TDIU11 – Föreläsning 4

Minneshantering (Memory management)

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

LINKÖPING UNIVERSITY

Hur vet vi vad som är ledigt?

Bitmap:

- 1 bit/logiskt block
- Blir snabbt stor

Länkad:

- Länka lediga block
- Svårt att hitta sekventiella block

Länkad med räknare:

- Räkna sekventiella lediga block
- Fortfarande dyr att traversera

Indexerad:

- Lagra en ”inod” med alla lediga block
- Kan enkelt plocka indexblock till ny fil

Abstraktion för inoder

Möjligt gränssnitt:

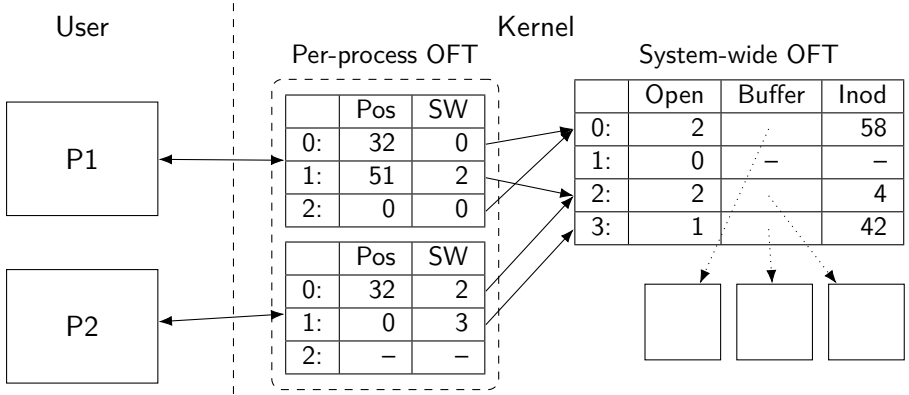
```
int inode_size(int id);

int inode_read(int id, int offset, byte *data, int size);

int inode_write(int id, int offset, const byte *data, int size);
```

Hur kan OS veta vad som borde laddas in i RAM?

Datastrukturer i kernel



Smidigare abstraktion för inoder

Idé: Vi kräver att man *öppnar* en inod först!

```
int  inode_open(int inode_id);
void inode_close(int fd);

int  inode_size(int fd);
void inode_seek(int fd, int pos);
int  inode_read(int fd, byte *data, int size);
int  inode_write(int fd, const byte *data, int size);
```

Hur namnger vi inoder?

LINKÖPING UNIVERSITY

Hur namnger vi filer?

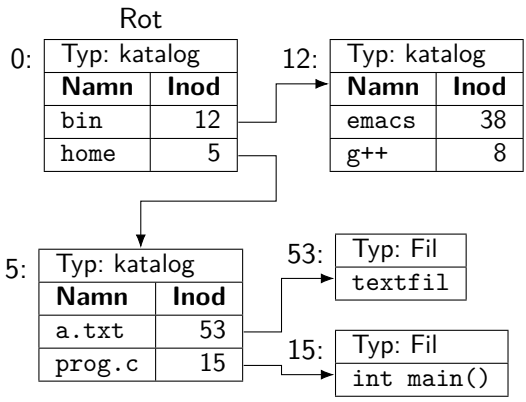
Idé: Lagra namn + id i inoder, vi kallar dem *katalog*-inoder

Tabell med:

- Namn
- Inod

Inod innehåller:

- Om katalog/fil
- Rättigheter



Länkar i systemet

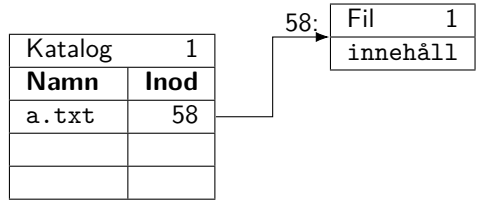
Idé: Tillåt flera referenser till samma fil

- Hårda länkar
- Symboliska länkar

I inod:

- Antal referenser

Varför inte hård länk till katalog?



Länkar i systemet

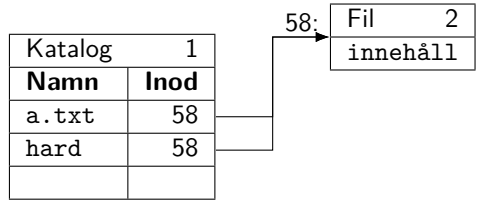
Idé: Tillåt flera referenser till samma fil

- Hårda länkar
- Symboliska länkar

I inod:

- Antal referenser

Varför inte hård länk till katalog?



Länkar i systemet

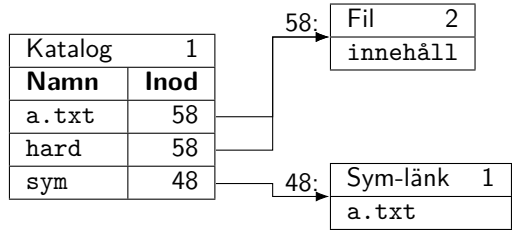
Idé: Tillåt flera referenser till samma fil

- Hårda länkar
- Symboliska länkar

I inod:

- Antal referenser

Varför inte hård länk till katalog?



Abstraktion för filer

Nu kan vi använda filnamn och sökvägar!

```
int  open(const char *name);
void close(int fd);

int  size(int fd);
void seek(int fd, int pos);
int  read(int fd, byte *data, int size);
int  write(int fd, const byte *data, int size);
```

Abstraktion för filer (forts.)

Nu kan vi använda filnamn och sökvägar!

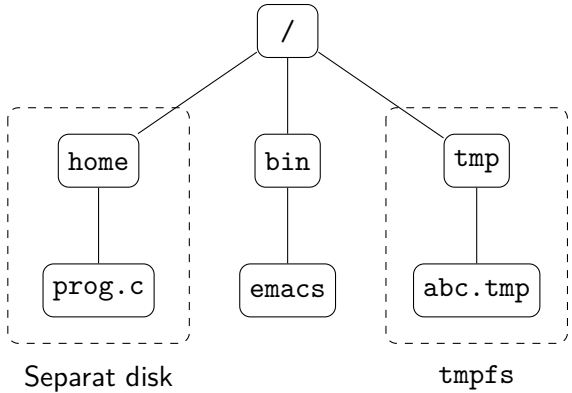
```
DIR *opendir(const char *name);
void closedir(DIR *dir);
dirent *readdir(DIR *dir);

void unlink(const char *name);
void rename(const char *old, const char *new);
```

Virtuellt filsystem

Många system har ett *virtuellt filsystem*:

- Program ser ett filsystem
- Finns egentligen olika filsystem



LINKÖPING UNIVERSITY

Mål

Process 1

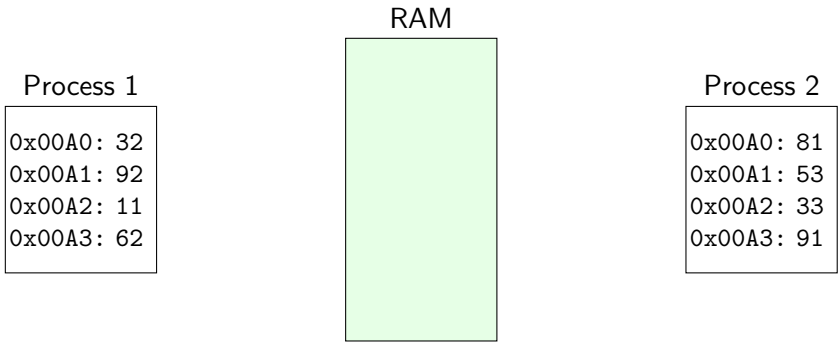
```
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
```

Hur går detta ihop?

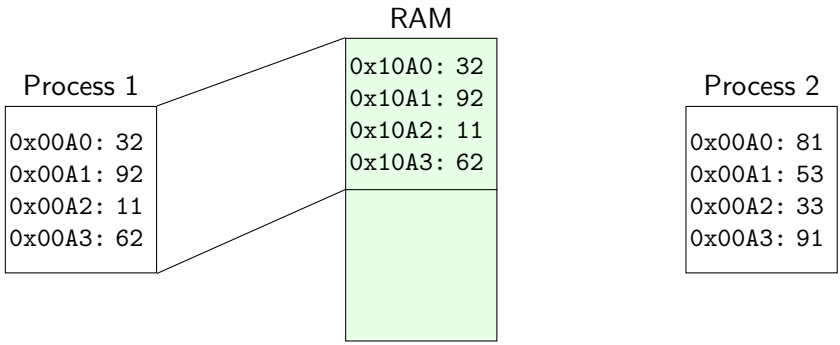
Process 2

```
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91
```

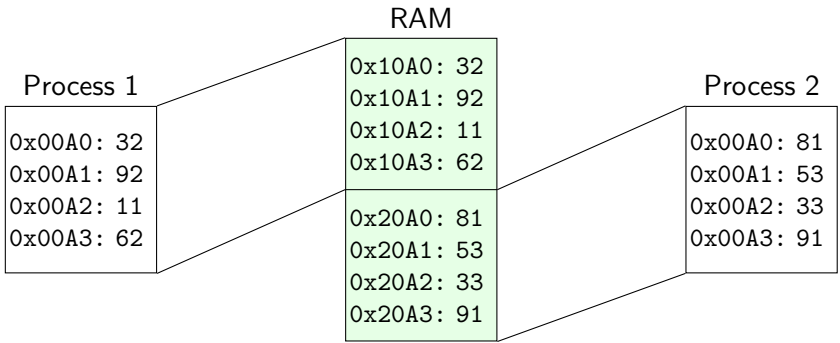
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91

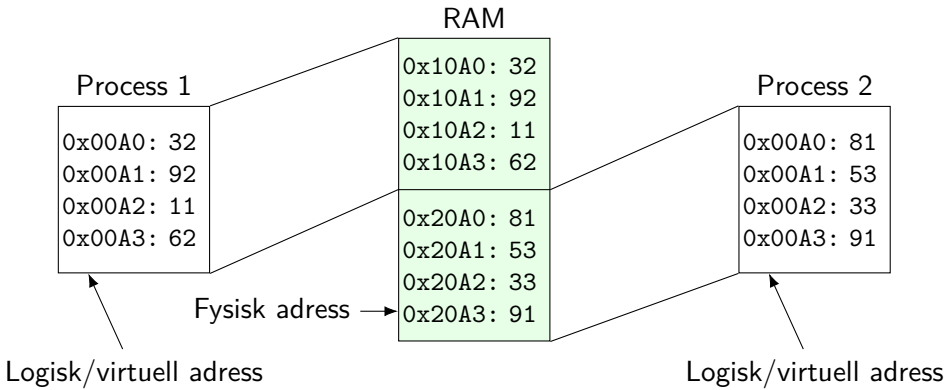
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
0x10A0: 32
0x10A1: 92
0x10A2: 11
0x10A3: 62
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91

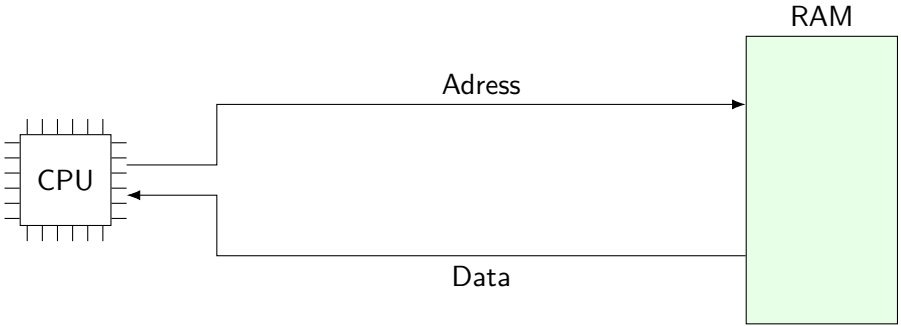
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
0x10A0: 32
0x10A1: 92
0x10A2: 11
0x10A3: 62
0x20A0: 81
0x20A1: 53
0x20A2: 33
0x20A3: 91
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91

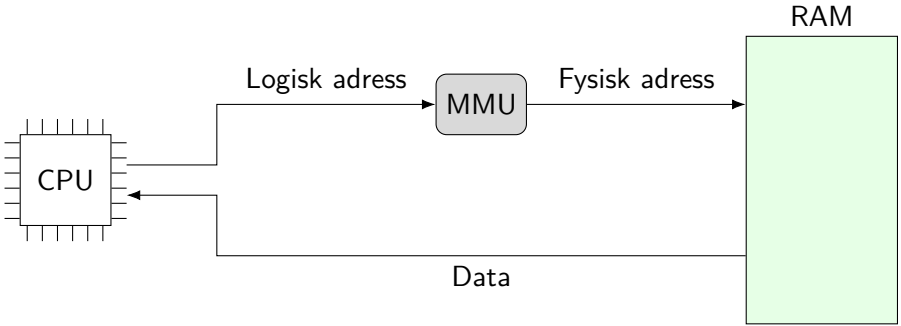
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
0x10A0: 32
0x10A1: 92
0x10A2: 11
0x10A3: 62
0x20A0: 81
0x20A1: 53
0x20A2: 33
0x20A3: 91
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91
Fysisk adress
Logisk/virtuell adress
Logisk/virtuell adress

Hur löser vi detta?



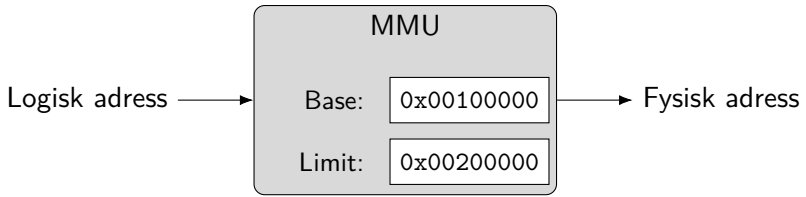
Hur löser vi detta?



Notera: logisk och fysisk adress kan vara olika stora!

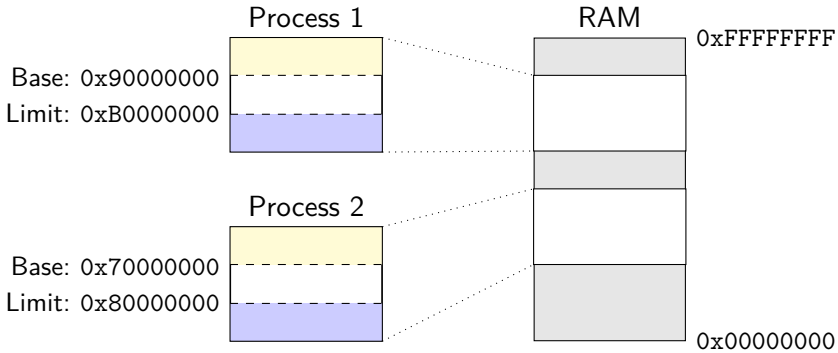
LINKÖPING UNIVERSITY

Kontinuerlig (Continuous allocation)

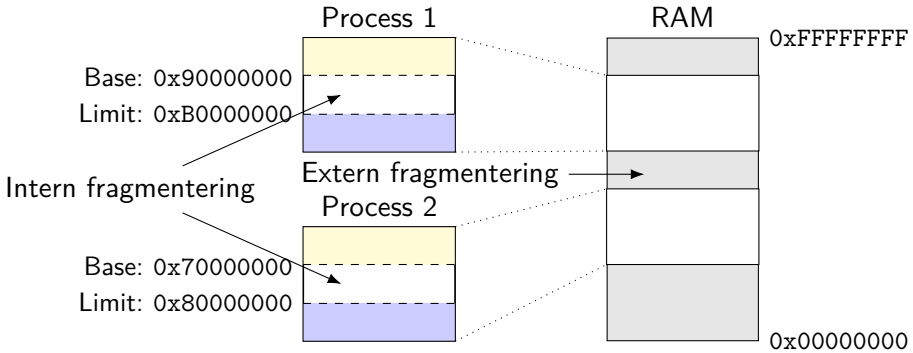


Notera: Varje process har olika MMU-inställningar!

Kontinuerlig (Continuous allocation)

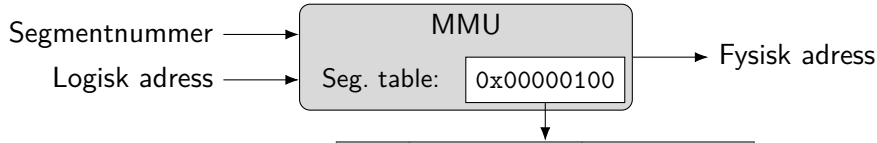


Kontinuerlig (Continuous allocation)



LINKÖPING UNIVERSITY

Segmentering

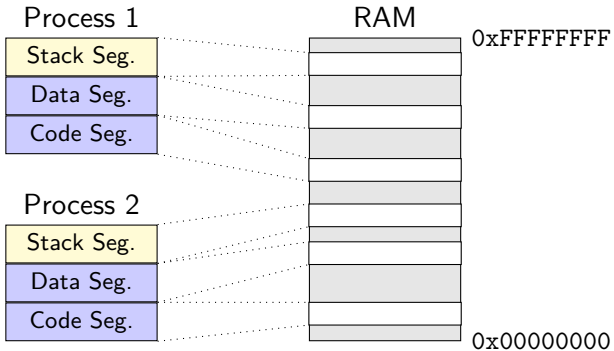


Seg	Base	Limit
0	0x00100000	0x00180000
1	0x00200000	0x00280000
2	0x00180000	0x001A0000

Mindre block $\Rightarrow$ Mindre problem med fragmentering

Notera: Varje process har olika segmenttabeller!

Segmentering

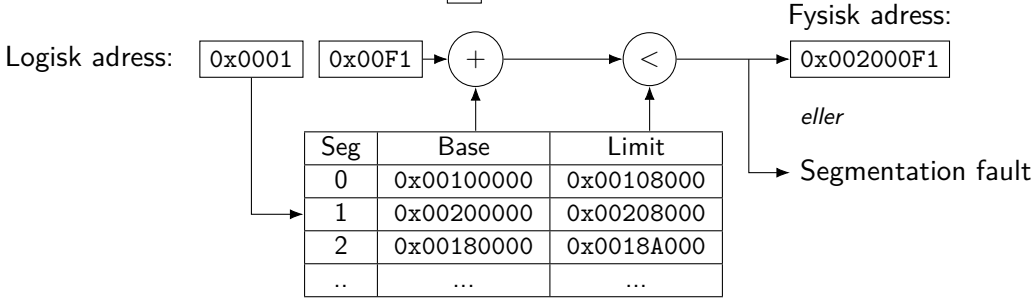


Segmentering: Hur ser logiska adresser ut?

Alternativ 1: Segmentregister (t.ex. Protected Mode i Intel x86)

```
mov ds:[0x00F1], 0x4F
```

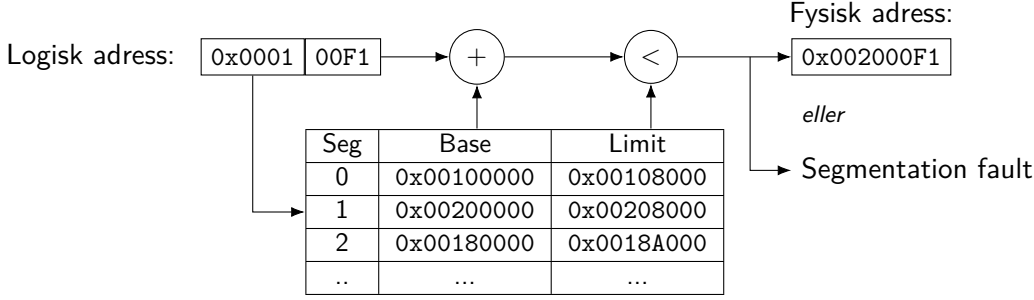
DS: 1



Segmentering: Hur ser logiska adresser ut?

Alternativ 2: Mest signifikanta bitar i logisk adress

```
mov [0x000100F1], 0x4F
```



LINKÖPING UNIVERSITY

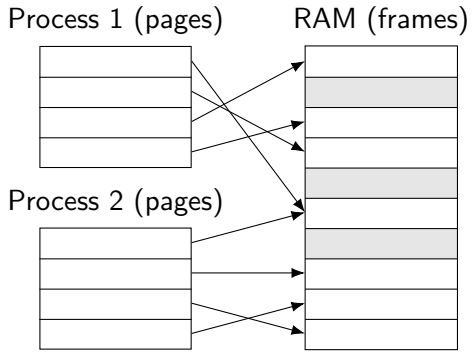
Paging

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

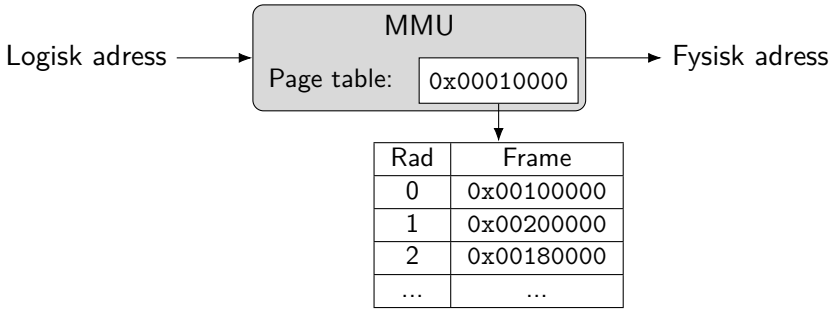
⇒ Vi kan lagra en *page* var som helst i RAM (i vilken *frame* som helst)

⇒ Bara *intern fragmentering* kvar!

- Hur stor ska en *page* vara?



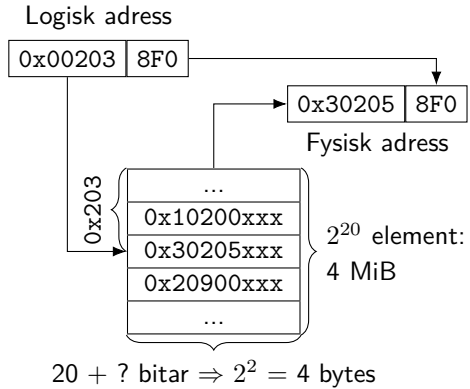
Paging



Olika page tables för olika processer!

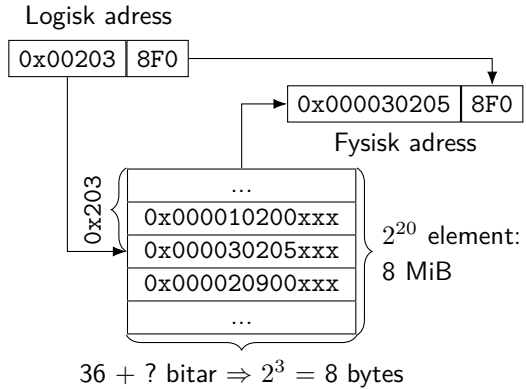
Paging: Hur stor blir page-table?

- 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?



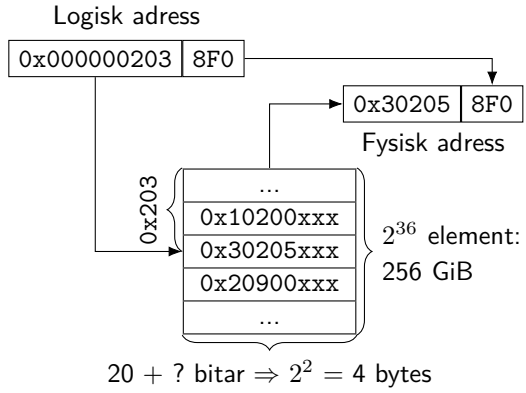
Paging: Hur stor blir page-table?

- 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?



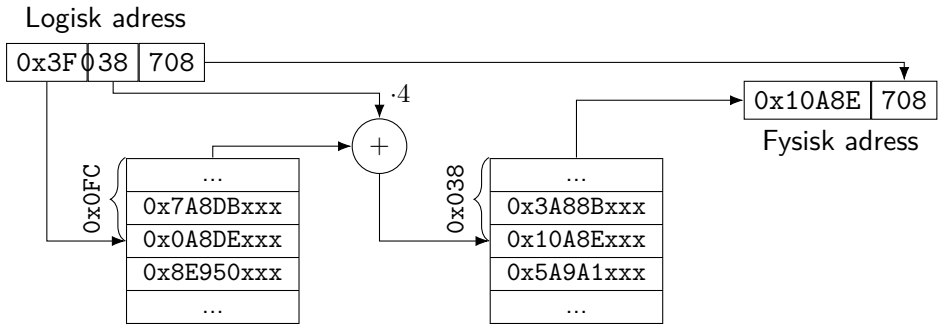
Paging: Hur stor blir page-table?

- 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:

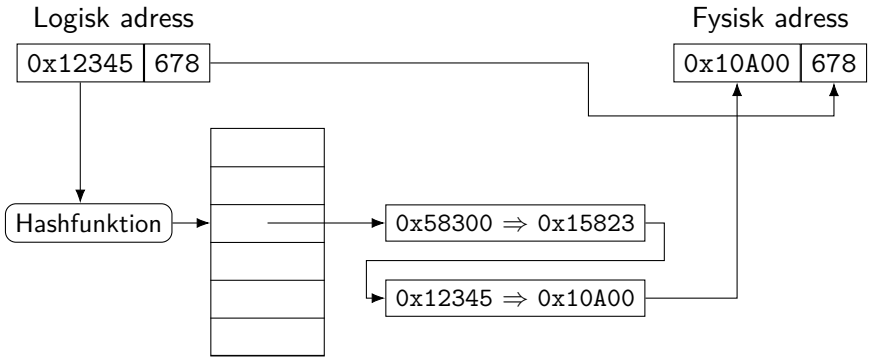
- α = hit-ratio, i %
- t = RAM-access
- ε = uppslagning i TLB
  Litet i förhållande till t

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

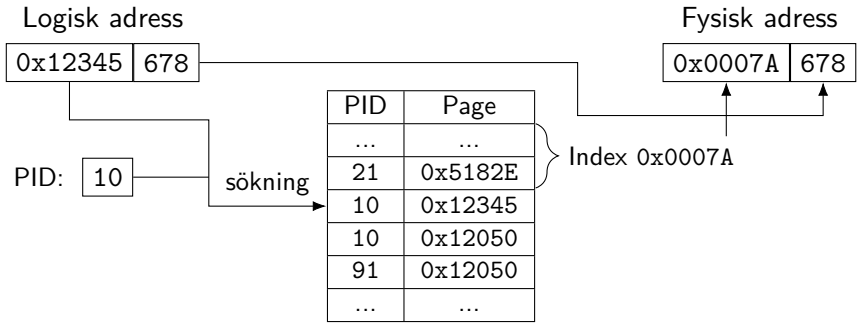
- $\alpha = 80\%$, $t = 100$ ns ger EAT = 120 ns
- $\alpha = 99\%$, $t = 100$ ns ger EAT = 101 ns

LINKÖPING UNIVERSITY

Hashad Page-table

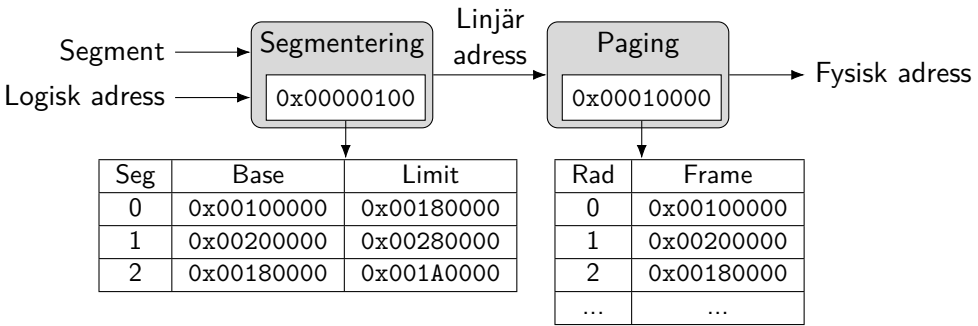


Inverterad Page-table



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
...	...

Nästa vecka

På 64-bitarssystem har vi 256 TiB logisk adressrymd, men mycket mindre RAM.

- Varför är det användbart?
- Hur kan vi använda mer minne än vad som finns tillgängligt?

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY