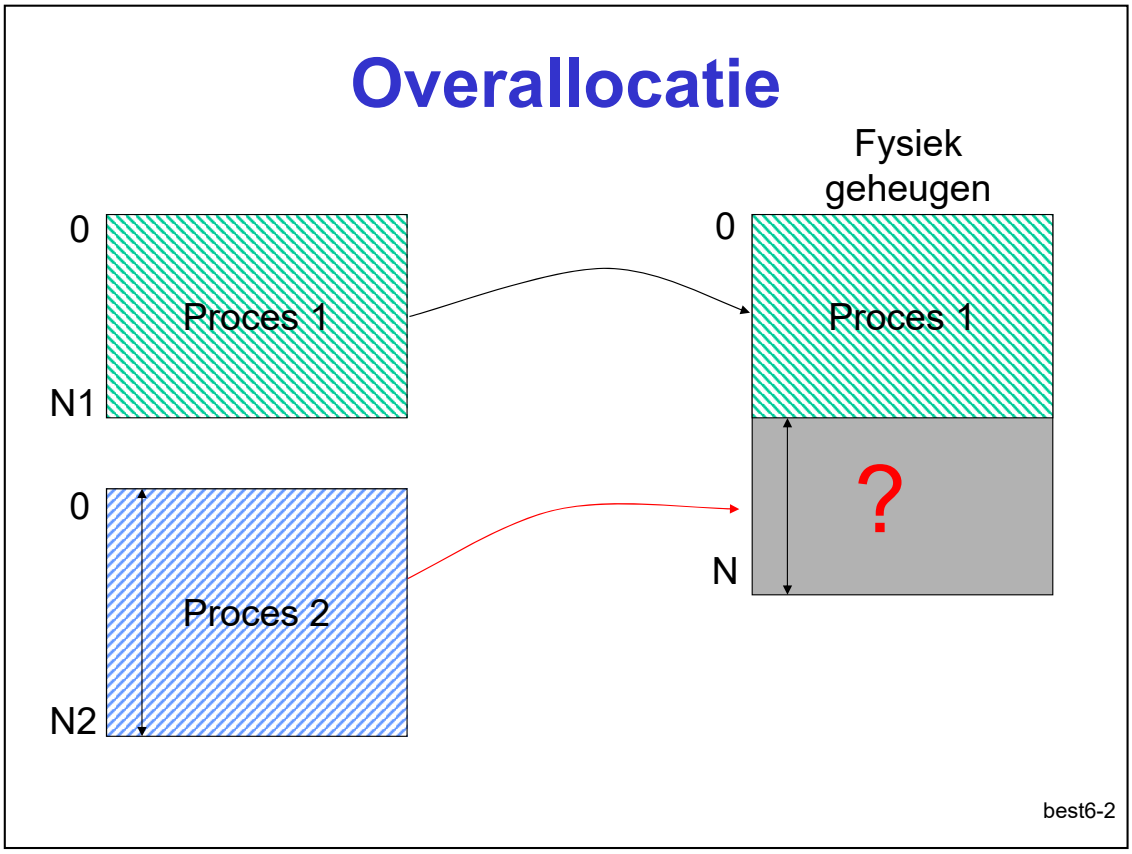
Les 6: Virtueel geheugen

“UNIX is simple. It just takes a genius to understand its simplicity.”
– *Dennis Ritchie*

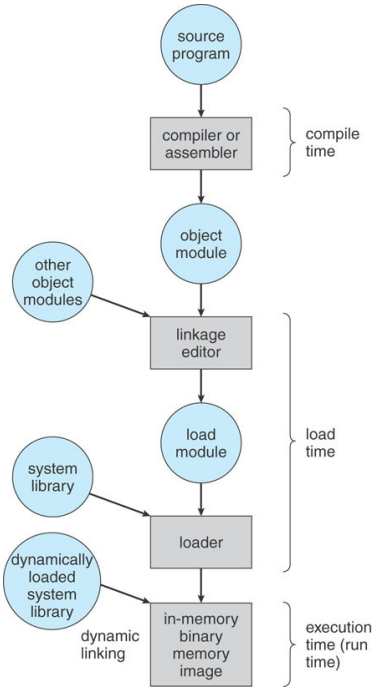
best6-1



Mechanismen zoals paginering en segmentatie vormen een afdoende oplossing voor het probleem van de fragmentatie van het geheugen, en de onderlinge afscherming van de processen. Dit laat ons toe om tot de laatste byte beschikbaar geheugen toe te wijzen aan de processen. Het is echter niet omdat een blok geheugen toegewezen werd aan een proces, dat het ook effectief zal gebruikt worden. Zo zal alle code die met de initialisatie te maken heeft, na het begin van het programma niet meer gebruikt worden. Eigenlijk is er geen reden waarom dit fysiek geheugen na die fase in de uitvoering van het programma niet zou kunnen toegewezen worden aan een ander proces. Het zal toch niet meer gebruikt worden. Als we dat doen, wordt het mogelijk om de som van de geheugenbehoeften van alle processen samen groter te laten worden dan de beschikbare hoeveelheid fysiek geheugen. Men noemt dit overallocatie.

Overzicht

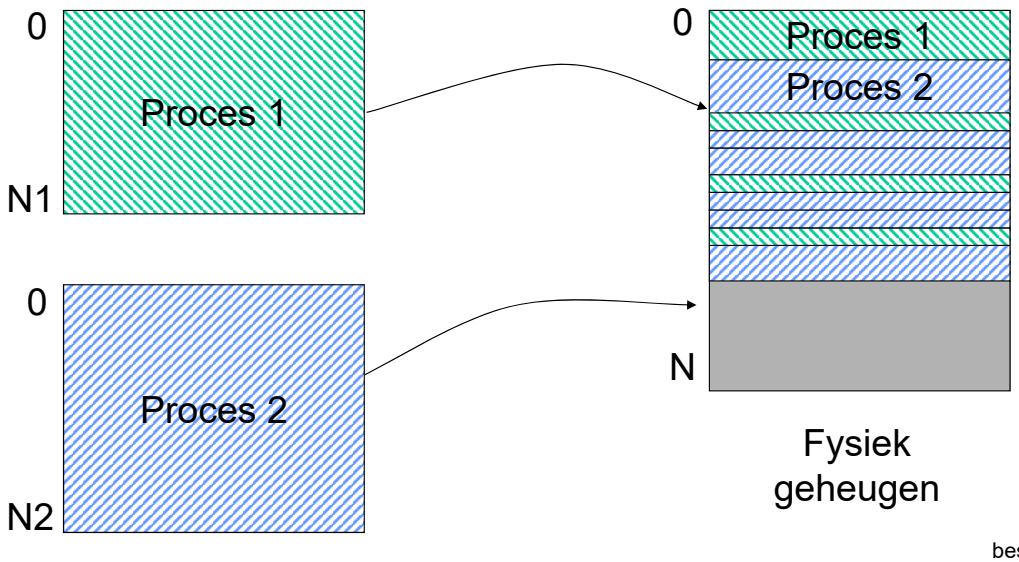
- Overallocatie
- Manuele methoden
  - Dynamisch laden
  - Dynamisch linken
  - Overlays
- Automatische methoden
  - Swapping
  - Virtueel geheugen
- Eindbeschouwingen



best6-3

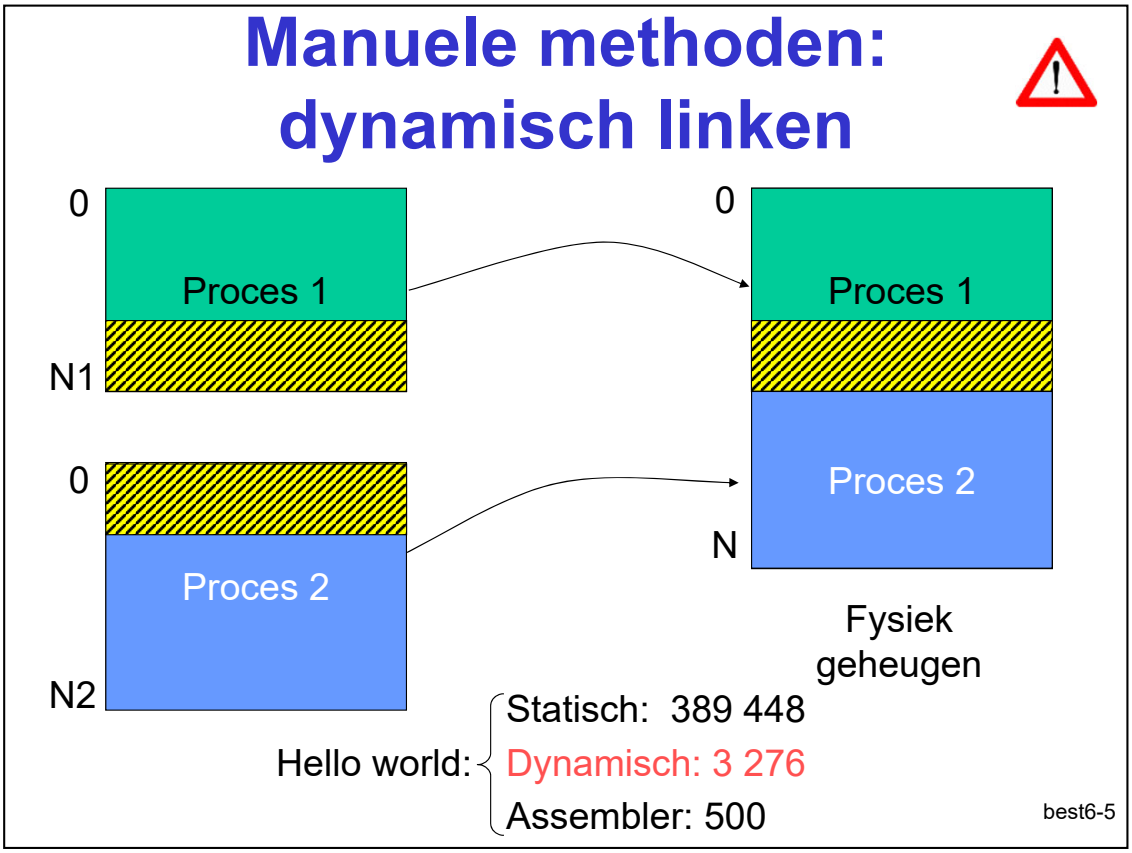
Er bestaan twee manieren om aan overallocatie te doen: manueel en automatisch. Onder de manuele methoden worden het dynamisch laden, dynamisch linken en het gebruik van overlays gerekend. Onder de automatische methoden valt alles wat men klassiek virtueel geheugen noemt. Het doel van beide technieken is hetzelfde: het uitvoeren van een proces in minder fysiek geheugen dan vereist door de omvang van het proces. We beginnen met de manuele methoden.

Manuele methoden: dynamisch laden



Hierbij wordt een programma in zijn geheel gecompileerd en gelinkt, maar niet in zijn geheel geladen. De niet-geladen delen zullen pas ingeladen worden (met aanpassing van alle relevante adrestabellen) indien dit nodig is. Onderdelen van het programma die niet gebruikt worden zullen op deze manier niet geladen worden. Dit is bijzonder nuttig bij zeer grote routine die maar sporadisch opgeroepen worden (denk b.v. maar aan de talrijke menu-opties die je in de praktijk zelden of niet gebruikt).

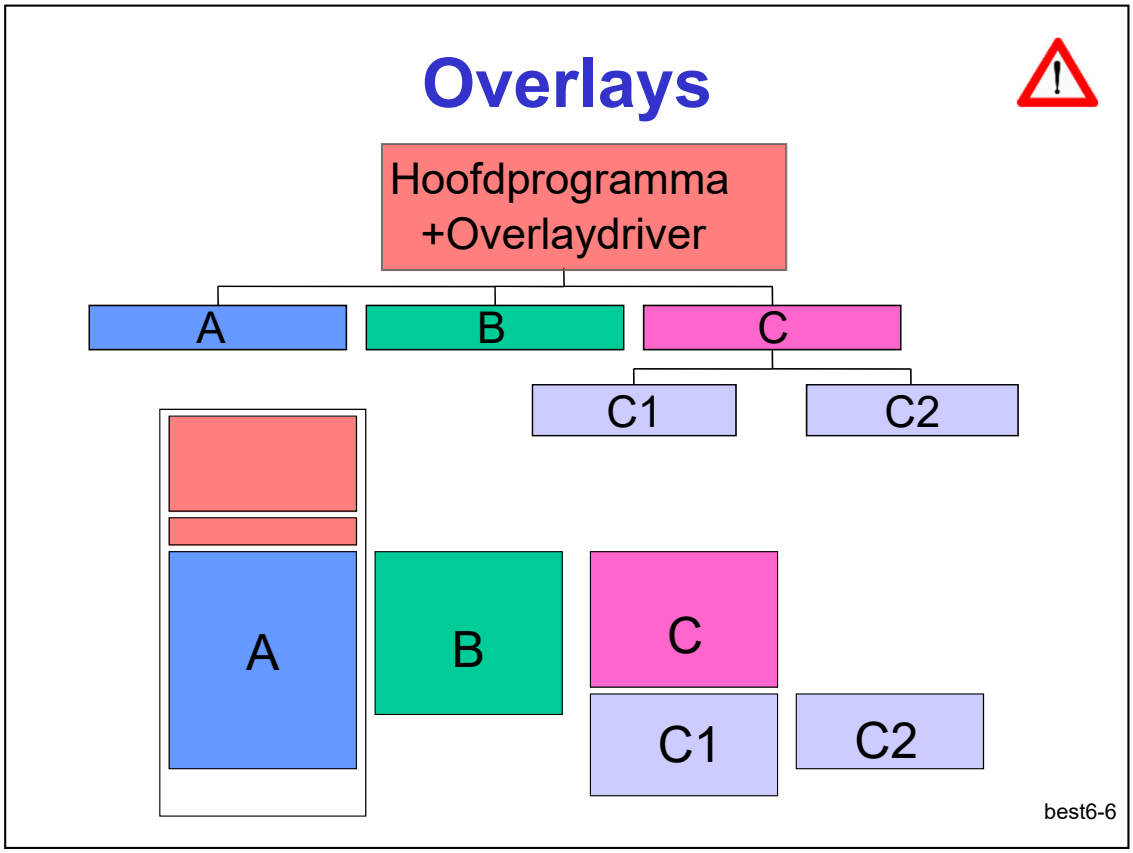
Dynamisch laden kan volledig in gebruikersmode gerealiseerd worden zonder tussenkomst van het besturingssysteem. Bij het opstarten van het programma worden enkel de meest courant gebruikte routines ingeladen. De minder frequent gebruikte routines worden vervangen door een stub, die de betrokken routine moet inladen in het geheugen, ze reloceert, en dan een controletransfer naar de net ingeladen routine uitvoert. Op dezelfde manier kan men desgewenst een routine ook weer verwijderen uit het geheugen.



Hierbij wordt een applicatie niet volledig gelinkt. Stukken code die als ‘shared library’ bekend staan worden niet opgenomen door de linker. In de plaats daarvan worden er opnieuw stubs in de code geplaatst. Het resulterend programma kan hierdoor aanzienlijk kleiner worden (sommige programmabibliotheken kunnen zeer groot zijn). Bij de uitvoering zal de eerste keer dat een dergelijke routine opgeroepen wordt de stub uitgevoerd worden die dan verantwoordelijk is om na te gaan of de gevraagde routine zich reeds in het geheugen bevindt. Indien deze routine zich nog niet in het geheugen bevindt, wordt ze ingeladen. Indien de routine zich wel reeds in het geheugen bevindt, wordt de stub vervangen door het adres van de routine en kan ze vanaf dat ogenblik zonder verdere vertraging gebruikt worden. Dynamisch linken is een techniek die vaak gebruikt wordt bij systeembibliotheken die men om die reden vaak permanent in het geheugen geladen houdt. Ook Windows maakt er uitgebreid gebruik van (DLL = Dynamic Link Library). Deze bibliotheken kunnen ook vernieuwd worden, zonder de programma's te moeten hercompileren. Om fouten tegen te gaan zullen ze wel een versienummer moeten krijgen en moeten nieuwe versies van een bibliotheek ook de oude versies blijven ondersteunen.

Dynamisch linken vereist een ondersteuning van het besturingssysteem omdat processen normaal gezien niet zomaar aan geheugen kunnen dat door andere processen gealloceerd werd (zoals de gemeenschappelijke bibliotheken).

Voor de geïnteresseerden: het kortst mogelijke programma
http://www.phreedom.org/research/tinype/ (97 bytes)



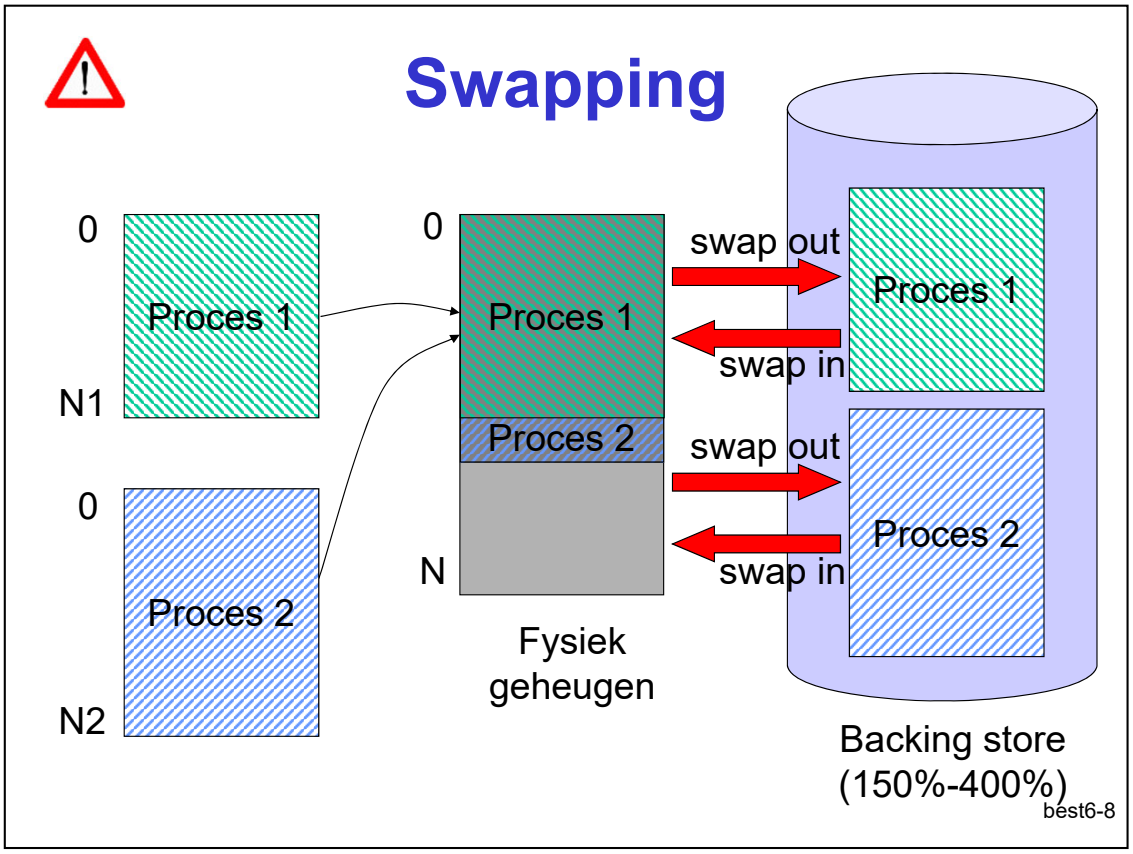
Bij overlays wordt een programma door de programmeur manueel opgesplitst in een aantal routines die onderling geen interactie hebben. De programmeur geeft aan welke stukken van de code op elk ogenblik van de uitvoering in het geheugen moeten zitten. Het programma bestaat dan uit die blokken die gedurende de volledige uitvoering in het geheugen moeten blijven, aangevuld met een zogenaamde ‘overlay driver’ die de transiënte blokken moet in- en uitladen.

Overlays vereisen geen speciale ondersteuning door het besturingssysteem. Modernere technieken hebben het gebruik van overlays grotendeels overbodig gemaakt.

Overzicht

- Overallocatie
- Manuele methoden
  - Dynamisch laden
  - Dynamisch linken
  - Overlays
- Automatische methoden
  - Swapping
  - Virtueel geheugen
- Eindbeschouwingen

best6-7



Als de voorgaande methoden niet volstaan om alle actieve processen in het geheugen te houden zal men bepaalde processen tijdelijk uit het geheugen moeten verwijderen om andere processen de gelegenheid te geven om te kunnen uitvoeren.

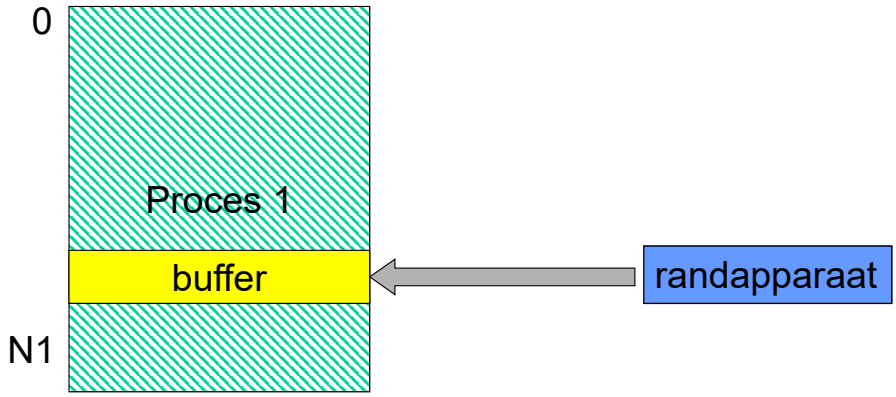
Swapping laat toe om ruimte in het fysiek geheugen vrij te maken door alle ruimte ingenomen door één proces tijdelijk op schijf weg te schrijven. Een proces kan zich dus zowel in het fysiek geheugen als op de schijf bevinden. Om kunnen uitgevoerd te worden moet het zich echter in het fysiek geheugen bevinden. Op schijf kan het zich in de toestand 'klaar' of 'wachtend' bevinden. Alle processen kunnen in de swapruimte over hun eigen geheugengebied beschikken.

Op systemen waar de binding tussen de logische en fysieke adressen tijdens de compilatie/linken of tijdens het inladen van een programma gebeurt, moet een proces bij het inswappen op precies dezelfde plaats in het geheugen terechtkomen, hetgeen een belangrijke beperking inhoudt. Voor systemen waarbij de binding tijdens de uitvoering gebeurt stelt dit probleem zich natuurlijk niet en kan een proces op een andere plaats in het fysiek geheugen heringeladen worden.

Swapping van complete processen wordt in de praktijk nog maar weinig toegepast. Het wordt in Unix nog gebruikt om een aantal processen in hun geheel uit het geheugen te verwijderen op het ogenblik dat het systeem overbelast geraakt. Oude versies van Windows gebruikten swapping om tussen twee applicaties om te schakelen. Op het ogenblik dat een applicatie de focus kreeg, werd het van schijf ingeladen.

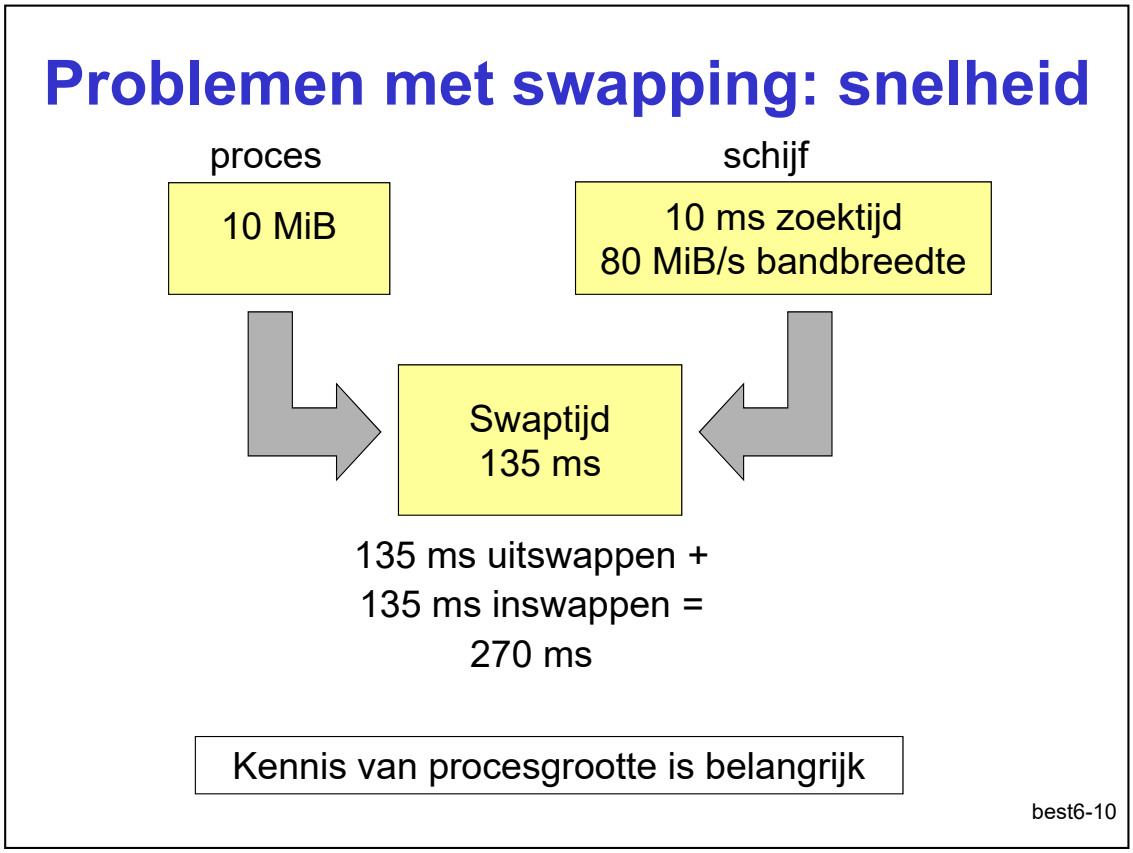
Swapping zal onmisbaar blijken te zijn voor de ondersteuning van de automatische methoden om lokaliteit uit te buiten, en ligt, in combinatie met paginering en segmentatie, aan de basis van virtueel geheugen.

Problemen met swapping: IO



best6-9

Een probleem dat zich voordoet bij swapping is dat een proces volledig uit het geheugen kan verdwijnen terwijl het staat te wachten op invoer. Men moet ervoor zorgen dat er geen hangende IO-operaties meer zijn die nadat een proces uitgeswapt is, toch nog zouden kunnen schrijven naar bepaalde gegevensstructuren. Men kan dit probleem oplossen door ervoor te zorgen dat alle buffers waarin men invoer verwacht eigendom zijn van de kern van het besturingssysteem en deze niet kunnen uitgeswapt worden. Dit veroorzaakt wel wat extra overlast doordat de ontvangen gegevens nogmaals zullen moeten gekopieerd worden naar de buffers van het applicatieprogramma.

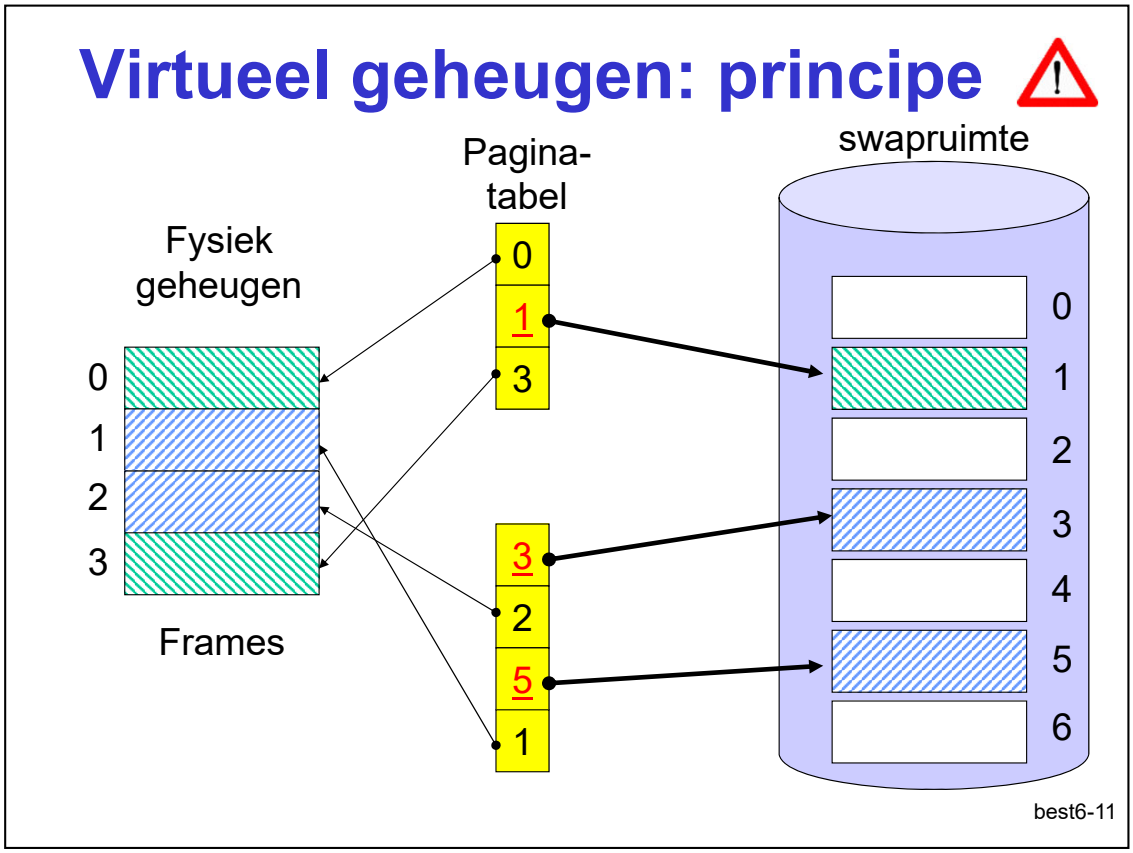


Doordat een proces zich ofwel volledig in het geheugen ofwel volledig op schijf moet bevinden, zijn de swaptijden evenredig met de grootte van het proces, bovenop de tijd nodig om de kop van de schijf te positioneren. Transfertijden van 0,1 s tot 1 s voor grote programma's zijn geen uitzondering. Vandaar dat het bij swapping essentieel is te weten hoe groot een programma nu precies is om de hoeveelheid te transfereren informatie zo klein mogelijk te houden. Dit is één van de redenen waarom dynamisch geheugen expliciet moet aangevraagd en vrijgegeven worden. Het besturingssysteem kan op die manier bijhouden hoe groot een proces precies is (en niet hoe groot het zou kunnen worden).

Anderzijds moet men zich de vraag stellen of het inswappen en uitswappen van een compleet proces wel zinvol is. De tweede keer dat een proces uitgeswapt wordt, zal het de eerder uitgeswapte versie in de swapruimte overschrijven. Alle stukken die niet veranderd werden (b.v. code) hoeven in principe niet teruggeschreven te worden. Bovendien is het omwille van de IO-buffers niet altijd wenselijk om alles terug te schrijven, maar zou het beter zijn mochten we een stukje van het proces in het fysiek geheugen kunnen houden.

Omgekeerd is ook het inswappen van een volledig proces niet zinvol. Sommige stukken zullen toch niet meer gebruikt worden (b.v. initialisatiecode). Het inswappen van procesdelen die achteraf niet gebruikt worden is een onnodige inspanning.

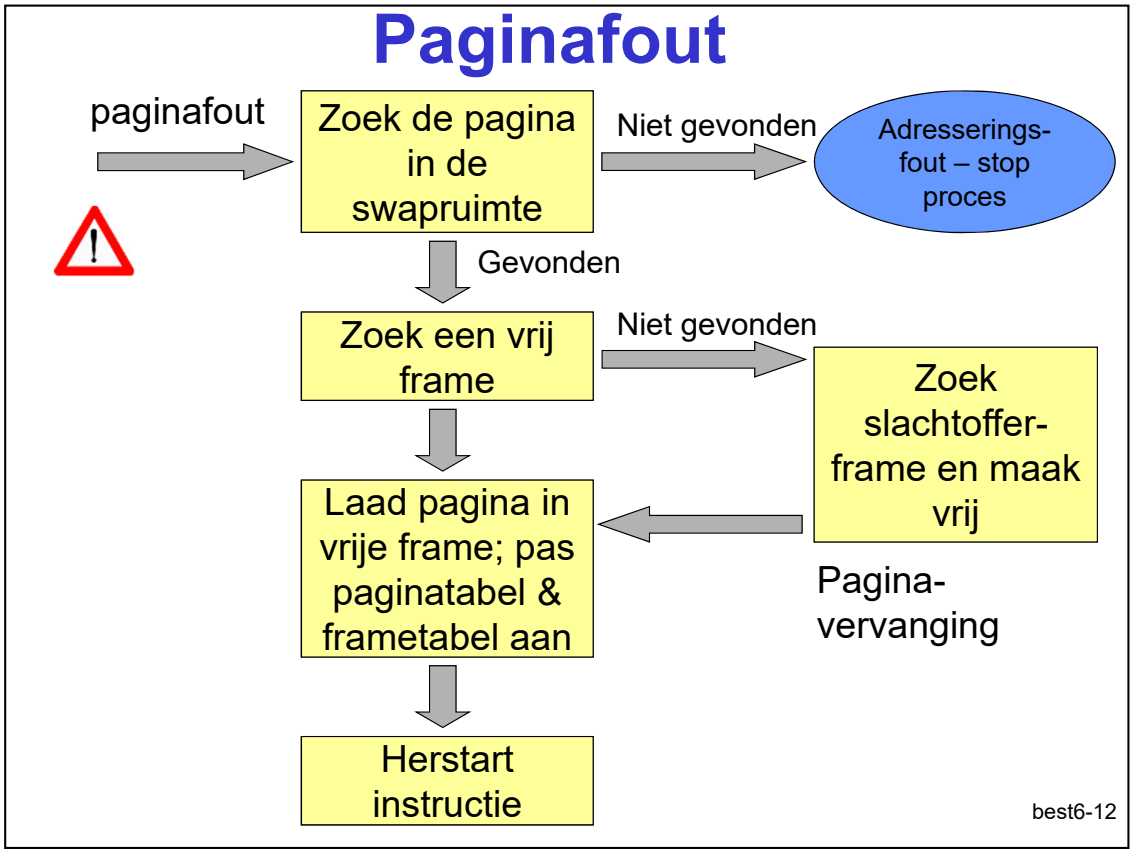
Bij virtueel geheugen zal men de processen niet compleet inswappen of uitswappen, maar zal men individuele pagina’s of segmenten uitwisselen met de swapruimte. De adresvertalingshardware zal daarbij goed van pas komen.



Om het onnodig inswappen en uitswappen automatisch te kunnen wegwerken moeten we beschikken over gedetailleerde informatie over het gedrag van het programma. Indien de swapruimte nu in plaats van hele processen pagina's of segmenten zou bevatten dan kunnen we deze stuk voor stuk in- en uitswappen. De referentiebits en dirty bits van de pagina's kunnen ons dan verklappen of een pagina of segment gebruikt of veranderd is geweest.

In plaats van een heel proces ineens in en uit te swappen, zullen enkel die onderdelen (pagina's of segmenten) die effectief gebruikt worden geswapt worden. Dit zal toelaten om (i) de hoeveelheid te transfereren informatie tussen het fysiek geheugen en de swapruimte drastisch te reduceren, (ii) per proces dus minder geheugen te gebruiken, en bijgevolg (iii) meer processen (gedeeltelijk) in het geheugen geladen te houden. Verder zal, doordat processen nu minder geheugen vereisen om uit voeren, een proces tijdens het wachten niet noodzakelijk (volledig) uitgeswapt worden hetgeen de doorlooptijd van een proces zal verbeteren.

Om virtueel geheugen te kunnen laten ondersteunen door het pagineringsmechanisme of door het segmenteringsmechanisme moet men ervoor zorgen dat de elementen in de paginatabellen nu niet enkel naar het fysiek geheugen kunnen wijzen, maar ook naar de swapruimte. De valid bit zal aangeven of een adres een geheugenadres is of een schijfadres (of ongeldig). Bij het gebruiken van een adres op een pagina die zich op dat ogenblik niet in het fysiek geheugen bevindt, zal er een **paginafout** (onderbreking) gegenereerd worden door de MMU. Op analoge manier zal bij het gebruiken van segmenten met de valid bit = 0 een segmentfout optreden.



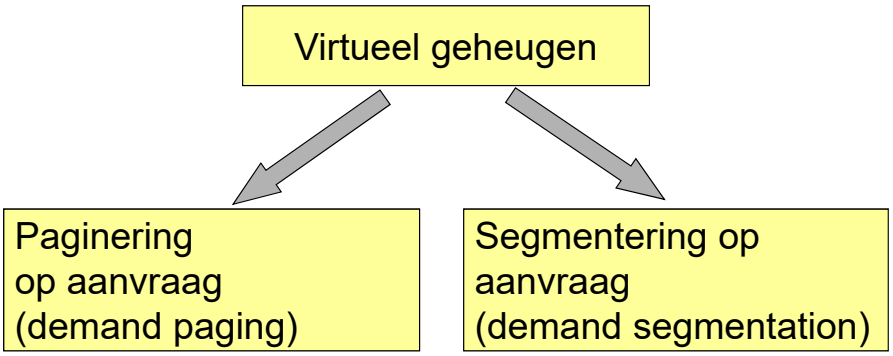
Bij het optreden van een paginafout wordt de zgn. paginator opgeroepen. Deze zal nagaan of de pagina die geassocieerd is met de paginafout zich in de swapruimte bevindt of niet. Indien niet, hebben we te maken met een echte adresseringsfout en wordt het programma onderbroken.

Indien de swapruimte de betrokken pagina wel bevat, moet er een vrije frame gezocht worden om die pagina van de swapruimte naar het fysiek geheugen te brengen. Daarbij kan het voorkomen dat een reeds ingeladen pagina moet wijken (de slachtofferpagina wordt geselecteerd door het paginavervangingsalgoritme, zie verder).

Tenslotte wordt de pagina van de swapruimte overgebracht naar het fysiek geheugen, worden de paginatabellen aangepast om de nieuwe situatie te reflecteren (misschien 1 pagina uitgeswapt, en 1 pagina ingeswapt), en wordt de instructie die de paginafout veroorzaakte opnieuw uitgevoerd.

Merk hierbij op dat één instructie verschillende paginafouten kan veroorzaken. Een load-instructie kan b.v. een paginafout veroorzaken bij het ophalen van de instructie, en bij het inladen van de waarde. Indien de instructie en het gegeven niet gealigneerd zijn, en toevallig over een paginagrens vallen, kan dit oplopen tot 4 paginafouten in 1 instructie. Bij instructie met meer dan 1 geheugenoperand kan het aantal nog groter zijn. Instructies die geheugenblokken ineens kunnen verplaatsen kunnen in dit verband bijzondere problemen stellen. Hedendaagse systemen proberen dit soort van problemen te vermijden door te eisen dat de instructies en de gegevens gealigneerd moeten zijn, en door enkel maar geheugenoperandi toe te laten in load- en store-instructies.

Virtueel geheugen: implementatie



Windows

Linux

…

Burroughs

OS/2

best6-13

Virtueel geheugen wordt meestal geïmplementeerd bovenop een pagineringssysteem. Het kan echter ook bovenop een segmenteringssysteem geïmplementeerd worden zoals dit bijvoorbeeld in OS/2 het geval is. Gezien segmenten een variabele lengte hebben is het implementeren van virtueel geheugen aan de hand van segmentering iets moeilijker. Segmentfouten treden wel minder vaak op en de hardware die men nodig heeft om segmentering te ondersteunen is ook minder uitgebreid en virtueel geheugen op basis van segmentering wordt om die reden soms nog toegepast.

De meest populaire vorm van virtueel geheugen is paginering op aanvraag (demand paging). In dat geval spreekt men niet meer van een swapper, maar van een paginator. Men spreekt soms ook van een luie swapper, een die pas swapt indien het echt niet anders kan, en dan nog wel pagina per pagina.

Bij pure paginering op aanvraag begint de uitvoering van een programma met 0 frames. De uitvoering van de eerste instructie zal minstens één paginafout veroorzaken. Tijdens de verdere uitvoering van het programma zullen gaandeweg meer pagina's in frames geladen worden. Van zodra alle frames die aan het proces werden toegewezen in gebruik zijn, zal de paginator bij een paginafout een pagina moeten selecteren die baan moet ruimen om een nieuwe pagina in een frame te kunnen laden. Op analoge manier kan men ook segmentering op aanvraag implemeteren.

Prestatieverlies

Globale gemiddelde geheugentoegangstijd

(1 - p) x ma + p x paginafouttijd

paginafouttijd = onderbrekingstijd
[+ swap-out tijd] +
swap-in tijd +
herstarttijd

ma = 2 ns, paginafouttijd = 8 ms

Vertraging van niet meer dan 10%:
p < 1 / 40 000 000

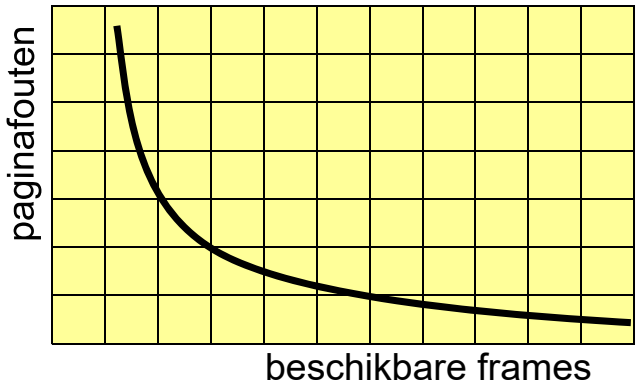
best6-14

Het gebruik van virtueel geheugen heeft een aanzienlijke impact op de prestatie van een programma, een impact die niet zomaar kan weggewerkt worden door b.v. het invoeren van een TLB. De swapruimte is nu eenmaal een mechanisch onderdeel van de computer en is om die reden traag.

Indien p de kans op het optreden van een paginafout is, en indien ma de geheugentoegangstijd naar het fysiek geheugen is (zonder de caches), dan wordt de globale gemiddelde geheugentoegangstijd gegeven door (1-p) x ma + p x paginafouttijd (ma is memory access time) waarbij de paginafouttijd bestaat uit de verschillende componenten zoals in de afbeelding aangegeven. Indien we de gemiddelde vertraging van het fysiek geheugen willen beperken tot 10% dan zal een paginafout een relatief zeldzame gebeurtenis moeten zijn: 1 paginafout per 40 000 000 toegangen naar het geheugen. Als de processor 1,2 miljard instructies per seconde uitvoert, spreken we over 30 paginafouten per seconde.

Paginavervanging

Doel: zo klein mogelijk aantal paginafouten bij een gegeven aantal frames



Anomalie van Belady

Referentieketen:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

best6-15

Er bestaan verschillende paginavervangingsalgoritmen. Al deze algoritmen hebben slechts één doel: het zo laag mogelijk houden van het aantal paginafouten. Paginavervangingsalgoritmen kunnen vergeleken worden door ze toe te passen op een zogenaamde referentieketen, dit is een opeenvolging van paginanummers waarvan de pagina's door het proces gebruikt worden. Gegeven een bepaald aantal frames kan er kan gekeken worden welk van de vervangingsalgoritmen het beste presteert.

Als algemene regel kan men stellen dat voor een goed vervangingsalgoritme het aantal paginafouten moet afnemen indien het aantal frames (systeemmiddelen) toeneemt (zie afbeelding). Zoals we later zullen zien geldt deze regel echter enkel voor die paginavervangingsalgoritmen die ook stapelalgoritmen zijn. Voor de andere algoritmen kan de anomalie van Belady optreden waarbij er ondanks het groter aantal frames toch ook meer paginafouten kunnen optreden (zie verder).

Paginavervangingsalgoritmen

- FIFO: First-In First-Out
- OPT: Optimaal algoritme
- LRU: Least Recently Used
- PLRU: Pseudo-LRU
- TK: Tweede kans algoritme
- LFU/MFU: Least/Most Frequently Used
- RR: Random Replacement

best6-16

De bovenstaande paginavervangingsalgoritmen zullen besproken worden, waar zinvol geïllustreerd op een referentieketen.

First-in First-out (FIFO)

FIFO

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	0	1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
	7	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
		7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
P	P	P	P		P	p	P	p	p	p			p	p			p	p	p

3 frames

6+9=15

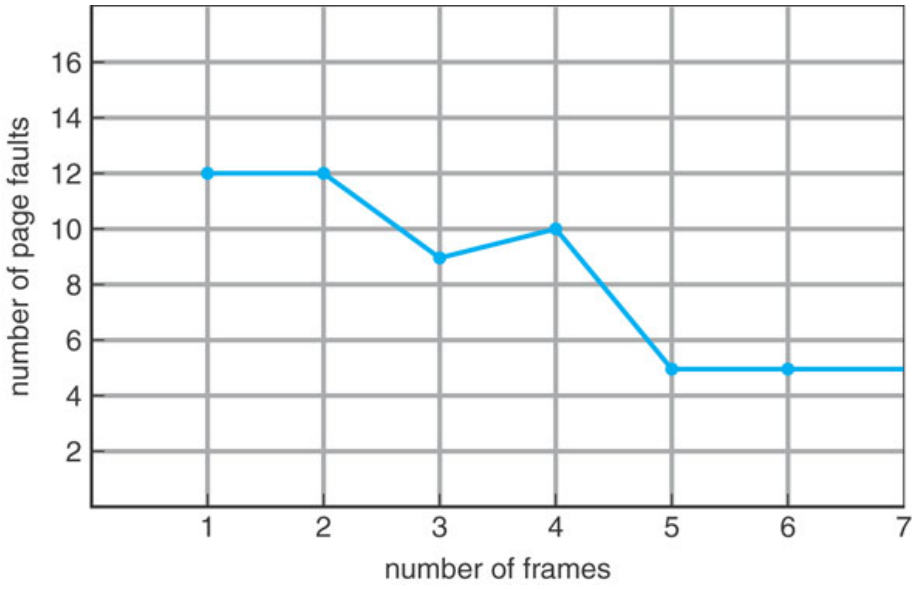
best6-17

Dit is het eenvoudigste algoritme. Indien er een slachtoffer voor vervanging moet gezocht worden kiest men steeds die pagina die zich reeds het langst in het geheugen bevindt. Hiertoe neemt men de pagina's op in een wachtlijn en kiest men steeds de pagina die zich aan de kop van de wachtlijn bevindt. Het is echter best mogelijk dat deze pagina een zeer actief gebruikte pagina is. Na uitgeladen te zijn zal hij in dat geval onmiddellijk terug ingeladen worden. Bij dit algoritme kan de anomalie van Belady optreden.

In de bovenste rij staat de referentieketen vermeld. Daaronder staan de drie pagina's die zich **na** het uitvoeren van een referentie naar de betrokken pagina in de frames zullen bevinden. Op de onderste rij wordt aangegeven wanneer er een paginafout optreedt. Met de letter 'P' wordt aangegeven dat het om een niet te vermijden paginafout gaat (een koude paginafout die dient om een pagina voor de eerste keer in het fysiek geheugen te brengen). De volgende keren dat er een paginafout optreedt op dezelfde pagina wordt deze met 'p' genoteerd. Dit is een capaciteitspaginafout die te wijten is aan een tekort aan frames.

Indien men het aantal frames opvoert tot het aantal verschillende pagina's die door een proces effectief gebruikt worden, dan zal het aantal paginafouten dalen tot het aantal koude paginafouten. We noteren het aantal paginafouten als P+p.

Anomalie van Belady



Referentieketen: 1 2 3 4 1 2 5 1 2 3 4 5

best6-18

De anomalie van Belady treedt op wanneer men ondanks een groter aantal beschikbare frames toch een groter aantal paginafouten noteert. In de praktijk is dit fenomeen niet zo heel belangrijk en treedt het maar op in zeer specifieke geconstrueerde gevallen. Over langere referentieketens is het bijna niet op te merken (omdat de toevallige toename met een paar paginafouten op enkele plaatsen meestal niet opweegt tegen de veel grotere reductie van het aantal paginafouten als gevolg van het groter aantal frames in de rest van de referentieketen). Op kleine geïsoleerde gevallen kan men het fenomeen wel vaststellen zoals geïllustreerd in de bovenstaande afbeelding en met de berekening van de volgende dia.

Opdat het fenomeen zich zou kunnen voordoen moet er voldaan zijn aan de voorwaarde dat de pagina's die zich op een bepaald ogenblik in de frames bevinden bij n frames geen deelverzameling zijn van de pagina's die zich in de frames zouden bevinden bij n+1 frames.

Algoritmen waarbij dit wel het geval is (en waar het fenomeen zich dus niet kan manifesteren) worden stapelalgoritmen genoemd. Bij stapelalgoritmen kan men nooit méér paginafouten krijgen bij meer frames omdat de pagina's bij n+1 frames zeker de pagina's bij n frames zullen bevatten en er dus onmogelijk paginafouten kunnen optreden bij n+1 frames die er al niet waren bij n frames.

Anomalie van Belady

FIFO

1	2	3	4	1	2	5	1	2	3	4	5	3 frames
1	2	3	4	1	2	5	5	5	3	4	4	
	1	2	3	4	1	2	2	2	5	3	3	
		1	2	3	4	1	1	1	2	5	5	
P	P	P	P	p	p	P			p	p		5+4=9

FIFO

1	2	3	4	1	2	5	1	2	3	4	5	4 frames
1	2	3	4	4	4	5	1	2	3	4	5	
	1	2	3	3	3	4	5	1	2	3	4	
		1	2	2	2	3	4	5	1	2	3	
			1	1	1	2	3	4	5	1	2	
P	P	P	P			P	p	p	p	p	p	5+5=10

Niet voldaan aan $\forall t.F(3,t) \subseteq F(4,t)$

best6-19

Hier wordt het voorbeeld van de vorige dia uitgewerkt. Zoals af te lezen is, is er bij vier frames 1 paginafout meer dan bij 3 frames. In de drie gemerkte kolommen is er niet voldaan aan de stapeleigenschap. De uitdrukking F(3,t) staat voor de inhoud van de frameset bij drie frames en op tijdstip t.

Optimaal algoritme (OPT)

OPT

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	3 frames
7	0	1	2	2	3	3	4	4	4	0	0	0	1	1	1	1	7	7	7	
	7	0	1	1	2	2	3	3	3	3	3	3	0	0	0	0	1	1	1	
		7	0	0	0	0	2	2	2	2	2	2	2	2	2	2	0	0	0	
P	P	P	P		P		P			p			p				p			6+3=9

best6-20

Net zoals bij de procesplanning kan men ook bij paginavervanging op zoek gaan naar het optimale algoritme, dit is het algoritme dat voor een gegeven aantal frames het minimale aantal paginafouten veroorzaakt. Net zoals bij de procesplanning betreft het ook hier een niet-causaal algoritme dat in de praktijk dus niet implementeerbaar zal zijn.

Het algoritme stelt dat die pagina moet vervangen worden die de langste tijd niet gebruikt zal worden. OPT is niet onderhevig aan de anomalie van Belady omdat de n pagina's die snelst opnieuw gebruikt zullen worden logischerwijze een deelverzameling zullen zijn van de n+1 pagina's die snelst opnieuw gebruikt zullen worden.

Least Recently Used (LRU)

LRU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	3 frames
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	
		7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7	
P	P	P	P		P		P	p	p	p			p		p		p			6+6=12

best6-21

Dit algoritme tracht de eenvoud van FIFO te combineren met de efficiëntie van OPT. In plaats van die pagina te kiezen die het langst niet gebruikt zal worden, wordt die pagina gekozen die het langst niet gebruikt is geweest (liever dan die pagina die het langst geleden werd ingeladen). Hierbij gaat men ervan uit dat de recente geschiedenis een goede voorspelling inhoudt voor de nabije toekomst.

Indien men de referentieketen omkeert en men daarop OPT toepast bekomt men LRU (met uitzondering van de uiteinden waar verschillen kunnen optreden). In de praktijk wordt LRU veel gebruikt voor theoretische studies, en worden afgeleiden van LRU vaak gebruikt in werkelijke implementaties.

LRU vertoont geen anomalie van Belady. Omdat de verzameling van pagina's die zich in het geheugen bevinden indien er n frames ter beschikking zijn op elk ogenblik een deelverzameling is van de pagina's die zich in het geheugen bevinden bij het aantal frames n+1. Dit is normaal: de n meest recent gebruikte pagina's vormen steeds een deelverzameling van de n+1 meest recent gebruikte pagina's.

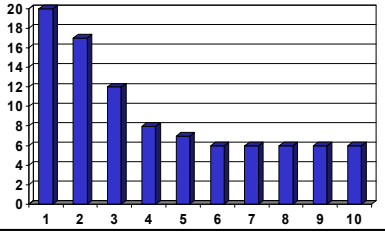
Afstandsketen

LRU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	∞ frames
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	
		7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7	
			7	7	1	1	2	3	0	4	4	4	0	0	3	3	2	2	2	
					7	7	1	1	1	1	1	1	4	4	4	4	3	3	3	
							7	7	7	7	7	7	7	7	7	7	4	4	4	
P	P	P	P		P		P													6+0=6
∞	∞	∞	∞	3	∞	2	∞	4	4	4	2	3	5	2	4	3	6	3	3	

D_1	D_2	D_3	D_4	D_5	D_6	D_∞
0	3	5	4	1	1	6

6



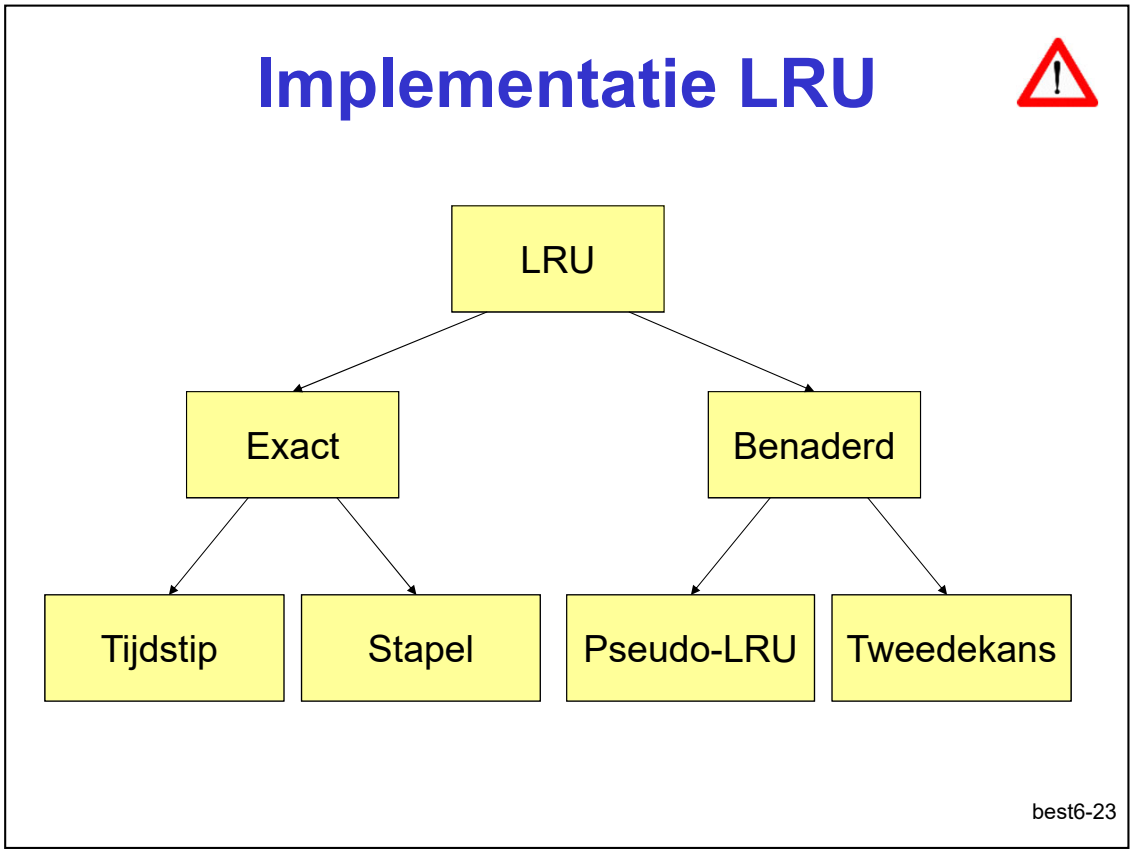
best6-22

Bij LRU kan het aantal paginafouten afgeleid worden uit de zgn. afstandsketen van de referentieketen. De afstandsketen wordt als volgt berekend: vooreerst wordt LRU toegepast zonder framebeperkingen. In het bovenstaande voorbeeld volstaan 6 frames. In dit geval wordt de stapel van paginanummers duidelijk zichtbaar. De bovenste drie rijen van de stapel zijn die pagina's die in frames ingeladen zullen zijn in geval er drie frames zijn, zij vormen dus altijd een deelverzameling van de pagina's die bij 4, 5 of 6 frames zullen ingeladen zijn.

De onderste rij geeft de afstandsketen weer. Deze geeft aan hoe diep een pagina uit de stapel naar boven moet gehaald worden (indien het element niet in de stapel voorkomt hebben we te maken met een koude paginafout en is de afstand ∞).

Vervolgens wordt er een histogram opgebouwd van de afstanden. Dit histogram laat nu toe om snel het aantal paginafouten bij een gegeven aantal frames af te lezen. Bij drie frames zullen b.v. alle afstanden die groter zijn dan 3 een paginafout veroorzaken. In dit geval 4+1+1=6 capaciteitsfouten en 6 koude paginafouten (D_∞). Uit de grafiek volgt dat het aantal paginafouten snel afneemt tot het aantal koude paginafouten en dat er hier geen anomalie van Belady optreedt (kan ook niet bij LRU).

Uit de tabel volgt ook dat door het toevoegen van precies 1 extra frame het aantal paginafouten met 4 kan gereduceerd worden. Het toevoegen van nog meer frames heeft een veel kleiner effect. Indien men een goede afweging wil maken tussen het aantal frames dat men ter beschikking stelt en het aantal paginafouten, zal men op zoek moeten gaan naar het punt in de grafiek waar de extra winst die men boekt door een frame toe te voegen begint af te nemen.

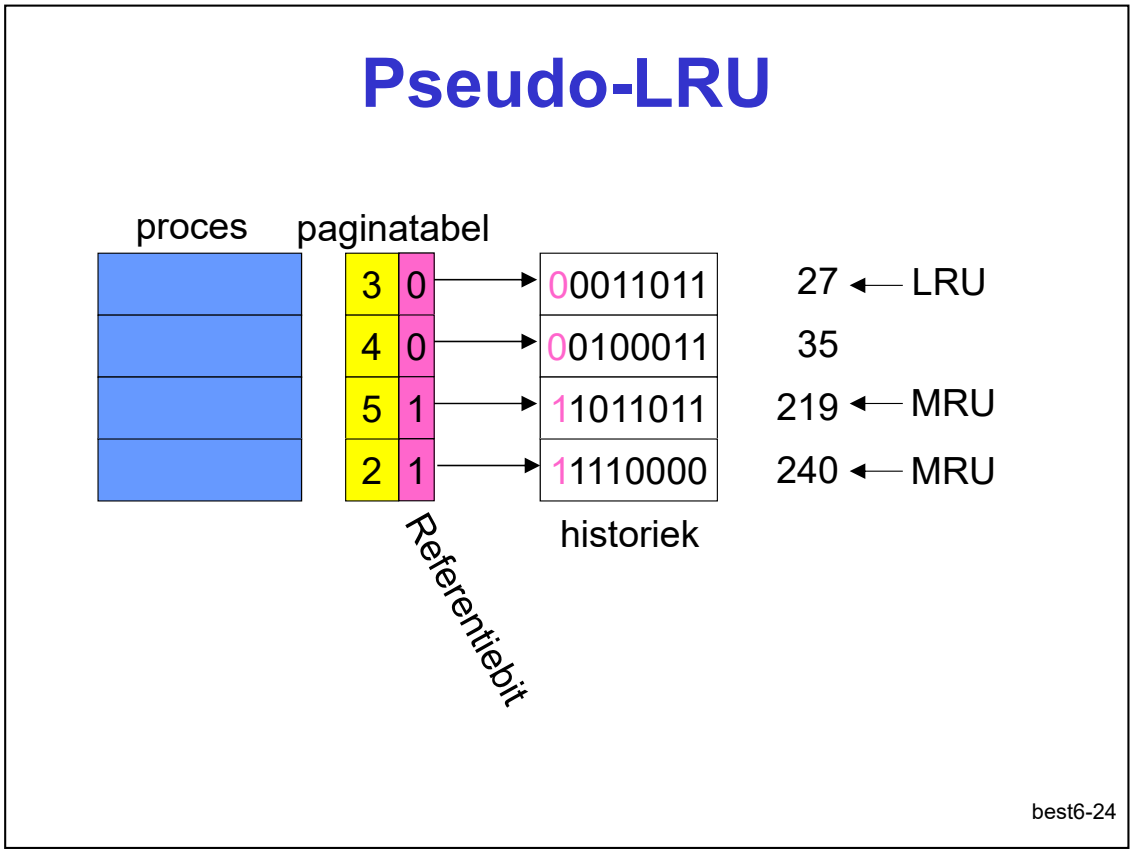


Een exacte implementatie van LRU vereist het bijhouden van informatie tot op het niveau van de individuele geheugentoegangen (ophalen van individuele instructies en de uitvoering van lees- en schrijfoperaties). Tenzij dit automatisch kan gebeuren door de hardware (zoals b.v. in caches), veroorzaakt dit een onaanvaardbare overhead. Er zijn twee methoden om LRU te implementeren.

Per pagina kan men het tijdstip van de laatste toegang bijhouden (extra veld in de paginatabel, aangepast door de hardware). Door die pagina op te zoeken met het oudste tijdstip kent men meteen de LRU-pagina.

Een alternatieve manier is het simuleren van de LRU-stapel. Alle pagina's worden opgenomen in een stapel (gelinkte lijst), en de gebruikte pagina wordt telkens naar de top van de stapel gebracht. Dit is een implementatie van het schema op de vorige dia. Zelfs bij een efficiënte implementatie veroorzaakt het bijhouden van dergelijke datastructuren een enorme overhead.

In de praktijk zijn de vermelde schema's niet zeer bruikbaar. Ze zijn wel zeer populair in simulators. De benaderende methoden komen aan bod op de volgende dia's.

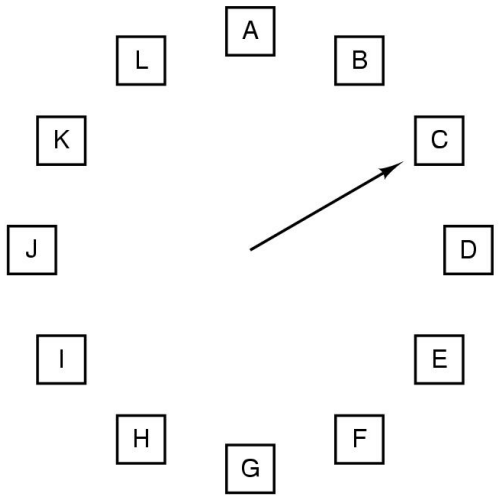


Pure LRU is moeilijk efficiënt te ondersteunen zonder veel bijkomende hardware. In de praktijk zal men LRU dan ook zo goed mogelijk trachten te benaderen. Daarbij kan men de referentiebit gebruiken dat bij de meeste geheugenbeheerssystemen aanwezig is. De referentiebit staat initieel op 0 en komt op 1 te staan van zodra een pagina gerefereerd wordt. Dit laat ons toe om te weten welke pagina's er gebruikt werden, niet in welke volgorde.

Deze volgorde kan gedeeltelijk gereconstrueerd worden door op geregelde tijdstippen een kopie te maken van de referentiebits, de referentiebits terug op 0 te zetten, en een historiek van de recentste referentiebits bij te houden. Indien men de nieuwe referentiebits aan de linker kant van de historiek inschuift kan men door de historiek numeriek te ordenen de LRU pagina bij benadering terugvinden (kleinste waarde).

De tijdsprecisie is beperkt tot de periode van de klokonderbreking, daarbinnen is geen volgorde meer gekend. Alles wat buiten de horizon van het algoritme valt (in dit geval 8 maal de klokperiode), krijgt de waarde 0, en is dus niet meer geordend.

Tweedekansalgoritme (Klokalgoritme)



- Circulaire lijst met pointer die huidige kop van de lijst aangeeft
- Indien referentiebit van C=0: vervang C en zet de referentiebit van de nieuwe pagina op 1; Indien referentiebit van C=1: zet de referentiebit van C op 0; schuif wijzer op naar D
- Eventueel verfijnen met via gebruik van dirty bit

best6-25

Dit algoritme is een verbetering van het eenvoudige FIFO algoritme, en maakt gebruik van de referentiebits. In plaats van een pagina onvoorwaardelijk uit te pagineren indien hij aan de kop van de wachtlijn komt, wordt er gekeken of deze pagina sinds het ogenblik dat hij de laatste keer in aanmerking kwam om uitgepagineerd te worden, gerefereerd werd (referentiebit = 1). Indien dit het geval is, wordt zijn referentiebit op 0 gezet en krijgt deze pagina een tweede kans. Indien dit niet zo is, wordt hij uitgepagineerd en vervangen door de nieuwe pagina. Indien alle pagina's in actief gebruik zijn, degenereert dit algoritme tot FIFO.

Een verfijning van dit algoritme is om niet enkel te kijken naar het referentiebit, maar ook naar de dirty bit. Van de niet-gebruikte pagina's worden dan eerst de niet-gewijzigde pagina's vervangen, en pas nadien de gewijzigde pagina's. Eventueel kan beslist worden om de gewijzigde pagina reeds te kopiëren naar de swapruimte zodat bij een volgende bezoek de dirty bit op 0 staat en de pagina voor vervanging in aanmerking komt.

Tweedekansalgoritme

1	3	6	1	7	3	4	6	7	2
1’	3’	6’	6’	7’	7’	4’	6’	6’	2’
	1’	3’	3’	6	6	3	7	7’	6
		1’	1’	3	3’	7’	4’	4’	7
P	P	P		P		P	p		P

best6-26

n betekent: pagina n ingeladen met referentiebit op 0

n’ betekent: pagina n ingeladen met referentiebit op 1

Een nieuwe pagina heeft altijd referentiebit op 1 (kolom 1,2,3,5,7,8,10)

Een bestaande pagina refereren zet de referentiebit steeds op 1 (kolom 6,9) tenzij hij al op 1 stond (kolom 4)

De overgang van kolom 4 naar 5 bestaat uit vier stappen:

1: oudste pagina weglaten (1’); gaat niet, krijgt een tweede kans en komt bovenaan te staan. Resultaat: 1 6’ 3’

2: dan oudste pagina weglaten (3’); gaat niet, krijgt een tweede kans en komt bovenaan te staan. Resultaat 3 1 6’

3: dan oudste pagina weglaten (6’); gaat niet, krijgt een tweede kans en komt bovenaan te staan. Resultaat 6 3 1

4: dan oudste pagina weglaten (1) gaat wel, en de nieuwe pagina 7 komt bovenaan te staan 7’ 6 3

Tellende algoritmen

- Houdt een teller van het aantal referenties per pagina bij
- Least Frequently Used (LFU): vervangt die pagina met de laagste tellerwaarde
- Most Frequently Used (MFU): vervangt die pagina met de hoogste tellerwaarde
- Not Frequently Used (NFU): past de tellerwaarde maar eenmaal aan per tijdskwantum

best6-27

Hiervoor is het noodzakelijk om het aantal geheugenreferenties te tellen. De pagina's met het kleinste aantal geheugenreferenties (LFU) zijn de eerste kandidaten voor vervanging. Dit algoritme heeft drie nadelen: (i) er is vrij complexe hardware nodig om de tellers te implementeren, (ii) pagina's die ooit zeer intensief gebruikt geweest zijn blijven een hoge waarde behouden, en (iii) actieve pagina's die zeer recent ingeladen werden zijn vaak slachtoffer omdat ze nog niet de gelegenheid gehad hebben om een hoge waarde op te bouwen.

Nadeel twee kan verholpen worden door de opgebouwde waarden in de tijd te laten vervallen door ze b.v. af en toe te delen door 2. Het laatste nadeel wordt kan vermeden worden door i.p.v. LFU MFU (Most Frequently Used) toe te passen die pas ingeladen pagina's bevoordeelt.

Een benaderende variant van deze algoritmen bestaat uit het niet aanpassen van de tellerwaarde per geheugentoegang, maar slechts 1x per tijdskwantum, en wel op basis van de waarde van het referentiebit (het referentiebit wordt dan opgeteld bij de tellerwaarde).

De tellende algoritmen vormen geen goede benadering van OPT en worden dan ook nauwelijks gebruikt.

Random Replacement (RR)

- Willekeurige pagina vervangen
- Eenvoudig
- Niet zeer doeltreffend

best6-28

Dit is een strategie waarbij willekeurig een pagina geselecteerd wordt ter vervanging. Dit algoritme is eenvoudig omdat het toestandsloos is en dus geen enkele boekhouding vereist. Het is echter niet zeer doeltreffend omdat er geen rekening gehouden wordt met lokaliteit (zie verder).

Paginabuffering

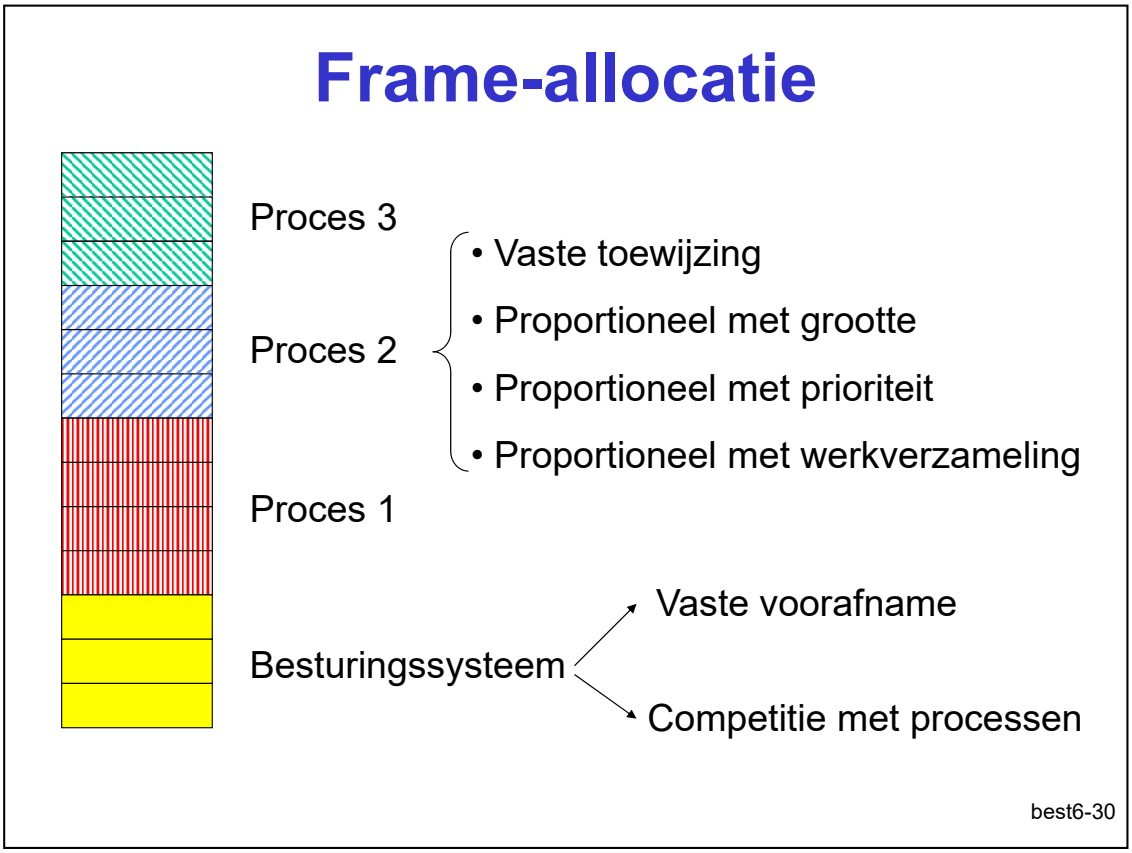
- Pool met vrije frames
  + men hoeft niet te zoeken naar een frames
  - men moet de pool regelmatig bijvullen

[gebruikt in VMS]

best6-29

Sommige systemen houden een voorraadje van vrije frames bij waaruit dan snel een vrij frame kan gekozen worden. Hierdoor kan een proces dat een paginafout veroorzaakt sneller verder werken. Het bijhouden van een voorraadje impliceert wel dat de voorraad op peil gehouden moet worden. Dit wil zeggen dat er een proces zal moeten zijn dat op geregelde tijdstippen (niet-gebruikte) pagina's verhuist naar de swapruimte om frames vrij te maken.

Het feit dat de pagina bewaard werd in de swapruimte hoeft echter niet te betekenen dat de pagina niet meer in het frame opgeslagen ligt terwijl hij zich in de pool bevindt. Mocht er opnieuw een vraag komen voor de betrokken pagina, dan kan deze zonder veel overhead terug toegevoegd worden aan de lijst met actieve pagina's. Dit systeem werd gebruikt in VMS. Men kon daar geen gebruik maken van de referentiebits waardoor men zijn toevlucht moest zoeken tot FIFO. Het systeem zorgde ervoor dat de staart van de wachtlijn bewaard werd in de swapruimte (FIFO), maar dat bij een heraanvraag van een uitgepagineerde pagina het frame snel kon gerecupereerd worden. Eigenlijk was dit een implementatie van het tweedekansalgoritme, maar dan zonder gebruik te maken van referentiebits.



Bij de verdeling van de frames over de verschillende processen zijn er verschillende alternatieven te overwegen. Vooreerst zijn er de noden van het besturingssysteem. Men kan ervoor opteren om het besturingssysteem sowieso de ruimte te geven die het nodig heeft om goed te kunnen werken (vaste voorafname), of men kan ervoor opteren om het besturingssysteem de competitie te laten aangaan met de andere processen in het systeem. Soms past men een hybride oplossing toe: de kern krijgt zeker de ruimte die hij nodig heeft, maar voor de bijkomende plaats (b.v. voor de buffering), moet het besturingssysteem op zoek naar bijkomende frames die het moet veroveren van de processen.

De verdeling van de frames tussen de processen onderling kan op de volgende manier gebeuren:

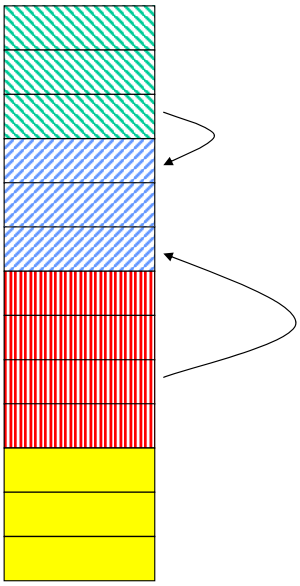
-Elk proces krijgt zijn eigen ruimte van b.v. 128 MiB en daarmee moet het het maar doen. Voor sommige processen zal dit te veel ruimte zijn, voor andere te weinig. Erg efficiënt is deze methode dan ook niet.

-Elk proces krijgt een aantal frames dat proportioneel is met het aantal pagina's van het proces. Indien alle processen samen 200% van de hoeveelheid beschikbaar fysiek geheugen nodig hebben, dan krijgen ze in dit geval allemaal de helft van wat ze vragen. Dit is vrij eerlijk, maar in de praktijk kan het mogelijk zijn dat een proces een kleine lus aan het uitvoeren is, en dat de meeste toegewezen frames niet gebruikt worden.

-Een proces krijgt een aantal frames dat proportioneel is met zijn prioriteit. Processen met hoge prioriteit krijgen dan ook meer frames waardoor ze minder paginafouten zullen genereren en dus sneller vooruit zullen komen (ten koste van de processen met lagere prioriteit). Ook hier kunnen we met het probleem zitten dat een proces weliswaar een hoge prioriteit heeft, maar geen nood aan veel frames.

-Proportioneel met de werkverzameling (zie verder).

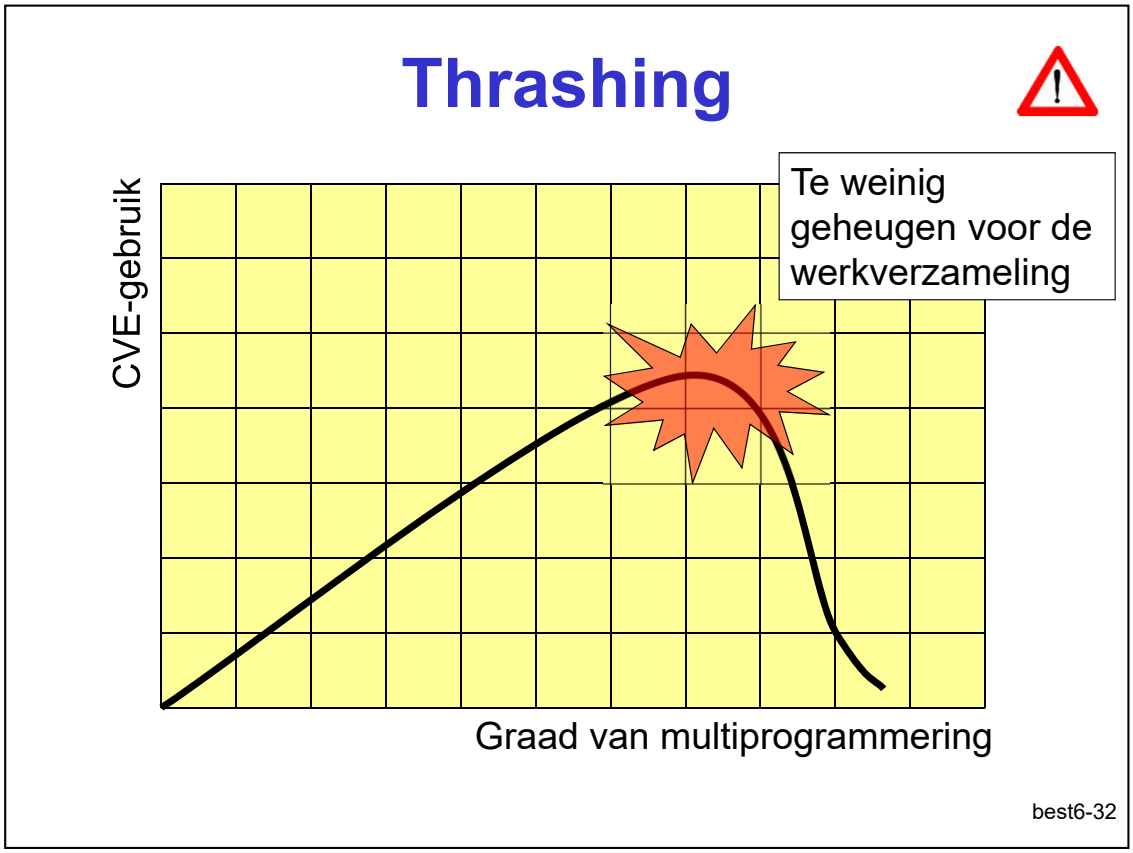
Globale vs. lokale vervanging



best6-31

De paginavervangingsstrategie kan lokaal of globaal zijn. Bij een lokale strategie zal bij de vervanging van een slachtoffer gezocht worden in de eigen frames. Bij een globale strategie zal er over alle frames gezocht worden naar een geschikte kandidaat voor vervanging. Bij de proportionele toewijzing zal men zelfs bij een lokale strategie toch soms ook frames van andere processen moeten ontnemen als de totale belasting van het systeem verandert (als er b.v. processen bijkomen) en er naar een nieuw evenwicht moet toegewerkt worden.

Bij een globale vervanging moet men er wel rekening mee houden dat men een proces niet minder frames kan geven dan er nodig zijn om 1 instructie te kunnen uitvoeren (afhankelijk van het type processor kan dit van 2 tot een tiental variëren).



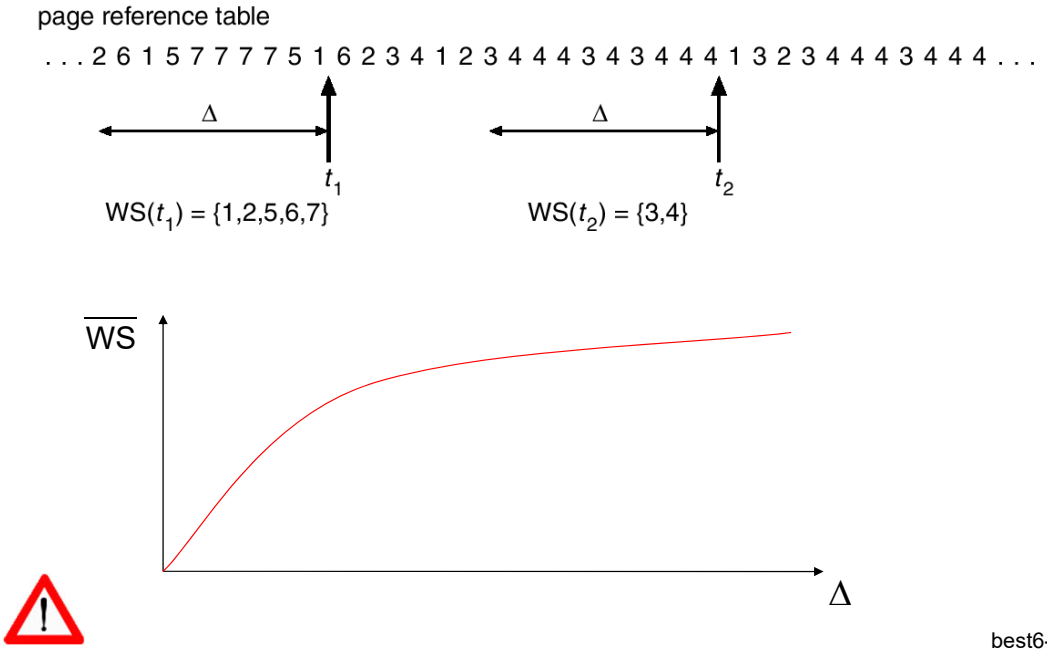
Ofschoon het technisch mogelijk is om het aantal frames te beperken tot de theoretische architecturale ondergrens zal dit in de praktijk niet realistisch zijn. Inderdaad, in dat geval zal telkens wanneer er een pagina moet ingeladen worden er ook een pagina moeten uitgeladen worden en doordat er maar net voldoende frames zijn, zal dit zonder twijfel een frame zijn dat actief gebruikt wordt, waardoor er dus vrij snel een nieuwe paginafout zal optreden. In een dergelijk geval spreekt men van thrashing, dit is een situatie waarbij een proces meer tijd steekt in pagineren dan in uitvoeren.

Thrashing is een instabiliteit in een besturingssysteem en moet daarom vermeden worden. Onderstel dat men een computersysteem heeft dat op nagenoeg volle belasting werkt en dat op een bepaald ogenblik een proces begint met bijvoorbeeld een grote matrix te initialiseren. De paginafouten volgen elkaar op. Indien men gebruik maakt van globale allocatie en globale vervangingsalgoritmen worden er pagina's afgenomen van de andere processen. Hierdoor beginnen ook deze processen bijkomende paginafouten te veroorzaken. Doordat tal van processen hierdoor staan te wachten op het inladen van de gevraagde pagina's zakt de benuttigingsgraad van de CVE, hetgeen een signaal is voor de jobplanner om meer processen tot de processor toe te laten, hetgeen er op zijn beurt weer voor zorgt dat er bijkomende frames gealloceerd worden voor deze processen, enz. Het uiteindelijke effect is dat het effectieve CVE-gebruik plotseling een scherpe duik naar beneden neemt.

Het gebruiken van een lokale vervangingsstrategie kan helpen om thrashing beperkt te houden tot één enkel proces. Het effect op de andere processen zal in dat geval enkel indirect zijn (doordat er meer dan gewone trafiek is met de swapruimte zullen de wachtlijnen daar gemiddeld langer zijn, en dus ook de wachttijd voor de processen die zelf niet thrashen).

Verder geldt als algemene regel dat indien thrashing optreedt het beter is om een proces tijdelijk uit te swappen en maar terug in te swappen van zodra er opnieuw voldoende frames ter beschikking zijn. Thrashing treedt op indien een proces over onvoldoende frames beschikt om zijn werkverzameling in het fysiek geheugen geladen te houden.

Werkverzameling (working set)



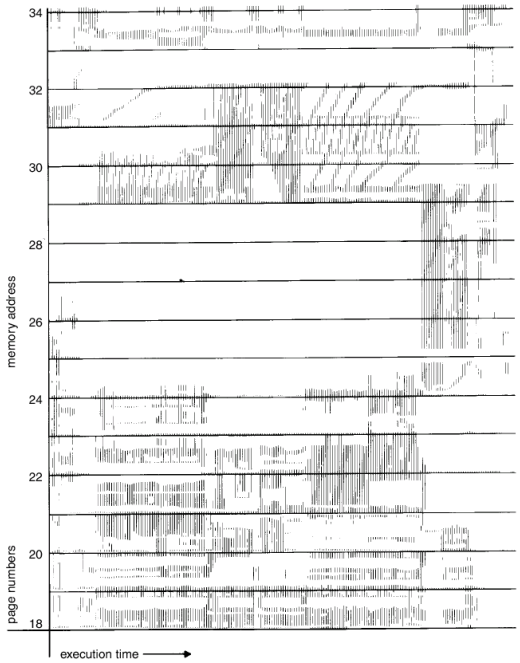
Werkverzameling WS(t) = alle gerefereerde pagina's door de Δ geheugenreferenties die tijdstip t voorafgaan.

De keuze van Δ is cruciaal. Neemt men Δ te klein, dan zal de werkverzameling niet alle pagina's bevatten die op dat ogenblik actief gebruikt worden. Indien men Δ te groot kiest, dan zal de werkverzameling ook een aantal pagina's bevatten die misschien niet langer gebruikt worden. Indien $\Delta = \infty$, dan krijgt men het totale aantal pagina's dat tot dan toe door het proces gebruikt werd.

Voor een geschikte Δ, kan men de som maken van de werkverzamelingen van de verschillende processen die uitgevoerd worden op een gegeven tijdstip t. Indien deze som groter is dan het totaal beschikbare aantal frames, dan zal er thrashing optreden op systeemniveau (indien er globale paginavervanging toegepast wordt). Het is beter om in dat geval een aantal processen uit te swappen om op die manier de som te verkleinen totdat ze past in het totale aantal beschikbare frames.

Bij lokale paginavervanging zal er thrashing optreden op procesniveau indien de werkverzameling van het proces groter is dan het lokale aantal toegewezen frames. Uiteraard dient men maar over thrashing te spreken indien een verhoogde paginafoutfrequentie gedurende langere tijd aanhoudt. Bij de overgang tussen 2 fasen in de uitvoering van een programma is het normaal dat er wat verhoogde pagineringsactiviteit is.

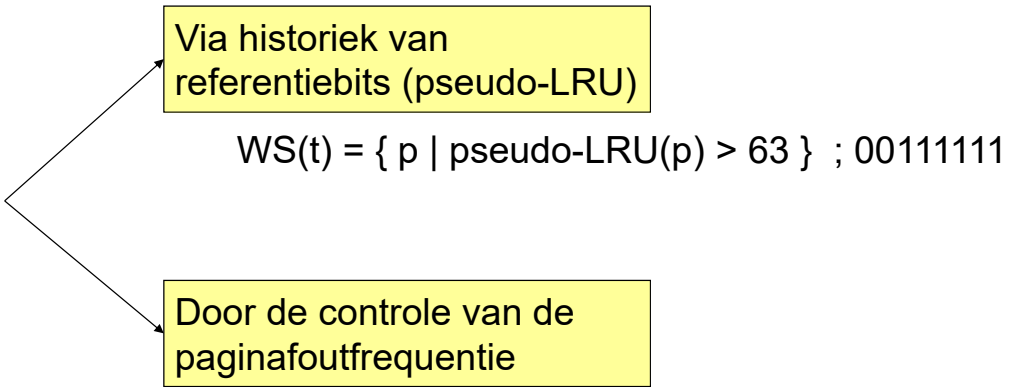
Lokaliteit



best6-34

De werkverzameling heeft te maken met lokaliteit. Hierboven staat een trace van een uitvoering van een programma. Zoals duidelijk te zien is, worden er op elk ogenblik tijdens de uitvoering maar een beperkt aantal pagina's gebruikt. Het zullen precies deze pagina's zijn die tot de werkverzameling zullen behoren. De Δ moet in de praktijk zodanig gekozen worden dat de werkverzameling zich maar geleidelijk aanpast aan de veranderende omstandigheden. Zoniet zal een pagineringsysteem dat tracht om de werkverzameling in het fysiek geheugen te houden te snel reageren op kleine verstoringen in de uitvoering (b.v. het eventjes oproepen van een routine uit een andere module). Men moet ervoor zorgen dat bij het terugkeren uit de routine de pagina met de oproeper inmiddels niet vervangen werd.

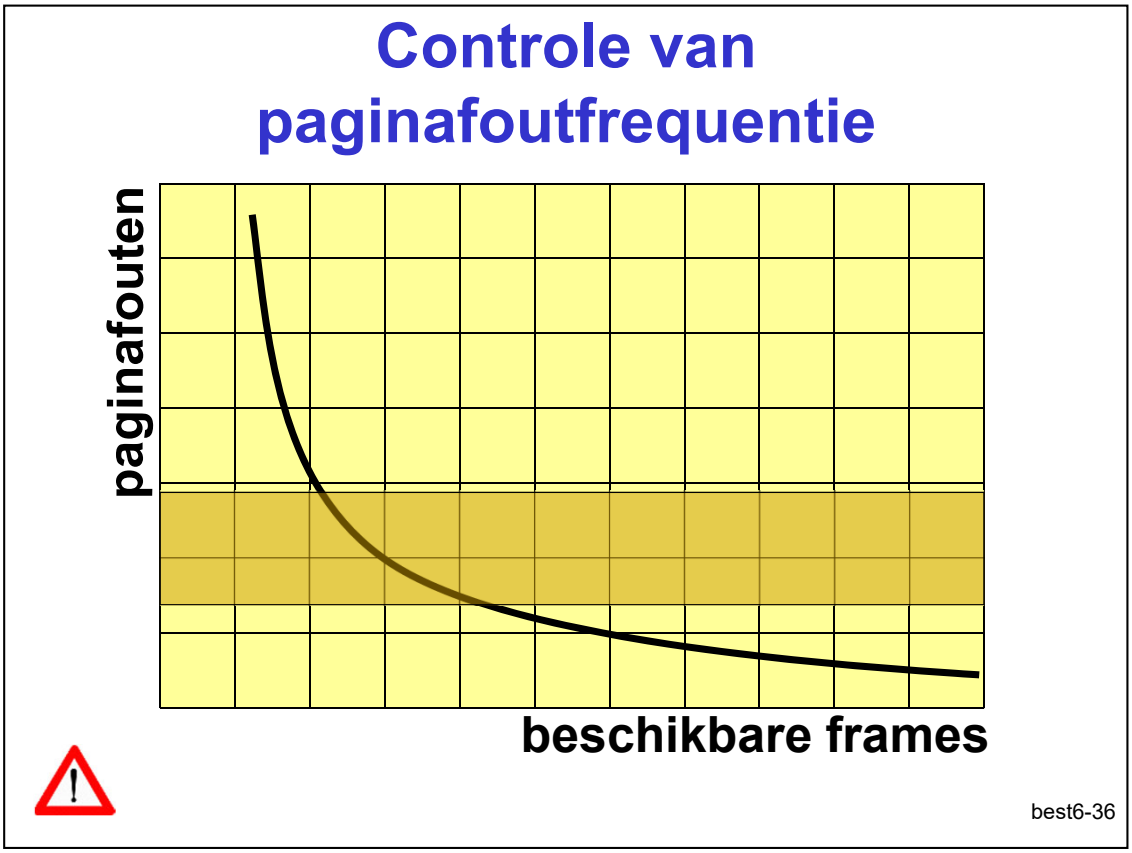
Werkverzameling bepalen



best6-35

Om de werkverzameling van een proces te bepalen kan er gebruik gemaakt worden van de referentiebits van de pagina's. Door op geregelde tijdstippen de referentiebits te kopiëren kan men een historiek van de referentiebits opbouwen (zoals bij pseudo-LRU) die kan gebruikt worden om de werkverzameling te bepalen. Pagina's die gedurende de laatste 2 klokperiodes b.v. niet gebruikt werden kunnen beschouwd worden als niet meer behorende tot de werkverzameling. Merk op dat men in dit geval de werkverzameling niet meer definieert aan de hand van het aantal geheugenreferenties, maar dat men hier de verstreken tijd als criterium gebruikt, wat niet hetzelfde is.

Een tweede methode maakt gebruik van de frequentie waarmee paginafouten optreden (zie volgende dia).



Een laag aantal paginafouten per seconde is een goede indicatie om vast te stellen of de werkverzameling van een proces zich in het fysiek geheugen bevindt of niet. Door het aantal paginafouten per seconde binnen twee opgelegde grenzen te proberen houden kan men het aantal frames zodanig aanpassen dat de werkverzameling in het fysiek geheugen gehouden wordt.

Indien het aantal paginafouten per seconde voor een gegeven proces boven een gegeven drempel uitkomt moeten er bijkomende frames aan het proces toegewezen worden om het aantal paginafouten te doen dalen. Anderzijds indien het aantal paginafouten beneden een bepaalde grens daalt, wijst dit op een mogelijks te groot aantal frames, en kan men beslissen om het aantal gealloceerde frames te verminderen. Dit is een dynamische manier om frames te alloceren. Door het aantal paginafouten per seconde gelijk te proberen houden voor alle aanwezige processen probeert men van elk proces een gelijk deel van de werkverzameling in het geheugen te houden. Processen met grotere werkverzamelingen zullen op deze manier meer frames toegewezen krijgen. Indien men dat mechanisme niet onbeperkt wenst te laten spelen, kan men bijkomende beperkingen opleggen.

Taken besturingssysteem

Procescreatie

- paginatabel alloceren en initialiseren
- initieel aantal paginaframes bepalen

Proceswisseling

- MMU instellen met beginadres paginatabel
- TLB leegmaken

Paginafout

- het adres dat de fout veroorzaakte bepalen
- vrije frame vinden, pagina van swapruimte inladen

Procesterminatie

- pagina's vrijgeven in geheugen en op schijf
- frames wissen

best6-37

Tot slot bekijken we nog even waar in het hele geheugenbeheer het besturingssysteem moet tussenkomen.

Overzicht

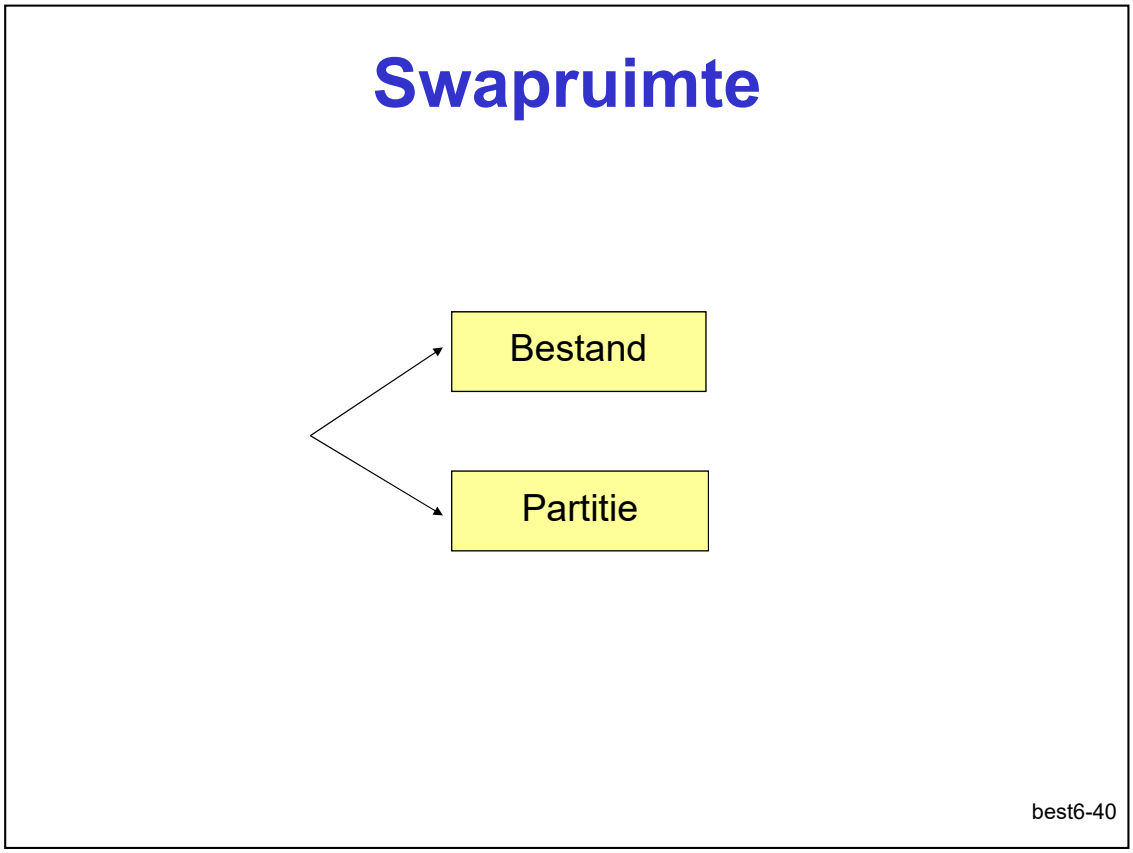
- Overallocatie
- Manuele methoden
  - Dynamisch laden
  - Dynamisch linken
  - Overlays
- Automatische methoden
  - Swapping
  - Virtueel geheugen
- Eindbeschouwingen

best6-38

Eindbeschouwingen

- Swapruimte
- Prepaginering
- Paginagrootte
- TLB-bereik
- Programmastructuur
- IO-interlock
- Ware tijd
- Memory mapped bestanden
- Copy-on-write
- Paging daemon
- Wissen frames
- Gedeelde pagina's
- Beleid vs. Mechanisme

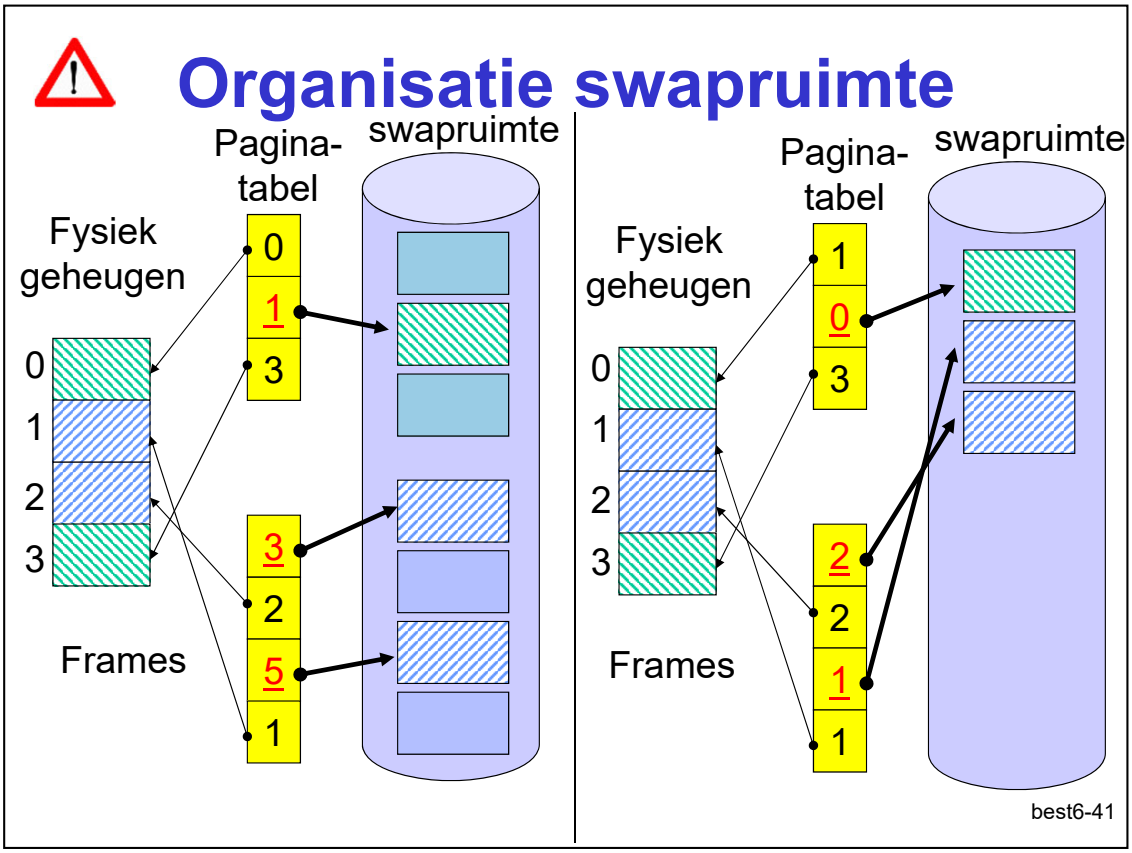
best6-39



Voor de swapruimte kan er gekozen worden tussen een bestand of een schijfpartitie. Een bestand heeft als voordeel dat het flexibel is, dat het kan groeien en krimpen indien de behoefte daartoe zou ontstaan, maar deze flexibiliteit heeft zijn prijs: het gebruik van een bestand is traag: vooreerst is er de overhead van het bestandensysteem (zie later), ten tweede kan het swapbestand gefragmenteerd zijn – wat de toegangssnelheid zeer nadelig kan beïnvloeden, en ten derde komt de organisatie van het bestand (b.v. de blokgrootte) niet steeds overeen met de grootte van een frame. Dit maakt een bestand minder geschikt als snelle swapruimte. Windows maakt gebruik van een bestand als swapruimte.

Een partitie daarentegen heeft de bovenstaande nadelen niet: ze kan optimaal georganiseerd worden als swapruimte zodat een snelle toegang gegarandeerd is. Het voornaamste nadeel van een partitie is dat de omvang vastgelegd wordt tijdens de oorspronkelijke partitionering van de schijf, en dat deze later niet zo eenvoudig kan veranderd worden.

De grootte van het virtuele geheugen is een parameter die bij de generatie van een besturingssysteem moet gekozen worden. In de regel wordt voor 200%-300% van het aanwezige fysiek geheugen gekozen. De grootte bepaalt hoeveel virtueel geheugen er maximaal kan gealloceerd worden aan alle processen samen.



De swapruimte zelf kan op twee manieren georganiseerd worden. Ofwel reserveert men in de swapruimte plaats voor de volledige processen en gedraagt het fysiek geheugen zich als een soort van cache t.o.v. de swapruimte. Processen worden in dit geval a.h.w. gecreëerd in de swapruimte. Een andere mogelijkheid is dat de swapruimte enkel plaats biedt aan die pagina's die geen frame ter beschikking hebben. Dit wil zeggen dat de pagina's van de processen verspreid liggen over het fysiek geheugen en de swapruimte. In dit tweede geval kan de swapruimte uiteraard kleiner gedimensioneerd worden. Het beheer ervan is echter wat complexer. Bij hedendaagse systemen die over zeer grote hoeveelheden fysiek geheugen beschikken is dit een te overwegen optie.

Indien men het fysiek geheugen als cache voor de swapruimte gebruikt, dan zal een te vervangen pagina niet dienen teruggeschreven te worden indien deze sinds het inswappen niet veranderd werd. In het andere geval moet een pagina steeds teruggeschreven worden. Voor read only pagina's (code en read-only data) kan men zelfs overwegen om deze telkens opnieuw op te halen uit de uitvoerbare bestanden. Indien dit niet zo vaak voorkomt (omdat er veel fysiek geheugen is), is ook dit een te overwegen optie.

Prepaginering

- Niet beginnen met 0 pagina's, maar *n* pagina's ineens inladen
- Bij inpagineren meteen een deelverzameling van uitgepagineerde pagina's selecteren (benadering van de working set)

best6-42

Bij pure paginering op aanvraag zal het aantal paginafouten bij het opstarten van een proces zeer groot zijn. Om dit ter vermijden kan men een aantal pagina's ineens in het geheugen brengen. Dit kan interessant zijn omdat de tijd nodig om een extra pagina van de swapruimte naar het fysiek geheugen te transfereren klein is in vergelijking met de tijd nodig om de kop van de schijf te positioneren. Anderzijds kan het wel verloren moeite zijn omdat men niet weet of de pagina's ooit gebruikt zullen worden.

Bij het inpagineren van een proces ligt het enigszins anders omdat men hier kan te weten komen welke pagina's uitgepagineerd werden. Gezien deze pagina's verband houden met de werkverzameling kan het hier interessant zijn om deze pagina's ineens in te laden alhoewel men ook hier geen zekerheid heeft over het uiteindelijk gebruik ervan.

Paginagrootte

- Vastgelegd door de hardware
- Trend naar grotere paginagroottes:
  - kleinere paginatabellen
  - minder paginafouten
  - meer interne fragmentatie
  - meer onnodige datatransfer

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2} \qquad p = \sqrt{2se}$$

best6-43

De paginagrootte wordt meestal vastgelegd door de aanwezige hardware. Wel is er een trend naar steeds groter wordende paginagroottes. De voordelen van een grote pagina zijn: kleinere paginatabellen, minder paginafouten. De nadelen zijn: meer interne fragmentatie en onnodige transfers. Deze laatste twee nadelen zijn echter van beperkt belang met de steeds groter wordende geheugens en de hogere bandbreedte naar de schijven. Windows gebruikt clusters van pagina's om te pagineren. Op die manier krijgt men een aantal van de voordelen van grotere pagina's.

Men kan de netto geheugenoverhead van de adresvertaling uitdrukken als de plaats die nodig is om tabelelementen in op te slaan (e = 4 bytes per gebruikte pagina), en een halve pagina interne fragmentatie (p/2). De procesgrootte is s. Indien men het minimum zoekt voor deze uitdrukking dan vindt men een uitdrukking voor p die voor een procesgrootte van 2 MiB en een tabelelementgrootte van 4 bytes precies een paginagrootte van 4 KiB oplevert. Voor een procesgrootte van 8 MiB zouden we dan 8 KiB uitkomen.

TLB-bereik

- **TLB-bereik** = bereikbaar geheugen vanuit de TLB =

  TLB grootte X paginagrootte

- Een TLB moet voldoende groot zijn om de werkverzameling van een proces in te kunnen bevatten

best6-44

Het TLB-bereik kan vergroot worden door ofwel de TLB te vergroten (dit is niet steeds eenvoudig omdat de TLB een associatief (duur) geheugen is, en een aanzienlijk grotere TLB ook trager zal zijn.

Een andere mogelijkheid is om de paginagrootte te doen toenemen (heeft ook nadelen, zie vorige dia). Een alternatieve mogelijkheid is om met verschillende paginagrootten te werken. Indien het proces baat heeft bij grote pagina's kan de TLB dan zodanig ingesteld worden dat hij met grotere pagina's kan werken.

Programmastructuur

```
void matrixoperatie()
{
 int A[][] = new int[1024][1024];
 for (i = 0; i < A.length; i++)
     for (j = 0; j < A.length; j++)
          A[i,j] = 0;
  …
}
```

1024 paginafouten

best6-45

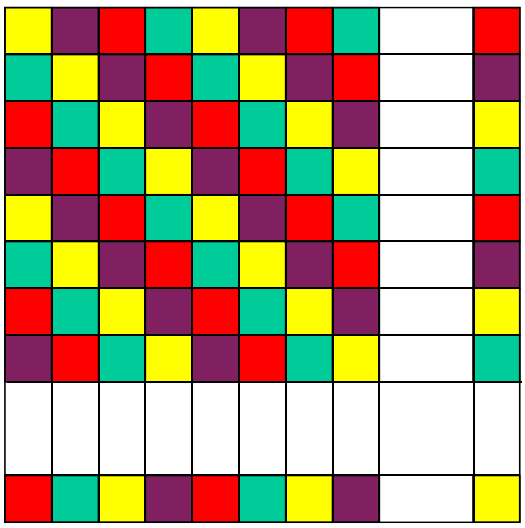
De programmeur kan door zijn code te herstructureren soms beter of minder goed gebruik maken van het virtueel geheugen.Algemeen kan men stellen dat door de lokaliteit van de programma's te verbeteren de werkverzameling zal verkleinen en het programma minder hinder zal ondervinden van de werking van het virtueel geheugen. Het schoolvoorbeeld om dit te illustreren is het initialiseren van een matrix.

De gebruikte matrix is 4 MiB groot en bestaat uit 1024 rijen van elk 4 KiB groot. Onderstellen we dat de paginagrootte 4 KiB is waardoor een rij precies in een pagina past. Indien de array op de hierboven manier geïnitialiseerd wordt zal er door de matrix voor elke rij precies één paginafout gegenereerd worden, dit zijn dus 1024 paginafouten.

Programmastructuur

```
void matrixoperatie()
{
 int A[][] = new int[1024][1024];
 for (j = 0; j < A.length; j++)
     for (i = 0; i < A.length; i++)
          A[i,j] = 0;
  …
}
```

1024 x 1024 paginafouten



best6-46

Indien de arrayindices omgewisseld worden dan zal er bij een LRU strategie per geheugentoegang een paginafout gegenereerd worden (indien het aantal frames kleiner is dan 1024). Het hoeft geen betoog dat dit een aanzienlijke invloed zal hebben op de prestatie van dit programma. Kort na het veralgemeend invoeren van virtueel geheugen in de jaren 70 heeft men heel wat programmabibliotheken moeten herstructureren om nog een redelijke prestatie te behouden.

I/O-interlock

- IO-operatie niet mogelijk indien een proces uitgepagineerd is
- Oplossingen
  - Buffering in de kern
  - Pagina's vergrendelen (lock bit)
- De lock-bit is ook bruikbaar om
  - nieuwe pagina's in het geheugen te houden
  - pagina's te beschermen tegen verplaatsing
  - pagina's te beschermen tegen uitpagineren (OS)
- Gevaarlijk: te veel locks verhinderen de goede werking van het geheugenbeheer

best6-47

We hebben reeds gezien dat paginering en IO niet steeds even goed samengaan. Om te verhinderen dat een hangende IO-operatie voltooid wordt en naar het (fysiek) geheugen probeert te schrijven terwijl een pagina niet ingeladen is kan men een pagina vergrendelen voor de duur van een IO-operatie aan de hand van de lock bit. Dit bit zal ervoor zorgen dat de pagina niet geselecteerd kan worden voor vervanging.

De lock bit kan trouwens ook nog voor andere toepassingen gebruikt worden. Stel dat een proces een paginafout genereert en dat de betreffende pagina in het geheugen geladen werd. Voor deze pagina zal de referentiebit op 0 staan, de dirty bit staat op 0, met andere woorden deze pagina is de ideale kandidaat voor vervanging. Om te vermijden dat een pagina vervangen wordt nog vóór dat hij de eerste keer gebruikt is geweest door het aanvragende proces kan men de lock bit op 1 zetten. Deze lock bit moet dan door de dispatcher terug op 0 gezet worden van zodra het proces de pagina voor het eerst gebruikt. De lock bit kan ook gebruikt worden om tijdens IO-operaties te verhinderen dan een pagina eventueel zou verplaatst worden in het fysiek geheugen (hierdoor zou het resultaat van een DMA-operaties b.v. fout kunnen zijn). Tenslotte wordt de lock bit ook gebruikt om ervoor te zorgen dat de pagina's die behoren tot de kern van het besturingssysteem in het fysiek geheugen gehouden worden.

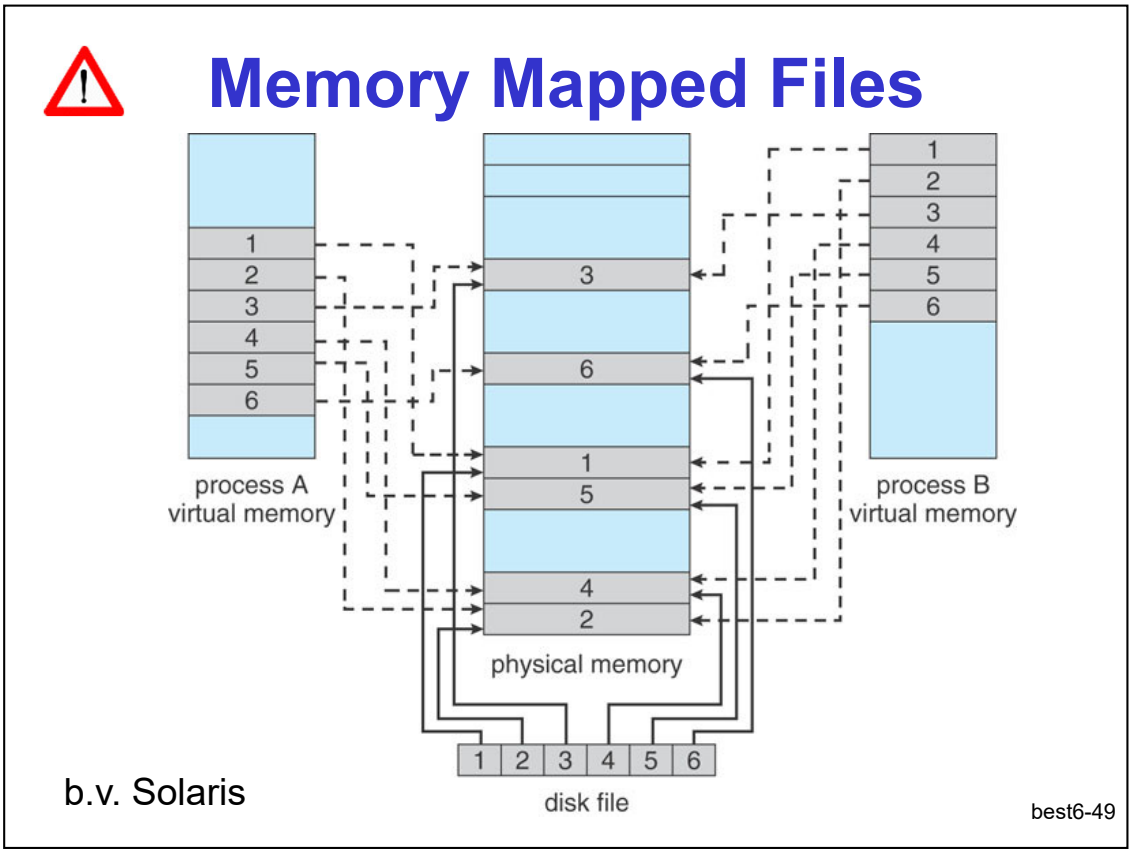
De lock bit houdt ook gevaren in: een lock bit dat men vergeet terug op 0 te zetten veroorzaakt een verloren frame. Bovendien kan het overmatig gebruik van de lock bit de werking van het virtueel geheugen hinderen.

Ware Tijd

- Virtueel geheugen is niet combineerbaar met ware tijd
- Lock bit kan soelaas brengen

best6-48

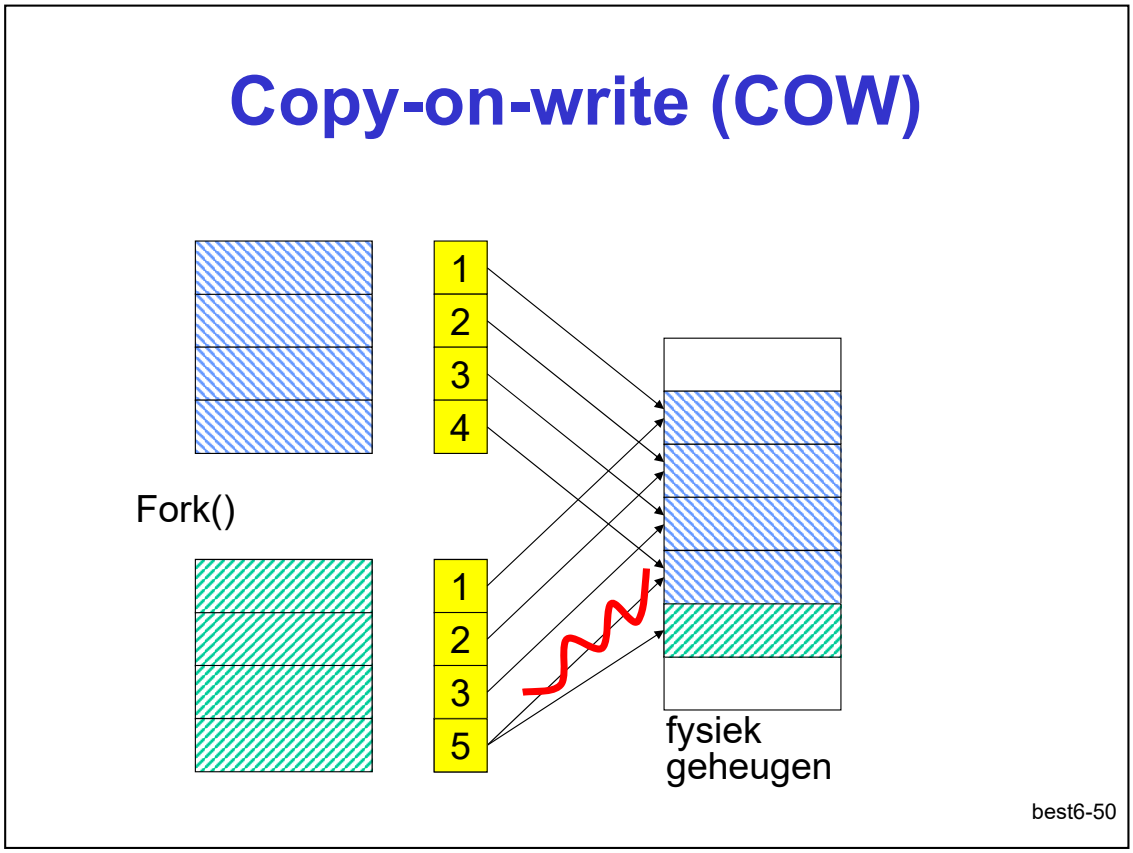
Het gebruik van virtueel geheugen is in principe niet te verzoenen met werking in ware tijd omdat men op ongeveer elk ogenblik een paginafout kan veroorzaken die de uitvoering van een proces met vele milliseconden zal vertragen. Door gebruik te maken van lock bits en dergelijke kan men in sommige gevallen toch een zacht real-time gedrag realiseren.



Het pagineringsmechanisme en virtueel geheugen zijn niet enkel beperkt tot interactie met de swapruimte, maar kunnen in principe ook interageren met gewone bestanden. Op die manier kan men b.v. een bestand afbeelden in het geheugen: men zegt gewoon dat een bepaald geheugengebied als swapruimte het betrokken bestand heeft. Als men dan een byte uit het bestand nodig heeft, volstaat het om die byte te lezen uit het geheugen. Er zal een paginafout optreden die de pagina die de byte bevat zal inladen in een frame en ter beschikking zal stellen. Alle verdere toegangen tot dezelfde pagina zijn gewone geheugentoegangen. Bij het sluiten van het bestand moet men er dan gewoon voor zorgen dat alle veranderde pagina's terug naar de schijf geschreven worden.

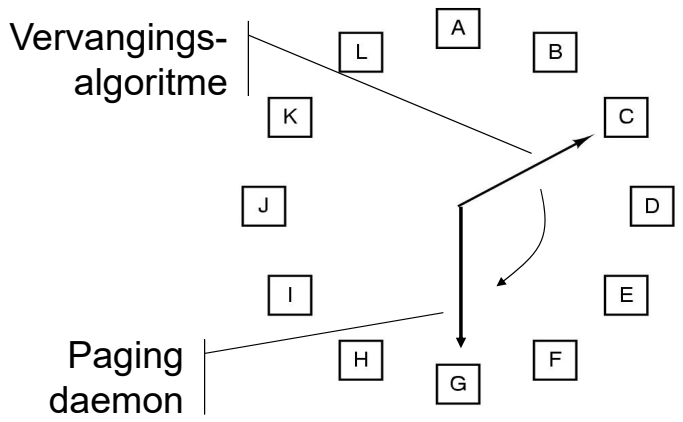
Verschillende processen kunnen hetzelfde bestand in hun adresruimte opnemen (sommige read-only, andere om te lezen en te schrijven).

Sommige besturingssystemen zoals Solaris gebruiken dit mechanisme zelfs als enig mechanisme om aan bestands-IO te doen.



Virtueel geheugen laat toe om fork() (het dupliceren van een procesbeeld) efficiënt te laten verlopen. Het idee is dat men ter gelegenheid van de fork()-oproep enkel de paginatabellen van het kindproces aanmaakt en deze laat wijzen naar de frames van het ouderproces. Tegelijk zet men de pagina's voor het ouderproces en voor het kindproces op read-only. Van zodra één van beide processen iets probeert te veranderen aan een pagina krijgen we een onderbreking die het betrokken frame dan dupliceert en de paginatabellen aanpast (opnieuw schrijfbaar in de beide processen). Op deze manier zullen er slechts bijkomende frames gealloceerd worden indien dit werkelijk nodig is. Indien het kind meteen na het uitvoeren van de fork() het proces overlaadt met een exec()-oproep, dan hoeven er zelfs geen pagina's gekopieerd te worden. Dit maakt het dupliceren van processen aanzienlijk efficiënter.

Paging daemon



Windows: automatic working set trimming

Solaris 2: pageout

best6-51

Paginering werkt beter als er altijd wat vrije frames beschikbaar zijn in de frame pool. De paging daemon is een proces dat moet instaan voor het aanvullen van de frame pool. Indien het aantal vrije frames onder een opgegeven drempel valt, dan zal hij dirty pagina's beginnen terugschrijven naar de swapruimte (en de referentiebit op 0 zetten). Hij gaat de wijzer van het vervangingsalgoritme vooraf zodat deze gemakkelijker pagina's vindt die kunnen vervangen worden. Indien de afstand tussen de paging daemon en het vervangingsalgoritme b.v. 1024 pagina's bedraagt (4 MiB), dan hebben de pagina's voldoende tijd om opnieuw gebruikt te worden en te verhinderen dat ze definitief verdwijnen.

Bij Windows wordt er per proces een minimale en maximale grootte van de werkverzameling bepaald. Het aantal frames dat toegewezen wordt aan een proces kan variëren tussen deze twee getallen. Indien het aantal frames in de frame pool te laag dreigt te worden wordt **automatic working set trimming** geactiveerd. Dit is een paging daemon die frames weghaalt bij die processen die meer frames hebben dan vereist door hun minimale werkverzameling.

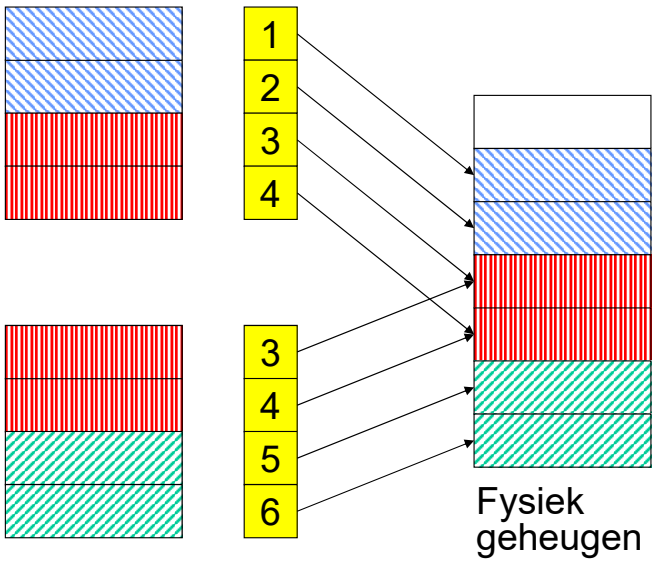
Ook Solaris gebruikt een vergelijkbaar systeem (pageout proces). Dit proces kan traag of snel vooruitgaan, afhankelijk van de noden.

Wissen frames

- Frames bevatten soms confidentiële informatie.
- Alvorens een frame toe te wijzen aan een proces moet het gewist worden (zero fill on demand)

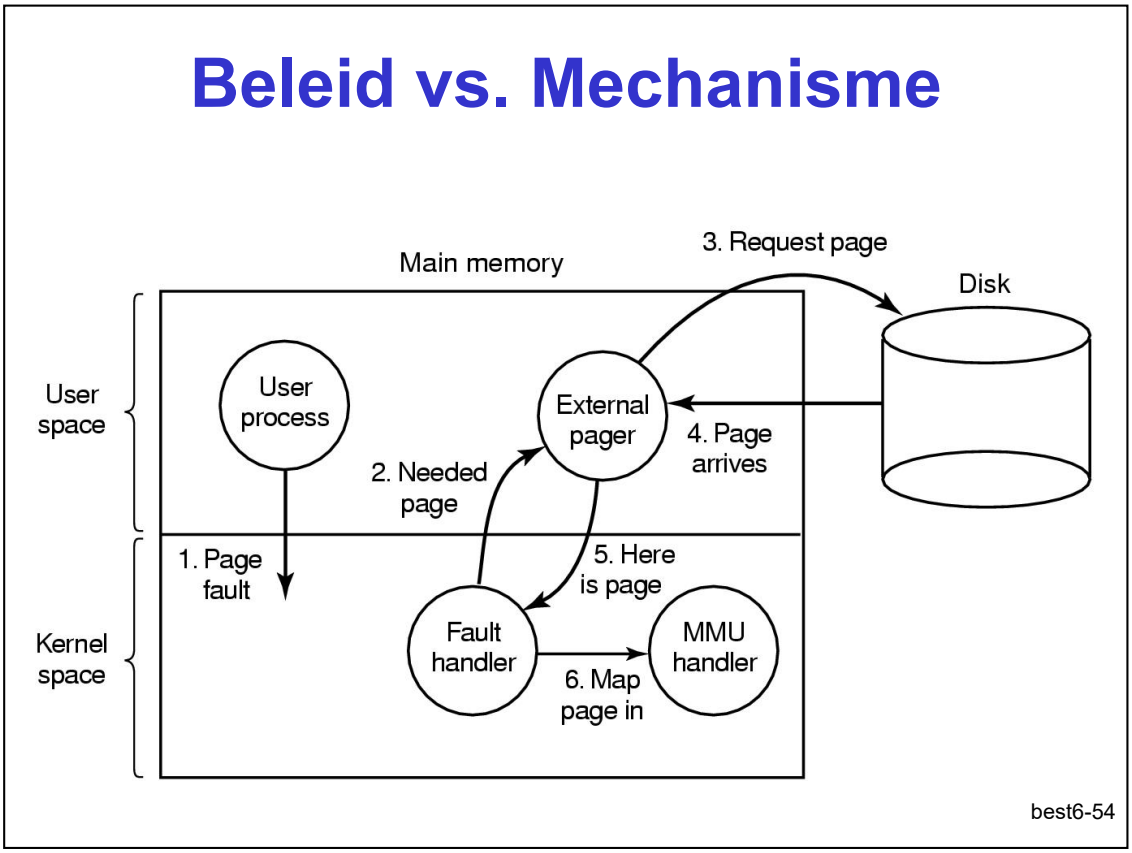
best6-52

Gedeelde pagina's



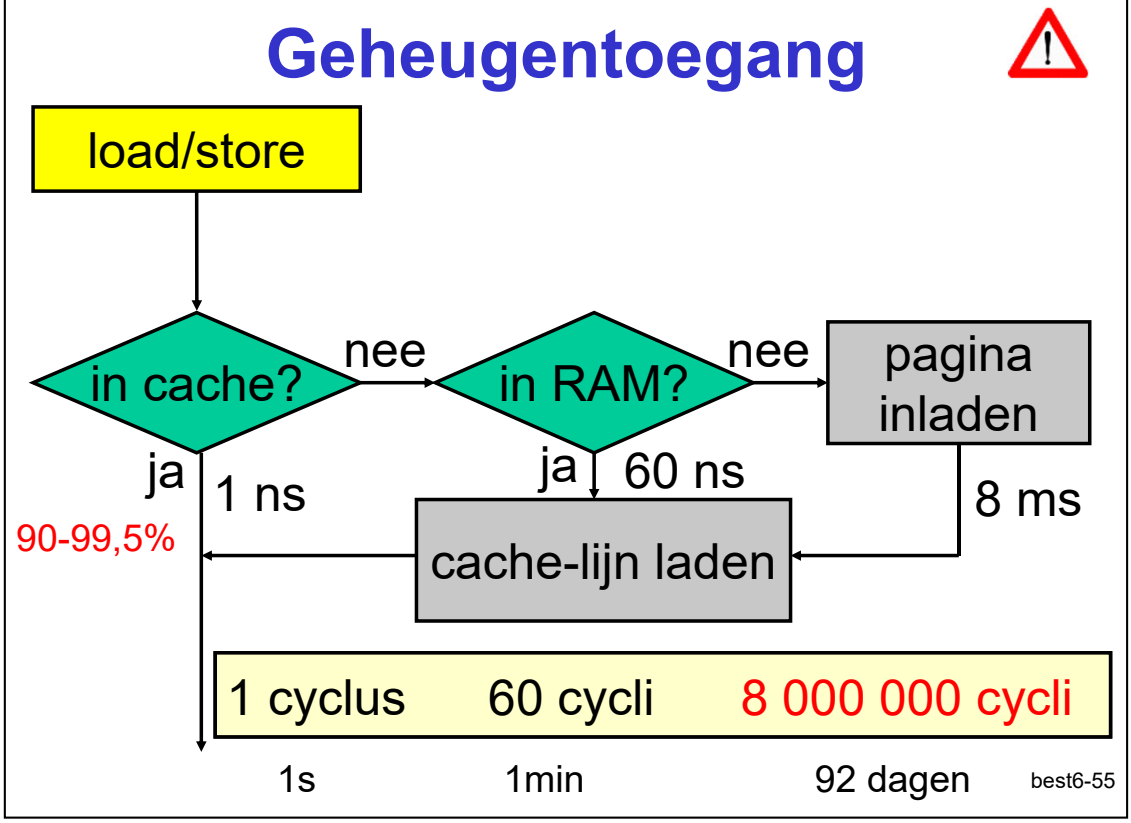
best6-53

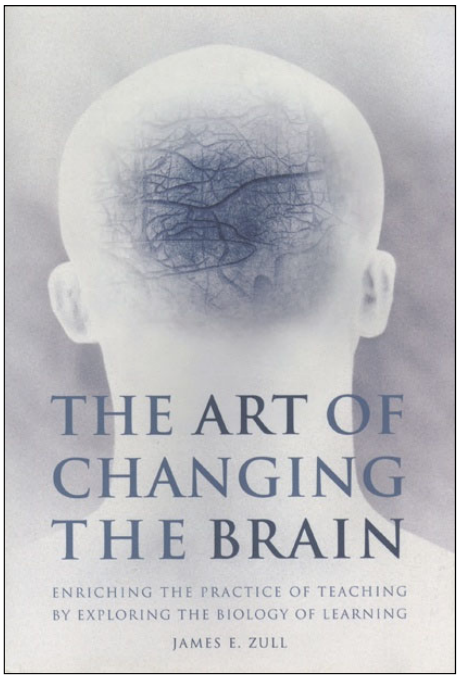
Instructiepagina's worden dikwijls gedeeld. Bij gedeelde pagina's moet men voorzichtig zijn met het uitpagineren. Als dezelfde frames ook in een andere adresruimte afgebeeld worden, zou het kunnen gebeuren dat ze meteen opnieuw moeten ingepagineerd worden. Nog gevaarlijker wordt het indien een gedeeld frame gewist zou worden omdat één van de processen die het frame gebruikt termineert.



Mach laat toe om de gebruiker zelf een vervangingsalgoritme te laten definiëren. Men kan zelf een paginator schrijven die de in te pagineren pagina moet meedelen aan de kern. De gebruiker kan in dat geval zelf bepalen op welke manier de swapruimte georganiseerd wordt.

Geheugentoegang





http://programma.ntr.nl/10577/ntr-academie/archief/detail/aflevering/6000007132/Justine-Pardoen%3A-Opvoeden-in-een-tijd-van-multimedia-prikkels

best6-56