TDIU11 – Föreläsning 3

Filsystem

Filip Strömbäck

LINKÖPING UNIVERSITY

Planering

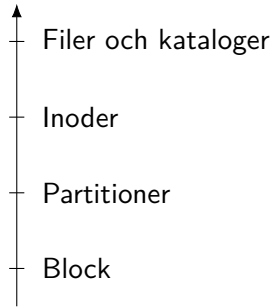
Vecka	Föreläsning	Seminarie
4	Processer och trådar	—
5	Filsystem och lagring	Utmaning 1: Schemaläggning
6	Minneshantering	Rapport 1: Filsystem I
7	Virtuellt minne	Utmaning 2: Filsystem
8	Säkerhet	Utmaning 3: Virtuellt minne
9	Repetition/utblickar	Rapport 2: Filsystem II
10	—	Utmaning 4: Säkerhet
11	Tentaförberedelse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

LINKÖPING UNIVERSITY

Mål med föreläsningen

- Operativsystemet ger oss:
  filer (*files*) och
  kataloger (*directories*)[a]
- En disk är en array av *block* där OS kan lagra data
- Hur lagrar vi *filer* i en array av *block*?

[a] Notera: *mapp* (*folder*) är tekniskt sett något annat



Hur fungerar en disk?

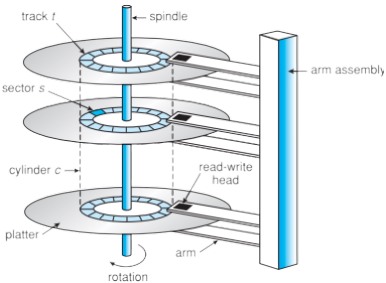
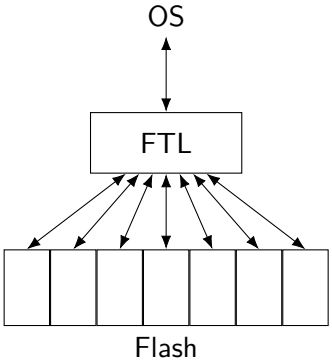


Bild från material till
Operating System Concepts

- Minsta enhet: sektor (block)
  Historiskt: 512 B, numera ca 4 KiB
- Åtkomstid: byt spår (*track*) + vänta på sektor
- Söktid: 3–12 ms
- Vänta på sektor: $\frac{60}{\text{RPM}}$
  För 5400 RPM: 11.2 ms max, 5.5 ms medel.

Hur fungerar en SSD?



- Minsta enhet: block, ca 4 KiB
- Flash Translation Layer (FTL):
  - Wear-levelling
  - Rensar använda block
  - Emulering av mindre blockstorlek
- Åtkomsttid: svarstid hos FTL och flash-minne, kan hantera flera operationer samtidigt

Abstraktion för att hantera diskar

Möjligt gränssnitt:

```
int get_block_size(int disk);

void read_block(int disk, int block, byte *data);

void write_block(int disk, int block, const byte *data);
```

Hur lagrar vi filer?

LINKÖPING UNIVERSITY

Hur stor är en megabyte/mebibyte?

- Många storlekar inom data är 2^n för något heltal $n \geq 0$
- Det är smidigt att säga 1 KiB $= 2^{10}$ bytes
- Hur många bytes är 1 MiB?

Prefix	SI		Tvåpotens			
kilo	1 kB =	10^3 B	1 KiB =	2^{10} B =	1024 B	
mega	1 MB =	10^6 B	1 MiB =	2^{20} B =	1024 KiB =	1048576 B
giga	1 GB =	10^9 B	1 GiB =	2^{30} B =	1024 MiB =	1073741824 B

Räkna med tvåpotenser

Ofta smidigt att behålla tal i bas 2:

Exempel: Vi har en disk som är 8 GiB stor med 512 byte block. Hur många block har vi?

$$8 \text{ GiB} = 2^3 \cdot 2^{30} \text{ bytes}$$

$$\frac{2^{33}}{2^9} = 2^{24} \text{ block}$$

(Tiotalet i exponenten ger rätt prefix)

$2^0 =$	1		
$2^1 =$	2		
$2^2 =$	4	$2^{10} =$	1 KiB
$2^3 =$	8	$2^{20} =$	1 MiB
$2^4 =$	16	$2^{30} =$	1 GiB
$2^5 =$	32	$2^{40} =$	1 TiB
$2^6 =$	64	$2^{50} =$	1 PiB
$2^7 =$	128	$2^{60} =$	1 EiB
$2^8 =$	256		
$2^9 =$	512		

