IN1805 I – Operating System Concepten

Hoofdstuk 9: Virtual memory

Virtueel Geheugen en demand paging (1)

- Programma's zijn vaak niet in hun geheel in het geheugen nodig, vanwege:
  - zelden gebruikte onderdelen
  - groter gedeclareerde arrays dan nodig
  - als programma helemaal nodig, dan niet alles tegelijk
- Virtueel geheugen maakt het mogelijk dat een programma gedeeltelijk in main memory aanwezig is , voordelen:
  - fysiek geheugen is geen beperking meer voor de programmeur
  - meer processen simultaan mogelijk
  - minder I/O nodig voor swapping
- Implementatie meestal d.m.v. *demand paging* (evt. *demand segmentation*)

Virtueel Geheugen en demand paging (2)

- Demand paging haalt pages binnen op het moment dat zij echt nodig zijn (*Lazy swapping)*
- *page-in* i.p.v. *swap-in*; page-out i.p.v. swap-out
- aan/afwezigheid van page in main memory aangegeven in pagetable (b.v. via valid/invalid bit)
- *pagefault* als bij vertaling page niet aanwezig blijkt

Virtueel Geheugen en demand paging (3)

- Pagefault geeft trap, resulterend in:
  - afbreken van de instructie
  - zoeken van een vrij frame
  - naar binnen halen van de gevraagde page
  - aanpassen van de pagetable
  - herstarten van de onderbroken instructie (eis aan architectuur)
- pagefaults beperkt vanwege localiteitsprincipe:
  - Ieder proces heeft op ieder moment maar een klein aantal pages echt in gebruik
- performance van demand paging:
  - pagefault rate laag houden
  - niet gewijzigde pages niet wegschrijven (modify/dirty bit)

Oefening (Effective Access Time)

Gegeven een paging systeem met de pagetable in main memory

1. Als een fysieke geheugenacces 200 ns duurt, hoe lang duurt dan een logische geheugenaccess?

2. Als we een TLB toevoegen en er vanuit gaan dat 75% van alle (logische) geheugen referenties gevonden worden in de TLB, hoe lang duurt dan de gemiddelde logische geheugenaccess?

3. Hoe lang duurt het afhandelen van een pagefault?

Virtueel Geheugen en demand paging (4)

- *Page replacement* is nodig, wanneer main memory volledig bezet is, en er treedt een pagefault op.

  actie:

  – zoek een slachtoffer voor page-out
  – copieer slachtoffer-page naar schijf
  – haal benodigde page naar binnen

Virtueel geheugen en demand paging (5)
page replacement

Algoritmen nodig voor :

- keuze van een slachtoffer bij het vrijmaken van een frame *(page replacement algoritme)*

- bepalen van het aantal frames dat ieder proces krijgt *(frame allocation algoritme)*

Page-replacement algoritmen (1)

- doel: page fault rate zo laag mogelijk houden.

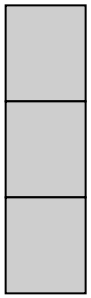
  Een kleine verbetering in demand paging, geeft een grote winst in de systeem performance.
- vergelijken van algoritmen door:
  - uitgaan van een *Page Reference String (PRS)*
  - aantal pagefaults te bepalen bij verschillende aantallen frames.
- overzicht van page-replacement algoritmen:
  - FIFO
  - Min (Optimal)
  - LRU
  - NUR

Page-replacement algoritmen (2)
FIFO

- Pagina die het langst geleden is binnengehaald , wordt slachtoffer.
- voordeel:
  - eenvoudige implementatie
- nadeel:
  - lang aanwezig zijn, wil niet zeggen niet meer in gebruik
  - Belady's anomalie

Belady's Anomaly: more frames ⇒ more page faults

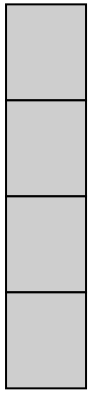
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames



1 4 5
2 1 3
3 2 4

9 page faults

- 4 frames



1 5 4
2 1 5
3 2
4 3

10 page faults

Page-replacement algoritmen (3)
Min (Optimal)

- Regel :

  Vervang die pagina die het langst niet meer nodig zal zijn
- Voordeel:

  Geeft de laagste page fault rate
- Nadeel:

  Is niet te implementeren zonder de toekomst te kennen.

Het (theoretische) resultaat van Min kan worden gebruikt om de kwaliteit van de andere algorimen te beoordelen.

Page-replacement algoritmen (4)
LRU

- Least Recently Used (LRU)
- Is een benadering van Min ; ziet het verleden als een spiegelbeeld van de toekomst, kiest die pagina die het minst recent voor het laatst gebruikt is
- voordelen :
  - benadert Min
  - geen last van Belady's anomalie
- nadeel: implementatie is moeilijk/duur.
  - virtuele klok, of
  - stack,

  bij iedere referentie aanpassen

Page-replacement algoritmen (5)
NUR

- Not Used Recently (NUR)
  - benadert LRU
  - kiest een pagina die niet pas nog gebruikt is.
  - implementatie d.m.v. *reference bit*

  varianten:
  - reference bit gecombineerd met shift register, geeft meer historie
  - *second chance met reference bit* geeft gerefereerde pages een nieuwe kans
  - *second chance met reference bit en dirty bit* kiest uit de laagste van de 4 mogelijke categorieën

Page-replacement algoritmen (5A)
(cyclic) second chance

R= Referenced bit = Used bit

	ronde n			*ronde n+1*		
	R			R		
start	1	→	0	0		0
	1	→	0	1		0
	0		*selected*	1		0
	1		1	*start* 1	→	0
	1		1	1	→	0
	0		0	0		*selected*

Page-replacement algoritmen (5B)
(cyclic) second chance with R and C bit

R= Referenced bit = Used bit, C = Changed bit = Dirty bit

Voorkeur en categorie	R	C
1	0	0
2	0	1
3	1	0
4	1	1

	ronde n		*ronde n+1*		
	R	C	R	C	
start	1	0→0	0	0	*selected*
	1	1→0	0	1	
	0	1→0	0	1	
	0	0 *selected*	1	1	
	0	1	*start* 0	1 → 0	1

Page-replacement algoritmen (6)
page buffering

page replacement te versnellen door:

- pool van vrije frames als bufferruimte
- modified (dirty) pages op stille momenten naar disk copiëren (dirty bit daarna 0)
- van ieder frame in de pool van vrije frames onthouden: welke page (procesnr en pagenr) hierin stond. Proces kan page reclaimen.

PAUZE

Frame allocation algoritmen(1)

- Bij single user systeem:
  - eerst OS geven wat het nodig heeft,
  - daarna kan het user proces alle vrije frames gebruiken
- bij multiprogrammering:
  - minimum aantal frames = het maximum aantal frames nodig om één instructie te kunnen uitvoeren. (Architectuur-afhankelijk)
    - b.v. 6 bij MVC in IBM /370
    - b.v. n+2 wanneer n levels van indirectie toegestaan

Frame allocation algoritmen (2)

Stel m frames, n processen

- eenvoudigst : per proces m/n frames (*gelijke allocatie*)
  niet reëel, er zijn 'kleine' en 'grote' processen.
- evenredig met de grootte van het virtuele geheugen van ieder proces: $a_i = s_i / S \times m$, (*evenredige allocatie*)
  Hierbij geldt: $S = \sum s_i$,
  s_i is 'grootte' van proces P_i
  a_i is toewijzing voor proces P_i
  m is aantal aanwezige frames
- eventueel ook rekening houden met prioriteit.

Global versus local page replacement

- bij local page replacement wordt t.b.v. een proces alleen de inhoud van eigen frames vervangen.
- bij global page replacement worden ook frames van andere processen afgepakt.

  probleem: Aantal pagefaults voor een proces wordt mede bepaald door het gedrag van andere processen. Geeft een een zekere onvoorspelbaarheid.

Thrashing

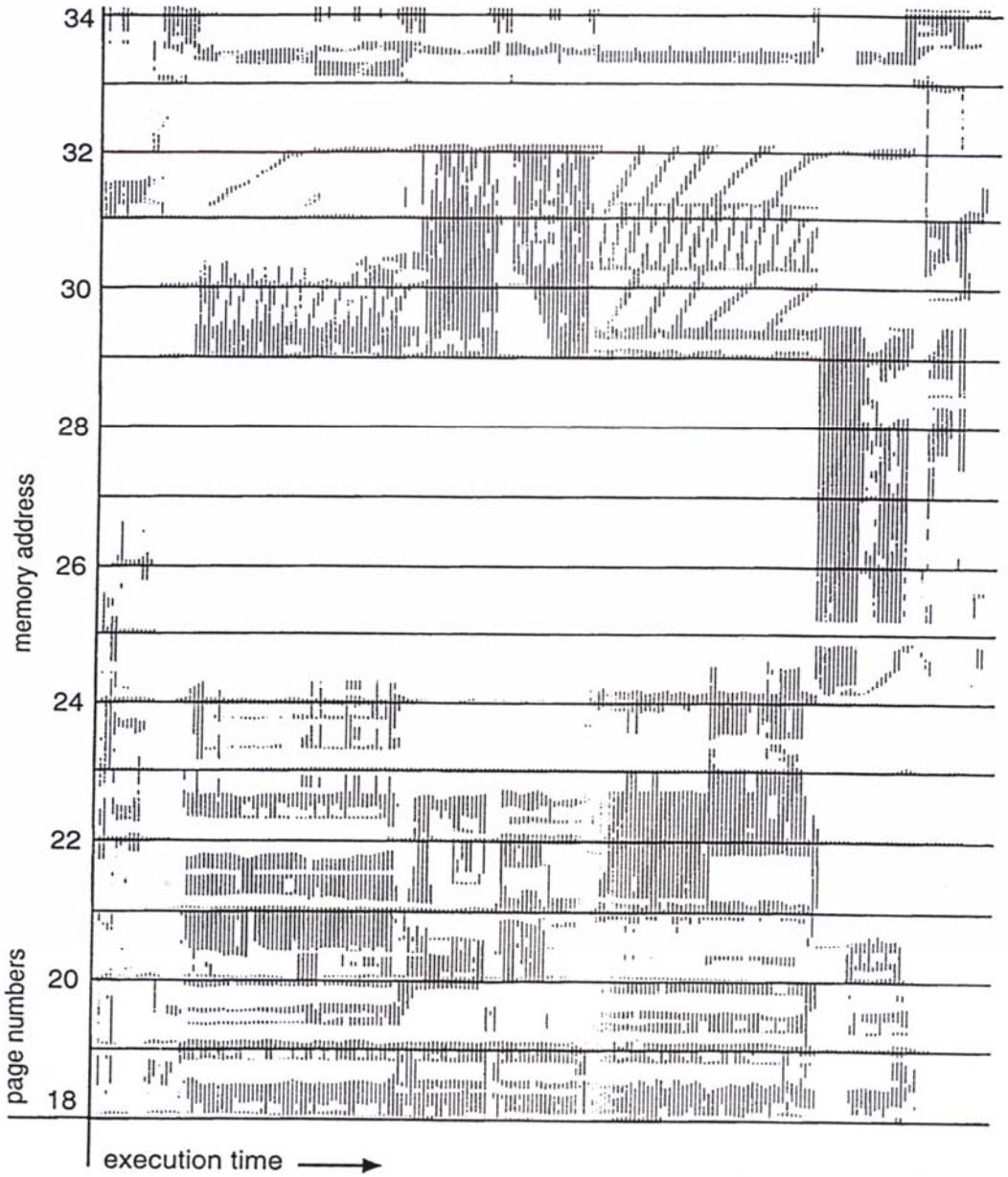
- *Thrashing* treedt op wanneer er teveel pagefaults zijn.
- oorzaak: te weing frames voor het thrashing proces,

  b.v. veroorzaakt door een te hoog *multiprogrammeringslevel (MPL)*

  B.v. stel OS verhoogt MPL bij te lage CPUbezetting, zonder te kijken naar de oorzaak

Thrashing voorkomen (1)
working set strategie

- *Working set strategie* gaat uit van *localiteitsprincipe*
- Δ = *working set window*
- *working set* = verzameling pages gerefereerd in de laatste Δ referenties
- ieder proces i heeft een workingset, size: wss_i
- totaal aantal benodigde frames: $D = \sum wss_i$ voor alle 'actieve' processen.
- Als D te groot worden enkele processen tijdelijk verwijderd (suspended)


34
32
30
28
26
24
22
20
18
memory address
page numbers
execution time

Thrashing voorkomen (2)
Page Fault Frequency (PFF)

Page Fault Frequency strategy:

- per proces bij iedere pagefault de PFF bepalen.
- als PFF $< f_{low}$, proces moet een frame afstaan
- als PFF $> f_{high}$, proces krijgt een frame extra
- als geen frame meer vrij en er is er een nodig, suspend een proces

N.B. PFF gemeten in virtuele tijd

Paging
voetnoten

- prepaging soms zeer effectief
  b.v. bij herstarten van een suspended proces
- paging transparant voor de gebruiker, maar onhandig programmeren maakt paging soms zeer inefficient,
  b.v. matrix kolomsgewijs bewerken.
- soms nodig pages te fixeren in het geheugen (locking)
  - b.v. I/O m.b.v. DMA
  - b.v. net binengehaalde page (referenced=0, dirty=0)
  - real time systemen