

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 1

BETRIEBSSYSTEME SPEICHERVERWALTUNG

Prof. Dr. Jörg Mielebacher
mail@mielebacher.de

Die folgenden Folien führen in die Verwaltung des Hauptspeichers durch das Betriebssystem ein. Zu jeder Folie sind Notizenseiten erfasst.

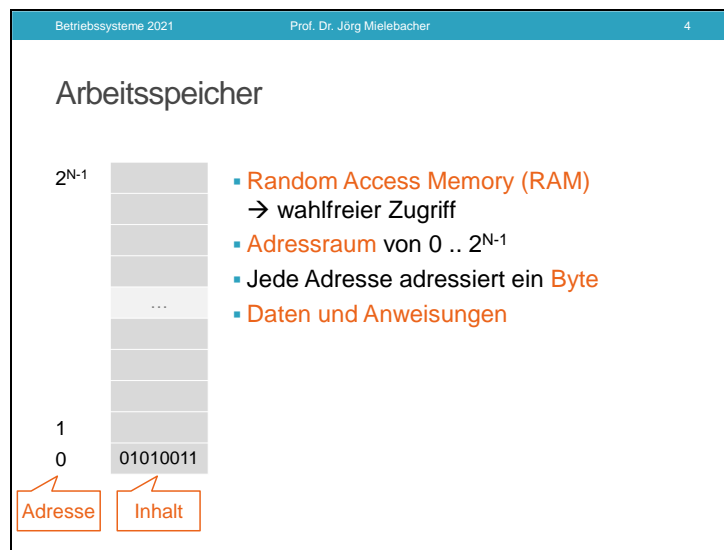
Verbesserungsvorschläge und Fehlerhinweise können Sie gerne an die Adresse mail@mielebacher.de senden.

Rechtliche Hinweise: Die Rechte an geschützten Marken liegen bei den jeweiligen Markeninhabern. Alle Rechte an diesen Folien, Notizen und sonstigen Materialien liegen bei ihrem Autor, Jörg Mielebacher. Jede Form der teilweisen oder vollständigen Weitergabe, Speicherung auf Servern oder Nutzung in Lehrveranstaltungen, die nicht von dem Autor selbst durchgeführt werden, erfordert seine schriftliche Zustimmung. Eine schriftliche Zustimmung ist darüber hinaus für jede kommerzielle Nutzung erforderlich. Für inhaltliche Fehler kann keine Haftung übernommen werden.

Wiederholung

- Prozesse sind Programme in Ausführung. Das Betriebssystem verwaltet sie anhand der PCBs in der Prozesstabelle.
- Prozesse unterscheiden sich in Häufigkeit und Dauer von CPU- und IO-Bursts.
- Die Umschaltung zwischen Prozessen erfordert den Wechsel des Kontextes. Diese Aufgabe übernimmt der Dispatcher.
- Der Scheduler teilt die Prozesse dem Prozessor zu; hierfür gibt es unterschiedliche Verfahren (z.B. FCFS, SJF, SRT oder RR).
- Threads sind Teil von Prozessen und erlauben die nebenläufige Ausführung auf mehreren Prozessoren.
- Systemaufrufe erlauben die Ausführen von Systemroutinen durch sog. Traps.
- IPC ist u.a. möglich durch Shared Memory, Message Queues, Pipes und Sockets.
- Das Betriebssystem stellt in Form von Systemaufrufen Techniken für die IPC zur Verfügung.

Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	3
<p data-bbox="467 586 911 636">Lokalität und Speicherhierarchie</p>		



Der Arbeitsspeicher erlaubt einen wahlfreien Zugriff, daher auch die englische Bezeichnung Random Access Memory (RAM). Wahlfrei bedeutet hierbei, dass man durch Angabe eines Adresse direkt den zugeordneten Speicherinhalt erreichen kann. Die Länge der Adressen N bestimmt dabei auch die Größe des sog. Adressraums, der die Adressen von 0 bis 2^N-1 umfasst, bei 32 bit also ca. 4 Mrd. Adressen. Unter jeder Adresse liegt ein Byte ab. Bei Von-Neumann-Rechnern enthält der Arbeitsspeicher sowohl Daten als auch Anweisungen.

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 5

Beispiel

```
int main()
{
    int w[ 5 ] = { 5, 1, 2, 4, 3 };
    int s = 0;

    for( int i = 0; i < 5; i++ )
        s += w[ i ];

    return 0;
}
```

...
4400 w[0]
4404 w[1]
4408 w[2]
4412 w[3]
4416 w[4]
4420 s
...

Das hier gezeigte Programm summiert die in einem Feld gespeicherten Werte. Die verwendeten Variablen liegen im Hauptspeicher. Der Vorteil einer Hochsprache wie C oder C++ besteht darin, dass sich Bereiche des Speichers unter einem symbolischen Namen ansprechen lassen. Bei Ausführung des Programms sind die Variablennamen jedoch bedeutungslos, da sie nicht Teil des Maschinencodes sind – im Maschinencode erfolgen nur noch Zugriffe anhand der Speicheradresse.

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 6

Beispiel

```
int main()
{
    int w[ 5 ] = { 5, 1, 2, 4, 3 };
    int s = 0;

    for( int i = 0; i < 5; i++ )
        s += w[ i ];
    return 0;
}
```

Räumlich benachbarter Zugriff

Wiederholter Zugriff

Zwei wichtige Beobachtungen werden uns im weiteren Verlauf immer wieder beschäftigen: Es fällt auf, dass manche Speicherbereiche (hier: die Variable *s*) immer wieder verwendet werden. Außerdem kann man bei dem Feld sehen, dass nach einem Zugriff auf ein Element, auch Zugriffe auf benachbarte Elemente folgen. Wiederholte Zugriffe und Zugriffe auf benachbarte Elemente sind typisch – sie treten insbesondere im Zusammenhang mit Feldern und Schleifen auf und betreffen nicht nur die verarbeiteten Daten, sondern auch die Anweisungen.

The slide is titled 'Definitionen' and is part of a presentation on 'Betriebssysteme 2021' by 'Prof. Dr. Jörg Mielebacher'. It features two blue rounded rectangular boxes stacked vertically. The top box contains the text 'Zeitliche Lokalität' and the bottom box contains the text 'Räumliche Lokalität'.

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 7

Definitionen

Zeitliche Lokalität

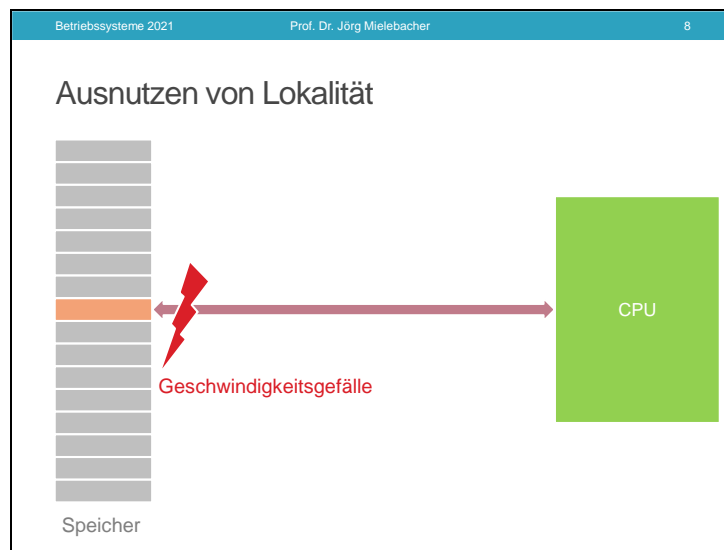
Räumliche Lokalität

Die beiden Beobachtungen führen uns auf zwei Arten von Lokalität:

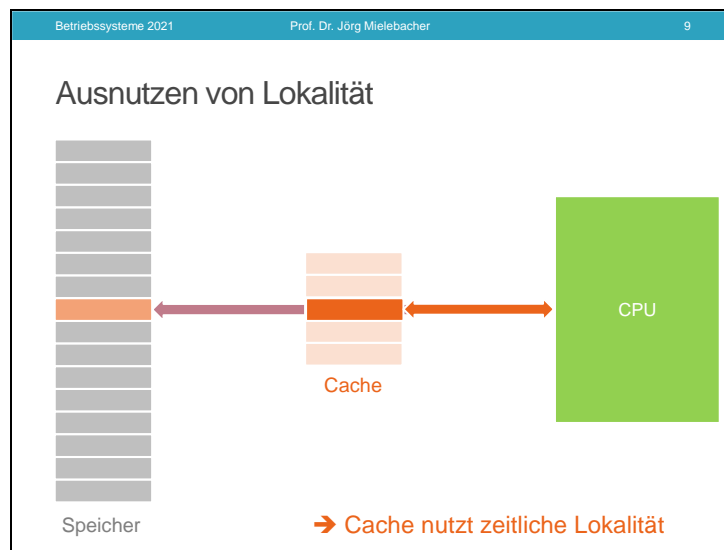
Zeitliche Lokalität bedeutet, dass nach einem Zugriff auf eine Speicheradresse meist weitere Zugriffe auf diese Adresse folgen, z.B. eine Variable initialisieren, Werte darin ablegen und auf diese Werte lesend zugreifen.

Räumliche Lokalität bedeutet, dass nach einem Zugriff auf eine Speicheradresse meist weitere Zugriffe auf benachbarte Adressen folgen, z.B. wenn ein Feld durchlaufen wird.

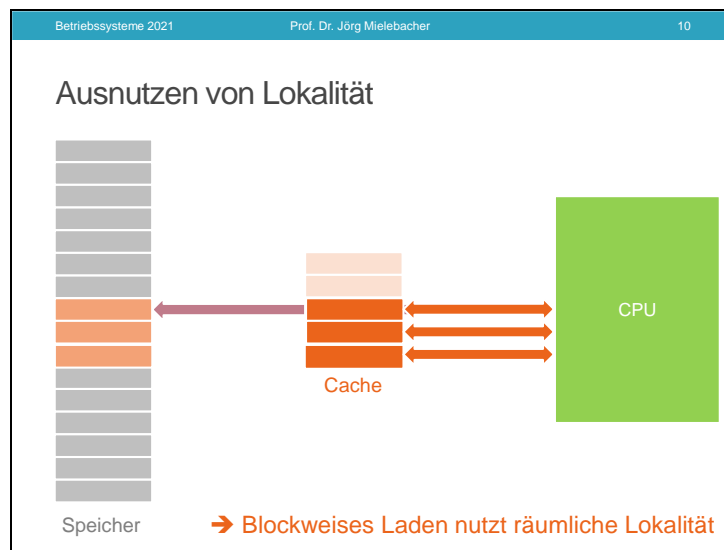
Zeitliche und räumliche Lokalität lässt sich auf unterschiedliche Ausnutzen, mehr dazu auf den folgenden Folien.



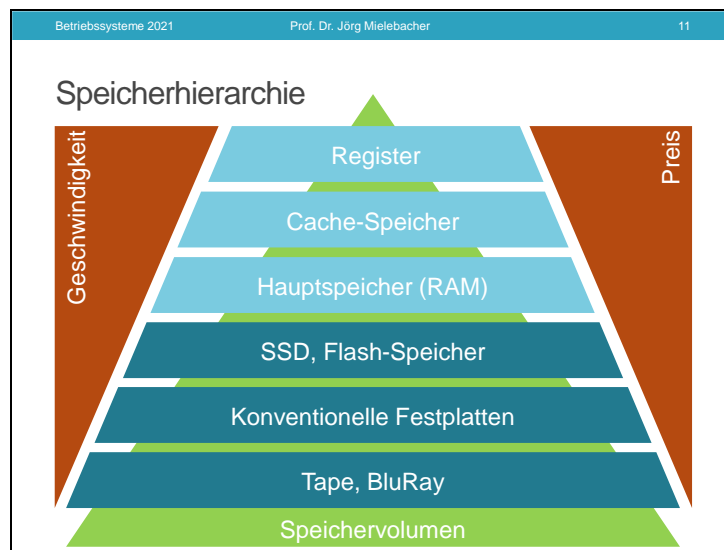
Ohne Ausnutzen der Lokalität würde der Prozessor jeden Wert direkt aus dem Speicher laden. Da jedoch der Prozessor deutlich schneller als der Speicher ist, liegt ein massives Geschwindigkeitsgefälle zwischen beiden vor, der Speicher wird zum Flaschenhals für den Prozessor. Die tatsächliche Leistung bleibt deutlich hinter den Möglichkeiten zurück.



Eine Möglichkeit ist die Nutzung eines sog. Cache-Speichers. Wird auf einen Wert im Speicher zugegriffen, wird dieser im Cache für nachfolgende Zugriffe abgelegt. Da Cache-Zugriffe deutlich schneller als Speicherzugriffe sind, wird das Geschwindigkeitsgefälle verringert. Der Prozessor könnte also bei wiederholten Zugriffen den Wert direkt aus dem Cache laden, ohne erneut auf den Speicher zugreifen zu müssen. Ein Cache nutzt folglich die zeitliche Lokalität.



In ähnlicher Weise kann man Vorgehen, um die räumliche Lokalität auszunutzen: Wird auf einen Wert zugegriffen, könnten auch die benachbarten Elemente in den Cache geladen werden, um nachfolgende Zugriffe auf diese Elemente zu beschleunigen. Das blockweise Laden nutzt somit die räumliche Lokalität. Dieses Vorgehen ist insbesondere auch bei Datenträgern wie Festplatten oder optischen Medien nützlich, bei denen das sequentielle Lesen besonders effizient ist.



Das Lokalitätsprinzip in Verbindung mit den Eigenschaften (speziell: Geschwindigkeit, Volumen und Preis) der verschiedenen Speichertechnologien führt auf die sog. Speicherhierarchie, die man in Computer nutzt: Auf den unteren Ebenen liegen die persistenten Speicher, wie Tape, BluRay usw., sowie konventionelle (also magnetische) Festplatten, SSD und andere Flash-Speicher. Auf ihnen bleiben die gespeicherten Daten auch nach dem Ausschalten erhalten. Zu den flüchtigen Speichern zählen der Hauptspeicher (Random Access Memory, RAM), die Cache-Speicher (meist L1, L2, L3) sowie die Prozessorregister. Von unten nach oben nimmt die Geschwindigkeit stark zu, jedoch auch der Preis pro Volumeneinheit. Umgekehrt nimmt von unten nach oben das Speichervolumen stark ab.

Teilweise unterscheidet man auch in Primärspeicher (Register, Cache, Hauptspeicher), Sekundärspeicher (SSD, Festplatten) und Tertiärspeicher (Optische Laufwerke, Tape).

Unser Fokus soll heute insbesondere auf den flüchtigen Speichern liegen, speziell auf dem sog. Hauptspeicher und wie er durch das Betriebssystem verwaltet wird.

Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	12
<h2>Aufgabe</h2> <ul style="list-style-type: none">Finden Sie typische Geschwindigkeiten und Volumina der einzelnen Speicher.		

Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	13
Aufgabe		
<300ps	Register	<1kB
<10ns	Cache-Speicher	64kB-16MB
10-100ns	Hauptspeicher (RAM)	1GB-1TB
0,1-1ms	SSD, Flash-Speicher	<2TB
<10ms	Konventionelle Festplatten	<20TB
ms - s	Tape, BluRay	Bis zu vielen TB

Die Speicherhierarchie umfasst Volumina von vielen TB bis zu einigen Bytes, auch die Zugriffszeiten bewegen sich von Sekunden bis zu Picosekunden, wobei man von Ebene zu Ebene eine Reduktion der Zugriffszeiten um Faktoren von 10-100 beobachtet, was den Nutzen der Speicherhierarchie verdeutlicht.

Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	14
<p data-bbox="469 586 1102 636">Speicherverwaltung durch das Betriebssystem</p>		

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 15

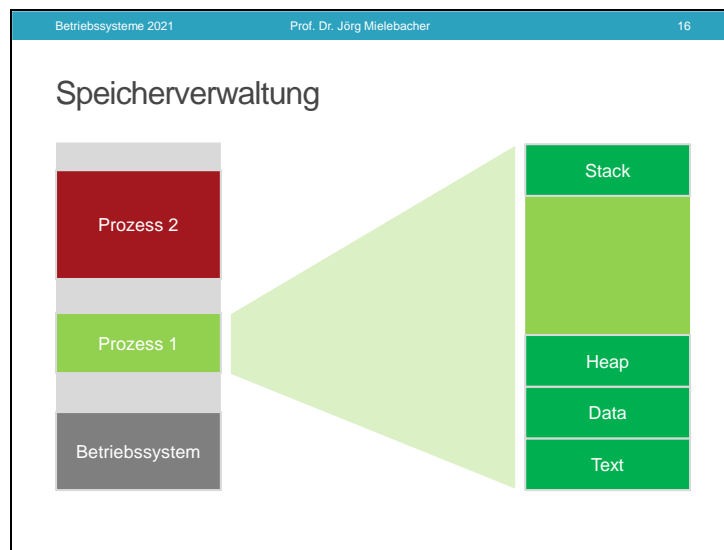
Aufgaben des Memory Managers

- Verteilen des Arbeitsspeichers an die Prozesse
- Belegung des Adressraums
- Optimieren des Zugriffs
- Schutz des Speichers

→ Ablage von Variablen usw. durch Compiler/Interpreter

Das Betriebssystem – genauer: der enthaltene Memory Manager - übernimmt zusammen mit der Memory Management Unit des Prozessors vielfältige Aufgaben in der Speicherverwaltung, vor allem die Verteilung des Arbeitsspeichers an die einzelnen Prozesse, somit auch wie der verfügbare Adressraum belegt wird. Darüber hinaus optimiert es den Zugriff auf den Arbeitsspeicher und schützt den Speicher, indem unberechtigte Zugriffe verhindert werden.

Wie die Daten eines Prozesses im Hauptspeicher abgelegt werden, legt nicht das Betriebssystem fest, sondern der Compiler bzw. Interpreter, wenn er die Daten im Daten- oder BSS-Segment arrangiert. Stack- und Heap-Segment ergeben sich zwangsläufig erst zur Laufzeit.



Wird ein Prozess gestartet – wird also das Programm in den Speicher geladen – umfasst der Speicherinhalt i.W. vier Teilsegmente:

Das Text-Segment enthält den Maschinencode des Programms.

Das Daten-Segment enthält die Daten, die bereits beim Start des Programms einen festgelegten Wert haben, also Konstanten sowie globale und statische Variablen. Lokale Variablen, die in einer Funktion angelegt werden, sind hier nicht abgelegt. Teilweise wird neben dem Data- auch noch das BSS-Segment (Block Started by Symbol) ausgewiesen, das alle globalen oder statischen, nicht initialisierten Variablen enthält.

Das Heap-Segment hat im Gegensatz zu Text und Data keine feste Größe, sondern kann während des Laufzeit des Prozesses wachsen. Das Heap-Segment enthält den dynamisch zur Laufzeit reservierten Speicher. In C erfolgt dies z.B. mit malloc, in C++ mit new – sie nutzen die Systemaufrufe brk und sbrk.

Das Stack-Segment wiederum wächst in Richtung des Heap-Segments. Es enthält den sog. Call Stack, d.h. immer wenn eine Funktion aufgerufen wird, werden die Informationen der Funktion, in der der Aufruf erfolgt, auf dem Stack abgelegt, vor allem Parameter, Rücksprungadresse und die lokalen Variablen. Beim Verlassen der Funktion wird der Stack-Eintrag wieder entfernt.



Betriebssysteme 2021

Prof. Dr. Jörg Mielebacher

18

Singletasking

- Betriebssystem und Prozess teilen sich Speicher
- Keine Aufteilung zwischen Prozessen notwendig


→ Heutzutage i.d.R. nicht verwendet

Der einfachste Fall der Speicherverwaltung ergibt sich beim sog. Singletasking, wenn also nur die Ausführung jeweils eines Prozesses neben dem Betriebssystem möglich ist. In diesem Fall teilen sich Betriebssystem und Prozess den vorhandenen Speicher, eine Aufteilung oder Isolation zwischen den Prozessen ist nicht notwendig. Dieser Fall ist sehr einfach, wird aber heutzutage i.d.R. nicht mehr verwendet.



Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 20

Feste Partitionen

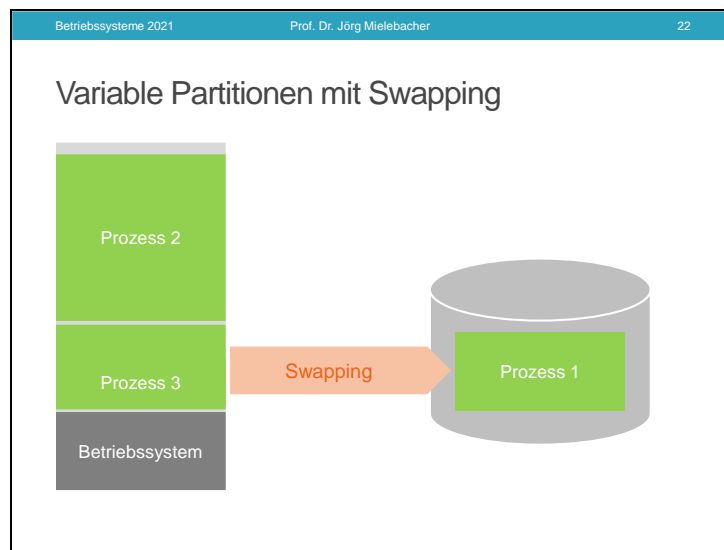


- Partitionen fester Größe
- Je Partition ein Prozess
- Auswahl der Partition passender Größe

→ Heutzutage i.d.R. nicht verwendet

Die Idee der festen Partitionierung beruht darauf, dass für den Computer eine zuvor manuell festgelegte Aufteilung in Partitionen fester Größe besteht. Diese Partitionen unterteilen den Hauptspeicher so, dass sich jeder Partition ein Prozess zuordnen lässt. Beim Eintreffen neuer Jobs wird anhand des Speicherbedarfs eine freie Partition ausgewählt. Diese Art der Speicherverwaltung wird heute i.d.R. nicht mehr verwendet.





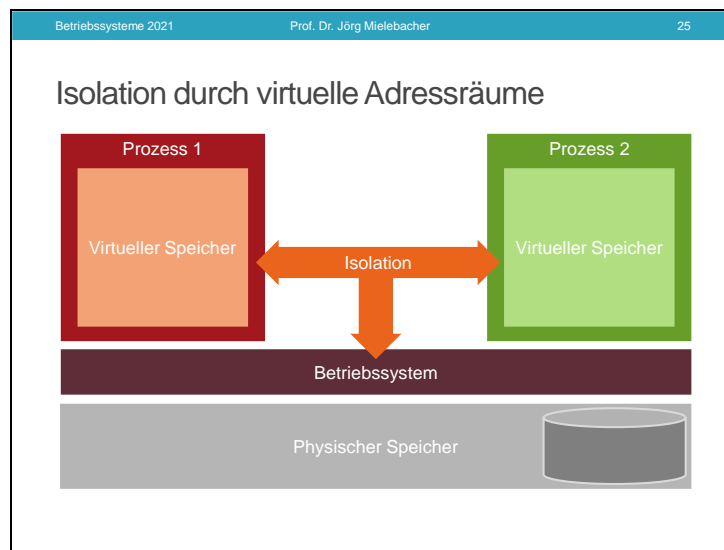
Mit dem Übergang zu Time-Sharing-Systemen stieß das Prinzip der Festen Partitionen an Grenzen, z.B. wurde deutlich, dass nicht alle Prozesse zur selben Zeit in den Hauptspeicher passen. Außerdem standen zunehmend Sekundärspeicher (Festplatten) zur Verfügung. Vor diesem Hintergrund entstand das Konzept der variablen Partitionen mit Swapping. Dabei erhält jeder Prozess eine Partition passender Größe. Inaktive Prozesse werden vorübergehend auf die Festplatte ausgelagert (Swapping). Der dadurch entstehende Platz kann durch den nächsten Prozess genutzt werden. Wird (wie hier Prozess 1) ein Prozess wieder aktiv, kann er von der Festplatte wieder in den Hauptspeicher geladen werden. Nachteil der variablen Partitionen ist das Auftreten von Lücken.



Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	24
<h3>Ausgangssituation</h3> <ul style="list-style-type: none">▪ Mehrere Prozesse benötigen Hauptspeicher.▪ Der Speicherbedarf eines oder aller Prozesse zusammen kann die Größe des Hauptspeichers übersteigen.▪ Prozesse müssen auch lauffähig sein, wenn sie teilweise ausgelagert sind.▪ Prozesse und Betriebssystem müssen voneinander getrennt sein.▪ Der tatsächliche Speicherort sollte für den Prozess bedeutungslos sein.▪ Gemeinsame Speicherbereiche sind notwendig.		

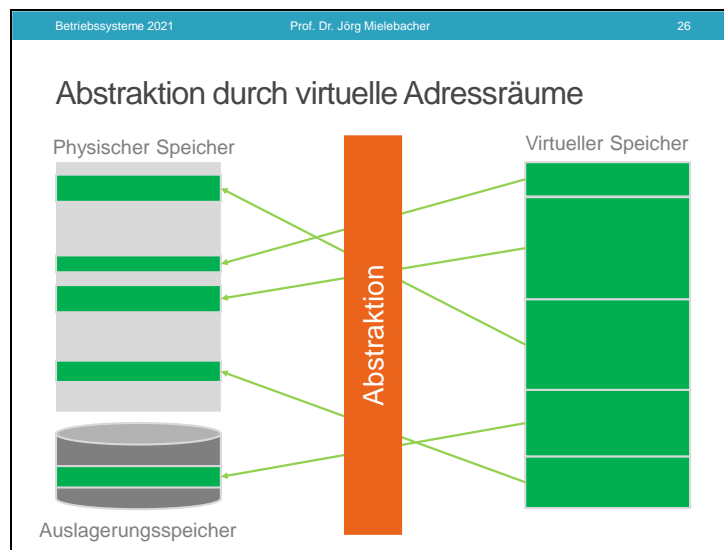
Die Idee des virtuellen Speichers, wie sie in heutigen Betriebssystemen umgesetzt wird, ist entstanden aus einer Reihe von Anforderungen: Natürlich müssen wir davon ausgehen, dass mehrere Prozesse zeitgleich ausgeführt werden, also zeitgleich Hauptspeicher benötigen. Auch ist davon auszugehen, dass der Speicherbedarf eines Prozesses (mindestens aber die Gesamtheit aller Prozesse) die Größe des Hauptspeichers übersteigt – Swapping ist also notwendig. Der Prozess muss aber auch dann lauffähig sein, wenn er teilweise ausgelagert ist.

Eine wichtige Aufgabe des Betriebssystems ist der Schutz des Speichers; Prozesse dürfen nicht gegenseitig ihre Speicherbereiche verändern, müssen also voneinander getrennt sein. Gleiches gilt für das Betriebssystem – andere Prozesse dürfen nicht die Speicherbereiche des Betriebssystems verändern. Wenn also Prozesse (bzw. ihr Speicher) teilweise ausgelagert sind, muss es für die Programmierung unerheblich sein, ob ausgelagert wurde und wo die Daten des Prozesses im physischen Speicher liegen. Außerdem werden für die Interprozesskommunikation und für gemeinsam genutzte Bibliotheken gemeinsame Speicherbereiche benötigt.



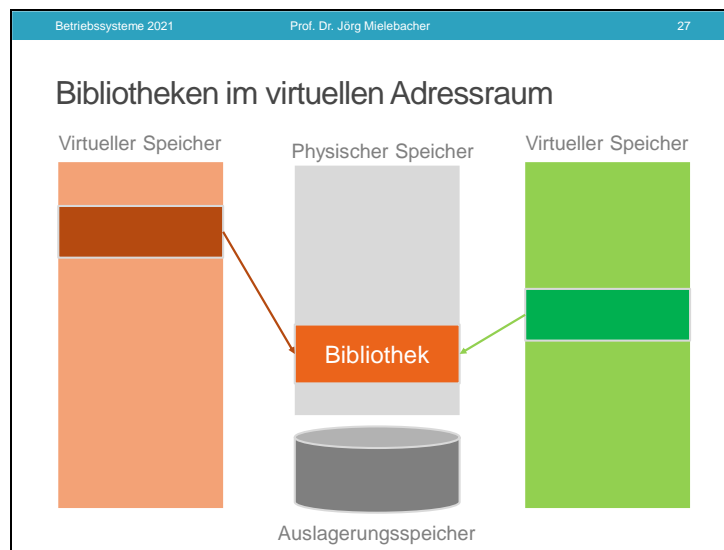
Heutzutage weisen Betriebssysteme i.d.R. jedem Prozess einen virtuellen Adressraum zu, d.h. der Prozess kann gar nicht direkt auf den physischen Hauptspeicher zugreifen, sondern sieht nur seinen eigenen Adressraum und greift auf diesen zu. Damit werden die Prozesse untereinander isoliert (horizontal), aber auch das von Betriebssystem von den Prozessen (vertikal).

Die Isolation von Prozessen ist auch eng verbunden mit dem Begriff des „Protected Mode“ (im Gegensatz zum sog. Real Mode), der in den 1980er Jahren eingeführt wurde.

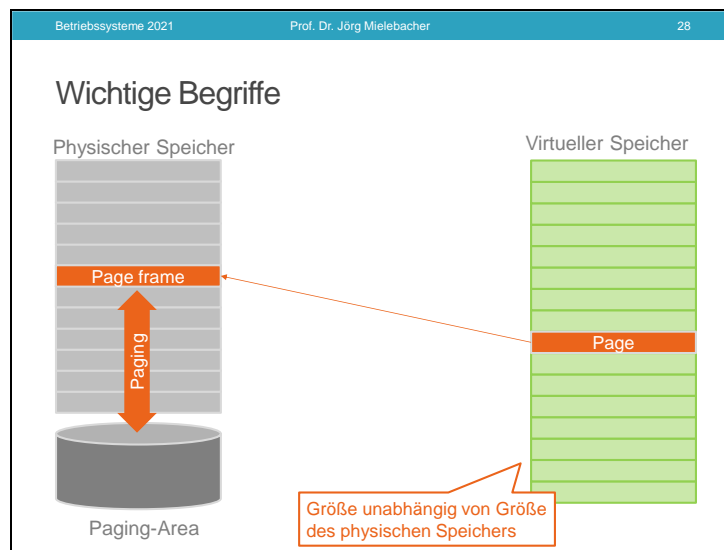


Der entscheidende Vorteil der virtuellen Speicherverwaltung ist der durchgängige, fragmentierungsfreie Adressraum, der dem Prozess zur Verfügung steht. Jeder Bereich des virtuellen Adressraums verweist dabei auf einen Bereich des physischen Speichers oder des Auslagerungsspeichers, wobei die Anordnung eine andere als im virtuellen Adressraum sein kann. Hierdurch kann einem Prozess weit mehr Speicher zur Verfügung gestellt werden, als physischer Speicher zur Verfügung steht. Lücken (d.h. Fragmentierung) des physischen Speichers, wie sie beim Swapping entstehen, sind bedeutungslos, da die einzelnen Bereiche des physischen Speichers in den durchgängigen virtuellen Adressraum abgebildet werden.

Vor allem wird wirkungsvoll von der physischen Speicherung abstrahiert – ob also Daten im Hauptspeicher liegen, wo sie liegen oder ob sie auf eine Festplatte ausgelagert sind, ist für den Prozess nicht erkennbar und auch irrelevant.



Durch die Abstraktion wird es auch möglich, mehrere Prozesse auf gemeinsame Speicherbereiche verweisen zu lassen – ein Gedanke, den wir neulich im Kontext der IPC als Shared Memory bezeichnet haben. Weit wichtiger ist dieses Vorgehen aber, wenn Bibliotheken (z.B. unter Windows DLLs) gemeinsam genutzt werden sollen. Wenn es sich um Bibliotheken handelt, die von mehreren Programmen benötigt werden, reicht es aus, sie einmal in den Speicher zu laden und dann die jeweiligen virtuellen Adressräume darauf verweisen zu lassen. In diesem Fall sind jedoch nur lesende Zugriffe auf den gemeinsamen Speicher erlaubt.

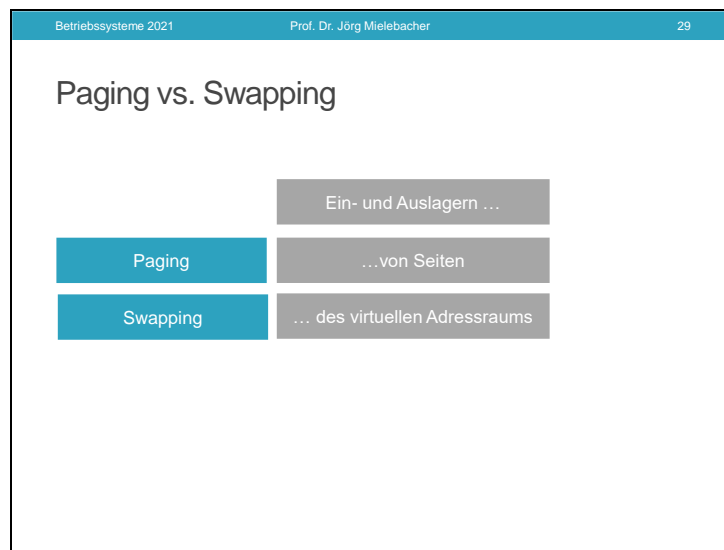


Der physische Speicher ist in gleich große Page frames – deutsch: Seitenrahmen – unterteilt. Der virtuelle Adressraum wiederum wird in gleich große Pages – deutsch: Seiten – unterteilt. Üblich sind z.B. Größen von 4kB.

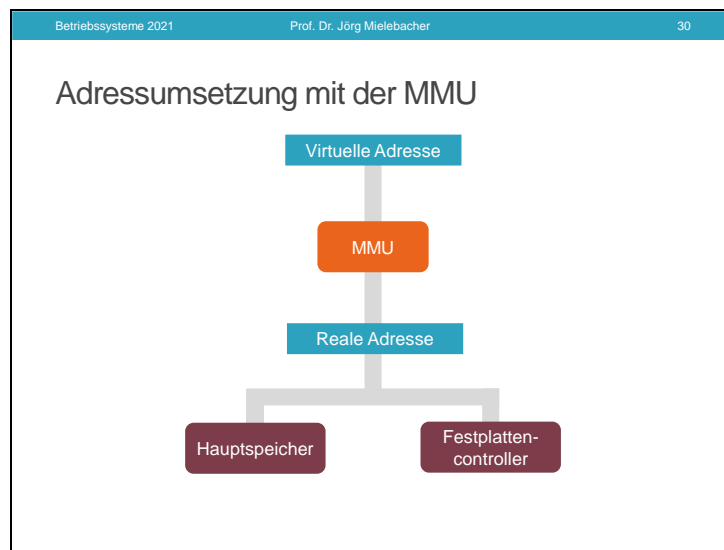
Die Idee des Pagings besteht nun darin, nicht benötigte Daten vom Hauptspeicher auf eine Festplatte, die sog. Paging Area, auszulagern, und benötigte Daten von dieser in den Hauptspeicher zu laden. Aufgrund der Lokalität des Datenzugriffs und des effizienteren Lesens/Schreibens zusammenhängender Blöcke von der Festplatte werden beim Paging immer ganze Seiten geladen bzw. geschrieben.

Unter Windows nennt man die Paging Area Auslagerungsdatei, meist hat sie eine Größe vom 1- bis 3-fachen des Hauptspeichers.

Die Größe des virtuellen Adressraums orientiert sich nicht an der Größe des Programms oder dergleichen; er ist auch unabhängig von der Größe des physischen Speichers – seine Größe richtet sich alleine nach der Länge der Adresse auf dieser Plattform. Der virtuelle Adressraum eines 32bit-Systems umfasst daher 2^{32} Adressen (also rund 4GB).



Auf den ersten Blick könnte man meinen, dass Paging und Swapping gleichbedeutend sind. Es gibt aber einen wichtigen Unterschied: Beim Paging werden einzelne Seiten ein- oder ausgelagert, beim Swapping der gesamte Adressraum eines Prozesses.



Eine besondere Rolle hat die Memory Management Unit (MMU), die üblicherweise Teil des Prozessors ist. Die MMU übernimmt vielfältige Aufgaben des Speichermanagement, vor allem die Umsetzung von virtuellen in physische Adressen, die man auch als Mapping bezeichnet. Soll auf eine virtuelle Adresse zugegriffen werden, übersetzt die MMU diese Adresse in eine reale Speicheradresse. Anhand dieser Adresse können die Daten dann – sofern vorhanden - aus dem Hauptspeicher geladen werden oder mit Unterstützung des Festplattencontrollers aus der Paging Area in den Hauptspeicher geladen werden.

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 31

Typische Ausnahmen

- Unerlaubter Speicherzugriff des Prozesses
→ **Segmentation Fault**

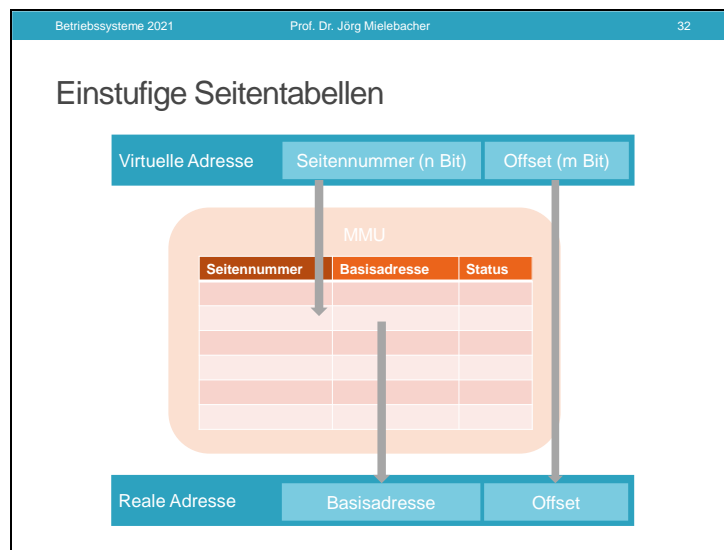
```
int main()
{
    int* p = 0;
    *p = 0;
    return 0;
}
```

Diagramm zur Illustration des Segmentation Faults:

- Die Zeile `int* p = 0;` ist mit einem Pfeil von einem Kasten mit der Aufschrift **Nullpointer** markiert.
- Die Zeile `*p = 0;` ist mit einem Pfeil von einem Kasten mit der Aufschrift **Zugriff** markiert.

- Seite ist noch nicht im Hauptspeicher
→ **Page Fault**

Bei der Umsetzung der Adressen können zwei wichtige Situationen eintreten: Versucht ein Prozess auf eine unerlaubte Adresse zuzugreifen, wird eine Ausnahme ausgelöst („Segmentation Fault“) – besonders oft beobachtet man das beim Versuch, einen Nullpointer zu dereferenzieren. Ist eine Seite nicht im Hauptspeicher, wird beim Versuch des Zugriffs ein Page Fault ausgelöst (ein Trap), der dazu führt, dass man vom User- in den Kernelmodus wechselt und das Betriebssystem das Laden der ausgelagerten Seite in den Hauptspeicher veranlasst. Ggf. muss es hierfür eine andere Seite ersetzen.



Die Umsetzung der virtuellen in eine reale Adressen erfolgt durch Seitentabellen, im einfachsten Fall durch eine einstufige Seitentabelle. Dabei nutzt man das folgende Prinzip: Die virtuelle Adresse hat eine Länge von $n+m$ Bit, wobei n Bit für die Seitennummer vorgesehen sind und m Bit für den Offset innerhalb der Seite. Die MMU verwaltet eine Seitentabelle, die der Seitennummer eine Basisadresse (d.h. den Anfang der Seite im Hauptspeicher) zuordnet, außerdem enthält sie für jede Seite Statusbits (z.B. Read-only). Die reale Adresse ergibt sich nun, indem man anhand der Seitennummer die Basisadresse ermittelt und diese Basisadresse mit dem Offset verbindet. Einstufige Seitentabellen können sehr groß werden, deshalb nutzt man meist mehrstufige Seitentabellen.

Betriebssysteme 2021

Prof. Dr. Jörg Mielebacher

33

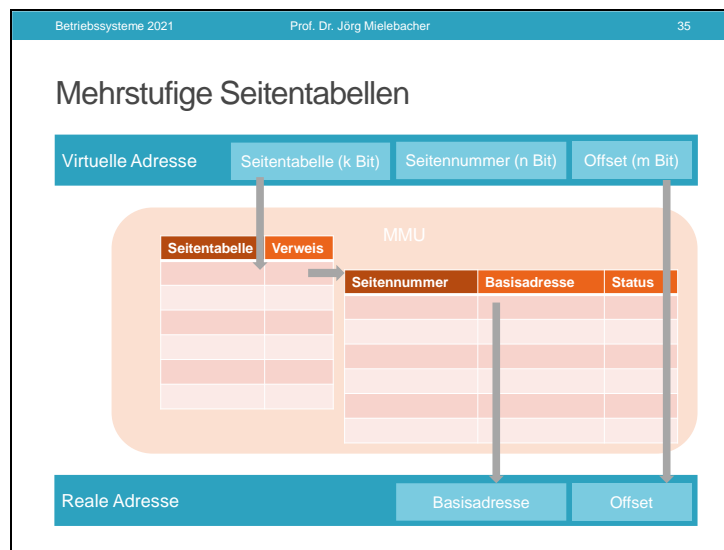
Statusbits der Seitentabelle

- Protection: lesender, schreibender, ausführender Zugriff
- Present/Absent: Seite im Hauptspeicher ja/nein
- Modified (Dirty): Seite wurde geändert
- Reference: Zugriff erfolgt
- ...

Die Seitentabelle speichert unterschiedliche Informationen zum Zustand der Seite, z.B. welche Art von Zugriff erlaubt ist (lesen, schreiben, ausführen), ob die Seite im Hauptspeicher liegt oder noch ausgelagert ist, ob Inhalte der Seite verändert wurden, ob Inhalte zugegriffen wurden, usw.



Im Rahmen der Prozessverwaltung hatten wir die Seitentabelle als Teil des Prozesskontextes eingeführt. Eine Seitentabelle ist also nicht systemweit angelegt, sondern bezieht sich immer auf einen Prozess. Bei der Prozessumschaltung wird also auch die Seitentabelle gewechselt.



Bei den mehrstufigen Seitentabellen werden in den virtuellen Adressen Bits für die Seitentabelle, die Seitennummer und den Offset vorgesehen. Die realen Adressen bestehen weiterhin aus der Basisadresse der Seite und dem zugehörigen Offset. Die MMU verwaltet nun mehrere Seitentabellen, die in einer zusätzlichen Tabelle organisiert sind. Die Umsetzung sucht also zunächst die verwendete Seitentabelle und sucht dann in dieser entsprechend der Seitennummer nach der Basisadresse. Der Offset wird direkt übernommen und wieder an die Basisadresse angehängt.

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 36

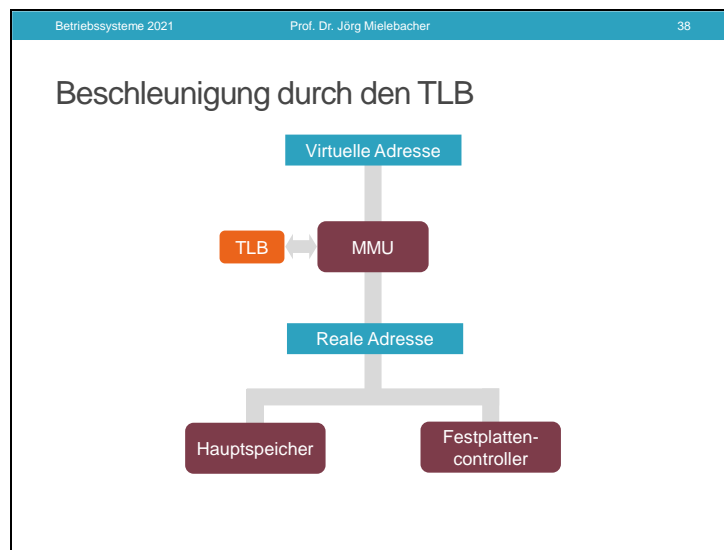
Aber...

- Bei 64bit-Systemen erheblicher Speicherbedarf für Seitentabellen
- Realer Speicher ist deutlich kleiner als virtueller Speicher

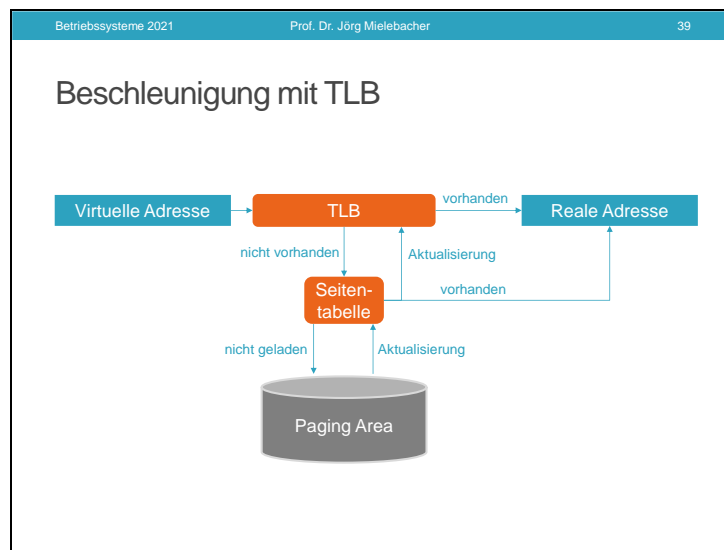
Bei 64bit-Systemen umfassen die virtuellen Adressräume 2^{64} Adressen, also ungefähr $2 \cdot 10^{19}$! Der Speicherbedarf für die Seitentabellen wäre erheblich. Offenkundig ist der reale Speicher signifikant kleiner als der virtuelle Speicher.

Betriebssysteme 2021 Prof. Dr. Jörg Mielebacher 37	
Invertierte Seitentabelle	
Seitentabelle	Invertierte Seitentabelle
<ul style="list-style-type: none">▪ Ein Eintrag je virtueller Seite▪ Direkter Zugriff anhand Seitennummer▪ Ausgelagerte Seiten enthalten	<ul style="list-style-type: none">▪ Ein Eintrag je Frame▪ Suche nach PID und Seitennummer anhand Hash▪ Ausgelagerte Seiten nicht vorhanden

Da in klassischen Seitentabellen je virtueller Seite ein Tabelleneintrag vorgehalten wird, um indexbasiert darauf zuzugreifen, sind die Tabellen sehr groß. In einer invertierten Seitentabellen wird je Frame ein Eintrag angelegt. Ein direkter Zugriff über die Seitennummer ist damit nicht möglich, stattdessen muss in der Tabelle anhand von PID und Seitennummer (meist als Hash) gesucht werden. Da nur Einträge für Frames vorhanden sind, sind ausgelagerte Seiten – im Gegensatz zu gewöhnlichen Seitentabellen nicht enthalten, sondern werden über separate Seitentabellen verwaltet.



Teil der MMU ist eine Speicherstruktur, die man als Translation Lookaside Buffer (TLB) bezeichnet. Dahinter steht das Ziel, die aufwendige Umsetzung von virtuellen in reale Adressen zu beschleunigen. Hierzu merkt der TLB die letzten N Paare von virtuellen und realen Adressen. Meist ist der TLB als sog. Assoziativspeicher aufgebaut, speichert also Schlüssel-Werte-Paare – die virtuelle Adresse samt Prozess-ID dient als Schlüssel, dem jeweils eine reale Adresse zugeordnet ist.



Ist eine virtuelle Adresse bereits im TLB vorhanden, ergibt sich direkt die reale Adresse. Ist sie es nicht, folgt die Umrechnung anhand der Seitentabelle; ist die Adresse dort ermittelt, wird der TLB aktualisiert und die reale Adresse weitergegeben. Ist die Seite der betreffenden Adresse aktuell ausgelagert, wird i.d.R. ein sog. Page Fault ausgelöst, der durch eine Trap signalisiert wird. In diesem Fall muss die ausgelagerte Seite geladen und die Seitentabelle aktualisiert werden.

Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	40
----------------------	----------------------------	----

Seitenersetzungsstrategien

- Not recently used (NRU)
- First in, first out (FIFO)
- Second Chance
- Least recently used (LRU)
- Not frequently used (NFU)
- U.v.a.

Muss eine Seite in den Hauptspeicher geladen werden und es gibt keinen freien Frame, so muss eine vorhandene Seite ausgelagert werden. Welche das ist, legt die sog. Seitenersetzungsstrategie fest. Optimal wäre das Verfahren nach Bélàdy: Es würde die Seite verdrängen, die am längsten in der Zukunft nicht verwendet wird. Da man jedoch nicht in die Zukunft schauen kann, um die nächsten Seitenzugriffe vorherzusagen, nutzt man Heuristiken.

Ein besonders einfacher Vertreter ist „Not recently used“ (NRU), bei der Seiten ausgelagert werden, die jüngst nicht gelesen oder verändert wurden (Referenced/Modified). Die Referenced-Bits können durch vom Kernel von Zeit zu Zeit zurückgesetzt werden. Seiten, die weder gelesen, noch verändert wurden kann man vorrangig ersetzen – bei Ihnen müssen auch keine Änderungen auf die Festplatte geschrieben werden. Bei Seiten, die nur verändert wurden, müssen die Änderungen vor der Ersetzung zurückgeschrieben werden; da kein Lesezugriff erfolgt ist, schließt man wegen der zeitlichen Lokalität, dass auch kein weiterer Zugriff erfolgen wird und erlaubt die Ersetzung. Gab es jedoch Lese- oder Lese- und Schreibzugriffe, wird dies als Indiz gewertet, dass weitere Zugriffe folgen; solche Seiten werden also nachrangig ersetzt.

First in, first out (FIFO) ersetzt die jeweils älteste Seite, d.h. diejenige, die bereits am längsten geladen wurde. Diese Seite steht am Anfang der FIFO-Warteschlange und wird bei Verdrängung aus der Warteschlange entfernt. Die an ihrer Stelle neue geladene Seite wird am Ende der FIFO eingereiht. Ein Nachteil ist offenkundig: Da die tatsächliche Nutzung der Seite keine Rolle spielt, könnten hierdurch häufig genutzte Seiten ausgelagert werden.

Das fälschliche Verdrängen häufig genutzter Seiten korrigiert man bei Second Chance. Hier wird zwar auch FIFO genutzt, jedoch nur dann verdrängt, wenn kein Zugriff auf die Seite erfolgt ist.

Least recently used (LRU) sortiert die Seiten nach dem Zeitpunkt der letzten Nutzung und verdrängt die Seite, die am längsten nicht mehr genutzt wurde. Da man jedoch nicht für jeden Zugriff den Zeitstempel speichert, ist die Umsetzung problematisch. Durch verkettete Listen könnte man eine Sortierung nachbilden, müsste jedoch bei jedem Zugriff diese Seite an das Ende der Liste verschoben werden.

Not frequently used (NFU) basiert darauf, dass Zugriffe auf Seiten gezählt werden, um die am wenigsten genutzten Seiten zu verdrängen. Allerdings wird dies dadurch verfälscht, dass viele Zugriffe in der Vergangenheit einer Verdrängung ähnlich entgegenstehen, wie viele Zugriffe in der Gegenwart.

Daneben gibt es zahlreiche weitere Verfahren, die wir hier nicht weiter verfolgen wollen.

Zusammenfassung

- Bei der Speichernutzung beobachtet man zeitliche und räumliche Lokalität.
- Die Speicherhierarchie vereint Technologien mit unterschiedlichen Zugriffszeiten und Speichervolumina.
- Aktuelle Betriebssysteme stellen jedem Prozess einen virtuellen Adressraum zur Verfügung und nutzen Paging.
- Die MMU übernimmt mit Unterstützung des TLB das Mapping zwischen virtuellen und realen Adressen durch Seitentabellen, teilweise mit invertierten Seitentabellen.
- Ein Segmentation Fault tritt bei unerlaubten Zugriffen auf virtuelle Adressen auf.
- Ein Page Fault tritt auf, wenn eine Seite noch nicht in den Hauptspeicher geladen ist.
- Es gibt verschiedene Ersetzungsstrategien (z.B. NRU, LRU, FIFO, NFU).

Aufgaben

1. Was versteht man unter räumlicher und zeitlicher Lokalität?
2. Wieso muss man auf eine Hierarchie unterschiedlicher Speichertechnologien zurückgreifen?
3. Wodurch könnte bei einer mit malloc oder new erzeugten, verketteten Liste das Prinzip der räumlichen Lokalität eingeschränkt sein?
4. Warum setzt man die virtuelle Speicherverwaltung ein?
5. Was ist der Unterschied zwischen Paging und Swapping?
6. Wie setzt die MMU virtuelle in reale Adressen bei mehrstufigen Seitentabellen um?
7. Was bedeuten die Offset-Bits einer virtuellen Adresse?
8. Wie beschleunigt der TLB die Arbeit der MMU?

Betriebssysteme 2021	Prof. Dr. Jörg Mielebacher	43
<h2>Aufgaben</h2> <ul style="list-style-type: none">9. Was versteht man unter Mapping?10. Welche Situationen können in einem C/C++-Programm zu einem Segmentation Fault führen?11. Was muss bei einem Page Fault geschehen?12. Weshalb sind konventionelle Seitentabellen bei 64bit-Systemen problematisch?		