacht sich die gemeinsame Nutzung von Programmcode oder Daten durch mehrere Prozesse, z.B. Shared Libraries
- jedes Segment bildet dabei eine logische Einheit, d.h. verschiedene Segmente können unterschiedlich gehandhabt werden

9.1 Implementierung reiner Segmentierung

- der Hauptunterschied zwischen Paging und Segmentierung ist der, dass Seiten eine feste Größe haben, Segmente nicht
- Problem bei der Segmentierung ist die **externe Fragmentierung**
- dabei entstehen durch Löschung einzelner Segmente Löcher im Speicher, dieses Phänomen nimmt im Zeitverlauf zu $\Rightarrow$ Verdichtung wird notwendig
- Beispiel: (Abbildung externe Fragmentierung)
  (a) Segmente 0 bis 4 füllen den Speicher komplett aus
  (b) Segment 1 wird entfernt und das kleinere Segment 7 tritt an seine Stelle $\Rightarrow$ Lücke entsteht
  (c) Segment 4 wird entfernt und das kleinere Segment 5 tritt an seine Stelle $\Rightarrow$ Lücke entsteht
  (d) Segment 3 wird entfernt und das kleiner Segment 6 tritt an seine Stelle $\Rightarrow$ Lücke entsteht
  (e) Behebung der einzelnen Lücken durch Speicherverdichtung

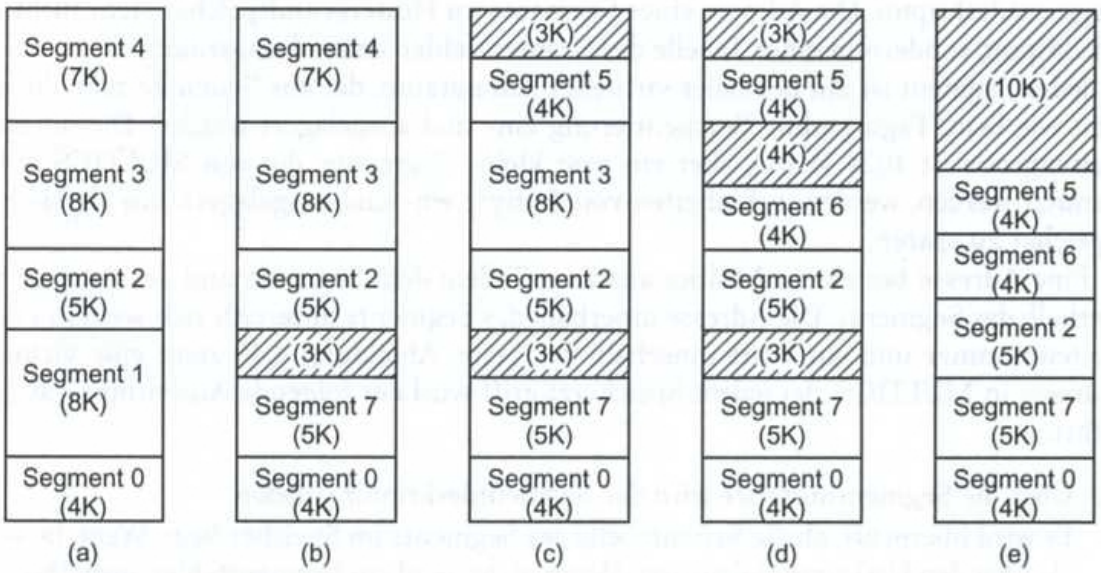

Abbildung 22: externe Fragmentierung

10 Übungsaufgaben

10.1 Aufgaben

1. Ein System hat genug Platz für 4 Programme im Hauptspeicher. Diese Programme warten jeweils die Hälfte der Zeit auf E/A. Welcher Anteil der CPU wird verschwendet?

2. Angenommen 2 Aufträge, die jeweils 10 Minuten CPU-Zeit benötigen, starten gleichzeitig. Wenn jeder Auftrag 50 Prozent der Zeit auf E/A wartet, wie lange dauert es, bis der letzte Prozess beendet ist? Wie lange dauert es, wenn sie parallel laufen? Die E/A-Operationen laufen nict parallel.

3. Ein Swapping-System entfernt Löcher aus dem Speicher durch Verdichtung. Wenn viele Löcher und Segmente zufällig über den Speicher verteilt sind und das Lesen oder Schreiben eines 32-Bit-Wortes 10ns dauert, wie lange dauert es 128MB Speicher zu verdichten?

4. Ein Speicher hat eine Gesamtgröße von 128MB und wird in Einheiten zu n Byte aufgeteilt. Für verkettete Listen nehmen wir an, dass der Speicher abwechselnd aus 64KB großen Löchern und Segmenten besteht. Außerdem wird angenommen, dass jeder Knoten 32 Bit für die Adresse, 16 Bit für die Länge und 16 Bit für den Zeiger auf den nächsten Knoten benötigt. Wieviel Speicher ist jeweils für die Verwaltung mittels Bitmaps und die Verwaltung mittels verketteter Listen erforderlich?

5. In einem Swapping-System sind folgende Löcher im Speicher nach aufsteigenden Adressen geordnet: 10KB, 4KB, 20KB, 18KB, 7KB, 9KB, 12KB. Welche Löcher wählen First Fit, Best Fit, Worst Fit und Next Fit jeweils aus, wenn nacheinander Segmente von 12KB, 10KB und 9KB Größe angefordert werden?

6. Was ist der Unterschied zwischen einer virtuellen und physischen Adresse?

7. Ein Befehl dauert 10ns und die Behandlung eines Seitenfehlers noch einmal $\boldsymbol{n}$ Nanosekunden. Geben sie eine Formel für die effektive Dauer eines Befehls an, falls ein Seitenfehler alle $\boldsymbol{k}$ Befehle auftritt.

8. Ein Rechner verfügt über einen 32Bit Adressraum und 8KB-Seiten. Die Seitentabelle liegt komplett in der Hardware, mit einem 32-Bit-Wort pro Eintrag. Wenn ein Prozess startet, wird die Seitentabelle aus dem Hauptspeicher in die Hardware kopiert, was für jedes Wort 100ns dauert. Wir groß ist der Anteil der Zeit, in der die CPU Seitentabellen lädt, wenn ein Prozess immer für 100ms läuft (eingeschlossen der Zeit, in der die Seitentabelle geladen wird)?

9. Ein Computer mit 32-Bit-Adressen benutzt eine zweistufige Seitentabelle. Virtuelle Adressen werden in ein 9-Bit-Feld für die oberste Seitentabelle, 11 Bit für die zweite Seitentabelle und einen Offset unterteilt. Wie viele Seiten sind im Adressraum und wie groß sind sie?

10. Eine 32-Bit virtuelle Adresse wird in 4 Teile a, b, c, d aufgeteilt. Die ersten 3 Felder sind die Indizes für eine dreistufige Seitentabelle. Das vierte Feld, d, ist der Offset. Hängt die Anzahl der Seiten von der Länge aller vier Felder ab? Wenn nein, welche Felder sind unwichtig?

11. Eine Festplatte hat eine durchschnittliche Such- und Rotationszeit von jeweils 10 ms und eine Spurgröße von 32 KB. Wie lange dauert es, ein 64 KB-Programm zu laden, bei einer Seitengröße von

    (a) 2KB

    (b) 4KB

    Die Seiten sind zufällig über die Platte verstreut und die Anzahl der Zylinder ist so groß, dass die Wahrscheinlichkeit, dass 2 Seiten auf demselben Zylinder liegen, vernachlässigbar ist.

12. Ein Computer hat 4 Seitenrahmen. Die Tabelle zeigt für jede Seite die Ladezeit, die Zeit des letzten Zugriffs sowie die R- und M-Bits. Die Zeiten sind jeweils in Uhrintervallen angegeben. Für die Zeit des Zugriffs ist der jeweilige Zeitstempel gesetzt.

Seite	Geladen	Zugriff	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

    Welche Seite wird ersetzt bei

    (a) NRU?

    (b) FIFO?

    (c) LRU?

    (d) Second-Chance?

13. Ein Computer stellt jedem Prozess einen Adressraum von 64 KB zur Verfügung, aufgeteilt in 4 KB-Seiten. Ein Programm hat 32768 Byte Programmcode, 16386 Byte Daten und 15870 Byte Stack. Passt dieses Programm in den Adressraum? Würde es passen, wenn die Seiten 512 Byte groß wären? Nicht vergessen, dass eine Seite nicht 2 Teile von verschiedenen Segmenten enthalten kann.

14. Was ist der Unterschied zwischen interner und externer Fragmentierung? Welcher der beiden kommt in Paging-Systemen vor und welche in Systemen mit reiner Segmentierung?

10.2 Lösungen

1. Es gilt die Formel: $A = 1 - p^n$. Für die CPU-Auslastung A folgt

$$A = 1 - 0,5^4 = 0,9375 = 93,75\%$$

   Es werden also $100 - 93,75 = 6,25\%$ verschwendet

2. a) sequentiell: die einzelnen Vorgänge werden nacheinander abgearbeitet, dafür werden insgesamt 30 Minuten benötigt.($2 \cdot 10$ Minuten für die reine Rechenzeit + $2 \cdot 5$ Minuten für die E/A)

   b) parallel: 20 Minuten (10 Minuten für die reine Rechenzeit (angenommen, die CPU-Kapazität reicht für beide Prozesse aus) + $2 \cdot 5$ Minuten für die E/A)

3. Der Speicher ist insgesamt 128 MB $= 128\text{Byte} \cdot 1024^2 = 134217728\text{Byte} = 1073741824\text{Bits}$. Die Umrechnung von Nanosekunde in Sekunde beträgt 10^{-9} . Für die jeweilige E/A-Ausgabe - Operation werden jeweils 10 Nanosekunden benötigt. Also $1073741824 \cdot 10 \cdot 10^{-9}\text{sec}/32 = 2,98\text{sec}$

4. 128 MB sind $128 \cdot 1024^2 = 134217728\text{Byte} = x$ Eine Einheit ist n Byte groß. Dass heißt, also $z = \frac{x}{n}$. Eine Bitmap teilt jeder Allokationseinheit einen Bit zu, benötigt also z Bits. Eine verkettete Liste braucht insgesamt $(32 + 16 + 16)$ Bits pro Allokationseinheit, also insdgesamt $64 \cdot z$ Bits.

5. - First Fit belegt die Blöcke 10KB, 20KB, 18KB
   - Best Fit belegt die Blöcke 10KB, 9KB, 12KB
   - Worst Fir belegt die Blöcke 20KB, 18KB, 12KB
   - Next Fir belegt die Blöcke 20KB, 18Kb, 9KB

6. - Der virtuelle Speicher befindet sich auf der Festplatte, währenddessen sich der physische Speicher im Arbeitsspeicher abgebildet ist. Weiterhin zeigt der virtuelle Speicher auf den physischen Speicher
   -

7. $10ns + \frac{n}{k}$

8.

9. a) $2 \cdot 20 \cdot 32 = 640\,\text{ms}$

   b) $2 \cdot 10 \cdot 16 = 320\,\text{ms}$

10. Die erste Seitentabelle ist $2^9 = 512$ Einträge groß. Die Seitentabellen der zweiten Ebene können jeweils 2^11 Einträge haben. Der Offset ist $2^{32-11-9} = 2^{12} = 4\,\text{KB}$ groß. Es ergeben sich also 2^{20} 4KB große Seiten.

11. Für die Anzahl der Seiten sind die Felder a,b,c von nöten. Dann errechnet sich die Anzahl der Seiten durch $2^{|a|} \cdot 2^{|b|} \cdot 2^{|c|}$. Die Länge der Seiten durch $2^{|d|}$ Byte. Wobei $|x|$ für die Länge der einzelnen Felder steht.

12. a) NRU lagert 2 aus, weil das R- und M-Bit auf 0.

    b) FIFO lagert 3 aus, weil die Zeit des letzten Ladens am weitesten zurück liegt

c) LRU lagert 1 aus, weil die Zeit des letzten Zugriffs am weitesten zurück liegt

d) Second-Chance lagert 2 aus, weil zuerst geht der Zeiger auf Seite 3, diese wurde referenziert, das R-Bit wird also auf 0 gesetzt und dies Seite an das Ende der Liste gepackt, das gleiche mit Seite 0, die als nächstes in der Liste kommt. Bei der nächsten Seite der Liste 2, ist das R-Bit nicht referenziert, also wird Seite 2 ausgelagert.

13. Der Programmcode benötigt genau 32768 Byte= 32KB. 16386 Byte = 16,00195 KB für die Daten und 15870 Byte = 15,498 KB für den Stack. Bei einer Seitengröße von 4 KB ergibt sich insgesamt (32 + 20 +16) KB Speicherbedarf, es pass also nicht. Bei einer Seitengröße von 4 KB ergibt sich insgesamt (32 + 16,5 +15,5) KB Speicherbedarf, es passt also.

14. a) interne Fragmentierung (Ist der Speicher in verschieden große Allokationseinheiten aufgeteilt mit einer Größe von n Byte und benötigt ein Prozess nur z Byte mit $n \neq z$, dann wird eine Allokationseinheit nicht vollständig ausgefüllt, aber trotzdem als belegt markiert.

    b) externe Fragmentierung (durch Löschung einzelner Segmente entstehen im Speicher Löcher, weil die nachfolgenden Segmente nicht die gleiche Größe haben, wie die vorhergehenden und dadurch Speicherplatz offen bleibt, dieses Problem nimmt im Zeitverlauf zu)

11 Begriffsübersicht

Archivspeicher	Speicher, in dem Daten systematisch abgelegt und extern verwaltet werden
Assembler	ein Computerprogramm, das eine Assemblersprache in Maschinensprache übersetzt
Assemblersprache	eine spezielle Programmiersprache, welche die Maschinensprache einer spezifischen Prozessorarchitektur in einer für den Menschen lesbaren Form repräsentiert
Bitmap	im Allgemeinen eine Folge von voneinander unabhängigen Bits, die jeweils die gleiche Bedeutung haben
Cache	schneller Puffer Speicher
Compiler	ist ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm - genannt Quellprogramm - in ein semantisch äquivalentes Programm einer Zielsprache (Zielprogramm) umwandelt
CPU	Central Processing Unit, Prozessor
E/A	Eingabe / Ausgabe (Operation)
Embedded Systems	elektronischer Rechner, der in ein technischen Kontext eingebunden ist
interne Fragmentierung	Verschnitt bei Nichtausfüllung einer Seite
Hash	Tabelle mit spezieller Indexstruktur
Hauptspeicher	ist der Speicher eines Computers, in dem Datenobjekte, also Programme und die von diesen in Mikroprozessoren zu verarbeitenden Nutzdaten, abgelegt und zu einem späteren Zeitpunkt (unverändert) abgerufen werden können
IBM Mainframes	Großrechner von IBM
Kernel	zentraler Bestandteil des BS, Organisation der Prozesse und Daten
MMU	Memory Management Unit, verwaltet die Zuteilung/Adressierung von virtueller Adresse auf physischer Adresse
MMU-Handler	Steuerungsprogramm für MMU
Offset	ganzzahliger Wert, der zu einer Adresse addiert wird; legt die Größe der einzelnen Seiten fest
OS/360	Betriebssystem mit Stapelverarbeitungsfähigkeit
Overhead	Daten, die nicht primär zu den Nutzdaten zählen, sondern als Zusatzinformation zur Übermittlung oder Speicherung benötigt werden
Paging	Methode der Hauptspeicherverwaltung, bei der die MMU zur Verwaltung eingesetzt wird
Palmtops	kompakter, tragbarer Computer, der neben vielen anderen Programmen hauptsächlich für die persönliche Kalender-, Adress- und Aufgabenverwaltung benutzt wird
physisches Speicherwort	Information über Ort der Speicheradresse
Plattenspeicher	Speicher auf der Festplatte
PPF	
Referenzkette	geordnete Liste von Seitennummern
Register	eine nach Funktion zusammengefasste Gruppe von einzelnen Speicherzellen
Scheduler	regelt die zeitliche Ausführung mehrerer Prozesse in Betriebssystemen
Seite	Einheiten des virtuellen Adresraumes
Seitenrahmen	den Seiten zugeordneten Einheiten im physischen Speicher
Speicherbus	dient zur Übertragung von Daten zwischen CPU und Speicher

Speicherverwaltung	Teil des Betriebssystems, der die Speicherhierarchie verwaltet
Stack	Stapelspeicher oder Kellerspeicher
Swapping	das Schreiben oder Lesen von Daten, die sich im schnellen, aber kleinen Hauptspeicher des Computers befinden, auf den langsamen, aber großen Hintergrundspeicher (wie z.B. eine Festplatte) und umgekehrt
SWAP-Bereich	Bereich auf der Festplatte, der zur Auslagerung von Seiten zur Verügung ste
τ	bestimmte Zeitspanne, die den Arbeitsbereich vorgibt
virtuelle Zeit	ist die CPU-Zeit, die ein Prozess seit seinem Start benutzt hat
virtueller Speicher	ist der vom tatsächlich vorhandenen Arbeitsspeicher unabhängige Adressraum, der einem Prozess für Daten und das Programm vom Betriebssystem zur Verfügung gestellt wird

12 Quellen

- Andrew S. Tanenbaum, Moderne Betriebssysteme, Pearson Education Deutschland GmbH 2002
-
- Wikipedia