TDIU11 – Föreläsning 5

Virtuellt minne (Virtual memory)

Filip Strömbäck

LINKÖPING UNIVERSITY

Planering

Vecka	Föreläsning	Seminarie
4	Processer och trådar	—
5	Filsystem och lagring	Utmaning 1: Schemaläggning
6	Minneshantering	Rapport 1: Filsystem I
7	Virtuellt minne	Utmaning 2: Filsystem
8	Säkerhet	Utmaning 3: Virtuellt minne
9	Repetition/utblickar	Rapport 2: Filsystem II
10	—	Utmaning 4: Säkerhet
11	Tentaförberedelse	Utmaning 5: Repetition (tisdag)
12	(omtenta-p)	Rapport: Uppsamling

1 Paging
2 Alternativ lagraing av page table
3 Virtuellt minne
4 Strategier för page replacement

LINKÖPING UNIVERSITY

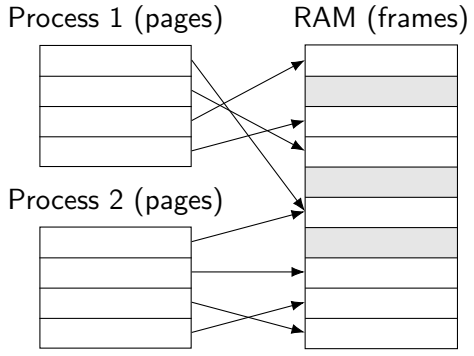
Paging

- Vi vill undvika *extrern fragmentering*
- Idé: Vi delar upp den virtuella adressrymden i *pages*
- Alla *pages* har *samma storlek*

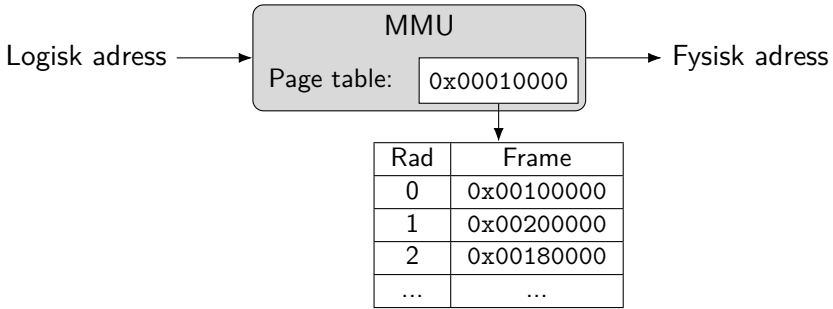
⇒ Vi kan lagra en *page* var som helst i RAM (i vilken *frame* som helst)

⇒ Bara *intern fragmentering* kvar!

- Hur stor ska en *page* vara?



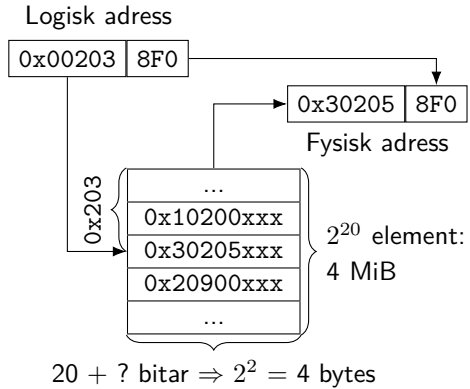
Paging



Olika page tables för olika processer!

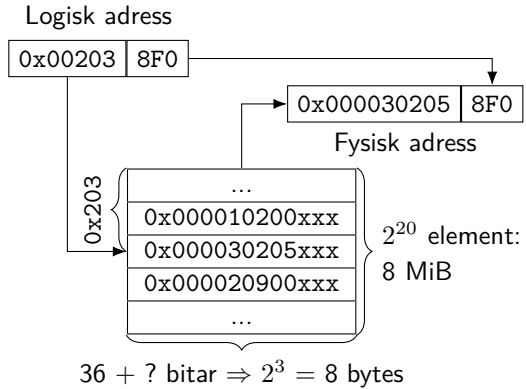
Paging: Hur stor blir page-table?

- 32-bit logiska adresser
- 32-bit fysiska adresser
- 4 KiB page och frame
- Page-table får inte plats i en page ⇒ Fragmentering!
- Kan vi välja en bra page-size?



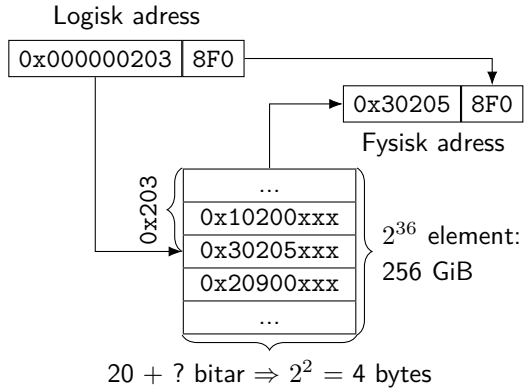
Paging: Hur stor blir page-table?

- 32-bit logiska adresser
- 48-bit fysiska adresser
- 4 KiB page och frame
- Page-table får inte plats i en page ⇒ Fragmentering!
- Kan vi välja en bra page-size?



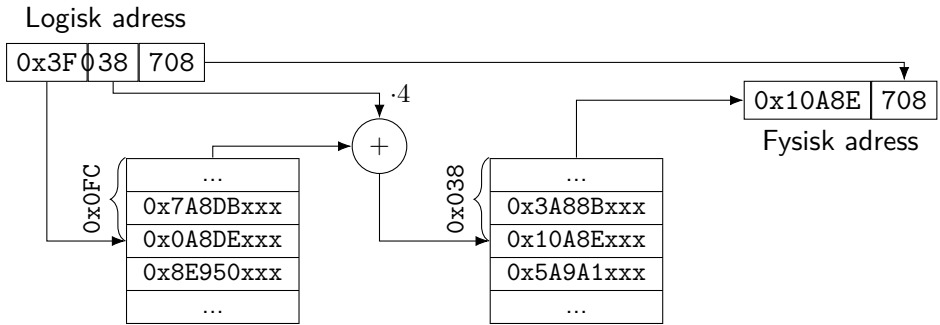
Paging: Hur stor blir page-table?

- 48-bit logiska adresser
- 32-bit fysiska adresser
- 4 KiB page och frame
- Page-table får inte plats i en page ⇒ Fragmentering!
- Kan vi välja en bra page-size?



Hierarkisk Page-table

Idé: Vi lagrar page-table i flera nivåer!



Kostnad av paging

Med 2 nivåer paging:

- Kostar 2 läsningar att slå upp adress
- Varje minnesaccess kostar som 3
- RAM är förhållandevis långsamt...

Lösning:

- TLB (Translation Lookaside Buffer)
- Cache för MMU-beräkningar (64–1024 element)
- Undviker minnesåtkomst om elementet finns

Kostnad av paging

Effective Access Time:
Antag:

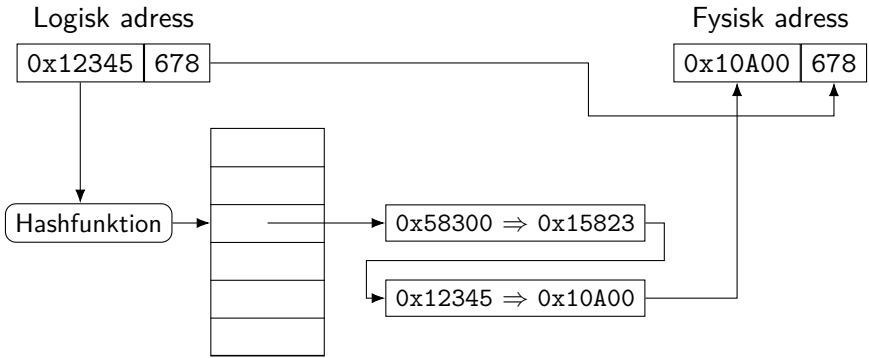
- $\alpha =$ hit-ratio, i %
- $t =$ RAM-access
- $\varepsilon =$ uppslagning i TLB
  Litet i förhållande till t

$$\mathrm{EAT} = (t + \varepsilon)\alpha + (3t + \varepsilon)(1 - \alpha)$$

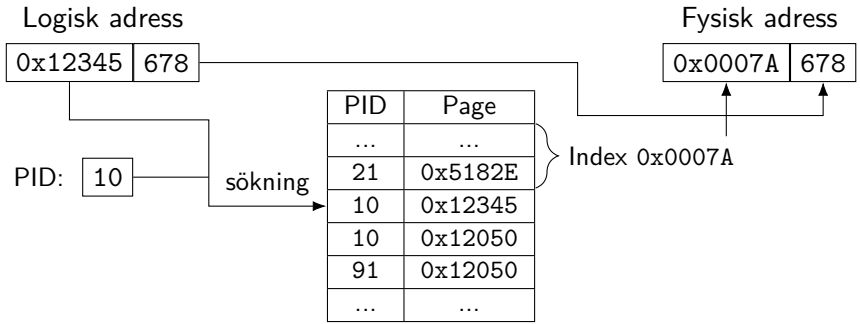
- $\alpha = 80\%$, $t = 100$ ns ger EAT $= 120$ ns
- $\alpha = 99\%$, $t = 100$ ns ger EAT $= 101$ ns

1 Paging
2 Alternativ lagraing av page table
3 Virtuellt minne
4 Strategier för page replacement

Hashad Page-table

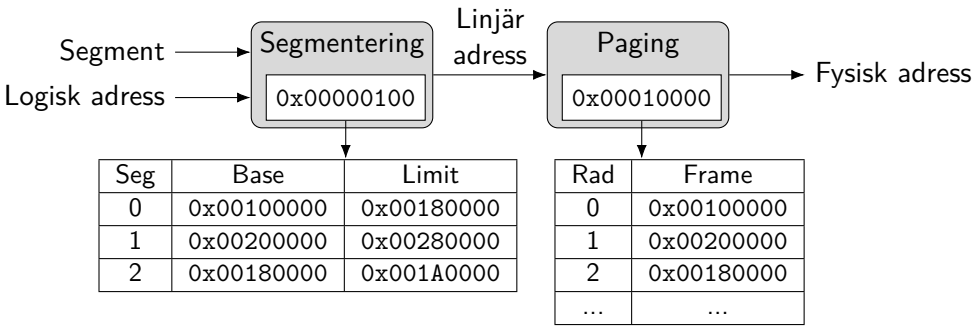


Inverterad Page-table



PT har ett element per *frame*. Svårt att dela minne mellan processer.

Segmentering + Paging (Intel x86)



Seg	Base	Limit
0	0x00100000	0x00180000
1	0x00200000	0x00280000
2	0x00180000	0x001A0000

Rad	Frame
0	0x00100000
1	0x00200000
2	0x00180000
...	...

1 Paging
2 Alternativ lagraing av page table
3 Virtuellt minne
4 Strategier för page replacement

LINKÖPING UNIVERSITY

Storlek på logisk minnesrymd

På 64-bitarssystem har vi 2^{48} bitar $=$ 256 TiB logisk adressrymd.

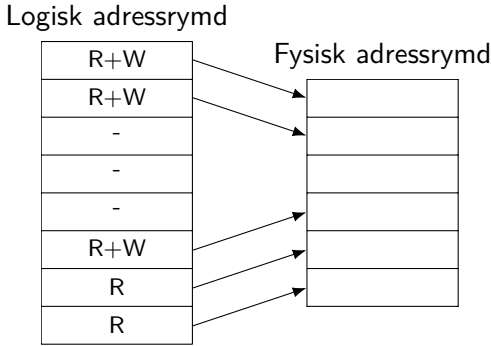
Vi har inte 256 TiB RAM.

Hur fungerar det?

Information i page-tabell

Pagetabell innehåller ytterligare information för varje rad:

- Är raden giltig (valid-bit/present-bit)
- Är raden skrivbar?
- Är raden tillgänglig ifrån user-mode?
- Information om åtkomst



Systemanrop

Windows	UNIX	Beskrivning
`VirtualAlloc`	`mmap`	Gör mer minne tillgängligt i den logiska adressrymden
`VirtualProtect`	`mprotect`	Ändra åtkomst till tillgängligt minne
`VirtualFree`	`munmap`	Frigör minne i den logiska adressrymden

Varför stor logisk adressrymd?

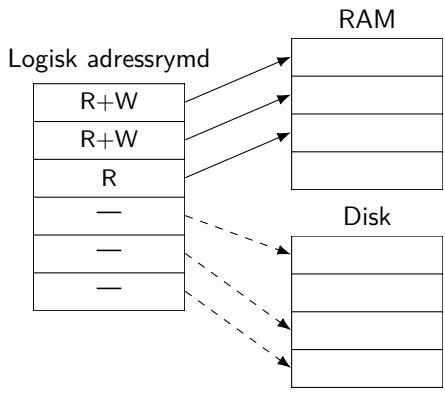
- Möjlighet att använda disk som RAM
- Möjlighet till mer RAM i framtiden
- ASLR (Address Space Layout Randomization)
- Möjlighet till filåtkomst via RAM

Virtuellt minne (Demand paging)

Idé: Använd disk för att lagra pages om vi har slut på frames i RAM!

För pages lagrade på disk:

- Sätt valid/present till 0
- Hårdvaran genererar interrupt när åtkomst sker (pagefault)
- OS kan då läsa page från disk, och starta om instruktionen

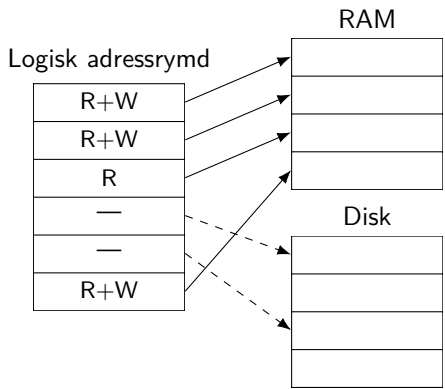


Virtuellt minne (Demand paging)

Idé: Använd disk för att lagra pages om vi har slut på frames i RAM!

För pages lagrade på disk:

- Sätt valid/present till 0
- Hårdvaran genererar interrupt när åtkomst sker (pagefault)
- OS kan då läsa page från disk, och starta om instruktionen

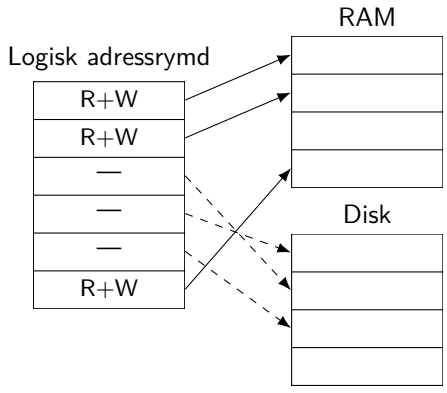


Virtuellt minne (Demand paging)

Idé: Använd disk för att lagra pages om vi har slut på frames i RAM!

För pages lagrade på disk:

- Sätt valid/present till 0
- Hårdvaran genererar interrupt när åtkomst sker (pagefault)
- OS kan då läsa page från disk, och starta om instruktionen

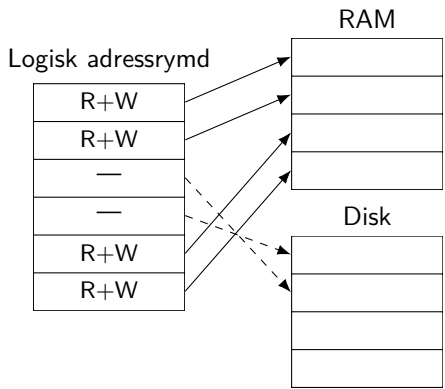


Virtuellt minne (Demand paging)

Idé: Använd disk för att lagra pages om vi har slut på frames i RAM!

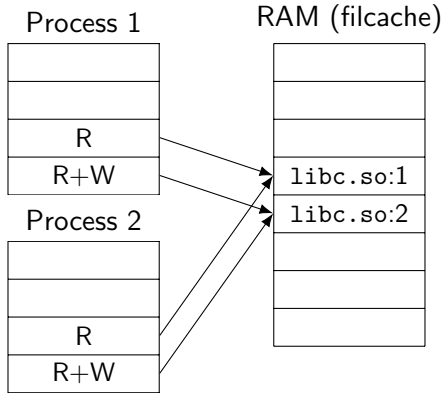
För pages lagrade på disk:

- Sätt valid/present till 0
- Hårdvaran genererar interrupt när åtkomst sker (pagefault)
- OS kan då läsa page från disk, och starta om instruktionen



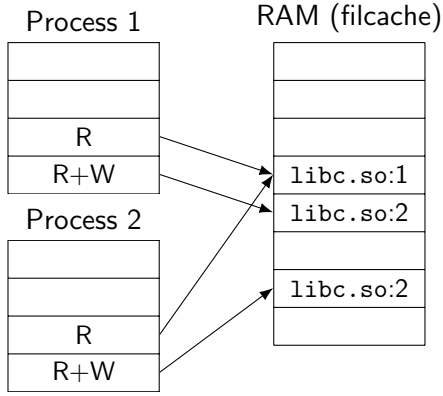
Filer från disk

- Vi kan använda samma teknik för filer på disk!
- Behöver inte "kopiera" filen till RAM
- Kan dela minne mellan olika processer
- Copy-on-write vid behov



Filer från disk

- Vi kan använda samma teknik för filer på disk!
- Behöver inte "kopiera" filen till RAM
- Kan dela minne mellan olika processer
- Copy-on-write vid behov



Demand paging

Fördelar:

- Vi kan köra fler processer än vad som får plats i RAM
- Vi behöver inte läsa hela programmet från disk
- (Vi behöver inte lagra pages som inte har använts)

Nackdelar:

- Disk (även SSD) är *mycket* långsammare än RAM
- Dyrt ifall pages måste flyttas fram och tillbaka till disk
- Extremfall: thrashing

Vad kostar demand paging?

Från förra föreläsningen:

$$\mathrm{EAT} = t_h \cdot \alpha + t_m \cdot (1 - \alpha)$$

- t_h – Tid för en *cache hit*
- t_m – Tid för en *cache miss*
- α – Andel av tiden vi får *cache hit*

För 2-nivå paging:

- $t_h = 100 \text{ ns} + \varepsilon$
- $t_m = 300 \text{ ns} + \varepsilon$
- $\alpha = 99\% = 0.99$

$\mathrm{EAT}_{\mathrm{ram}} = 102 \text{ ns}$

Vad kostar demand paging?

Från förra föreläsningen:

$$\text{EAT} = t_h \cdot \alpha + t_m \cdot (1 - \alpha)$$

- t_h – Tid för en *cache hit*
- t_m – Tid för en *cache miss*
- α – Andel av tiden vi får *cache hit*

Demand paging (med SSD):

- $t_h = 102$ ns
- $t_m = 100$ µs $= 10^5$ ns
- $\alpha = 99\% = 0.99$

$\text{EAT}_{\text{disk}} = 1101$ ns
– För högt!

Vad kostar demand paging?

Från förra föreläsningen:

$$\text{EAT} = t_h \cdot \alpha + t_m \cdot (1 - \alpha)$$

- t_h – Tid för en *cache hit*
- t_m – Tid för en *cache miss*
- α – Andel av tiden vi får *cache hit*

Demand paging (med SSD):

- $t_h = 102$ ns
- $t_m = 100$ µs $= 10^5$ ns
- $\alpha = 99.99\% = 0.9999$

$\text{EAT} \approx 112$ ns

Ett pagefault för varje 10000:e minnesåtkomst

1 Paging
2 Alternativ lagraing av page table
3 Virtuellt minne
4 Strategier för page replacement

Översikt

När en process använder en page som inte finns i RAM:

1. Hitta page på disk
2. Hitta en ledig frame i RAM:
   - Finns ingen: välj en frame som ska ersättas, skriv till disk
3. Kopiera page från disk till vald frame
4. Fortsätt exekevering av processen

Om RAM är fullt blir EAT större

Vilken page ska ersättas?

Om RAM är fullt måste någon page flyttas till disk.

Vilken ska vi välja?

Finns olika algoritmer:

- FIFO (First In First Out)
- Optimal
- LRU (Least Recently Used)

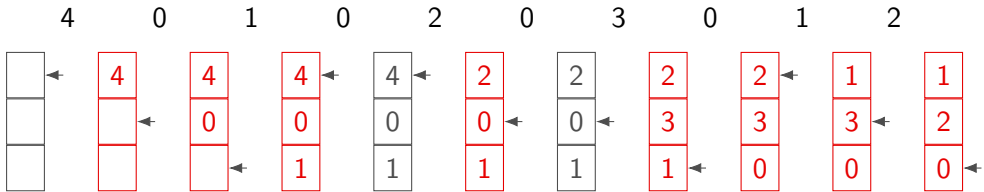
FIFO (First In First Out)

Idé: Välj den äldsta (använd en kö)

4 0 1 0 2 0 3 0 1 2

FIFO (First In First Out)

Idé: Välj den äldsta (använd en kö)



Optimal

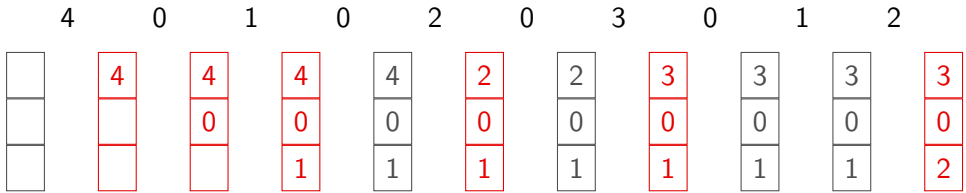
Idé: Välj den page som inte kommer användas på längst tid

4 0 1 0 2 0 3 0 1 2

Problem: Kan inte se in i framtiden... Bra referenspunkt.

Optimal

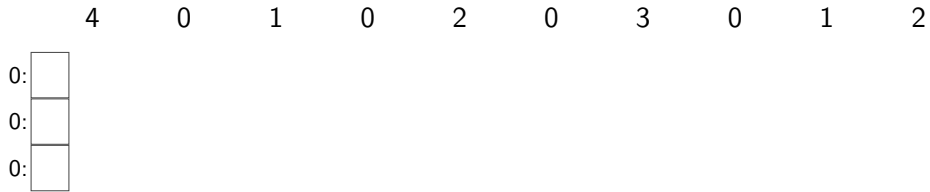
Idé: Välj den page som inte kommer användas på längst tid



Problem: Kan inte se in i framtiden... Bra referenspunkt.

LRU (Least Recently Used)

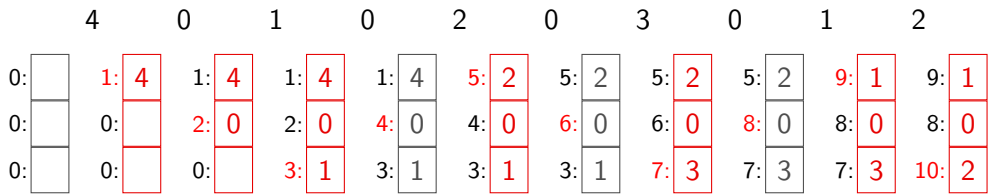
Idé: Approximera optimal algoritm genom att använda den som inte använts på längst tid



Problem: Hur håller vi koll på detta?

LRU (Least Recently Used)

Idé: Approximera optimal algoritm genom att använda den som inte använts på längst tid



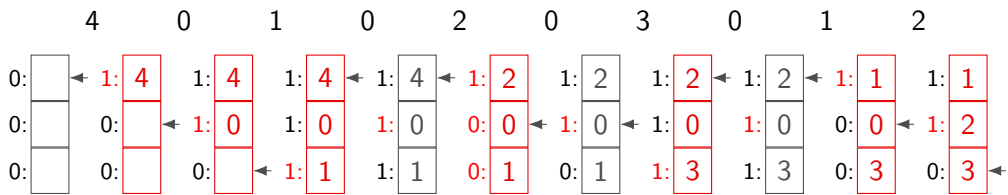
Problem: Hur håller vi koll på detta?

Approximativ LRU – Klockalgoritm (Second-chance algorithm)

Idé: Hårdvaran noterar när pages har använts, vi kör FIFO men hoppar över pages som använts!

Approximativ LRU – Klockalgoritm (Second-chance algorithm)

Idé: Hårdvaran noterar när pages har använts, vi kör FIFO men hoppar över pages som använts!



Hur många frames ska varje process få?

Finns olika metoder:

- Lika många frames/process
- Proportionerligt mot storlek

Om för liten $\Rightarrow$ thrashing

Working set kan estimera minimum:

- Antal pages som aktivt används
- Approximation: antal pages som använts senaste x tiden
- Varierar under programmets körning

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY