TDIU11 – Föreläsning 4

Virtuellt minne (Virtual memory)

Filip Strömbäck

LINKÖPING UNIVERSITY

Planering

Vecka	Föreläsning	Seminarie
3	Processer och trådar	—
4	Minneshantering	Challenge 1: Schemaläggning
5	Virtuellt minne	Artikel 1: Schemaläggning
6	Filsystem och lagring	Challenge 2: Virtuellt minne
7	Säkerhet	Challenge 3: Filsystem
8	Repetition/utblickar	Artikel 2: Filsystem
9	—	Challenge 4: Säkerhet
10	Tentaförberedelse	Challenge 5: Repetition
11	(omtenta-p)	Artikel 3: Säkerhet (reserv)

LINKÖPING UNIVERSITY

Hierarkisk paging – Repetition

Vi har:

- 48 bitar logiska adresser
- 38 bitar fysiska adresser
- 1 KiB pages

Hur många nivåer behövs?
Hur sker adressuppslagning?
Hur mycket minne tar page-tabeller?
(`cat /proc/meminfo`)

Hierarkisk paging – Repetition

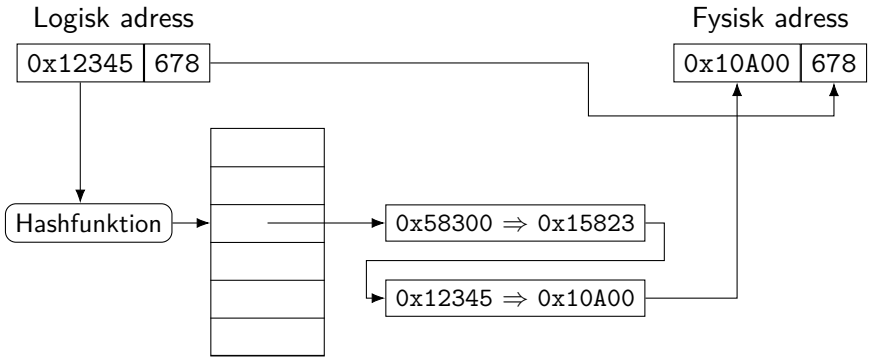
Vi har:

- 48 bitar logiska adresser
- 38 bitar fysiska adresser
- 1 KiB pages

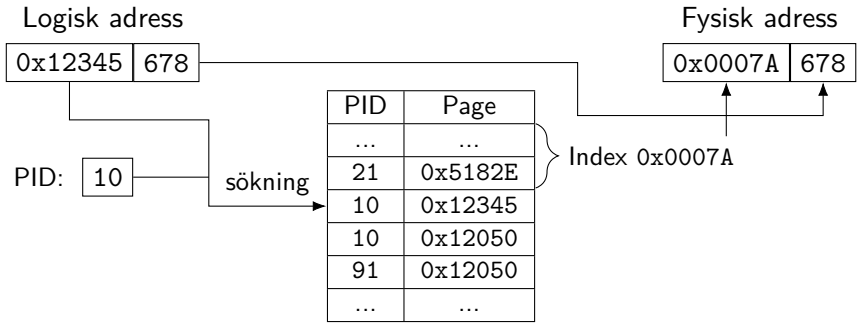
Hur många nivåer behövs?
Hur sker adressuppslagning?
Hur mycket minne tar page-tabeller?
(`cat /proc/meminfo`)

- 1 KiB $= 2^{10}$ bytes
- Offset: 10 bitar
- Frame-nummer: 28 bitar
- 4 bytes/rad i pagetabell
- Index i pagetabell: 8 bitar
- Logisk adress:
  $6 + 8 + 8 + 8 + 8 + 10$ bitar

Hashad Page-tabell

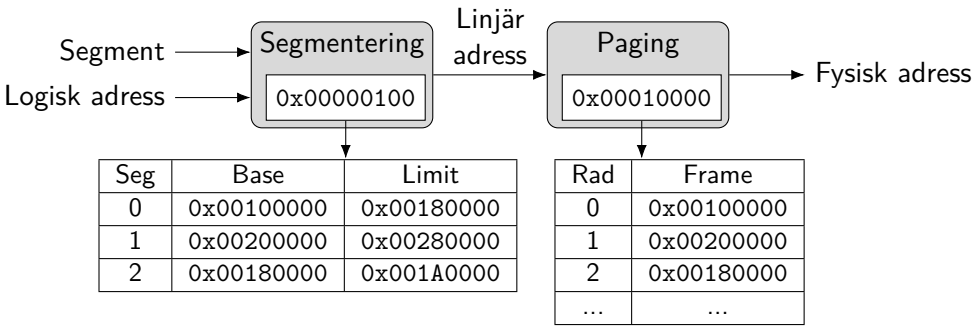


Inverterad Page-tabell



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
...	...

LINKÖPING UNIVERSITY

Storlek på logisk minnesrymd

På 64-bitarssystem har vi 256 TiB logisk adressrymd.

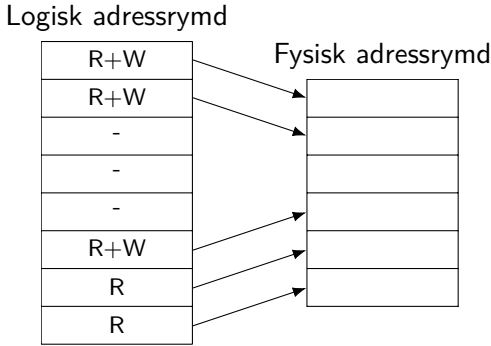
Vi har inte 256 TiB RAM.

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?

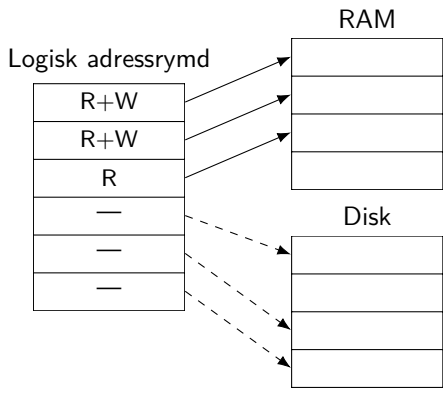
- Möjlighet till mer RAM i framtiden
- ASLR (Address Space Layout Randomization)
- Möjlighet till filåtkomst via RAM

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

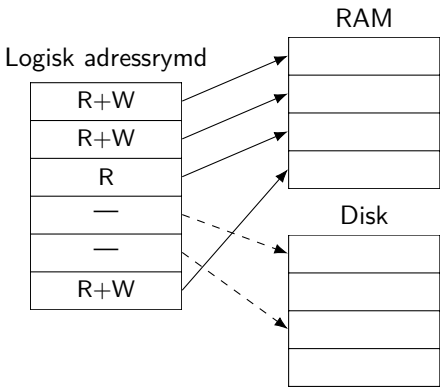


Virtuellt minne (Demand paging)

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

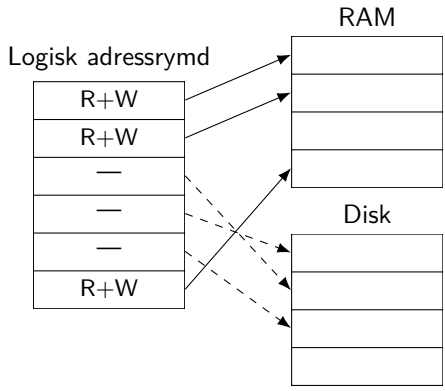


Virtuellt minne (Demand paging)

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

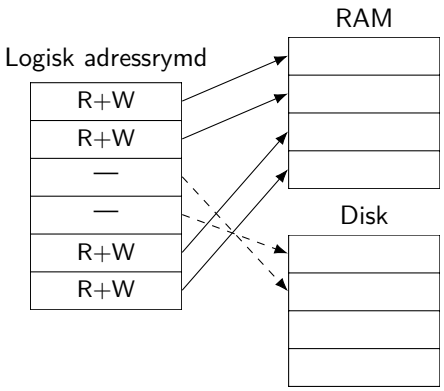


Virtuellt minne (Demand paging)

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



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:

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

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

För 2-nivå paging:

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

$\text{EAT} = 102 \text{ ns}$

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):

- $t_m = 100 \text{ ns}$
- $t_m = 100 \text{ µs} = 10^5 \text{ ns}$
- $\alpha = 99\% = 0.99$

$\text{EAT} = 1099 \text{ ns}$ – 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):

- $t_m = 100$ ns
- $t_m = 100$ µs $= 10^5$ ns
- $\alpha = 99.98\% = 0.9998$

$\text{EAT} \approx 110$ ns

Ett pagefault för varje 5000:e minnesåtkomst

LINKÖPING UNIVERSITY

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)

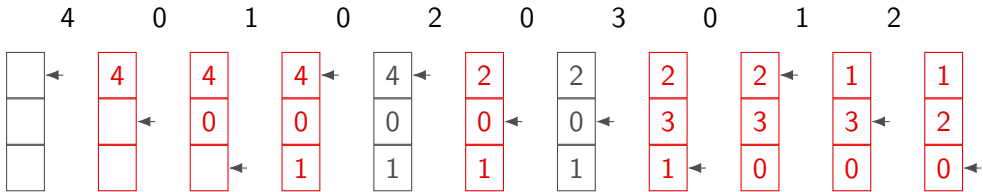
FIFO (First In First Out)

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

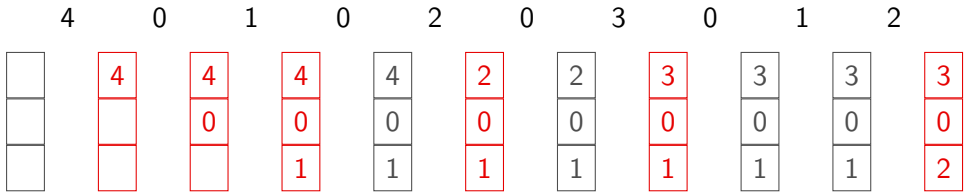
Idé: Välj den page som inte kommer användas på längst tid

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)

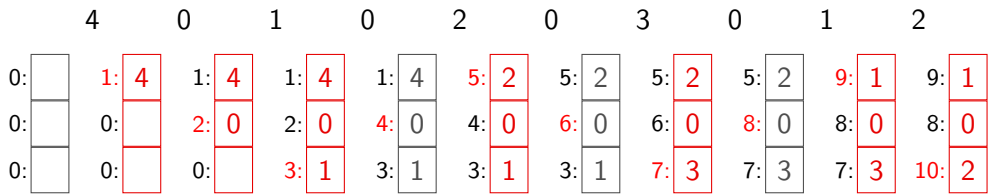
Idé: Approximera optimal algoritm genom att använda den som inte använts på längst tid

	4	0	1	0	2	0	3	0	1	2
0:										
0:										
0:										

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



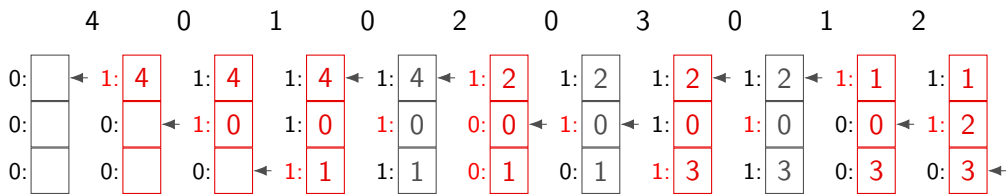
Problem: Hur håller vi koll på detta?

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

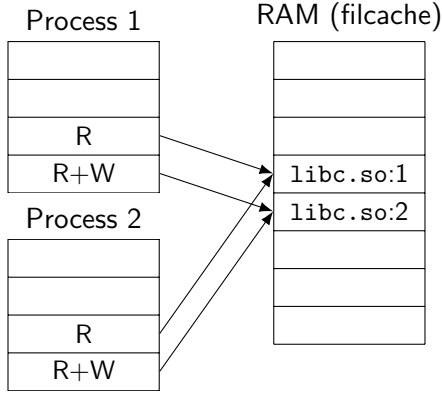
Om för liten $\Rightarrow$ thrashing

Working set kan estimera minimum:

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

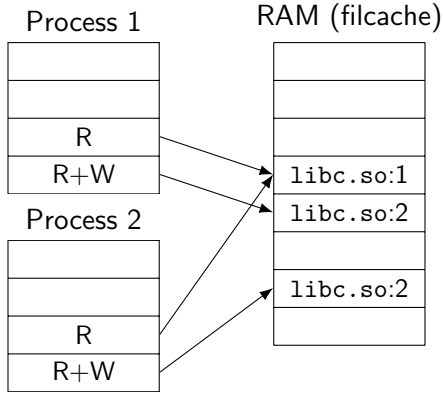
Filer från disk

- 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



LINKÖPING UNIVERSITY

Nästa föreläsning

- Hur lagrar vi data på disk?

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY