Systemprogrammierung

Grundlage von Betriebssystemen

Teil C – XII.3 Speicherverwaltung: Virtualisierung

Wolfgang Schröder-Preikschat

14. Januar 2016

Agenda

Gliederung

Lehrstoff

- **Speichervirtualisierung** im Detail behandeln und in Bezug auf ihre beiden zentralen Aufgaben untersuchen:
  - Ladestrategie (*fetch policy*)
    - wann muss ein Datum im Hauptspeicher liegen?
  - Ersetzungsstrategie (*replacement policy*)
    - welches Datum im Hauptspeicher ist ersetzbar?
- **Vor- und Nachteile** erkennen, das heißt, als optionales Merkmal der Speicherverwaltung eines Betriebssystems verstehen
  - als benutzerorientiertes oder systemorientiertes Kriterium begreifen
  - entweder einen Prozess oder mehrere Prozesse weitestgehend unabhängig von der Größe des Hauptspeichers ermöglichen
  - bei vielen Prozessen, den Grad an **Mehrprogrammbetrieb** maximieren
- dynamische Bindung zwischen virtueller Adresse eines Prozesses und realer Adresse im **Arbeitsspeicher** verinnerlichen
  - d.h., die **Synergie** von Haupt- und Ablagespeicher
  - desselben Rechensystems oder verschiedener (vernetzter) Rechensysteme

Gliederung

Einlagerung der Gebrauchsstücke

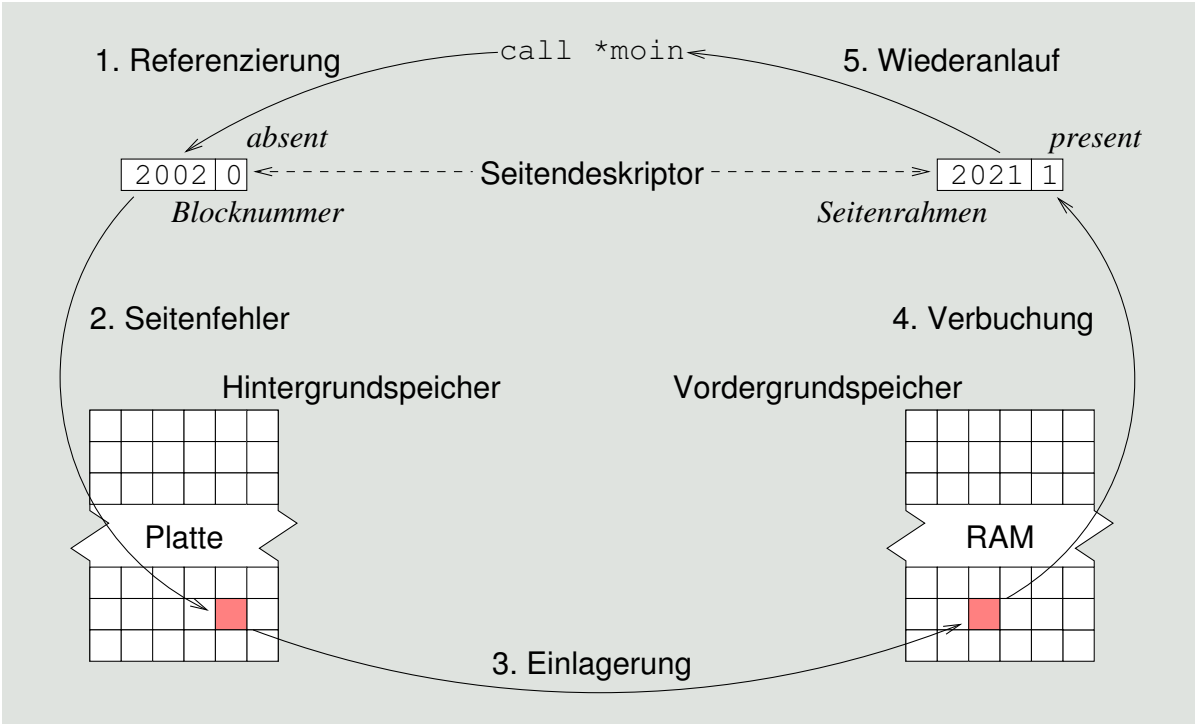
Hinweis (Gebrauchsstück (vgl. [11, S. 14]))

*Ein **umlagegrungsfähiges Bestandteil** eines Prozessadressadraums, in Format (Seite, Segment) und Größe (fest, veränderlich) bestimmt durch die Adressumsetzungseinheit (MMU).*

- **Einzelanforderung**: *on demand*
  - gesteuert durch das **Präsenzbit** eines jedem Stücks
    - anwesend – Zugriff möglich
    - abwesend – *page/segment fault*
  - bei Abwesenheit erfolgt die **partielle Interpretation** des Zugriffs
    - Ausnahmebehandlung
    - durch das Betriebssystem
- **Vorausladen**: *anticipatory*
  - der Einzelanforderung höchstmöglich zuvorkommen
    - **Vorabruf** (*prefetch*)
    - Vermeidung von Folgefehlern
  - **Heuristiken** liefern Hinweise über zukünftige Zugriffe
    - Prozesslokalität
    - Arbeitsmenge (*working set*)
- ggf. fällt die **Verdrängung** (Ersetzung) von anwesenden Stücken an

Einzelanforderung

- die **Seitenumlagerungsfunktion** (*pager*) des Betriebssystems



Einzelanforderung mit Vorausladen

Hinweis (`call *moin` — und mehr dieser Art)

Bei der Ausführung ein und desselben Maschinenbefehls durch die CPU kann es mehr als einen Zugriffsfehler geben.

- Vorbeugung ggf. nachfolgender Zugriffsfehler desselben Befehls:
  1. den gescheiterten Befehl dekodieren, Adressierungsart feststellen
  2. da der Operand die Adresse einer Zeigervariablen (`moin`) ist, den Adresswert auf Überschreitung einer Seitengrenze prüfen
  3. da der Befehl die Rücksprungadresse stapeln wird, die gleiche Überprüfung mit dem Stapelzeiger durchführen
  4. in der Seitentabelle die entsprechenden Deskriptoren lokalisieren und prüfen, ob die Seiten anwesend sind
     - jede abwesende Seite (*present bit* $= 0$) ist einzulagern
  5. da jetzt die Zeigervariable (`moin`) vorliegt, sie dereferenzieren und ihren Wert auf Überschreitung einer Seitengrenze prüfen
     - hierzu wie bei 4. verfahren
  6. den unterbrochenen Prozess den Befehl wiederholen lassen

↪ **Teilemulation** fast aller Maschinenbefehle durch das Betriebssystem ☹

Zugriffsfehler

- **Seitenfehler** (*page fault*) oder **Segmentfehler** (*segment fault*)
  present bit = 0
  - je nach Befehlssatz und Adressierungsarten der CPU kann der **Behandlungsaufwand** im Betriebssystem und somit der **Leistungsverlust** beträchtlich sein

- Fallstudie: Aufruf einer Prozedur indirekt über einen Funktionszeiger

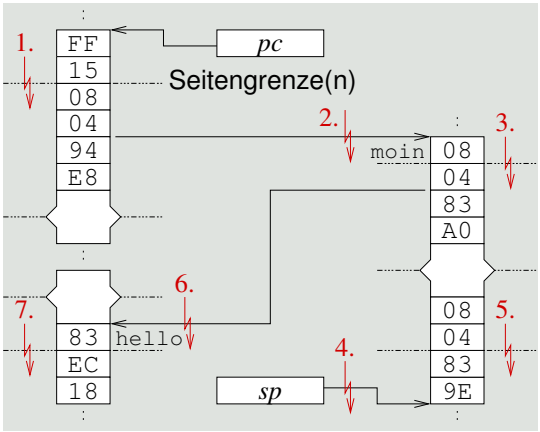
```
1  void hello () {
2      printf("Hi!\n");
3  }
4
5  void (*moin)() = &hello;
6
7  main () {
8      (*moin)();
9  }
```

```
10  main:
11      pushl %ebp
12      movl  %esp,%ebp
13      pushl %eax
14      pushl %eax
15      andl  $-16,%esp          :
16      call  *moin          FF15080494E8
17      leave                    :
18      ret
```

  - wie viele Seitenfehler sind bei Ausführung dieses einen Befehls möglich?

Zugriffsfehler: Schlimm(st)er Fall eines Seitenfehlers

- Aufruf einer indirekt adressierten Prozedur: `call *moin`
  - angenommen, der Operationskode (FF15) wurde bereits gelesen



  - Interpretation des Befehls (`call`):
    1. Operandenadresse holen (080494E8)
    2. Funktionszeiger lesen (08)
    3. Funktionszeiger weiterlesen (0483A0)
    4. Rücksprungadresse stapeln (0804)
    5. Rücksprungadresse weiterstapeln (839E)
  - Aufnahme des Folgebefehls (`sub`):
    6. Operationskode holen (83)
    7. Operanden holen (EC18)
  - Seitenfehler 6. und 7. sind bereits der Ausführung des ersten Befehls der aufgerufenen Prozedur (`hello`) zuzurechnen

- **Ausrichtung** (*alignment*) von Programmtext und -daten im logischen Adressraum hilft, die Anzahl von Seitenfehlern zu reduzieren
  - eine Aufgabe des Kompilierers, Assemblierers oder Binders

Aufwandsabschätzung von Einzelzugriffen

- Speichervirtualisierung bringt **nichtfunktionale Eigenschaften** bei der Ausführung von Programmen mit sich
  - **Interferenz** in Bezug auf den Prozess, der einen Zugriffsfehler produziert
- ein Seitenfehler bewirkt eine Prozessverzögerung, dehnt die **effektive Zugriffszeit** (*effective access time*, *eat*) auf den Hauptspeicher
  - hängt stark ab von der **Seitenfehlerwahrscheinlichkeit** (p) und verhält sich direkt proportional zur **Seitenfehlerrate**:

$$eat = (1 - p) \cdot pat + p \cdot pft, 0 \leq p \leq 1$$

  - angenommen, folgende Systemparameter sind gegeben:
    - 50 ns Zugriffszeit auf den RAM (*physical access time*, *pat*)
    - 10 ms mittlere Zugriffszeit auf eine Festplatte (*page fault time*, *pft*)
    - 1 % Wahrscheinlichkeit eines Seitenfehlers ($p = 0,01$)
  - dann ergibt sich:
    $eat = 0,99 \cdot 50\,\text{ns} + 0,01 \cdot 10\,\text{ms} = 49,5\,\text{ns} + 10^5\,\text{ns} \approx 0,1\,\text{ms}$

⇒ Einzelzugriffe sind im Ausnahmefall um den Faktor 2000 langsamer !

Aufwandsabschätzung bei Folgezugriffen

- anzunehmen ist eine **mittlere Zugriffszeit** (*mean access time*, *mat*) auf den Hauptspeicher
  - hängt stark ab von der effektiven **Seitenzugriffszeit** und der Anzahl der **Seitenreferenzierungen**:
    - schlimmstenfalls der Seitengröße (in Bytes pro Seite bzw. Seitenrahmen)

$$mat = (eat + (sizeof(page) - 1) \cdot pat)/sizeof(page)$$

  - angenommen, folgende Systemparameter sind gegeben:
    - Seitengröße von 4 096 Bytes (4 KB)
    - 50 ns Zugriffszeit (*pat*) auf ein Byte im RAM
    - effektive Zugriffszeit (*eat*) wie eben berechnet bzw. abgeschätzt
  - dann ergibt sich: $mat = (eat + 4\,095 \cdot 50\,\text{ns})/4\,096 \approx 74,41\,\text{ns}$
    - aber nur, wenn jede Speicherstelle der Seite einmal referenziert wird

⇒ daraus resultiert bestenfalls eine Verlangsamung um den Faktor 1,5

- ***Seitenfehler sind nicht wirklich transparent***...
  - ihre Auswirkungen stehen und fallen mit der **Ersetzungsstrategie**
  - ihre Häufigkeiten stehen und fallen zusätzlich mit der **Prozesslokalität**

Gliederung

Zeitmultiplex von Seitenrahmen

Hinweis (Speichervirtualisierung)

Abbildung der logisch abzählbar unendlichen Menge von Seiten eines oder mehrerer virtueller Adressräume auf die abzählbar beschränkte Menge von Seitenrahmen des realen Adressraums.

- logisch, da angenommen wird, dass die Anzahl virtueller Adressräume im Allgemeinen unbestimmbar ist
  - jeder virtuelle Adressraum ist einem Prozessexemplar zugeordnet
  - die Anzahl dieser Exemplare ist in Mehrbenutzersystemen i.A. unbekannt
- aber physisch ist diese Anzahl auf Grund der **endlichen Darstellung** solcher Adressräume in Rechensystemen nach oben beschränkt
  - jeder virtuelle Adressraum ist durch mehrere Datenstrukturen beschrieben
  - Datenstrukturen belegen Speicherplatz, der nur begrenzt vorhanden ist
- gleichwohl ist die Seitenmenge (virtueller Adressraum) größer als die Seitenrahmenmenge (realer Adressraum)
  - mehrere Seiten müssen sich ein und denselben Seitenrahmen teilen
  - Seitenrahmeninhalte sind durch (logisch) verschiedene Seiten zu ersetzen

Menge residenter Seiten

Definition (*resident set*)

Die abzählbar endliche Menge der Seiten des virtuellen Adressraums eines Prozessexemplars, die gegenwärtig auf Seitenrahmen abgebildet ist und damit im Hauptspeicher platziert vorliegt.

- das Betriebssystem entscheidet, wie viel Seiten in den Hauptspeicher gebracht, d.h., wie viel Seitenrahmen einem Prozess zugeteilt werden
  - nicht vorliegende Seiten werden automatisch nachgezogen (Ladestrategie)
- **verschiedene Faktoren** bestimmen die Anzahl residenter Seiten, die einem jeweiligen Prozess zugestanden werden können/sollten:
  - i je kleiner die Menge, desto höher der **Grad an Mehrprogrammbetrieb**
  - ii bei zu kleiner Menge steigt jedoch die **Seitenfehlerwahrscheinlichkeit**
  - iii und bei zu großer Menge sinkt die **Seitenfehlerrate** nur unwesentlich
- vor diesem Hintergrund wird die Menge residenter Seiten einem jeden Prozess in **fester oder variabler Größe** zugeteilt
  - wobei die Befehlssatzebene eine **Mindestmenge** vorgibt (vgl. S. 10)
  - entsprechend folgt Ersetzung einem statischen oder dynamischen Ansatz

Einzugsbereich der Seitenersetzung

- die Ladestrategie fordert, dass ein Seitenrahmen zur Aufnahme der jeweils einzulagernden Seite verfügbar sein muss
  - solange nicht alle einem Prozess zugeteilten Seitenrahmen belegt sind, ist die Entscheidung, wohin die Seite einzulagern ist, einfach
    - der erstbeste zugeteilte unbelegte (d.h., freie) Seitenrahmen wird genommen
  - anderenfalls ist ein Seitenrahmen freizumachen, d.h., die dort eingelagerte Seite ist durch die einzulagernde Seit zu ersetzen
    - es ist zu prüfen, ob die eingelagerte Seite zuvor ausgelagert werden muss !
- dazu geschieht eine **lokale oder globale Suche** nach Seitenrahmen, die zur Einlagerung der Seite freigemacht werden können

  lokal
  - Suchraum ist nur die Menge der residenten Seiten des Prozesses, der den Seitenfehler verursacht hat
  - ein Seitenfehler ist vorhersag-/reproduzierbar [10, S. 12]

  global
  - Suchraum ist die Menge aller residenten Seiten aller Prozesse im System, unabhängig vom Verursacher des Seitenfehlers
  - ein Seitenfehler ist unvorhersag-/unreproduzierbar [10, S. 13]
- so besteht ein Zusammenhang zwischen der Anzahl residenter Seiten eines Prozesses und dem Wirkungskreis der Seitenersetzung

Gegenüberstellung

vgl. [14, S. 364]

Zuteilung	***Ersetzung***	
	lokal	*global*
fest	Die Anzahl der Seitenrahmen des Prozesses ist fest. Die zu ersetzende Seite wird unter den dem Prozess zugeteilten Seitenrahmen ausgewählt.	Nicht möglich!
variabel	Die Anzahl der Seitenrahmen des Prozesses ist nicht fest, sie variiert mit der **Arbeitsmenge** (S. 27) über die Zeit. Die zu ersetzende Seite wird unter den dem Prozess zugeteilten Seitenrahmen ausgewählt.	Die Anzahl der Seitenrahmen des Prozesses ist nicht fest, sie kann sich zu seiner **Lebenszeit** verändern. Die zu ersetzende Seite wird unter allen verfügbaren Seitenrahmen ausgewählt.

- feste Zuteilung von Seitenrahmen erfolgt spätestens zur **Ladezeit** des Maschinenprogramms, d.h., zum Zeitpunkt der Prozesserzeugung
  - basierend auf Anwendungswissen oder Vorgabe des Betriebssystems

Lokalitätsprinzip

Hinweis

Ein Prozess, der zu einem Zeitpunkt eine Stelle in seinem Adressraum referenziert, wird mit gewisser Wahrscheinlichkeit dieselbe Stelle oder eine andere Stelle in direkter Umgebung referenzieren.

- festgelegt in der **Programmvorschrift** — aber auch bestimmt durch Struktur und Übersetzung der jew. Software (vgl. [3, S. 250]):
  - i Programmausführung ist meist sequentiell, d.h., der nächste abzurufende Maschinenbefehl folgt dem jetzigen an nächster Stelle
  - ii Programmschleifen umfassen vergleichsweise wenig Maschinenbefehle, d.h., Berechnungen finden in kleinen begrenzten Bereichen statt
  - iii Programmberechnungen verwenden oft große Datenstrukturen wie Felder, Verbünde, Dateien, d.h., sie neigen zu benachbarten Datenzugriffen
- daher kann davon ausgegangen werden, dass die **Referenzfolge** eines Prozesses abschnitt- bzw. phasenweise nahezu stetig ist
  - sie gut abschätzen zu können, ist für jedes Ersetzungsverfahren wichtig
  - hier helfen „klebrige“ **Statusbits** im Seitendeskriptor (vgl. [9, S. 15])

Seitenersetzung ist eine Zukunftsfrage. . .

Auswahl jenes Seitenrahmens, dessen Seite (am längsten) nicht mehr referenziert werden wird.

- diese Strategie ist nachweislich optimal, da sie die Anzahl möglicher Seitenfehler in Bezug auf eine beliebige Referenzfolge minimiert [1]

  MIN
  - wähle eine Seite, die zukünftig nicht erneut referenziert wird
  - falls es keine solche Seite gibt, wähle eine, auf die relativ zur Zeit des Seitenfehlers am weitesten entfernt zugegriffen werden wird
  - auch als OPT bezeichnet

- gleichsam ist sie unrealistisch, nicht implementierbar, weil praktisch keine Referenzfolge eines Prozesses im Voraus bekannt ist[1]
  - das würde nämlich bedeuten:
    - der Ablaufpfad in seinem Programm ist absehbar ?
    - die diesen Pfad beeinflussende Eingabewerte sind vorherbestimmt ?
    - und Verzweigungen durch Programmunterbrechungen sind vorhersagbar ?
  - bestenfalls ist es möglich, eine gute **Approximation** anzugeben

[1] Ausnahmen (Echtzeitsysteme: WCET-Analyse) bestätigen die Regel.

Approximation der optimalen Strategie

- Grundlage bildet Wissen über die **Vergangenheit** und **Gegenwart** von Seitenzugriffen, um Annahmen über die **Zukunft** zu treffen:

  FIFO (*first-in, first-out*) ✓
  - ersetzt wird die zuerst eingelagerte Seite ⇝ verketten
  - Belady's Anomalie [2]: mehr Seitenrahmen, ggf. mehr Seitenfehler

  LFU (*least frequently used*)
  - ersetzt wird die am seltensten referenzierte Seite ⇝ zählen
  - Alternative: MFU (*most frequently used*)

  LRU (*least recently used*) ✓
  - ersetzt wird die am längsten nicht mehr referenzierte Seite
    - pro Seitendeskriptor **Zeitstempel** setzen oder **Kondensator** aufladen
    - eine **Stapeltechnik** einsetzen: die referenzierte Seite „oben ablegen"
    - die **Alterungsstruktur** einer Seite in einem Schieberegister erfassen ☺
    - bzw. weniger aufwändig durch einzelne **Statusbits** abschätzen ☺
  - die Seite mit dem **größten Rückwärtsabstand** wird ausgewählt
    - größter Zeitabstand, kleinste Ladung, unten liegend, wenigsten Einsen

- die Effektivität der Verfahren hängt ab von der **Prozesslokalität**, der Stetigkeit seiner gegenwärtigen Referenzfolge

LRU: Alterungstruktur

aging

- dazu ist jedem Seitendeskriptor ein **Schieberegister** (*aging register*) zugeordnet, meist in Software implementiert (Schattendeskriptor)
- zusätzlich wird ein **Zeitgeber** (*timer*) benötigt, der eine **periodische Unterbrechung** des Prozesses verursacht $\rightsquigarrow$ **Hintergrundrauschen**
- bei jedem Ablauf des Zeitintervalls wird das **Referenzbit** einer jeden eingelagerten Seite des unterbrochenen Prozesses eingeprüft:
  - i ist es gesetzt, wurde die Seite referenziert: das Bit wird zurückgesetzt
  - ii ist es nicht gesetzt, wurde die Seite nicht referenziert

tick	Ref.	*aging register*
		00000000
n	1	10000000
$n+1$	1	11000000
$n+2$	0	01100000
$n+3$	1	10110000
$\vdots$	$\vdots$	$\vdots$

- Registerinhalt (*age*) als Ganzzahl interpretiert liefert ein Maß für die Aktivität einer Seite
- mit abnehmendem Betrag, d.h. einer sinkenden Prozessaktivität, steigt die Ersetzungspriorität
- bei Seitenfehler/-ersetzung wird die global älteste Seite gewählt

LRU: zweite Chance I

second-chance replacement

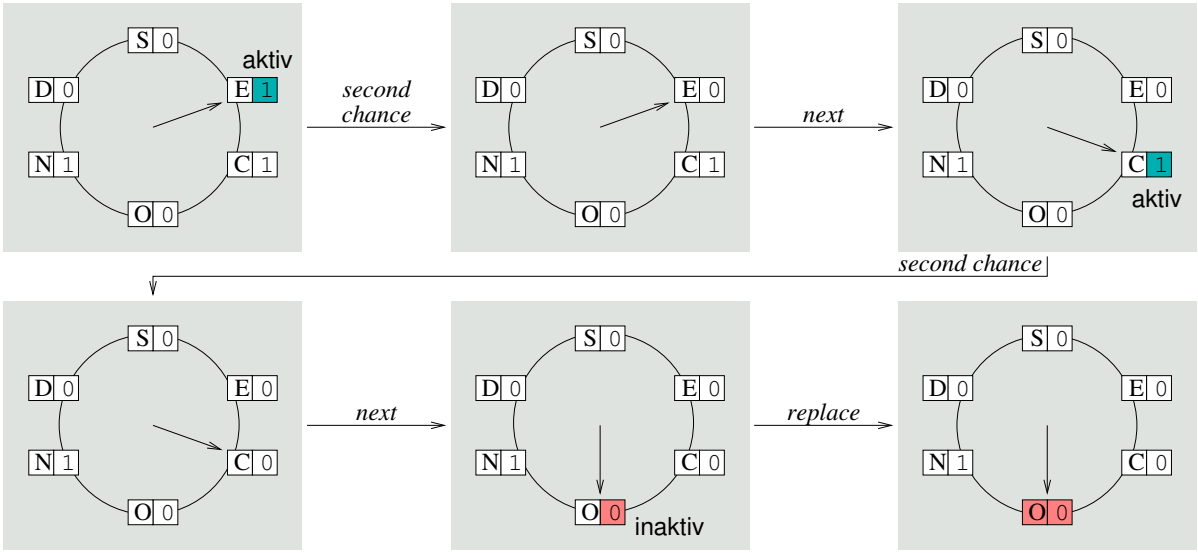
- arbeitet im Grunde nach **FIFO**, berücksichtigt jedoch zusätzlich noch **Statusbits** der jeweils in Betracht zu ziehenden Seiten
- nutzt einen **Zeitgeber** (*timer*), der eine **periodische Unterbrechung** des Prozesses verursacht $\rightsquigarrow$ **Hintergrundrauschen**
- bei jedem Ablauf des Zeitintervalls wird das **Referenzbit** (*use*) einer eingelagerten Seite des unterbrochenen Prozesses geprüft:

use	Aktion	Bedeutung
1	Referenzbit zurücksetzen	Seite erhält eine zweite Chance
0	—	Seite kann ersetzt werden

  - bei einem Seitenfehler, der eine Seitenersetzung nach sich zieht, wird die global zuerst eingelagerte und unreferenzierte Seite gewählt
- schlimmstenfalls erfolgt ein Rundumschlag über alle Seiten, wenn die Referenzbits aller betrachteten Seiten (auf 1) gesetzt waren
  - die Strategie „entartet“ dann zu FIFO

LRU: zweite Chance II

clock replacement



- Annahme ist, referenzierte Seiten sind vermeintlich aktive Seiten:
  - E ▪ aktiv, Referenzbit zurücksetzen, Seite im Hauptspeicher behalten
  - C ▪ aktiv, Referenzbit zurücksetzen, Seite im Hauptspeicher behalten
  - O ▪ inaktiv, Seite ist ersetzbar, Seitenrahmen für andere Seite nutzen
- allgemein konzipiert für $n \geq 0$ *use*-Bits pro Seite [4]: FIFO bei $n = 0$, bestes Kosten-Nutzen-Verhältnis bei $n = 1$, Alterung bei $n > 1$

LRU: dritte Chance

third-chance replacement

- übernimmt alle Merkmale von *second chance*:
  - FIFO als Basis, periodische Unterbrechung, Referenzbit (*use bit*)
  - Hintergrundrauschen
- zusätzlich zum Referenz- wird das **Modifikationsbit** (*dirty bit*) jeder eingelagerten Seite des unterbrochenen Prozesses geprüft
  - dieses Bit wird bei Schreibzugriffen gesetzt, bleibt sonst unverändert
  - zusammen mit dem Referenzbit zeigen sich vier Paarungen (*use*, *dirty*):

	Bedeutung	Entscheidung
(0, 0)	ungenutzt	beste Wahl
(0, 1)	beschrieben	keine schlechte Wahl
(1, 0)	kürzlich gelesen	keine gute Wahl
(1, 1)	kürzlich beschrieben	schlechteste Wahl

  - ausgewählt wird die global zuerst eingelagerte, unreferenzierte und wenn möglich unveränderte Seite
- kann für jede eingelagerte Seite zwei Umläufe erwirken, gibt aktiven, d.h., referenzierten Seiten damit eine dritte Chance [8]
  - auch als ***enhanced second chance*** bezeichnet

Seitenflattern

thrashing

Kritisches Systemverhalten, wenn die durch Seitenein-/auslagerungen verursachte E/A die gesamten Systemaktivitäten dominiert [5]

- eben erst ausgelagerte Seiten werden sofort wieder eingelagert
  - es wurde die falsche Seite ausgewählt, die Vorhersage stimmte nicht
  - Folge: Prozesse verbringen mehr Zeit beim Umlagern als beim Rechnen
- ein mögliches **Phänomen der globalen Seitenersetzung**
  - Prozesse bewegen sich zu nahe am Seitenrahmenminimum
    - die (fest) oder ihre jeweilige (variabel) Menge residenter Seite ist zu klein
    - d.h., sie ist kaum größer als die durch die Hardware definierte Mindestmenge
  - zum Lastprofil eher ungünstige Ersetzungsstrategie
  - zu hoher Grad an Mehrprogrammbetrieb
- verschwindet ggf. so plötzlich von allein, wie es aufgetreten ist...

Hinweis

Ein ernstes Problem, das allerdings auch immer in Relation zu der Zeit, die Prozesse mit sinnvoller Arbeit verbringen, zu setzen ist.

Freiseitenppuffer

page buffering

- arbeitet im Grunde nach **FIFO**, verwaltet aber zusätzlich noch einen **Zwischenspeicher** (*cache*) potentiell zu ersetzender Seiten [13]
  - getrennt in zwei Listen für modifizierte und unmodifizierte Seiten
  - modifizierte Seiten sind auszulagern, bevor sie ersetzt werden können
- Seiten im Zwischenspeicher sind logisch abwesend, ihr **Präsenzbit** ist gelöscht, physisch aber noch anwesend, bis sie ersetzt wurden
  - sie kommen jeweils als **Fußseite** in die ihrem Zustand entsprechende Liste
- bei einem Zugriffsfehler auf eine im Zwischenspeicher liegende Seite, erfolgt ihre **Reklamierung** und ihr Präsenzbit wird wieder gesetzt
  - anderenfalls wird die **Kopfseite** (aus einer der beiden Listen) entfernt
  - ausgewählt wird die lokal zuerst eingelagerte und längst ungenutze Seite
- der Zwischenspeicher ist die **Reserve** „ungebundener“ Seitenrahmen, *pager*-gesteuert durch **Schwellwerte** $\rightsquigarrow$ **Hintergrundrauschen**:
  - *low*: Seitenrahmen als frei markieren, Seiten zwischenspeichern
  - *high*: Seitenrahmen liegen brach, Seiten einlagern $\rightsquigarrow$ **Vorausladen**
  - damit entspricht die Ersetzungszeit einer Seite ihrer Ladezeit

Arbeitsmenge

working set (vgl. [6, S. 326] und [7])

Definition (Standpunkt eines Prozesses)

Die kleinste Sammlung von Programmtext und -daten, die in einem Hauptspeicher vorliegen muss, damit effiziente Programmausführung zugesichert werden kann.

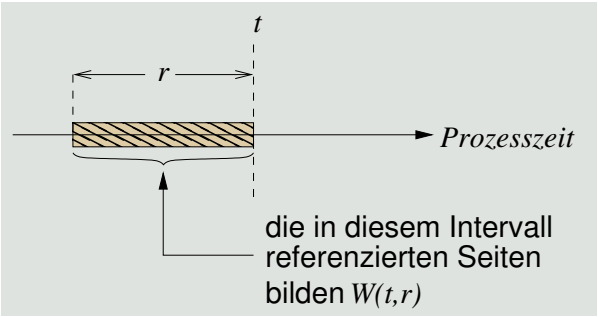
- eine Forderung, die ohne **exakte Voranzeigen** zum Platzbedarf über die Zeit der Programmausführung nicht umsetzbar ist
- damit einhergehendes **Vorabwissen** ist nicht verfügbar, es lässt sich nie vollständig durch statische Analyse der Programme herleiten
- wie viel Hauptspeicher faktisch belegt sein wird, ist nur **zur Laufzeit** durch Aufzählung der wirklich benutzten Seiten feststellbar

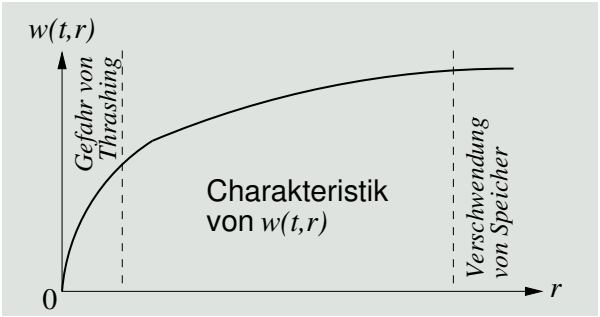
Definition (Standpunkt eines Betriebssystems)

Die **Teilmenge** *WS* der zuletzt (*most recently*) referenzierten Seiten eines Prozesses aus der Menge seiner residenten Seiten *RS*, $WS \subseteq RS$.

Aktive Seiten eines Prozesses

- die Menge von Seiten ***W(t, r)***, die ein Prozess zum Zeitpunkt t und im vorangegangenen relativen Zeitfenster r in Benutzung hatte:
  t Beobachtungszeitpunkt
  r Arbeitsmengenparameter (*working set parameter*), Konstante
- die Anzahl aktiver Seiten in $W(t, r)$, die also referenziert und damit genutzt wurden, legt die **Arbeitsmengengröße** $\omega(t, r)$ fest



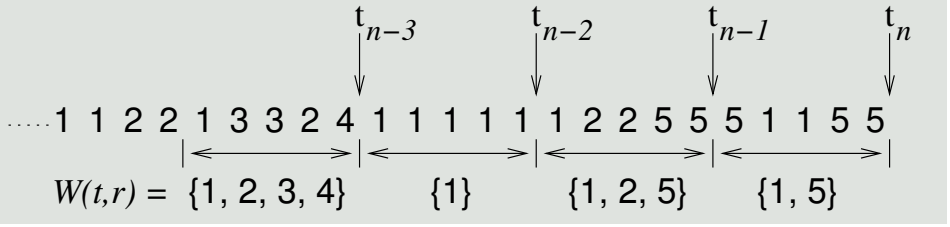


- die im **Zeitinterval** $(t - r, t)$ jüngst von einem Prozess referenzierten Seiten werden von ihm wahrscheinlich weiter benutzt
  - nur einzelne davon auszulagern, erhöht das Risiko zum Seitenflattern...

Arbeitsmengendynamik

moving working set window

- die Arbeitsmengengröße hängt von der **Prozesslokalität** ab und kann nur näherungsweise bestimmt werden
  - gegeben sei die **Seitenreferenzfolge** der jüngeren Vergangenheit
  - darauf wird ein **Arbeitsmengenfenster** (*working set window*) geöffnet
  - dieses bewegt sich schrittweise vorwärts mit jeder weiteren Referenz
- die **Fensterbreite** r gibt eine „feste Anzahl von Maschinenbefehlen“
  - realisiert als **Zeitfenster** (*timer*), das eine **periodische Unterbrechung** des Prozesses verursacht $\rightsquigarrow$ **Hintergrundrauschen**
  - die **Periodenlänge** macht in etwa eine feste Anzahl von Befehlen aus



  - zu kleine Fenster halten benutzte Seiten draußen (Seitenfehlerrate steigt), zu große halten unbenutzte Seiten drinnen (Speicherverschwendung)
  - WS ähnelt lokalem LRU, begrenzt aber durch das Fenster den Suchraum

Operative Umsetzung

- eine Ersetzungsstrategie auf Basis des Arbeitsmengenmodells verfährt sodann nach den folgenden beiden **Regeln**:
  - i bei jeder Seitenreferenz wird die Arbeitsmenge aktualisiert, woraufhin nur die Seiten eben dieser Menge im Hauptspeicher gehalten werden
  - ii ein Prozess kann voranschreiten genau dann, wenn seine Arbeitsmenge im Hauptspeicher vollständig vorliegt
- obgleich konzeptionell attraktiv, so ist die Umsetzung des Konzepts schwierig und sie zieht auch **hohe Unkosten** nach sich
  - i die Bestimmung der Fenstergröße muss empirisch geschehen, indem r solange variiert wird, bis die beste Leistung verzeichnet wird
  - ii die aktuelle Arbeitsmenge kann sich mit jeder Seitenreferenz verändern und damit die Zuteilung von Seitenrahmen an den Prozess ☹
- nur **Approximation** lässt eine praxistaugliche Lösung erhoffen, z.B.:
  - die **Alterung** von Seiten erfassen (vgl. S. 21) mit $r = \delta * n$, wobei δ das Zeitinterval festlegt und n die Bitanzahl im Alterungsregister
  - ungenutzte Seiten verlieren graduell an Bedeutung, ab einem festgelegten Schwellwert $l \geq 0$ wird die Seite aus der Arbeitsmenge entfernt
    - die Seite fiel mit Erreichen von l aus dem Arbeitsmengenfenster heraus

Gliederung

Resümee

. . . optionales Merkmal

- die **Ladestrategie** bestimmt, wann ein Datum im Hauptspeicher liegt und wie es dort hingebracht wird
  - **Einzelanforderung** (*demand paging*) oder **Vorausladen** (*anticipatory*)
  - ersteres setzt Zugriffsfehler voraus, letzteres versucht diesem vorzubeugen
- die **Ersetzungsstrategie** bestimmt, welches Datum seinen Platz im Hauptspeicher für ein anderes Datum freimachen muss
  - **globale Verfahren** untersuchen die Mengen aller residenten Seiten aller Prozesse im System, unabhängig vom Verursacher des Zugriffsfehlers
    - FIFO, LRU (*aging*, *second chance*/*clock*, *advanced second chance*)
    - als unerwünschter Effekt ist **Seitenflattern** (*thrashing*) möglich
  - wohingegen **lokale Verfahren** nur die Menge der residenten Seiten des Prozesses, der den Zugriffsfehler verursacht hat, in Betracht ziehen
    - Freiseitenpuffer, Arbeitsmenge (insb. in Kombinbation mit lokalem LRU)
    - mit letzterem Konzept sind hohe **Unkosten** (*overhead*) verbunden
- die **residente Menge von Seiten** (*resident set*) eines Prozesses ist nicht mit seiner **Arbeitsmenge** (*working set*) zu verwechseln
  - letztere ist eine Teilmenge ersterer und hoch dynamisch (s. auch S. 36)

Literaturverzeichnis I

[1] Bélády, L. A.:
A Study of Replacement Algorithms for a Virtual Storage Computer.
In: *IBM Systems Journal* 5 (1966), Nr. 2, S. 78–101

[2] Bélády, L. A. ; Nelson, R. A. ; Shelder, G. S.:
An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine.
In: *Communications of the ACM* 12 (1969), Jun., Nr. 6, S. 349–353

[3] Bic, L. F. ; Shaw, A. C.:
Operating System Principles.
Pearson Education, Inc., 2003

[4] Corbató, F. J.:
A Paging Experiment with the Multics System / Project MAC, Defense Technical Information Center.
1968. –
Forschungsbericht

Literaturverzeichnis II

[5] Denning, P. J.:
Thrashing: Its Causes and Prevention.
In: *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference (AFIPS '68), December 9–11, 1968, San Francisco, CA, USA* Bd. 33, ACM, 1968 (Part I), S. 915–922

[6] Denning, P. J.:
The Working Set Model for Program Behavior.
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 323–333

[7] Denning, P. J.:
Working Sets Past and Present.
In: *IEEE Transactions on Software Engineering* SE-6 (1980), Jan., Nr. 1, S. 64–84

[8] Goldman, P. :
Mac VM Revealed.
In: *BYTE* 14 (1989), Nov., Nr. 12, S. 350–360

[9] Kleinöder, J. ; Schröder-Preikschat, W. :
Adressräume.
In: [12], Kapitel 12.1

Literaturverzeichnis III

[10] Kleinöder, J. ; Schröder-Preikschat, W. :
Betriebssystemmaschine.
In: [12], Kapitel 5.3

[11] Kleinöder, J. ; Schröder-Preikschat, W. :
Speicherzuteilung.
In: [12], Kapitel 12.2

[12] Kleinöder, J. ; Schröder-Preikschat, W. ; Lehrstuhl Informatik 4 (Hrsg.):
Systemprogrammierung.
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien)

[13] Levy, H. M. ; Lipman, P. H.:
Virtual Memory Management in the VAX/VMS Operating System.
In: *IEEE Computer* 15 (1982), März, Nr. 3, S. 35–41

[14] Stallings, W. :
Operating Systems: Internals and Design Principles.
Prentice Hall, 2001

Größe der Menge residenter Seiten

Linux 3.2.0-4-amd64

```
wosch@faui40 102$ ps
  PID TTY          TIME CMD
26125 pts/11   00:00:00 csh
28439 pts/11   00:00:00 ps
wosch@faui40 103$ pidstat -r -p 26125 1
09:02:14          PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
09:02:15        26125      0.00      0.00   10216   2028   0.00  csh
09:02:16        26125      0.00      0.00   10216   2028   0.00  csh
09:02:17        26125      0.00      0.00   10216   2028   0.00  csh
^C
wosch@faui40 104$ pidstat -r -p SELF 1
09:15:45          PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
09:15:46        24996     14.00      0.00    4128    756   0.00  pidstat
09:15:47        24996     12.00      0.00    4128    788   0.00  pidstat
09:15:48        24996      4.00      0.00    4128    788   0.00  pidstat
^C
wosch@faui40 105$ getconf PAGESIZE
4096
```

- `csh` ▪ $RSS = 2028\,KiB = 507$ Seiten á 4096 Bytes
- `pidstat` ▪ $RSS = 756\,KiB = 189$ Seiten initial, dann 197 Seiten á 4 KiB
- die Menge residenter Seiten der Prozesse ist variabel (s. `pistat`) — Linux verwaltet aber keine Arbeitsmengen, die viel variabler wären