Räkna med tvåpotenser

Ofta smidigt att behålla tal i bas 2:

Exempel: Vi använder 32-bitars heltal för att indexera 1 KiB stora block. Hur stor kan vår disk vara?

$$2^{32} \cdot 2^{10} \text{ bytes} = 2^{42} \text{ bytes} =$$
$$= 2^2 \cdot 2^{40} \text{ bytes} = 2^2 \text{ TiB} = 4 \text{ TiB}$$

(Tiotalet i exponenten ger rätt prefix)

$2^0 =$	1		
$2^1 =$	2		
$2^2 =$	4	$2^{10} =$	1 KiB
$2^3 =$	8	$2^{20} =$	1 MiB
$2^4 =$	16	$2^{30} =$	1 GiB
$2^5 =$	32	$2^{40} =$	1 TiB
$2^6 =$	64	$2^{50} =$	1 PiB
$2^7 =$	128	$2^{60} =$	1 EiB
$2^8 =$	256		
$2^9 =$	512		

Hexadecimala tal

Ofta smidigt att jobba i bas 16 (hexadecimalt):

- 0-9, A-F $\Rightarrow$ Varje position är 4 bitar
- Konvention i kursen: Prefix `0x` som i C och C++

Exempel: 32-bitars tal

Hex:	`0x`	`1`	`2`	`4`	`8`	`1`	`3`	`7`	`F`
Bin:		`0001`	`0010`	`0100`	`1000`	`0001`	`0011`	`0111`	`1111`

LINKÖPING UNIVERSITY

Schemaläggning av diskåtkomst

För mekaniska diskar:

- Söktid (accesstid) tar mest tid
- Läs block ”nära” varandra om möjligt!

Några alternativ:

- FCFS
- SSTF
- SCAN, C-SCAN
- LOOK, C-LOOK

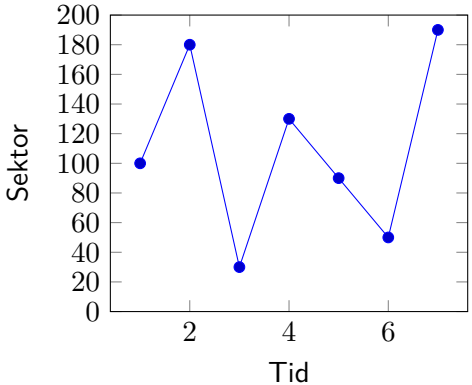
Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

FCFS – First Come First Served

Idé: Använd ordningen i kön

Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

Totalt: 550

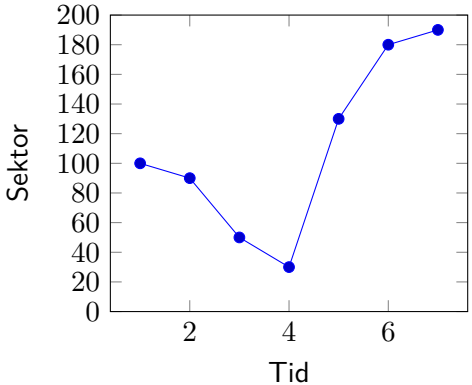


SSTF – Shortest Search Time First

Idé: Välj det närmst nuvarande position först

Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

Totalt: 230
Risk för *starvation*

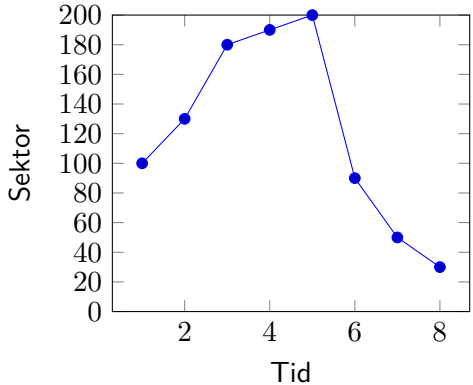


SCAN

Idé: Gå fram och tillbaka mellan minsta och största sektorn

Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

Totalt: 270

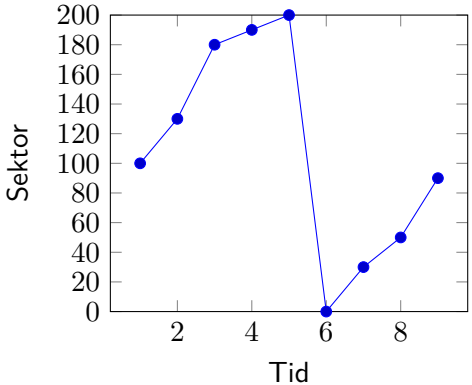


C-SCAN

Idé: Gå från 0 till 200, börja sedan om igen.

Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

Totalt: 390
Jämnare accesstid

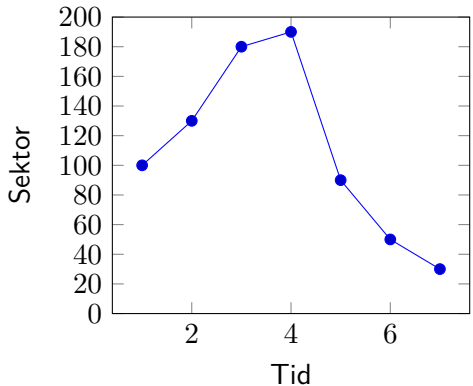


LOOK

Idé: Gå mellan största och minsta access, fram och tillbaka.

Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

Totalt: 250

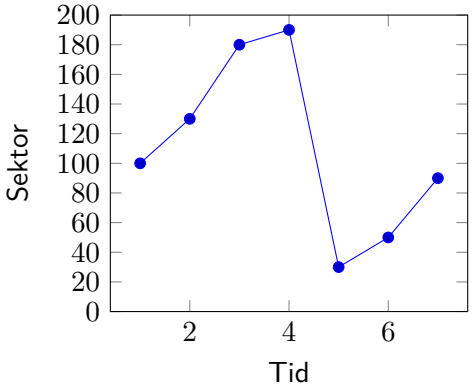


C-LOOK

Idé: Gå mellan största och minsta access, cirkulärt

Exempel:
Disk med 200 sektorer
Start: 100, kom från 10
Kö: 180, 30, 130, 90, 50, 190

Totalt: 310
Jämnare accesstid



Räkna total söktid – I Emacs-lisp!

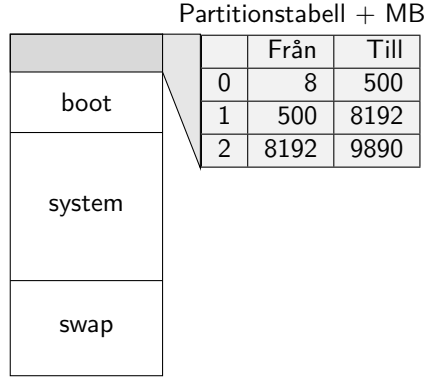
```
(defun travel (elems)
  (if (null (cdr elems))
      0
    (+ (abs (- (car elems) (car (cdr elems))))
       (travel (cdr elems)))))

(travel '(100 180 30 130 90 50 190))        ;; FCFS
(travel '(100 90 50 30 130 180 190))        ;; SSTF
(travel '(100 130 180 190 200 90 50 30))    ;; SCAN
(travel '(100 130 180 190 200 0 30 50 90)) ;; C-SCAN
(travel '(100 130 180 190 90 50 30))        ;; LOOK
(travel '(100 130 180 190 30 50 90))        ;; C-LOOK
```

LINKÖPING UNIVERSITY

Partitioner

- En disk kan delas upp i *partitioner*
- Varje *partition* är en samling block som ligger efter varandra
- Olika *partitioner* är oberoende av varandra, kan innehålla olika filsystem
- Exempelvis:
  - FAT32 för uppstartsfiler (UEFI)
  - ext4/NTFS för systemet
  - Partition för swap
  - Återställningspartition



Abstraktion för partitioner

Möjligt gränssnitt:

```
int get_block_size(int disk);

int get_partition_count(int disk);

void read_block(int disk, int part, int block, byte *data);

void write_block(int disk, int part, int block, const byte *data);
```

Hur lagrar vi filer?

LINKÖPING UNIVERSITY

Filsystem

- Lager ovanpå en partition
- Beskriver hur vi lagrar filer och kataloger
- Många olika med olika för- och nackdelar
  - FAT32, exFAT
  - NTFS
  - ext3, ext4, ...
  - HFS
  - ...

Blockstorlek:

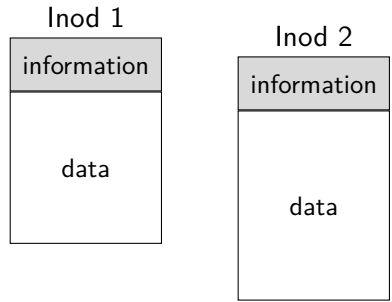
- FS arbetar med *logiska block* eller *kluster*
- Kan ha annan storlek än *fysiska block*
- Ett logiskt block brukar vara 2^n fysiska block
- Om vi lagrar block-index som 32-bitars tal, hur stor disk kan vi ha med en viss blockstorlek?

Vad är en inod?

Inod (UNIX) / MFT Record (NTFS)

- Sekvens av *bytes* (inte *block*)
- Godtycklig längd
- Attribut (ägare, rättigheter, ...)
- Identitet i form av ID (`ls -i`)

Tänk *fil utan namn* – vi använder tal som "namn" (ex.vis nummer på logiskt block där inoden börjar)



Hur lagrar vi inoder? – Kontinuerlig allokering

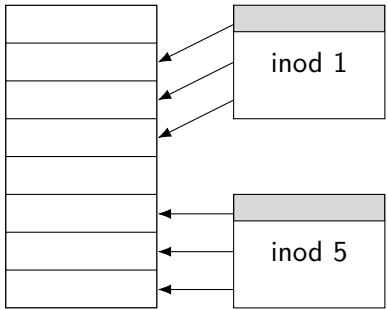
Idé: Lagra inoder sekventiellt

Fördelar:

- enkel att implementera
- sekventiell åtkomst

Nackdelar:

- fragmentering

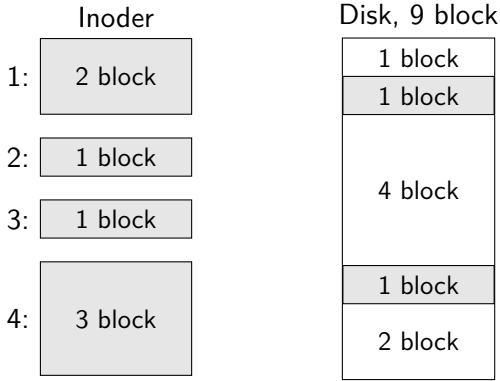


Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



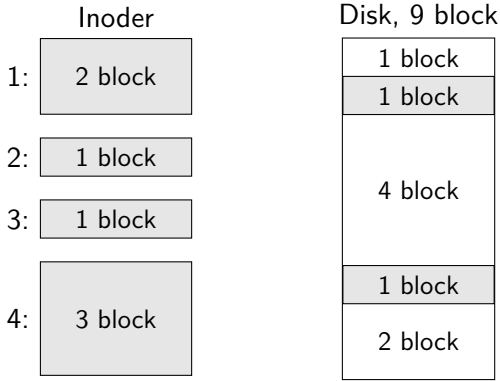
Allokering av block

Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



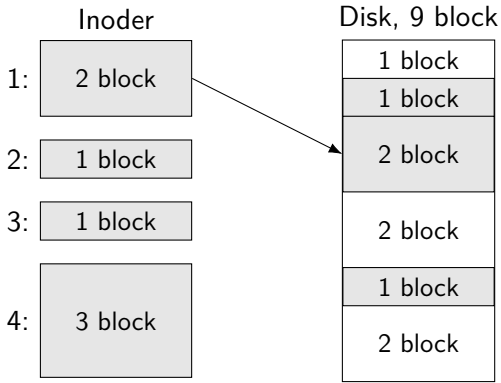
Allokering av block

Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



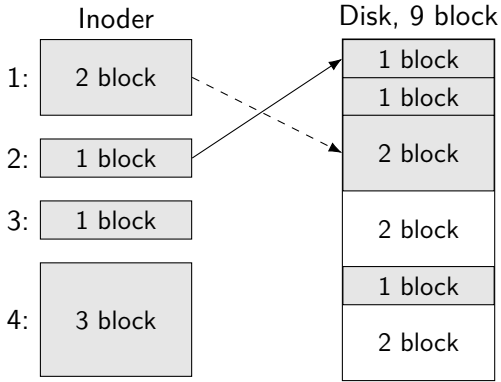
Allokering av block

Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.

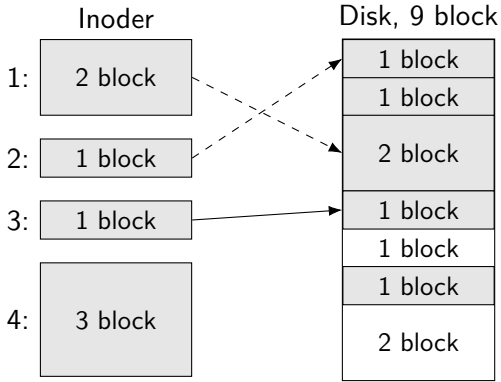


Allokering av block

Finns olika strategier:

- ⇒ *First-fit*
- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



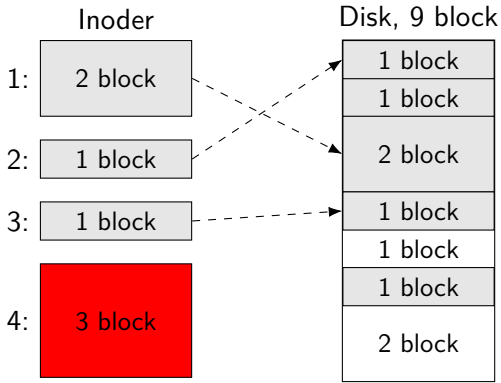
Allokering av block

Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

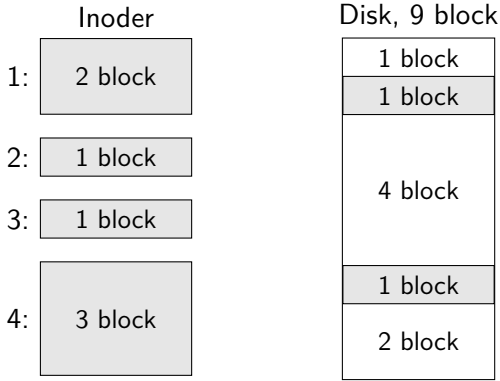
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

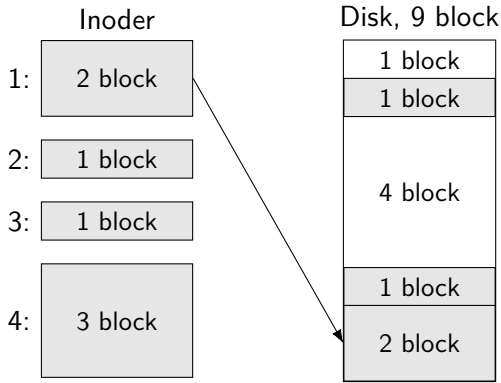
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

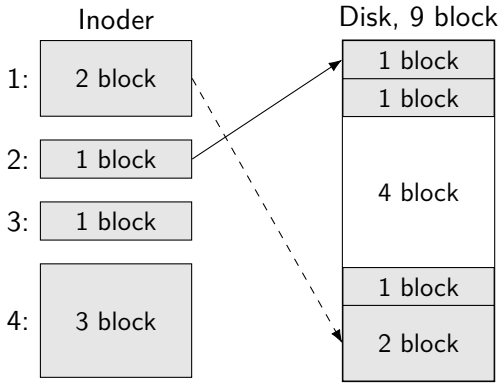
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

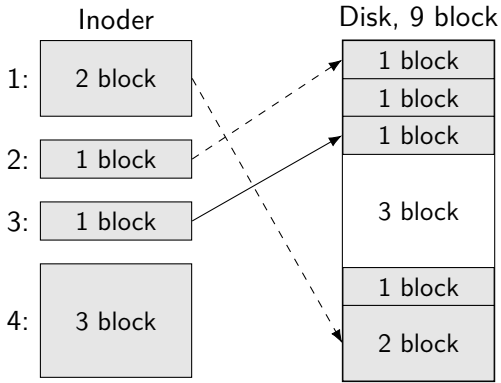
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

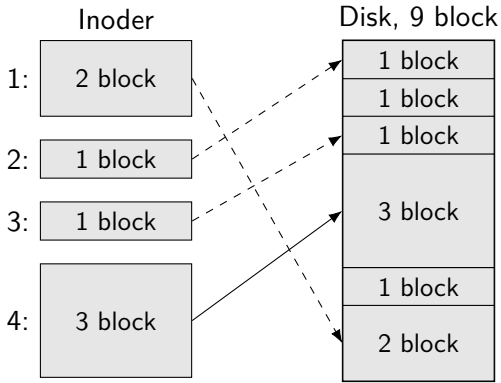
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

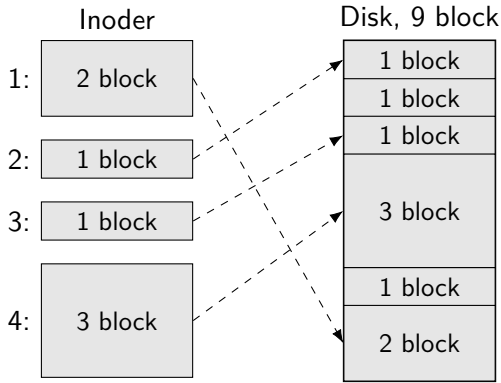
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



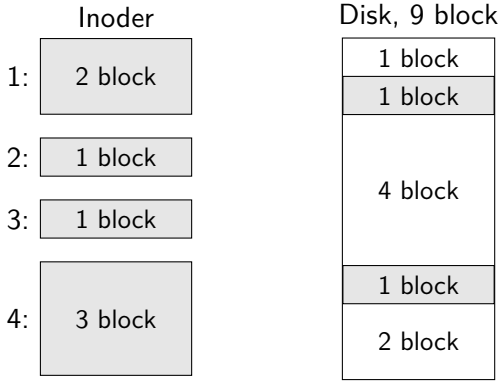
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



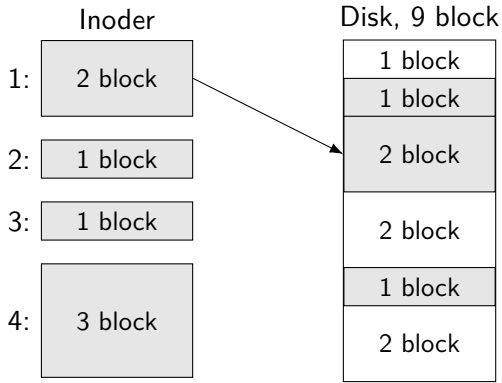
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



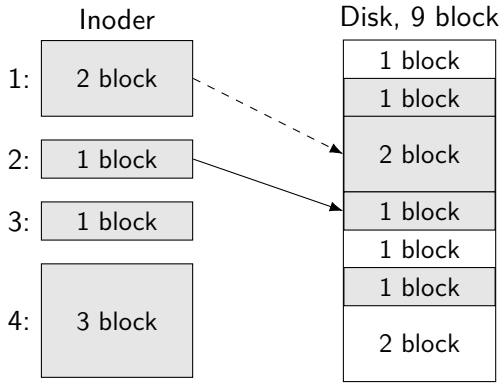
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



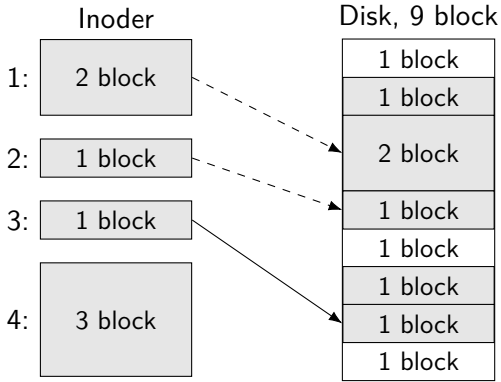
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



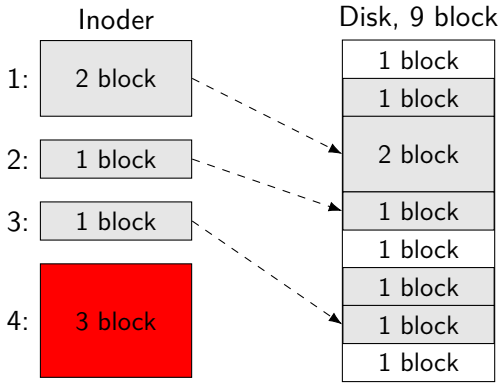
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Hur lagrar vi inoder? – Länkad allokering

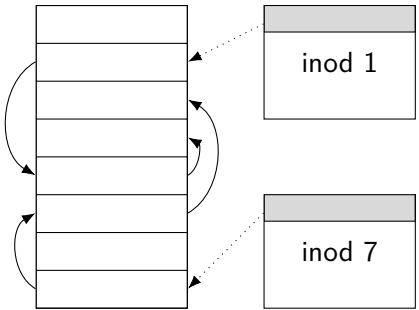
Idé: Undvik fragmentering genom att länka inoder

Fördelar:

- enkel att implementera
- ingen extern fragmentering

Nackdelar:

- dyrt att hoppa runt i filer



Hur lagrar vi inoder? – File Allocation Table (FAT)

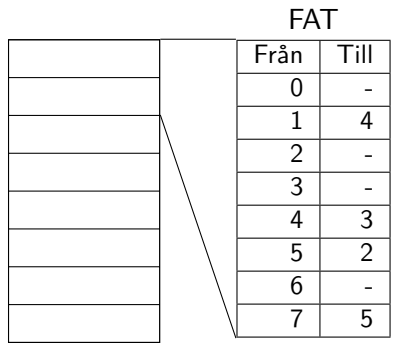
Idé: Flytta ihop länkarna!

Fördelar:

- ingen extern fragmentering
- enkelt att veta vad som är ledigt

Nackdelar:

- vi vill ha FAT i RAM
- om FAT skadas är



Hur lagrar vi inoder? – Indexerad allokering

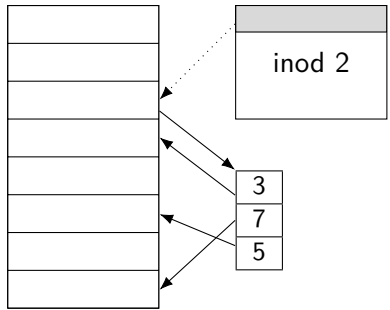
Idé: Använd ett block för att lagra "pekare" till andra block!

Fördelar:

- ingen extern fragmentering
- enkelt att söka

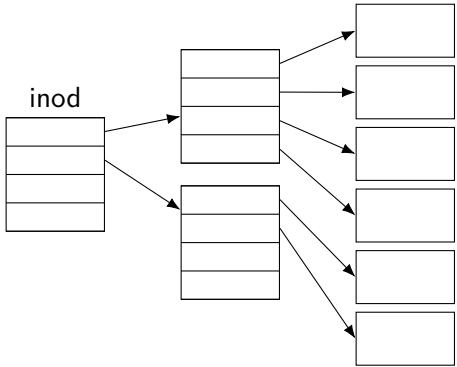
Nackdelar:

- begränsad filstorlek



Hur lagrar vi inoder? – Fler nivåer av indexering

Idé: Använd flera nivåer av indirekta block för större filer!



Fördelar:

- ingen fragmentering
- ok sökning
- hanterar stora filer

Nackdel:

- mycket overhead för små inoder

Hur lagrar vi inoder? – Varierande antal nivåer

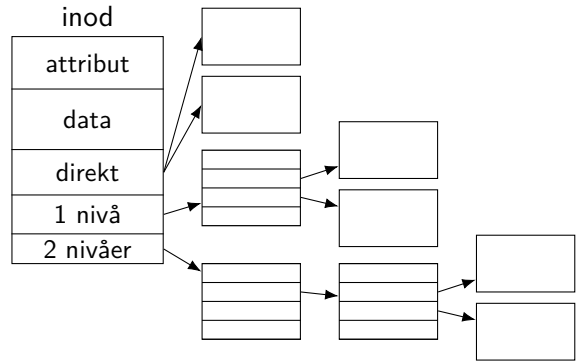
Idé: Använd flera nivåer av indirekta block!

Fördelar:

- ingen fragmentering
- ok sökning
- hanterar stora filer

Nackdel:

- komplex att implementera



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:

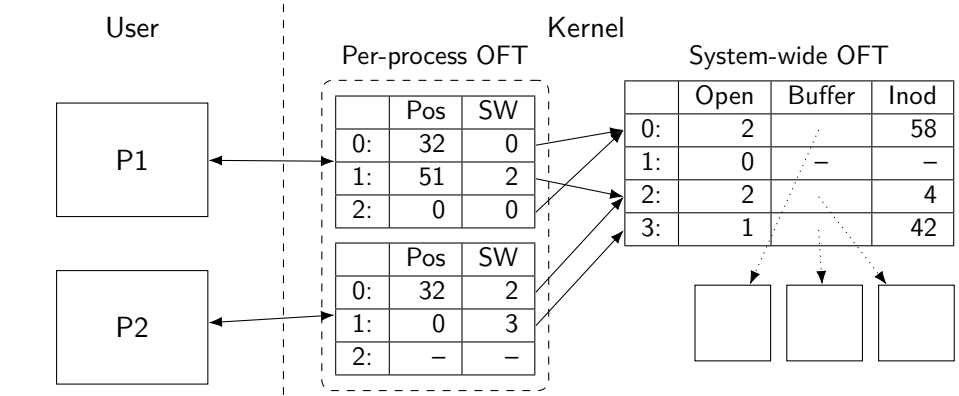
```
int inode_size(int id);

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?

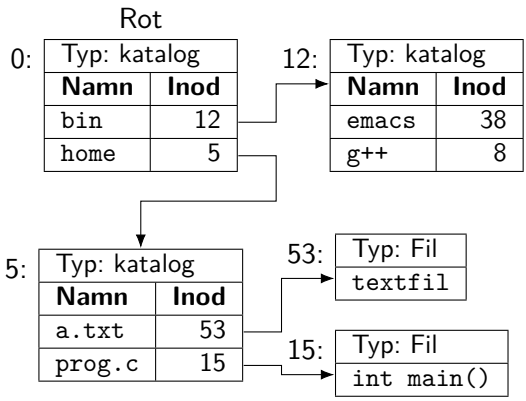
Idé: Lagra namn + id i inoder, vi kallar dem *katalog*-inoder

Tabell med:

- Namn
- Inod

Inod innehåller:

- Om katalog/fil
- Rättigheter



Länkar i systemet

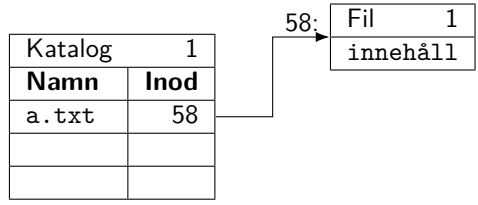
Idé: Tillåt flera referenser till samma fil

- Hårda länkar
- Symboliska länkar

I inod:

- Antal referenser

Varför inte hård länk till katalog?



Länkar i systemet

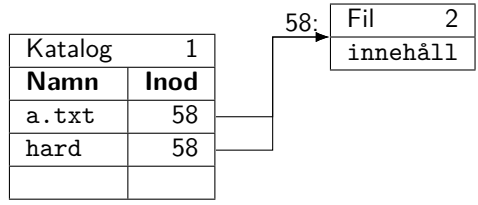
Idé: Tillåt flera referenser till samma fil

- Hårda länkar
- Symboliska länkar

I inod:

- Antal referenser

Varför inte hård länk till katalog?



Länkar i systemet

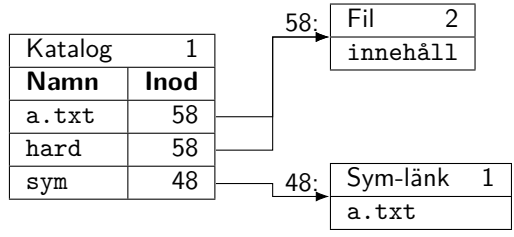
Idé: Tillåt flera referenser till samma fil

- 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!

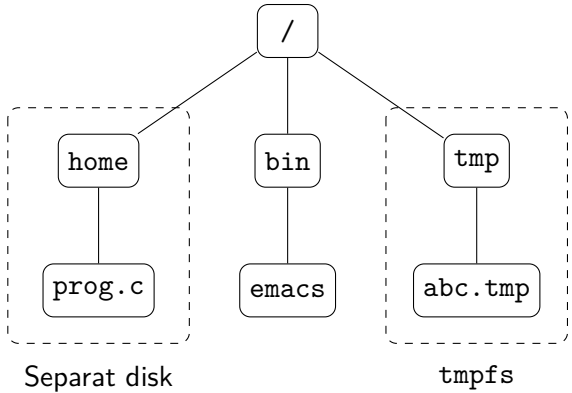
```
DIR *opendir(const char *name);
void closedir(DIR *dir);
dirent *readdir(DIR *dir);

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



Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY