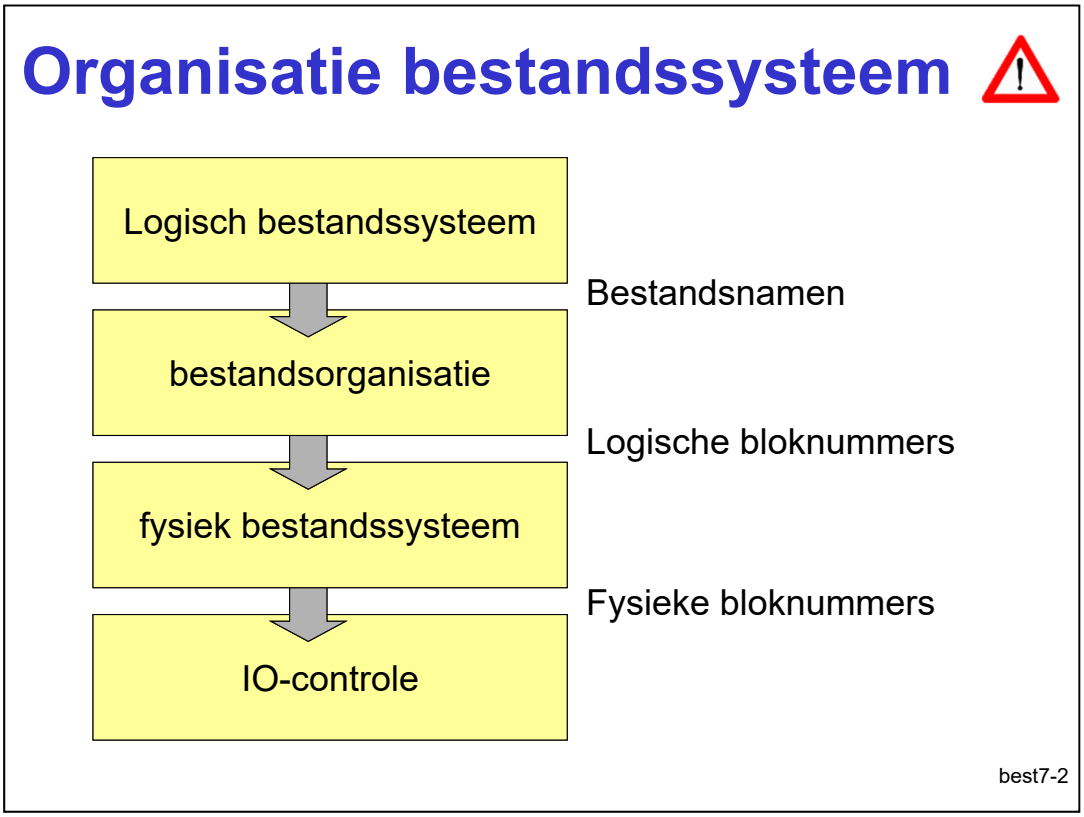
Les 7: Bestandssystemen

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." **Brian Kernighan**

best7-1

Naast het beheer van de processor en het geheugen is er nog een derde component in een computersysteem die door alle gebruikers gedeeld wordt, met name het secundair geheugen, doorgaans geïmplementeerd aan de hand van roterende media (magnetische of optisch schijven). De schijf behoort tot de traagste onderdelen van de geheugenhiërarchie van de computer. Dit is te wijten aan het feit dat het een mechanische component is. Anderzijds is de informatie die op de schijf opgeslagen ligt wel het meest waardevolle onderdeel van de computer. De prijs om de gegevens van een schijf te reconstrueren (indien mogelijk) overtreft vele malen de aanschafprijs van de schijf of zelfs van het gehele computersysteem.

Het feit dat de gebruikers niet geconfronteerd worden met de technische details van de roterende media is te danken aan het feit dat het secundaire geheugen geabstraheerd wordt tot een bestandssysteem bestaande uit gegevensbestanden om gegevens in op te slaan, directory's om al die gegevensbestanden te organiseren en schijfpartities om verzamelingen directory's logisch of fysiek van elkaar te scheiden.



Naast het gebruik van de naakte schijf zoals dit door sommige toepassingen gebeurt, zal het besturingssysteem de gebruikers de schijf aanbieden op een hoger abstractieniveau, met name als een gestructureerde verzameling van bestanden. Dit wordt gerealiseerd door middel van een bestandssysteem.

Een bestandssysteem bestaat uit verschillende lagen die elk van de diensten van de onderliggende lagen gebruik maken.

Op het bovenste niveau is er het **logisch bestandssysteem**. Dit is het niveau van de gegevensbestanden, directory's, speciale bestanden, enz. Het biedt ook de oproepen aan om bestanden te manipuleren.

Het logische bestandssysteem maakt gebruik van de **bestandsorganisatie** die de verbinding vormt tussen het logische en het fysieke niveau. Hier wordt aan het beheer van de vrije schijfruimte gedaan, hier worden de bestandsnamen omgezet naar de logische schijfadressen. Het zet als het ware de bestandsnaam om in een verzameling van logische geheugenblokken op de schijf (genummerd van 0 tot het laatste blok).

Het **fysiek bestandssysteem** krijgt van de bestandsorganisatie de logische schijfadressen binnen en vertaalt deze naar fysieke schijfadressen voor de betreffende schijven.

IO-controle zijn de drijvers voor de schijven. Deze drijvers verbergen alle details van de schijfhardware voor de bovenliggende lagen.

Voor de gebruiker zijn de kenmerken van een bestand totaal losgekoppeld van de fysieke kenmerken van het medium waarop het opgeslagen werd (harde

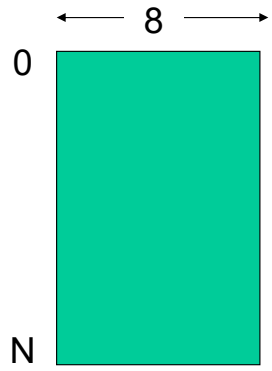
schijf, usb-sleutel, DVD, magneetband, netwerk, enz.). Het besturingssysteem zorgt voor een automatische vertaling naar de onderliggende implementatietechnologie.

Overzicht

- Logisch bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruimte
  - Partities
  - Reservekopieën
- Optimalisaties

best7-3

Gegevensbestanden



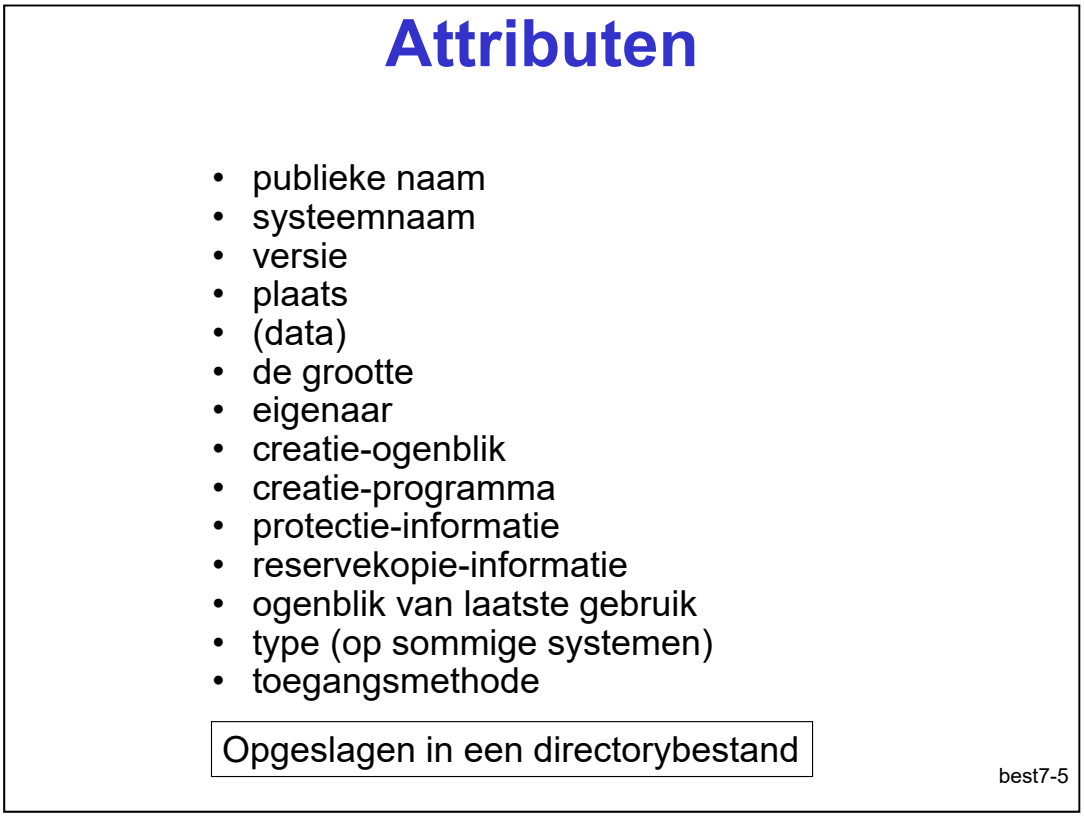
- Attributen
- Operaties
- Types
- Toegangsmethoden

Bestand
= benoemde verzameling bij elkaar horende gegevens
= eenheid van opslag.

best7-4

Een gegevensbestand bevat gegevens (namen van personen, de uitslag van de lotto, een Pascalprogramma, een uitvoerbaar programma). Deze gegevens worden voorgesteld door een opeenvolging van bytes. Elke byte in een gegevensbestand heeft een adres. De eerste byte heeft adres 0. Het bestandssysteem benoemt deze gegevens, en zorgt ervoor dat ze via de gegeven naam kunnen teruggevonden worden. Men kan een bestand beschouwen als de eenheid van opslag in het secundaire geheugen.

Doorgaans maakt men een onderscheid tussen twee types van bestanden: ascii en binaire bestanden, al naargelang wij een bestand rechtstreeks kunnen lezen, of er een bijkomende transformatie nodig is om de inhoud van het bestand leesbaar te maken. Dit onderscheid is echter een kwestie van interpretatie en in deze sectie zullen wij er dan ook van uitgaan dat een bestand een opeenvolging is van bytes en niets meer. De interpretatie wordt overgelaten aan de programma's die de bestanden aanmaken of gebruiken. Op het niveau van het besturingssysteem speelt de inhoud van een bestand dan ook geen enkele rol – behalve voor die bestanden die door het besturingssysteem zelf onderhouden worden uiteraard.



Per bestand houdt het besturingssysteem een aantal zogenaamde attributen bij. Daartoe zullen zeker behoren:

Publieke naam al dan niet beperkt in lengte of in de tekens die in de naam kunnen voorkomen. Soms maakt men in een bestandsnaam een onderscheid tussen de eigenlijke naam en de extensie.

Systeemnaam intern in het bestandssysteem zal er naar het bestand gerefereerd worden met interne wijzers.

Versie sommige besturingssystemen houden opeenvolgende versies van bestanden bij. In VMS worden er heuse versies gebruikt (b.v. t.out;2). Oude versies kunnen automatisch tot een bepaald aantal beperkt worden of kunnen manueel verwijderd worden.

Plaats dit is een verwijzing naar de schijf en de plaats op de schijf waar het bestand opgeslagen ligt.

Data sommige systemen beschouwen de eigenlijke data ook als attribuut, andere gebruiken het plaatsattribuut om naar de data te verwijzen.

Grootte uitgedrukt in bytes, woorden, blokken, afhankelijk van de organisatie van het bestandssysteem.

Eigenaar doorgaans diegene die het bestand aangemaakt heeft.

Creatie-ogenblik, creatie-programma, ogenblik van laatste gebruik (lezen,schrijven)

Protectie-informatie wie heeft welke rechten op dit bestand (lezen, schrijven, uitvoeren); eventueel een wachtwoord

Reservekopie-informatie werd dit bestand sinds de laatste reservekopie (back-up) gewijzigd?

Type ascii, binair. Vaak nemen bestandsnaamextensies de rol van types over.

De totale hoeveelheid informatie kan variëren in lengte van een paar tientallen tot een paar honderden bytes. Deze informatie moet permanent bijgehouden worden en zal daarom ook op de schijf ondergebracht worden (in de directory, zie verder). De directory's kunnen een aanzienlijk deel van de schijf innemen.

Naamgeving

- String van letters,cijfers en leestekens,
  - Beperkt in lengte
  - Soms onderscheid kleine/hoofdletters (Unix, NTFS), soms niet (Windows, FAT)
  - NTFS gebruikt unicode: βαДи‌ה탁탫敿
- Meestal ook bestandsextentie
  - 8.3 notatie: abcdefgh.ijk
  - Via extensies worden applicaties geassocieerd

best7-6

Tegenwoordig is er zeer veel vrijheid in de naamgeving van bestanden. Dit is ooit anders geweest toen de 8.3 notatie de wereld beheerste. Indien men universele uitwisseling van bestandsnamen nastreeft is de 8.3 notatie nog steeds de beste garantie. Een aantal bestandssystemen hebben het moeilijk met vreemde tekens en spaties in bestandsnamen.

Operaties op bestanden

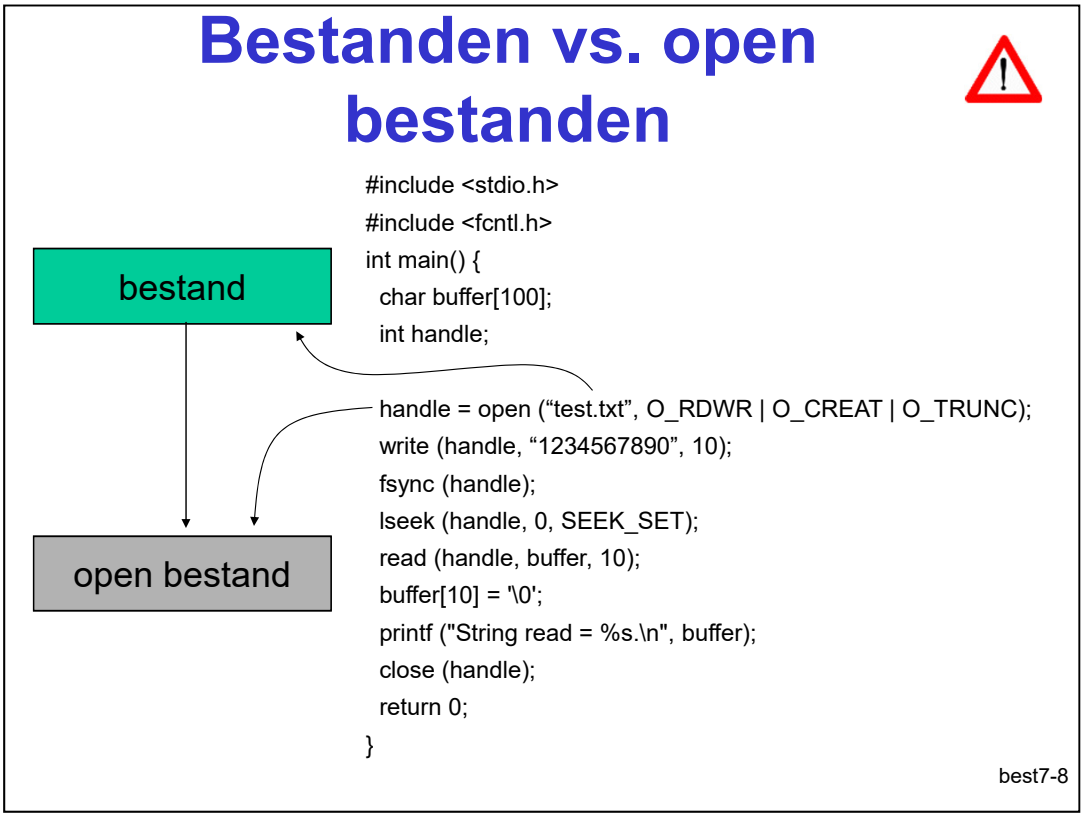
Win32	Unix	Omschrijving
CreateFile	creat	Creëer een bestand
OpenFile	open	Open een bestand
CreateProcess	exec	Start een proces

best7-7

Er zijn eigenlijk maar 2 basisoperaties die op gegevensbestanden werken:

Create of open: het creëren of openen van een gegevensbestand om te lezen of te schrijven. De oproepen geven een open bestand terug.

Createprocess, het creëren van een proces, uitgaande van een laadmodule. Dit is natuurlijk enkel van toepassing voor gegevensbestanden die uitvoerbaar zijn. Createprocess geeft een procesidentificatie terug.



Het is belangrijk een onderscheid te maken tussen het concept van een bestand voorgesteld door een bestandsnaam, en het concept van een open bestand (open file) voorgesteld door een handle of bestandscontroleblok (file control block).

Een bestand verwijst doorgaans naar gegevens die opgeslagen liggen op een opslagmedium, terwijl een open bestand kan refereren naar een veel grotere variëteit aan gegevens. Zo kan men b.v. een open bestand creëren om gegevens naar een randapparaat te sturen, of gegevens uit de kern te lezen. In geen van beide gevallen is het open bestand geassocieerd met gegevens die op een schijf opgeslagen liggen.

En open bestand wordt voorgesteld door een handle (dit is gewoon een index in een kerntabel met open bestanden). Elke programmeertaal heeft haar eigen manier om op het programmaniveau open bestanden te beheren. Doorgaans gebeurt dit aan de hand van een speciaal datatype dat naast de handle ook nog extra informatie bijhoudt en bijkomende functionaliteit kan aanbieden (b.v. buffering).

Bij Unix zijn de handles gewone nummertjes: 0 stelt stdin voor (console-input), 1 stelt stdout voor (console-output), 2 stelt stderr voor (foutboodschappen). De bijkomende bestanden die geopend worden zullen handles 3, 4, 5, … krijgen. Het is belangrijk dat de open bestanden per proces bijgehouden worden. Bij het onderbreken van een programma moet de kern ervoor zorgen dat alle systeemmiddelen opnieuw vrijgegeven worden, inclusief de open bestanden. Dat kan enkel indien de kern in het PCB bijhoudt welke bestanden er in gebruik zijn.

Operaties op open gegevensbestanden

Win32	Unix	Omschrijving
CloseHandle	close	Sluit een bestand
ReadFile	read	Lees data uit bestand
WriteFile	write	Schrijf data naar bestand
SetFilePointer	lseek	Stel de lees/schrijfwijzer in
SetEndOfFile	truncate	Stel de lengte van het bestand in

best7-9

Deze dia bevat een lijst van de voornaamste operaties op open bestanden in Win32 en in Unix.

close het sluiten van een open bestand.

read het lezen van een aantal bytes vanaf een gegeven plaats (aangewezen door de leeswijzer) in het open bestand.

write het schrijven van een gegeven aantal bytes op een gegeven plaats (aangewezen door de schrijfwijzer) in het open bestand.

seek het lokaliseren van een bepaalde plaats in een open bestand. De lees/schrijfwijzer krijgt hierbij een nieuwe waarde. Dit is een pure softwareaangelegenheid. Deze oproep vereist geen interactie met het secundaire geheugen zelf.

truncate het afkappen van een open bestand op een gegeven lengte.

Verder worden er ook vaak nog andere operaties aangeboden die implementeerbaar zijn aan de hand van de net voorgestelde primitieve operaties: het toevoegen van informatie aan een bestand (append), het kopiëren van een bestand (copy), enz.

Voorbeeldprogramma Unix

```
in_fd = open(argv[1], O_RDONLY);    /* open the source file */
if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
if (out_fd < 0) exit(3);            /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
      rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;           /* if end of file or error, exit loop */
      wt_count = write(out_fd, buffer, rd_count); /* write data */
      if (wt_count <= 0) exit(4);   /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                  /* no error on last read */
      exit(0);
else
      exit(5);                      /* error on last read */
```

Details: man 2 <system call>

best7-10

Dit is een voorbeeld van een programma dat een bestand kopieert.

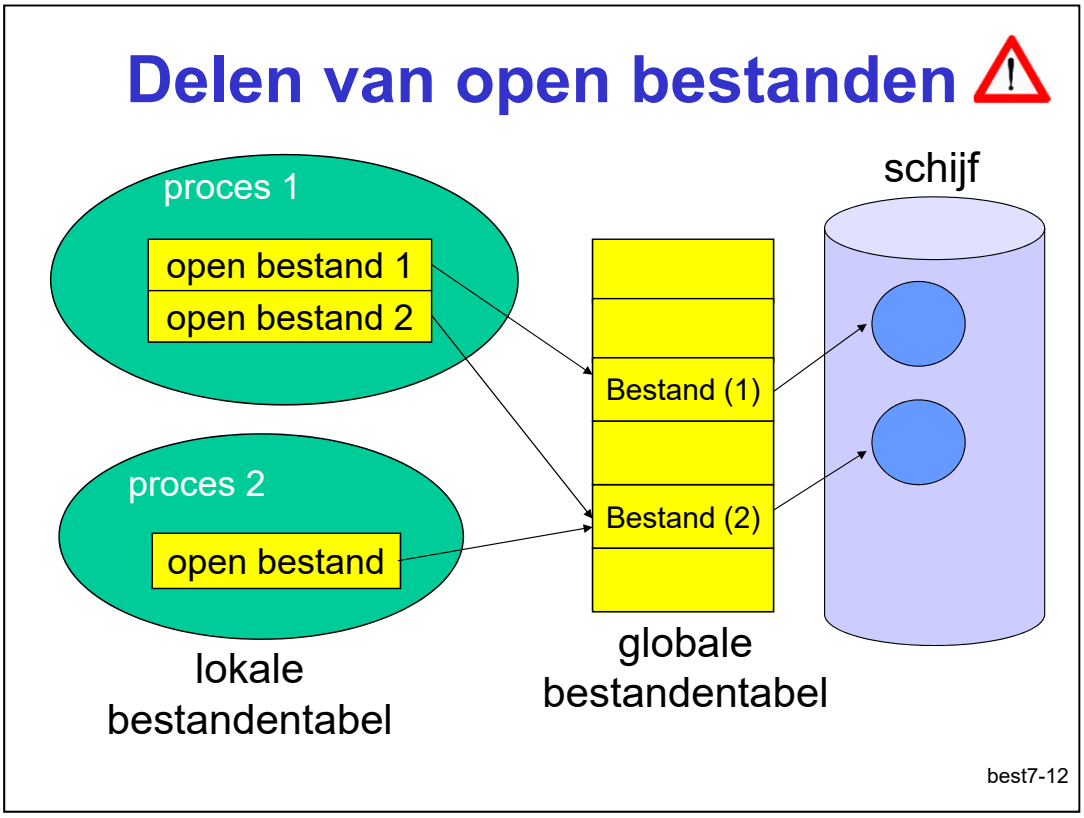
Voorbeeldprogramma Windows

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
    s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
    if (s && count > 0) WriteFile(outhandle, buffer, count, &ocnt, NULL);
} while (s > 0 && count > 0);

/* Close the files. */
CloseHandle(inhandle);
CloseHandle(outhandle);
```

```
if (!copyfile(argv[1], argv[2], false)) exit(3);
```

best7-11

Dit Win32 programma kopieert een bestand. Merk op dat er een Win32 oproep bestaat om een bestand in één keer te kopiëren.



In systemen met verschillende gebruikers kan het voorvallen dat twee gebruikers hetzelfde open bestand willen gebruiken. Om dit mogelijk te maken zal er een bijkomend niveau nodig zijn tussen de bestandentabel per proces en de eigenlijke bestanden: de globale bestandentabel.

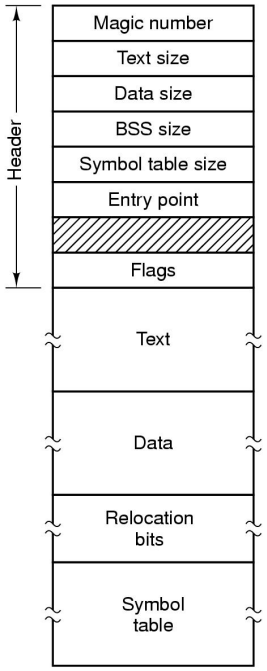
De globale tabel houdt op het niveau van het systeem bij welke bestanden er open zijn, door hoeveel processen ze gebruikt worden, waar ze opgeslagen liggen op de harde schijf enz. De lokale tabellen bevatten een verwijzing naar de globale tabel, de huidige bestandswijzer voor dat proces, en eventuele andere informatie die lokaal moet bijgehouden worden (zoals de rechten waarmee het bestand geopend werd: lezen, schrijven, enz.). Bij het beëindigen van een proces zullen alle bestanden die nog open staan automatisch gesloten worden aan de hand van de informatie in deze tabel.

In de globale tabel wordt een tellertje bijgehouden van het aantal keer dit bestand open staat. Als het tellertje op 0 komt, wordt het bestand verwijderd uit de globale tabel.

Het kernobject dat een bestand beheert tijdens zijn gebruik wordt een bestandscontroleblok genoemd (FCB: File Control Block).

Bestandstypes

- directory's
- Speciale bestanden (Unix, zie I/O)
- Links, shortcuts
- Gegevensbestanden
  - ASCII: leesbaar
  - Binair: betekenis enkel zinvol voor programma dat indeling kent
  - Uitvoerbaar
  - Magic numbers om type te bepalen
    - Speciaal geval: uitvoerbaar bestand, OS kent indeling



Het toekennen van een type aan een bestand kan op verschillende manieren gebeuren: door een type-attribuut, door de naamgeving van het bestand, of door beide.

Elk besturingssysteem kent een aantal speciale bestandstypes: directory's, speciale bestanden, links. Deze worden herkend aan de hand van een bestandsattribuut. Daarnaast is er ook nog de grote groep van gegevensbestanden die tegenwoordig meestal beschouwd worden als een opeenvolging van bytes. Als gebruiker heeft men wel de indruk dat ze ook een type hebben op basis van de extensie van de bestanden. Het gebruik van extensies is bijzonder handig. We beseffen pas hoe onmisbaar ze zijn van zodra iemand op een niet-conventionele manier gebruik maakt van de extensies. Het gebruik van extensies om een informeel type aan te geven is echter vrijblijvend en de meeste programma's laten ook toe bestanden te gebruiken die niet de vereiste extensies hebben. Extensies worden gebruikt om gegevensbestanden met applicaties te koppelen. Technisch gezien is dit niet de beste oplossing. Het zou beter zijn om een applicatie via een afzonderlijk attribuut aan een gegevensbestand te koppelen.

De bestandstypes voor gegevensbestanden kunnen echter een wezenlijk onderdeel van het bestandssysteem zijn. VMS kent diverse expliciete bestandstypes: binair, ascii met vaste en veranderlijke recordlengte, enz. Sommige van de gespecialiseerde besturingssystemen van IBM hebben ook een aantal voorgedefinieerde bestandstypes. Het voornaamste voordeel van voorgedefinieerde bestandstypes is dat het besturingssysteem geoptimaliseerde bestandsmanipulatieroutines per bestandstype kan gebruiken. Een dergelijke organisatie heeft echter ook nadelen zoals de noodzaak tot conversies tussen bestandstypes, de grotere complexiteit van het besturingssysteem. Hedendaagse besturingssystemen laten de gespecialiseerde operaties over aan de applicaties: indexering is b.v. een taak voor een databank.

Toegangsmethoden

- Sequentiële toegang
- Directe toegang
- Geïndexeerde toegang

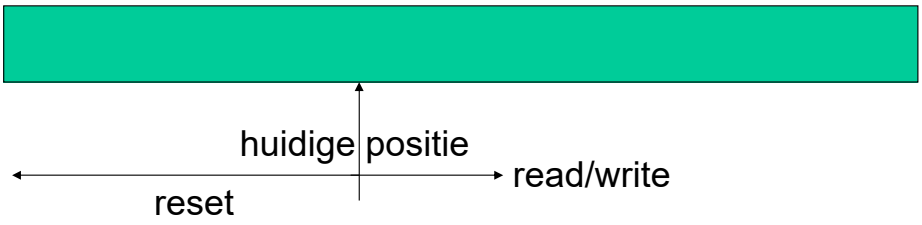
Tegenwoordig:

Directe toegang op byte-niveau

best7-14

Bestanden kunnen op drie verschillende manier gebruikt worden: sequentieel, direct of geïndexeerd. Deze drie toegangsmethoden worden nu kort toegelicht.

Sequentiële toegang



3 operaties:
- reset
- read next
- write next

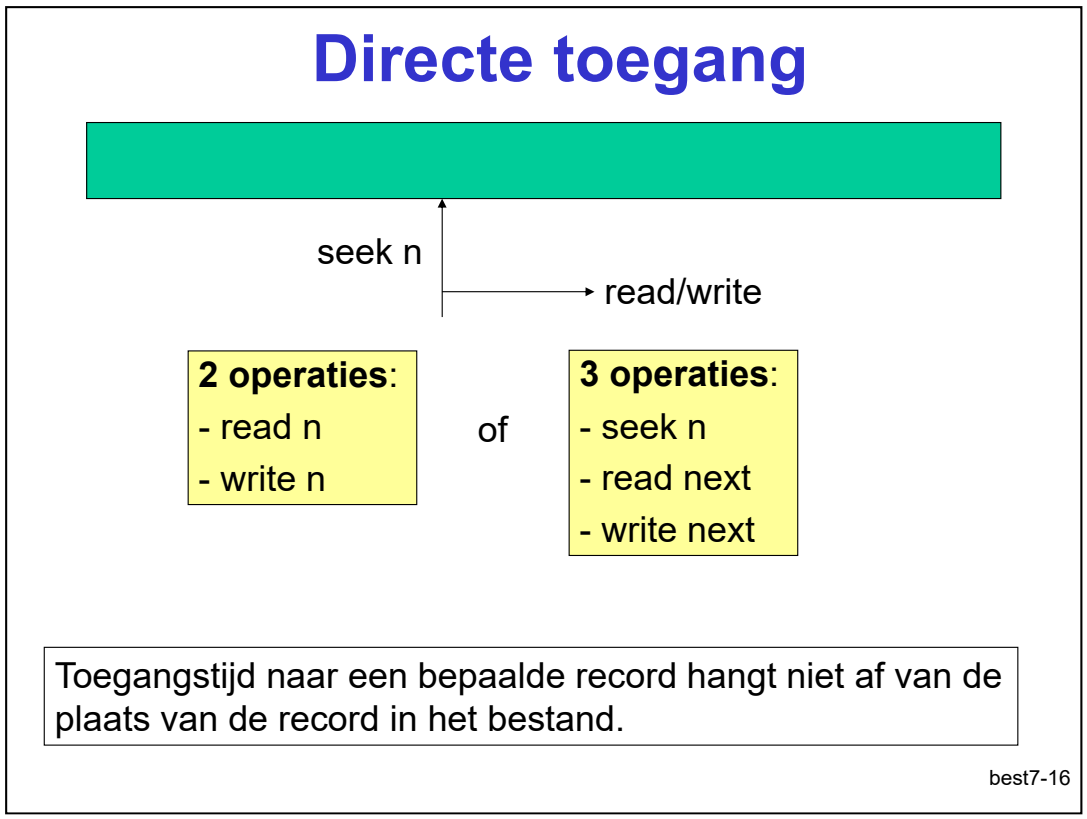
Toegangstijd naar een bepaalde record hangt af van de plaats van de record in het bestand.

best7-15

Dit is de eenvoudigste manier die ook toepasbaar is op magneetbanden. Opeenvolgende lees- of schrijfbewerkingen worden uitgevoerd op opeenvolgende records van het bestand. Editors, compilers en alle stream-based programma's hebben genoeg aan sequentiële toegang. Bij deze toegangsmethode mogen opeenvolgende records een verschillende lengte bezitten (van belang voor tekstbestanden). Programma's zullen steeds van vooraf aan beginnen lezen of schrijven.

Er zijn maar 3 bestandsoperaties:

- Reset: ga naar het begin van het bestand
- Read next: lees de volgende record
- Write next: schrijf de volgende record



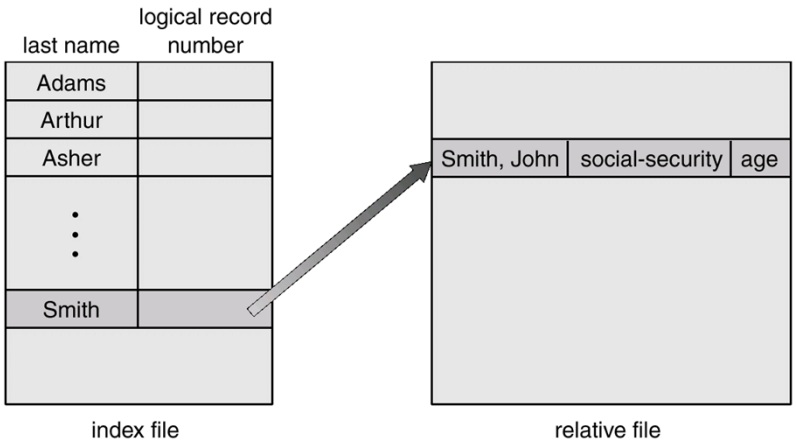
Directe toegang is enkel mogelijk indien men werkt met records van vaste lengte (die eventueel maar voor een deel effectief gebruikt worden). Gegeven het nummer van een record, kan het beginadres van de record berekend worden. Bij directe toegang zal men bij het lezen of schrijven steeds het nummer van de record moeten opgeven.

Sommige (oudere) systemen eisen dat een bestand van bij zijn creatie als 'sequentieel' of 'direct' gedeclareerd wordt. Bestanden die initieel als sequentieel gecreëerd werden, kunnen naderhand niet meer direct gebruikt worden. Het omgekeerde is minder erg: directe bestanden kunnen wel sequentieel geraadpleegd worden door het recordnummer zelf bij te houden. Bij andere systemen kan men bij het openen van een bestand kiezen op welke manier men het zal gebruiken: sequentieel of direct. Een bestand dat sequentieel gecreëerd werd kan naderhand direct gebruikt worden (in de mate waarin dit zin heeft natuurlijk).

Er zijn 2 operaties bij directe toegang: **read n**: lees record n en **write n**: schrijf record n

In de praktijk maakt men vaak gebruik van een positioneeroperatie, gevolgd door de operaties van sequentiële toegang. **seek n, read next**, en **write next**

Geïndexeerde toegang



best7-17

Vaak is het recordnummer niet het gegeven waarop we een bepaalde record willen selecteren. In een databank met adressen zullen we veeleer een record willen selecteren op basis van b.v. de naam van een individu. Geïndexeerde bestanden laten ons toe om rechtstreeks die record te selecteren die correspondeert met de naam van een gegeven individu. Om dit doel te bereiken wordt er een index voor het bestand bijgehouden. De technieken om indices aan te maken komen elders aan bod en worden hier niet besproken. Tegenwoordig vindt men dat de creatie en het bijhouden van indices eigenlijk geen taak is voor het besturingssysteem, maar dat dit veeleer een taak is voor de toepassingsprogramma's die er nood aan hebben.

Directe toegang in Java

```
import java.io.File;
import java.io.RandomAccessFile;
import java.io.IOException;

public class DemoRandomAccessFile {

    public static void main(String[] args) {

        try {
            RandomAccessFile raf = new RandomAccessFile(“test.txt”, "rw");

            byte ch = raf.readByte();
            System.out.println("Read first character of file: " + (char)ch);
            System.out.println("Read full line: " + raf.readLine());

            raf.seek(raf.getFilePointer() - 10);
            raf.writeBytes(“???”);
            raf.seek(0);
            System.out.println("Read full line: " + raf.readLine());
            raf.seek(file.length());
            raf.writeBytes("This will complete the Demo");
            raf.close();

        } catch (IOException e) {
            System.out.println("IOException:");
            e.printStackTrace();
        }
    }
}
```

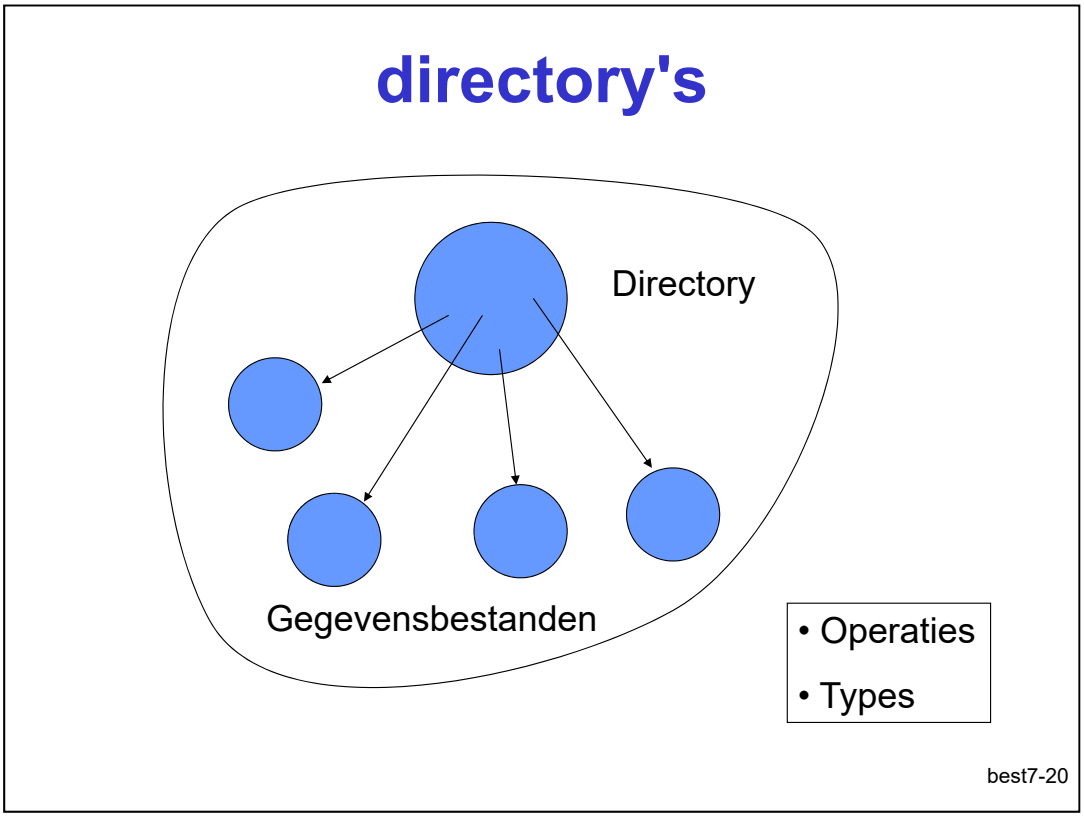
best7-18

Het voorbeeld spreekt voor zichzelf.

Overzicht

- Logische bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruimte
  - Partities
  - reservekopieën
- Optimalisaties

best7-19



Naast de gegevens die opgeslagen worden in de bestanden, zijn er ook nog de attributen van de bestanden. Deze worden niet opgeslagen in de bestanden zelf, maar in zogenaamde directory's. Een directory is een speciaal bestand dat voor dit doel gebruikt wordt. Afhankelijk van het type van besturingssysteem zal men de directorybestanden wel of niet kunnen manipuleren met een editeerprogramma. Dit is echter niet aan te bevelen gezien de schade die men hierdoor kan aanrichten. Een corrupte directory kan immers zeer veel bestanden verloren laten gaan.

directory's moeten aan een aantal voorwaarden voldoen. Ze moeten efficiënt implementeerbaar zijn zodat bestanden snel kunnen teruggevonden worden, ze moeten een vrije en flexibele naamkeuze van de bestanden mogelijk maken, en tenslotte moeten ze het mogelijk maken dat bestanden gegroepeerd worden – in afzonderlijke directory's, of op basis van de bestandsnaam.

Operaties op directory's

Win32	Unix	Omschrijving
CreateDirectory	mkdir	Maak een nieuwe directory
RemoveDirectory	rmdir	Laat een directory weg
FindFirstFile	opendir	Open een directory
FindNextFile	readdir	Lees een directory-element
FindClose	closedir	Sluit een directory
MoveFile	rename	Hernoem een bestand
SetCurrentDirectory	chdir	Verander van huidige directory
DeleteFile	unlink	Laat een bestand weg
GetFileAttributes	fstat	Lees een bestandsattribuut

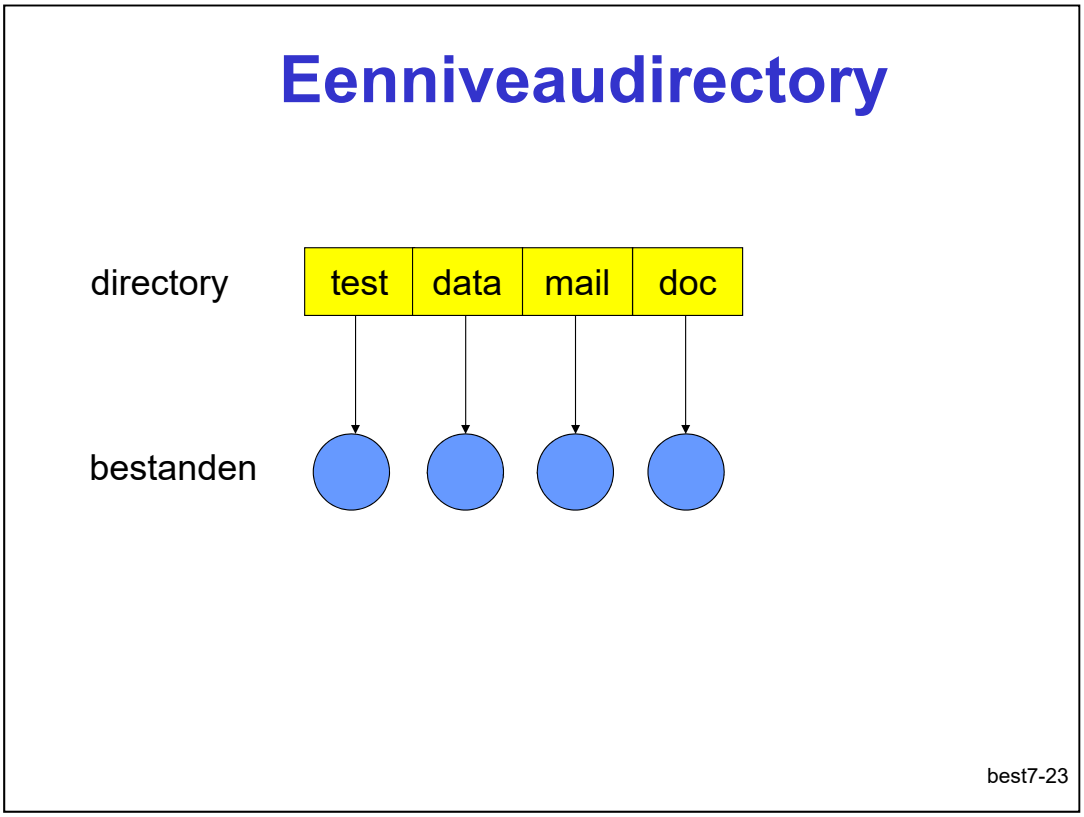
best7-21

Een directory zal voornamelijk gebruikt worden om, gegeven de naam van een bestand (**gegevensbestand of directory**), op te zoeken waar de gegevens van het bestand zich bevinden op de schijf. Een aantal basisoperaties in Win32 en Unix die ondersteund worden, worden hierboven opgesomd.

Directorytypes

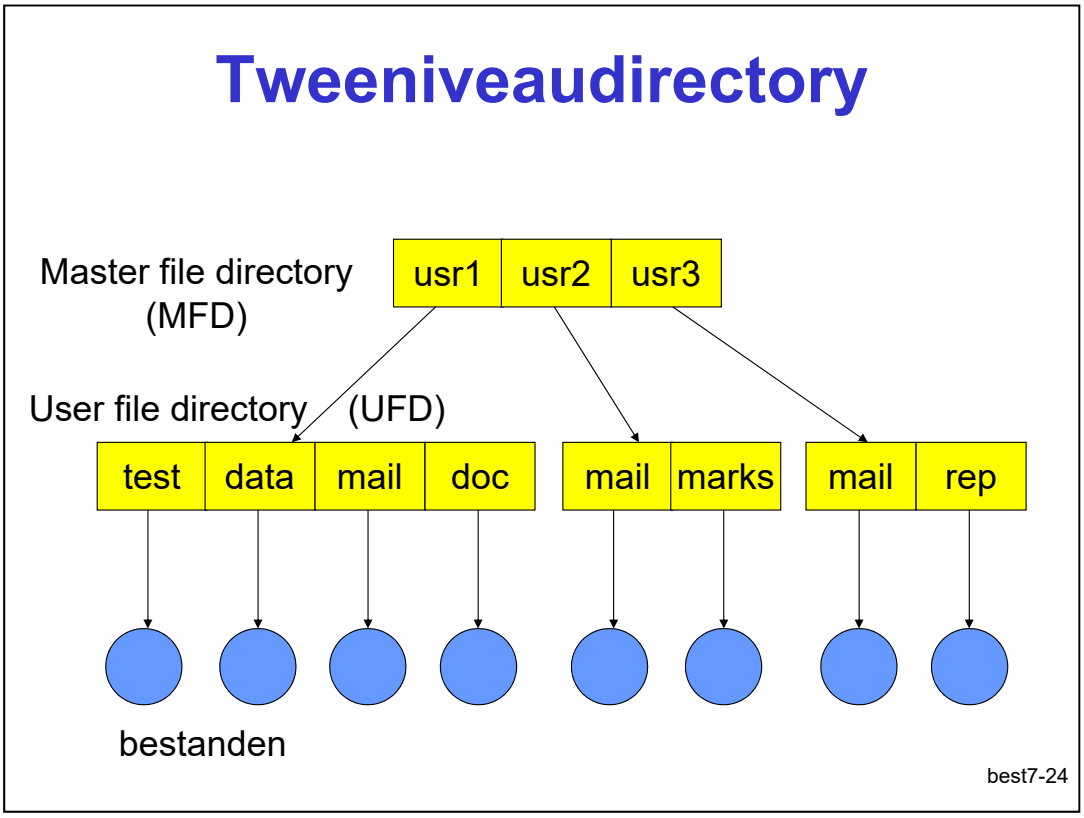
- 1 niveau
- 2 niveaus
- Boomstructuur
- Acyclische graaf
- Graaf

best7-22



Bij dit directorytype bevinden alle bestanden zich in de enige directory van het systeem. Dit creëert potentiële naamconflicten tussen bestanden. Een dergelijk type is niet bruikbaar door verschillende gebruikers.

Uiteraard hebben gebruikers altijd wel de vrijheid om zich b.v. op een usb-sleutel vrijwillig te beperken tot 1 enkel niveau.



Een voor de hand liggende uitbreiding van een directorytype van 1 niveau is een directorytype van 2 niveaus. Het eerste niveau creëert een onderverdeling per gebruiker. Elke gebruiker heeft een directory van 1 niveau voor zijn of haar bestanden. De directory met de gebruikers wordt de **master file directory** genoemd, de directory's per gebruiker worden de **user file directory's** genoemd.

Dit is prima indien gebruikers niet hoeven samen te werken. Als ze wel willen samenwerken moet er een methode gevonden worden om elkaars bestanden te kunnen benoemen. Dit is trouwens ook nodig om ervoor te zorgen dat men niet alle systeemprogramma's in de directory's van alle gebruikers moet kopiëren.

De oplossing bestaat erin om gebruik te maken van een zgn. padnaam: /usr1/doc. Op die manier kan de ene gebruiker bestanden benoemen van een andere gebruiker. Voor de systeemprogramma's maakt men doorgaans gebruik van nog een andere techniek, nl. het zoekpad. Het zoekpad bevat een lijst van directory's die doorzocht moeten worden op zoek naar een bepaald programma.

Een alternatieve methode om bestanden te delen is gebruik te maken van logische of symbolische namen. Dit is de methode die gebruikt werd in VMS. De vertaling van de logische en symbolische namen gebeurde door de besturingssysteem en kon verschillende stappen vereisten om de finale bestandsnaam te bereiken.

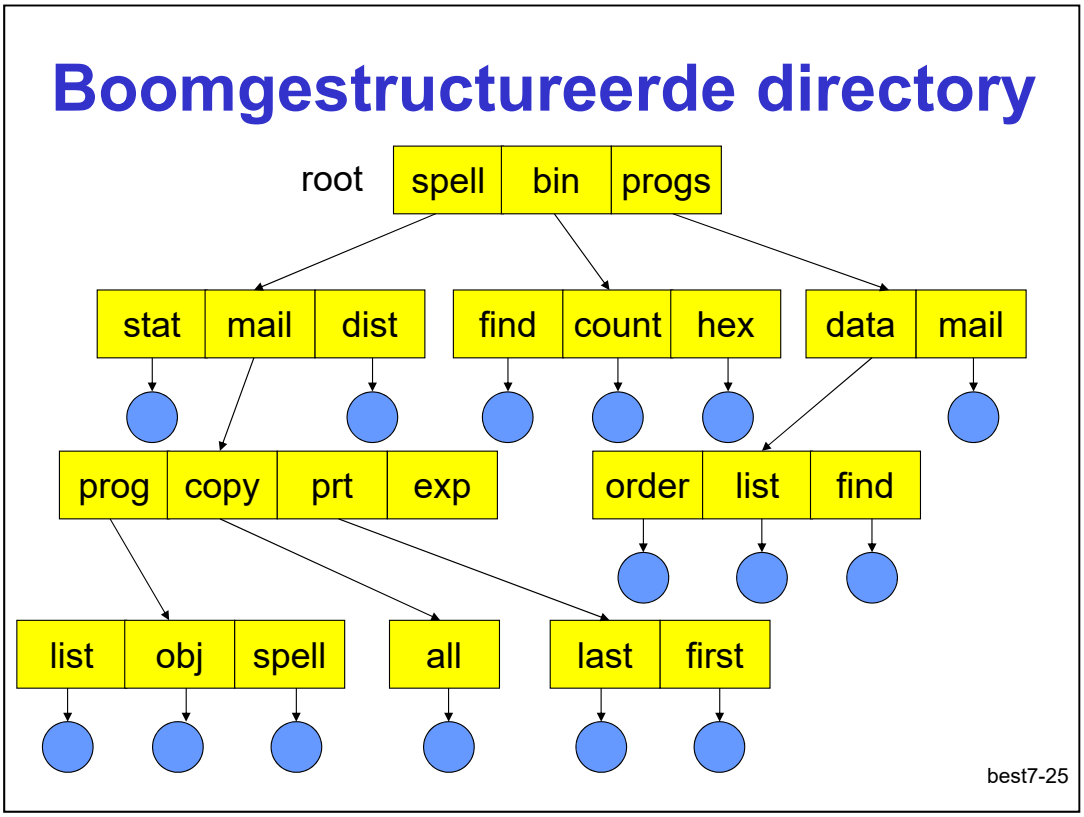
b.v. Applicaties = "c:\Program Files"

Suite = "gnu C compiler"

Compiler = "gcc"

cc = "Applicaties\Suite\Compiler"

cc test.c wordt dan vertaald in c:\Program Files\gnu C compiler\gcc test.c

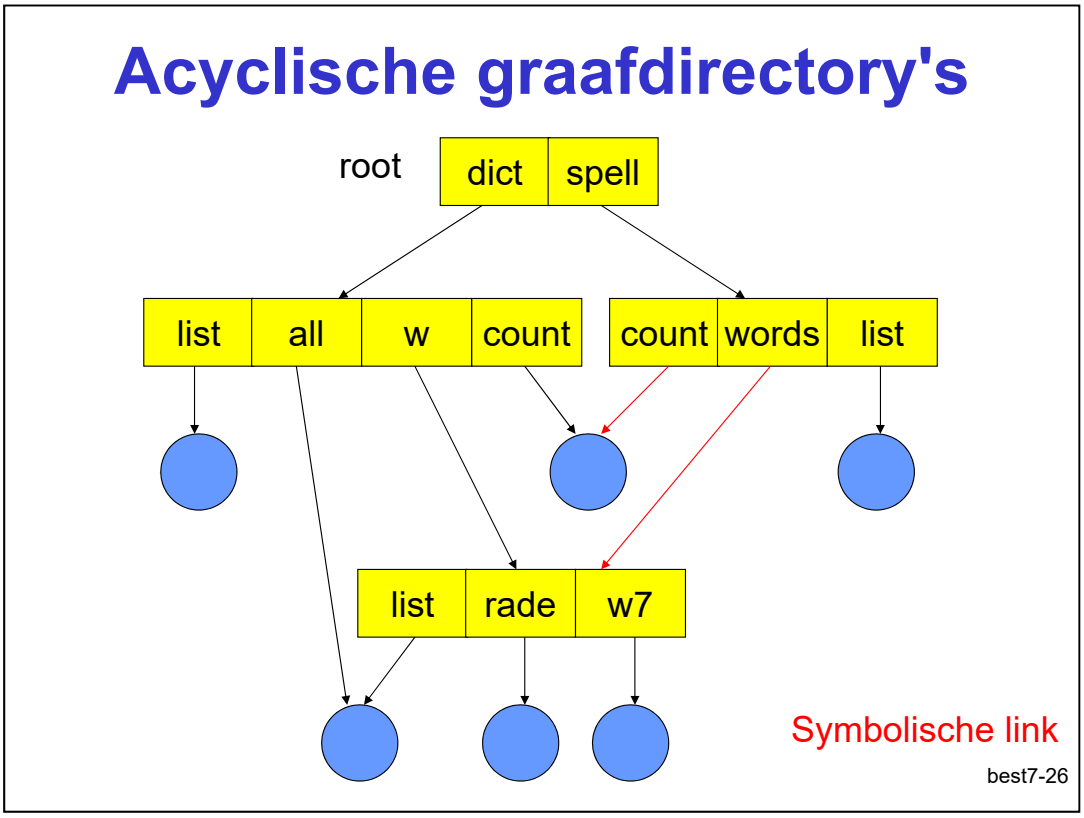


Een verdere uitbreiding van het directorytype van 2 niveaus is het boomgestructureerde directorytype. Hierbij kan een directory niet alleen gegevensbestanden, maar ook subdirectory's bevatten. De padnaam wordt in dit geval uitgebreid tot alle niveaus van de boom en vormt dan een eenduidige specificatie van een bestand. Doordat alle gebruikers opgenomen zijn in dezelfde boomstructuur is het ook mogelijk om naar bestanden van andere gebruikers te refereren door hun padnaam op te geven.

De verschillende onderdelen van de padnaam worden gescheiden door een speciaal scheidingsteken. Voor Windows is dit ‘\’, voor Unix is dat ‘/’, en voor Multics was dit ‘>’.

Aangezien een gebruiker nu over verschillende directory's beschikt voert men het concept van de ‘huidige directory’ in. Als men bestanden opvraagt zonder verder specificatie (met ‘dir’ of ‘ls’) dan krijgt men de bestanden van de huidige directory te zien. Met het commando ‘cd’ kan men de huidige directory veranderen.

Padnamen kunnen zowel absoluut (ten opzichte van de root-directory, b.v. /spell/mail/exp), als relatief (ten opzichte van de huidige directory b.v. copy/all in de directory mail) zijn. Absolute padnamen hebben als voordeel dat ze in alle omstandigheden verwijzen naar dezelfde directory, maar ze kunnen hierbij wel aardig lang worden (100 tekens zijn geen uitzondering). Relatieve padnamen hebben als voordeel dat ze korter kunnen zijn, en dat ze in zekere zin ‘positie-onafhankelijk’ zijn, dit wil zeggen dat ze binnen een bepaalde boomstructuur geldig blijven, ook wanneer de totale subboom verplaatst wordt. Absolute padnamen worden herkend aan het feit dat ze beginnen met een scheidingsteken.



Boomvormige directorystructuren hebben echter ook hun nadelen. Gemeenschappelijke bestanden (b.v. systeemprogramma's, maar ook gegevensbestanden die door verschillende gebruikers gedeeld worden) zullen ofwel met lange absolute padnamen moeten opgegeven worden, ofwel moeten ze gedupliceerd worden in de directory's van een gebruiker. Het zou handig zijn mochten bepaalde bestanden en directory's maar éénmaal moeten opgeslagen worden, maar op verschillende plaatsen in de directorystructuur te voorschijn kunnen komen zonder ze te moeten dupliceren.

Dit wordt mogelijk in de acyclische graafdirectory's. Hierbij maakt men gebruik van zogenaamde links of shortcuts. Unix onderscheidt twee soorten links: symbolische links en harde links. Een symbolische link is gewoon een verwijzing naar een ander bestand. Uitwendig ziet een symbolische link eruit als een gewoon bestand, maar het is wel een bestand van een speciaal type, en het bevat geen gegevens, enkel de naam van het bestand waarnaar het wijst. Als het bestand waarnaar gewezen wordt verdwijnt, dan zal de symbolische link blijven bestaan, maar verwijzen naar een niet langer bestaand bestand en een foutboodschap produceren indien gebruikt. Het analoge concept in Windows wordt een shortcut genoemd.

De tweede soort link is de harde link. Deze is te vergelijken met het delen van geheugen door dezelfde frame-adressen op te nemen in twee paginatabellen. In dit geval zullen twee elementen in de directorytabel wijzen naar hetzelfde fysieke bestand. Hierbij wordt een teller bijgehouden met het aantal links die verwijzen naar het bestand. Het wissen van een harde link zal enkel het wissen van het bestand tot gevolg hebben indien deze link de laatste link naar een bestand was.

Links in Unix

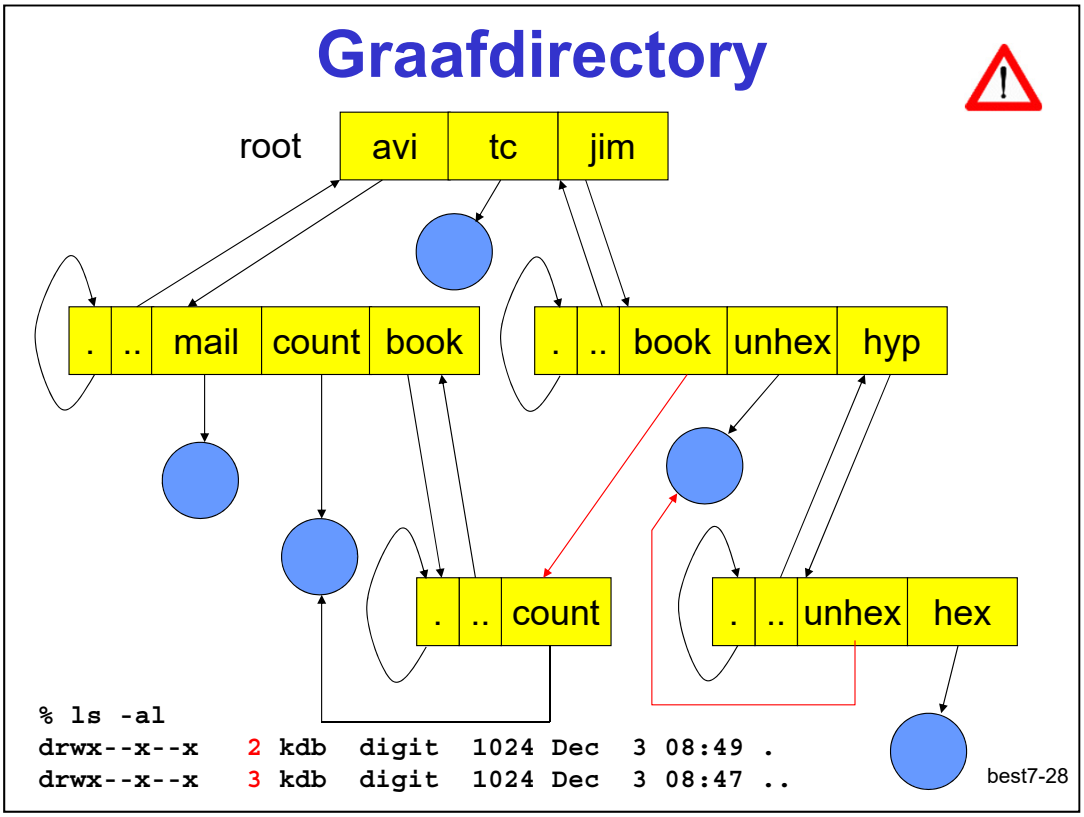
```
% ln -s ttt symlink
% ls -al
drwx--x--x   2 els  users   1024 Dec  3 08:48 .
drwx--x--x   3 els  users   1024 Dec  3 08:47 ..
lrwx--x--x   1 els  users      3 Dec  3 08:48 symlink -> ttt
-rw-------   1 tom  users     23 Dec  3 08:48 ttt
% cat symlink
dit is een testbestand
% ln ttt hardlink
% ls -al
drwx--x--x   2 els  users   1024 Dec  3 08:49 .
drwx--x--x   3 els  users   1024 Dec  3 08:47 ..
-rw-------   2 tom  users     23 Dec  3 08:48 hardlink
lrwx--x--x   1 els  users      3 Dec  3 08:48 symlink -> ttt
-rw-------   2 tom  users     23 Dec  3 08:48 ttt
% rm ttt
% ls -al
drwx--x--x   2 els  users   1024 Dec  3 08:49 .
drwx--x--x   3 els  users   1024 Dec  3 08:47 ..
-rw-------   1 tom  users     23 Dec  3 08:48 hardlink
lrwx--x--x   1 els  users      3 Dec  3 08:48 symlink -> ttt
% cat symlink
cat: cannot open symlink: No such file or directory
% cat hardlink
dit is een testbestand
```

best7-27

In dit voorbeeld worden links in Unix gedemonstreerd. Het bestand ‘ttt’ bevat de tekst ‘dit is een tekstbestand’. Met het commando ‘ln –s ttt symlink’ wordt een symbolische link ‘symlink’ gecreëerd naar het bestand ‘ttt’. Het bestand ‘symlink’ is van het type ‘l’ (zie eerste kolom). Het bevat een verwijzing naar het bestand ‘ttt’ (het is om die reden ook 3 bytes groot!).

Met het commando ‘ln ttt hardlink’ wordt een harde link gecreëerd naar het bestand ‘ttt’. In dit geval gaat het over een nieuw directoryelement dat aangemaakt wordt, maar wijst naar hetzelfde gegevensbestand. Het bestand met als inhoud ‘dit is een tekstbestand’ heeft nu 2 namen: ‘ttt’ en ‘hardlink’. Het feit dat er twee directoryelementen zijn die ernaar wijzen is af te lezen uit het cijfertje ‘2’ in de tweede kolom. Dit is het aantal links die verwijzen naar het bestand. Deze twee directoryelementen zijn equivalent. Dit blijkt uit het feit dat als men het oorspronkelijke bestand ‘ttt’ weglaat, het bestand ‘hardlink’ blijft bestaan. Het bestand ‘symlink’ blijft ook bestaan, maar geeft bij het gebruik ervan wel de boodschap dat het gerefereerde bestand (‘ttt’) niet bestaat. Bij hardlink is dat niet zo, deze heeft de rol van ‘ttt’ overgenomen. Een bestand in Unix wordt gewist als het aantal links op 0 komt, in dit geval bij het weglaten van het directoryelement ‘hardlink’. Het weglaten van een directoryelement wordt in de Unix wereld om die reden ‘unlink’ genoemd.

Merk op dat het bestand ‘ttt’ oorspronkelijk door de gebruiker tom aangemaakt werd en dat ‘hardlink’ blijkbaar ook eigendom is van tom, ofschoon niet door hem aangemaakt. Zelfs na het verdwijnen van ‘ttt’ blijft hardlink eigendom van tom en zal de plaats ingenomen door het bestand blijvend aangerekend worden aan de account van tom – ook al komt dat bestand niet langer voor in zijn eigen directorystructuur (gesteld dat de link in twee verschillende subdirectory's zou voorkomen).



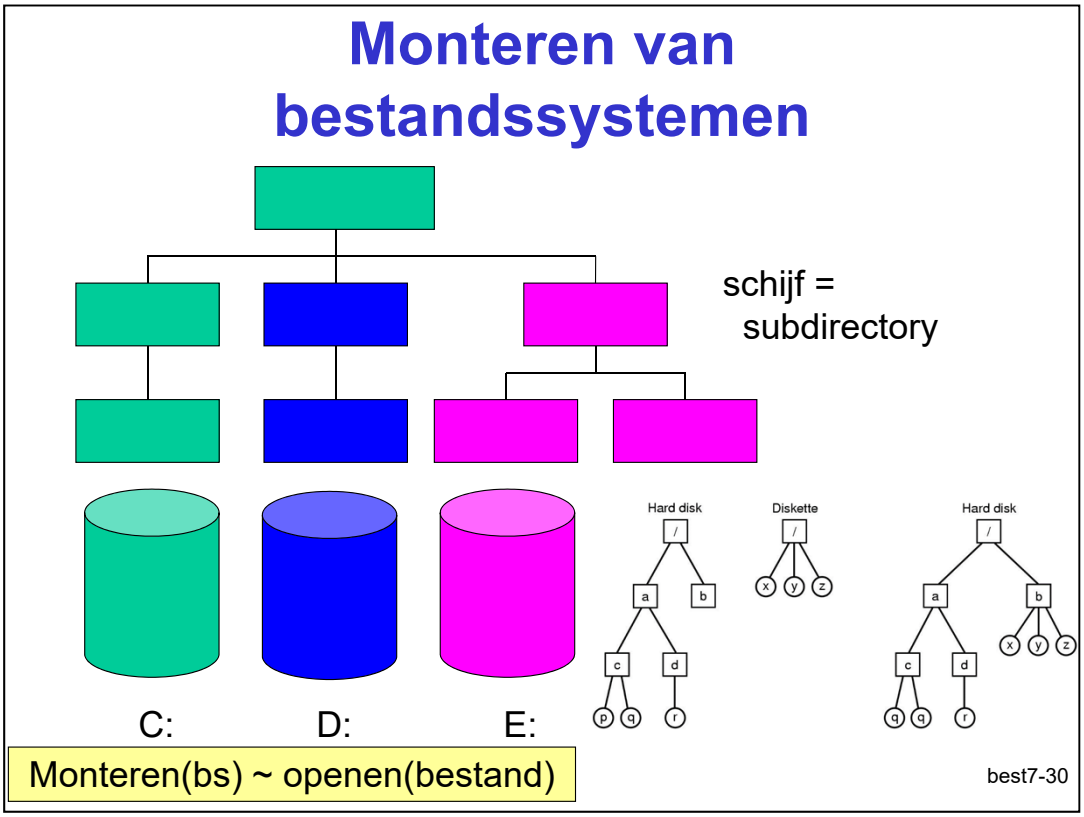
Het gebruik van links kan ervoor zorgen dat een directorystructuur niet langer een boomstructuur is maar een graaf. Een probleem met grafen is dat het doorzoeken van een graaf complexer is dan het doorzoeken van een boom omdat men moet vermijden tweemaal hetzelfde deel van een graaf te doorzoeken. Bij acyclische grafen heeft het meermaals doorzoeken van een graaf enkel een effect op de snelheid waarmee een algoritme uitgevoerd wordt (een gemeenschappelijke subdirectory kan b.v. tweemaal doorzocht worden). Bij een algemene graaf kunnen lussen voorkomen waardoor bepaalde subdirectory's telkens weer opnieuw bezocht worden. Het doorzoeken van een algemene graaf vereist dan ook een meer gesofistikeerd algoritme.

De directorystructuur van Unix en Windows is een voorbeeld van een graaf. De subdirectory '.' verwijst naar de directory waarin het bestand voorkomt (zelfreferentie), en de subdirectory '..' verwijst naar de directory waarin de subdirectory voorkomt (ouderdirectory). Dit zorgt ervoor dat een subdirectory minstens 2 links zal hebben, met name een link vanuit de ouderdirectory, en '.'. Het zorgt er ook voor dat een directory per bijkomende subdirectory een bijkomende link zal krijgen ('..' vanuit die subdirectory). Los van '.' en '..' kunnen er geen hardlinks naar directory's gemaakt worden. Mocht dit mogelijk zijn, dan zou dit leiden tot extra complexiteit. Zo zou '..' naar twee ouderdirectory's moeten verwijzen, en zouden bestanden ook twee verschillende padnamen kunnen hebben (via de twee mogelijke ouderdirectory's).

Overzicht

- Logische bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruime
  - Partities
  - Reservekopieën
- Optimalisaties

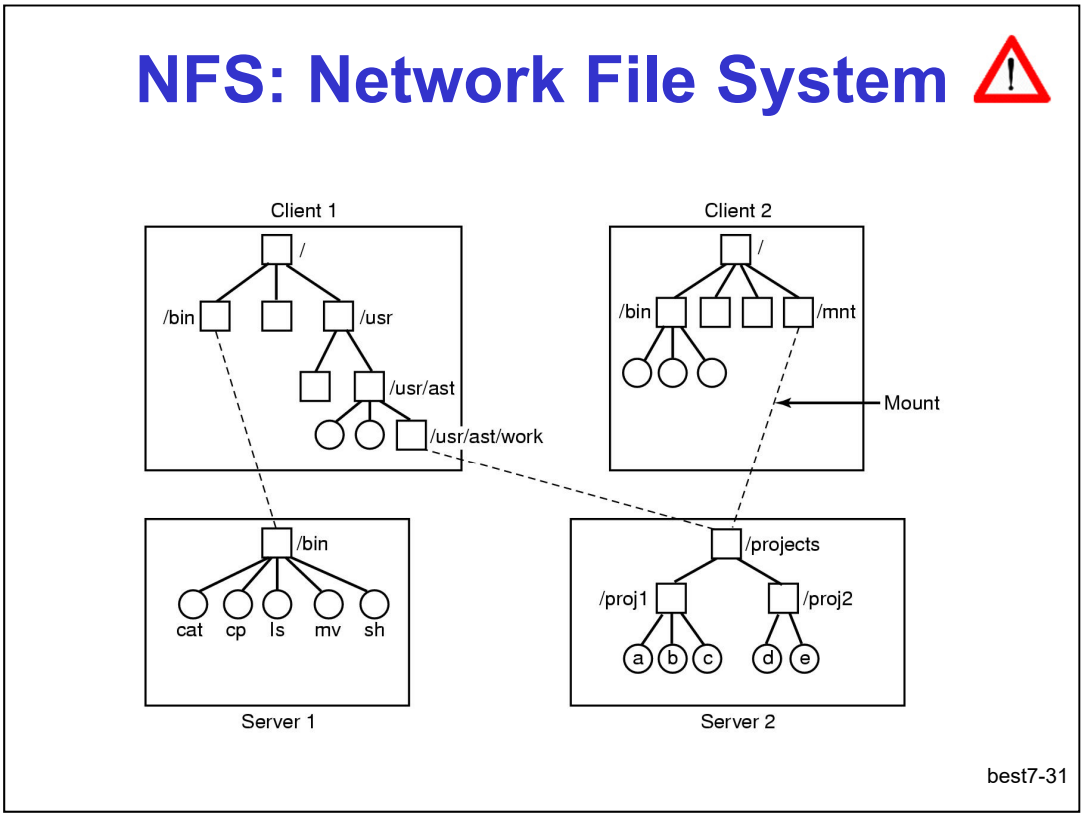
best7-29



Het secundair geheugen van een computersysteem kan uit verschillende gescheiden bestandssystemen bestaan (één per partitie, zie verder). Deze bestandssystemen kunnen op verschillende manieren aan de gebruiker aangeboden worden. Ofwel krijgen ze een naam zoals in Windows: A: B: C: enz. waardoor ze expliciet zichtbaar blijven, ofwel worden ze opgenomen als subdirectory in een bestandssysteem waardoor ze transparant worden voor de gebruiker.

Indien een partitie niet fysiek aanwezig is (b.v. bij usb-sleutels of CD-ROMS), zal deze subdirectory gewoon leeg zijn; indien de partitie aanwezig is, zal men in deze directory gegevens kunnen opvragen en bewaren. Eenmaal geconfigureerd is dit totaal transparant voor de gebruiker. Bij opslagmedia die normaal gezien slechts door één gebruiker op een zinvolle manier gebruikt kunnen worden zoals een usb-sleutel zal men niet enkel een directory moeten associëren met een bepaalde partitite, maar er ook moeten voor zorgen dat er slechts één gebruiker echt toegang heeft tot die directory.

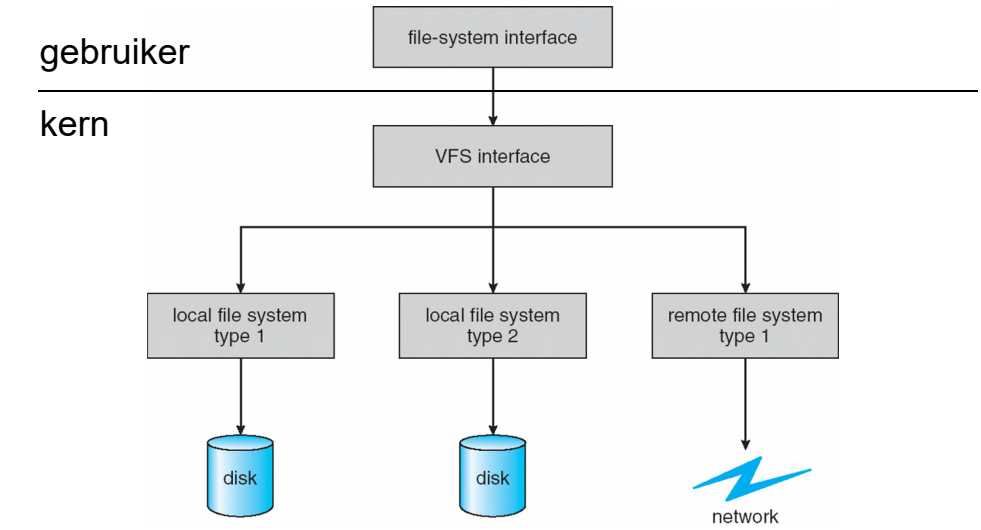
Het opnemen van een bijkomend bestandssysteem noemt men het ‘monteren’ van dat bestandssysteem. Het monteren van een bestandssysteem doet voor een bestandssysteem wat ‘open’ doet voor een bestand: het maakt het bestandssysteem klaar voor gebruik.



Het monteren van bestandssystemen hoeft niet beperkt te blijven tot 1 machine, maar kan ook uitgebreid worden tot bestandssystemen op andere machines. Een voorbeeld hiervan is NFS of network file system.

Het principe is precies hetzelfde. Enkel zal er nu een directory op een andere machine opgegeven worden. Eenmaal gemonteerd gedraagt de subdirectory van die andere machine zich net zoals de lokale directory's.

Virtueel bestandssysteem



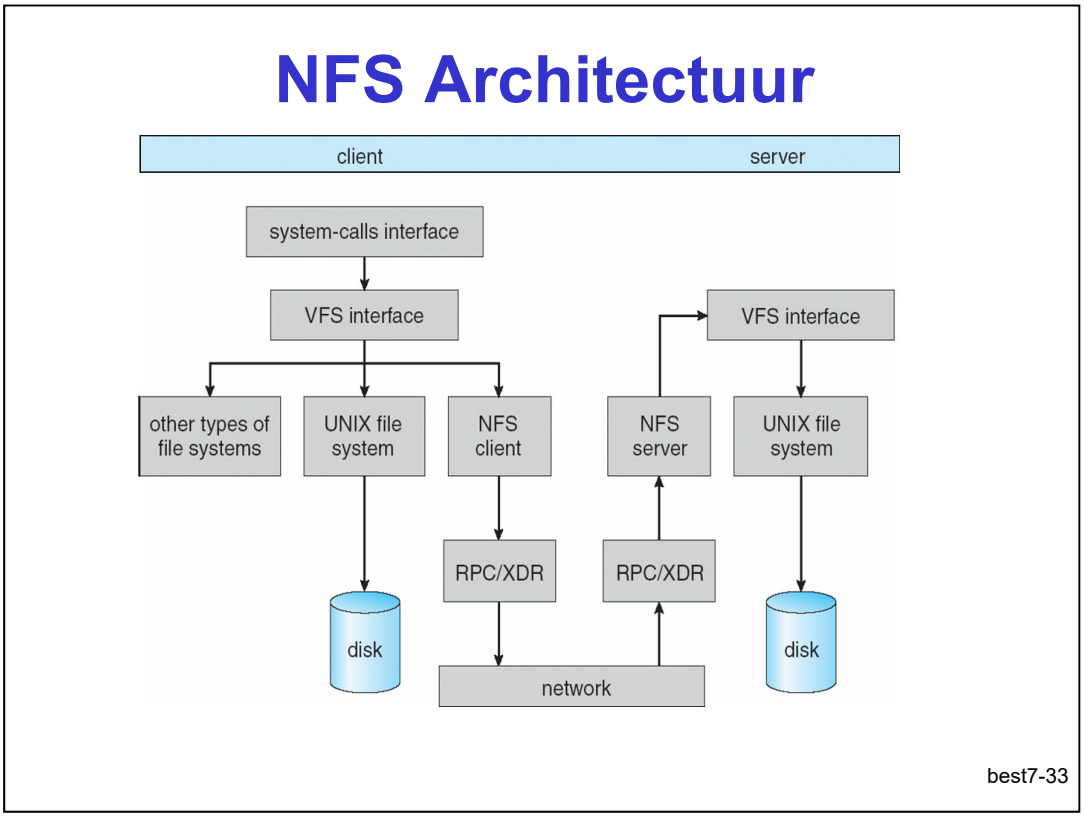
http://en.wikipedia.org/wiki/List_of_file_systems

best7-32

De meeste computersystemen zullen met verschillende bestandssystemen tegelijk werken (b.v. NTFS voor de harde schijf, en FAT voor de USB-sleutel). De verschillende bestandssystemen moeten echter in de mate van het mogelijke transparant zijn voor de gebruikersprogramma's. Dit wordt gerealiseerd aan de hand van een gestandaardiseerde interface met de bestandssystemen (open, close, read, write, ...).

Het virtueel bestandssysteem biedt de gebruikersprocessen een uniforme API aan die in het besturingssysteem zelf dan vertaald wordt naar oproepen van de verschillende bestandssystemen – zij het een harde schijf, een stuk van het fysiek geheugen of via het netwerk naar een schijf van een andere computer. Dit maakt de implementatie van de verschillende onderliggende systemen compleet onzichtbaar voor de gebruikersprogramma's die enkel interageren met het virtuele bestandssysteem.

Deze interface kan echter niet verhinderen dat sommige aspecten van het bestandssysteem toch nog doorschemeren (b.v. gevoeligheid voor hoofdletters en kleine letters, het al dan niet mogen voorkomen van bepaalde lettertekens in bestandsnamen).



NFS maakt gebruik van VFS om zijn diensten aan te bieden aan de applicaties. Deze zijn zich dus niet bewust van het feit dat de bestanden niet van de lokale schijf afkomstig zijn. NFS maakt gebruik van boodschappen die via RPC verstuurd worden tussen de client en de server. Om representatieverschillen tussen verschillende platformen op te vangen worden alle boodschappen genormaliseerd aan de hand van XDR (External Data Representation).

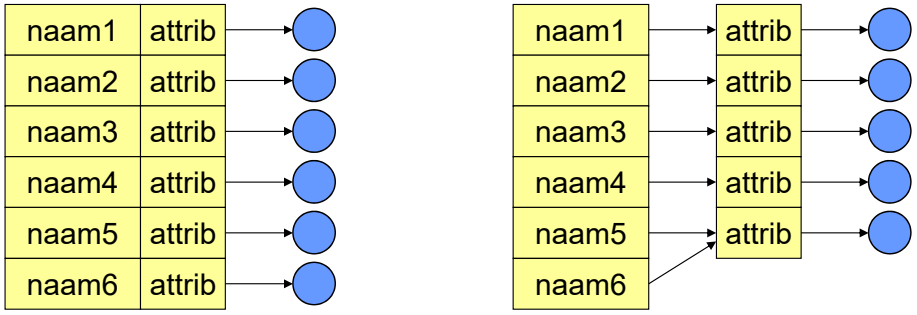
Overzicht

- Logische bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruimte
  - Partities
  - Reservekopieën
- Optimalisaties

best7-34

Organisatie van directory's

Directory: f(naam) → adres_van_attributen



Lineaire lijst of via hakselfunctie, of b-tree

best7-35

Doordat een directorybestand vrij vaak gebruikt wordt, moet het liefst zo efficiënt mogelijk georganiseerd zijn. Een directorybestand bestaat essentieel uit een lijst met directoryelementen. Men kan een directoryelement opzoeken door de lijst sequentieel te doorzoeken of door gebruik te maken van een hakselfunctie. De tweede methode gaat sneller, maar vereist wel een meer gesofistikeerde organisatie van de directoryinformatie. De belangrijkste functie is eigenlijk het terugzoeken van het adres van de attributen van een bestand, gegeven de naam (net zoals men in een telefoonboek het adres van iemands woning kan opzoeken in functie van zijn of haar naam, in het Engels spreekt men dan ook van een telephone directory).

Verder kan men beslissen om alle directoryattributen bij de naam van het bestand op te slaan, of daarentegen (een deel van) de attributen afzonderlijk op te slaan. De laatste voorstelling heeft als voordeel dat een bestand meer dan 1 naam kan dragen terwijl sommige attributen van het bestand centraal kunnen opgeslagen worden.

Organisatie van gegevensbestanden

- Contigue allocatie
- Gelinkte allocatie
- Geïndexeerde allocatie

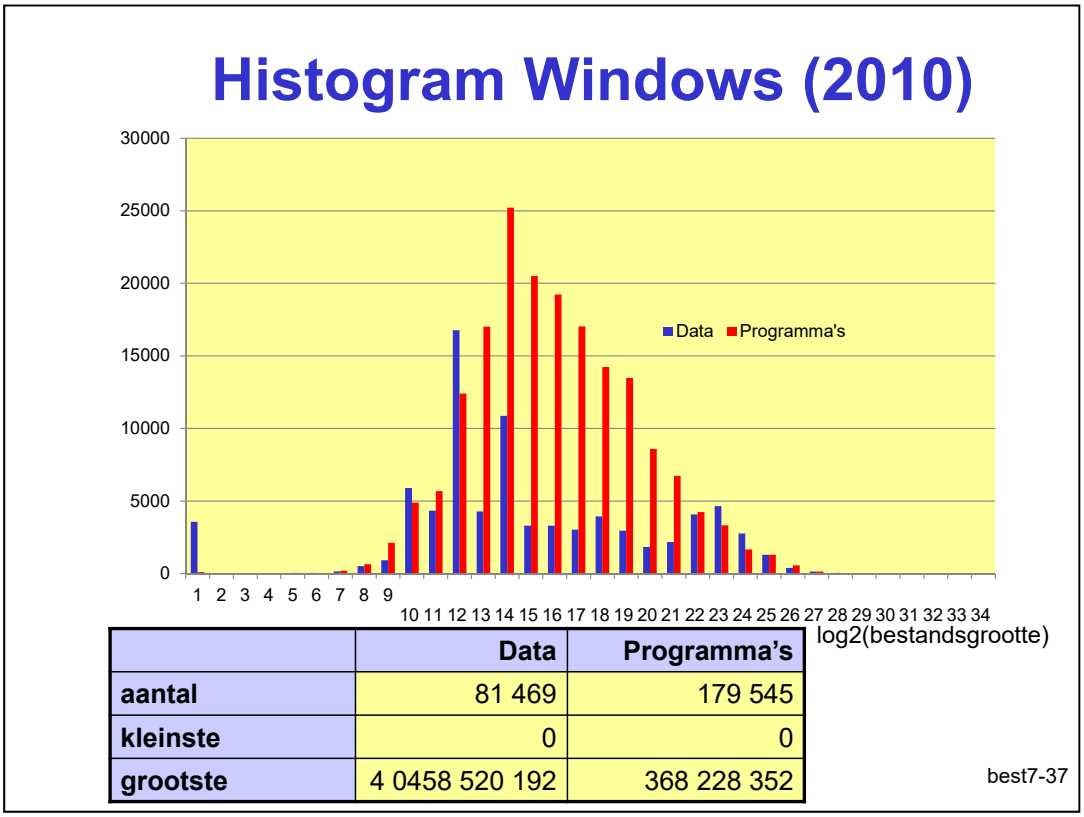
best7-36

De vraag die door de bestandsorganisatie dient beantwoord te worden is op welke manier bestanden best gebruik kunnen maken van de schijfblokken. Net zoals bij het geheugenbeheer het geheugen verdeeld moest worden over de verschillende processen, moet hier het schijfgeheugen verdeeld worden over de verschillende bestanden. Daarbij moeten we rekening houden met een aantal randvoorwaarden.

- De bestanden moeten direct toegankelijk zijn. Dit wil zeggen dat de tijd nodig om een bepaalde byte te lezen niet (te veel) mag afhangen van de plaats van dat byte in het bestand.
- De bestanden moeten betrouwbaar opgeslagen worden. Indien een bepaald blok van de schijf defecten vertoont, zal de schade best beperkt blijven tot het bestand of de directory waarin het defecte blok werd opgenomen.

Het probleem van de allocatie van bestanden betreft de vraag op welke manier de blokken of clusters van de schijf samengevoegd kunnen worden tot een bestand. Er zijn verschillende mogelijkheden: contigu, gelinkt, gebaseerd op een allocatietabel, en gebaseerd op een indextabel. We gaan er hier voor de eenvoud van uit dat de blokken (of clusters) 8 KiB groot zijn (of 16 sectoren). Dit is een courante keuze. Merk op dat deze ontwerpskeuze volledig los staat van b.v. de keuze van de framegrootte.

De keuze van een bepaalde allocatiemethode zal onder meer afhangen van de manier waarop de gegevens van een schijf gebruikt zullen worden: sequentieel, direct of geïndexeerd, en van de prestaties die men van de schijf verwacht. Doordat de schijf zeer traag is in vergelijking met de processor, zal men doorgaans trachten om het aantal toegangen naar de schijf zoveel mogelijk te beperken door bepaalde tabellen in het geheugen te houden, door gegevens intelligent over de schijf/schijven te verdelen zodat de kopbewegingen beperkt worden, enz. Vaak kan het de moeite lonen om de processor een complexer algoritme te laten uitvoeren indien dit het gebruik van de schijf kan verminderen.

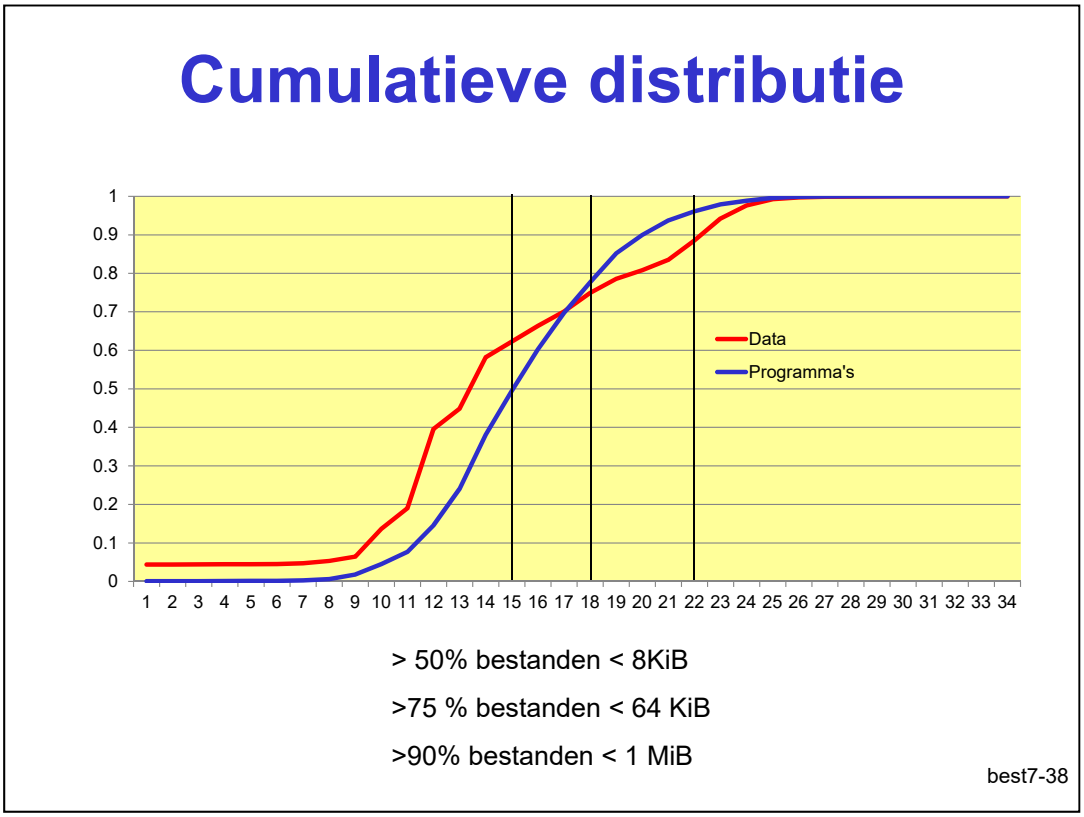


	Data	Programma's
aantal	81 469	179 545
kleinste	0	0
grootste	4 0458 520 192	368 228 352

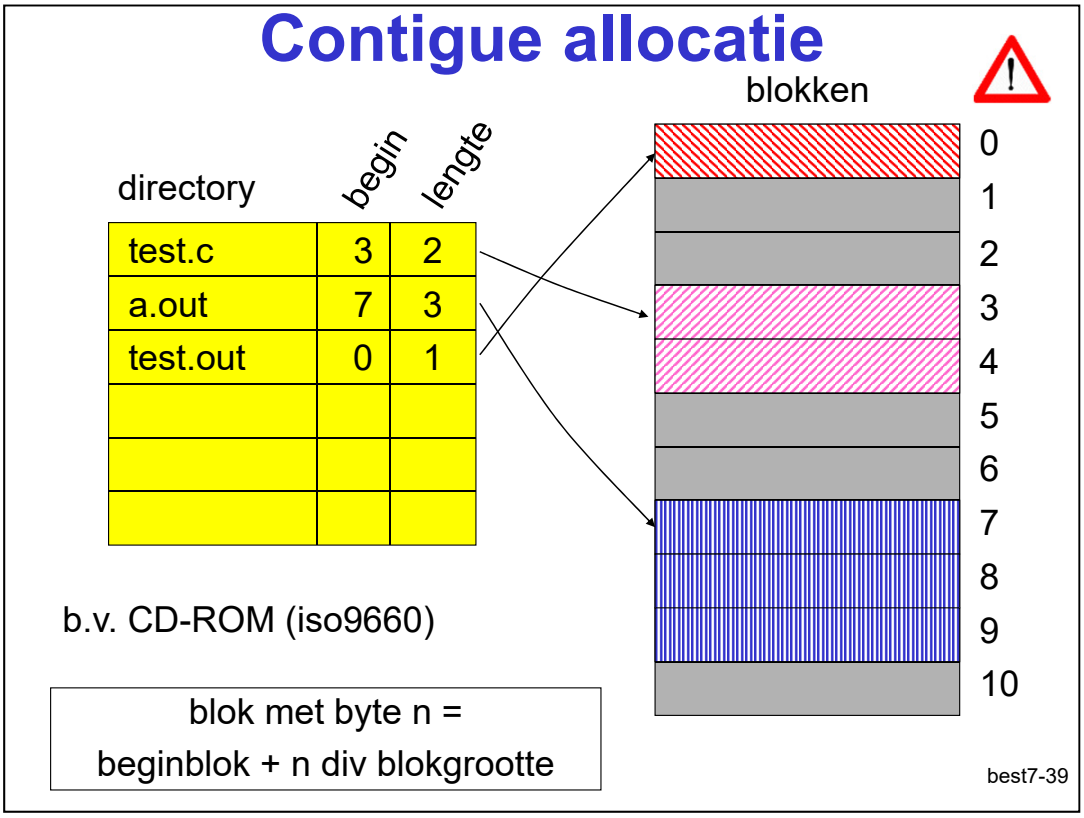
best7-37

Deze afbeelding toont een histogram van ruim 150 000 bestanden die op een Windows XP computer gevonden werden. Hierbij wordt een onderscheid gemaakt tussen de programmabestanden (schijf C:) en de gegevensbestanden (schijf D:). Uit deze figuur volgt dat de meeste bestanden kleiner zijn dan 1 MiB, maar dat er ook heel grote bestanden kunnen voorkomen.

Het loont met andere woorden zeer zeker de moeite om een bestandssysteem (i) te optimaliseren voor kleine bestanden, en (ii) toch ook aandacht te hebben voor de grote tot zeer grote bestanden.



Uit de bovenstaande figuur blijkt dat de meeste bestanden zeer klein zijn, de helft ervan is kleiner dan 8 KiB. Er is zelfs een zeer grote fractie van bestanden met lengte 0. In een beperkt aantal gevallen zijn dit bestanden die gecreëerd werden om een boolese waarde bij te houden (bestand bestaat = true; bestand bestaat niet = false). In de meeste gevallen blijken het echter vooral tijdelijke internetbestanden te zijn waarvan de transfer onderbroken werd, en uiteindelijk resulteerden in een leeg bestand. De toename tussen 128 en 512 bytes is voor een stuk te wijten aan de aanwezigheid van cookies.



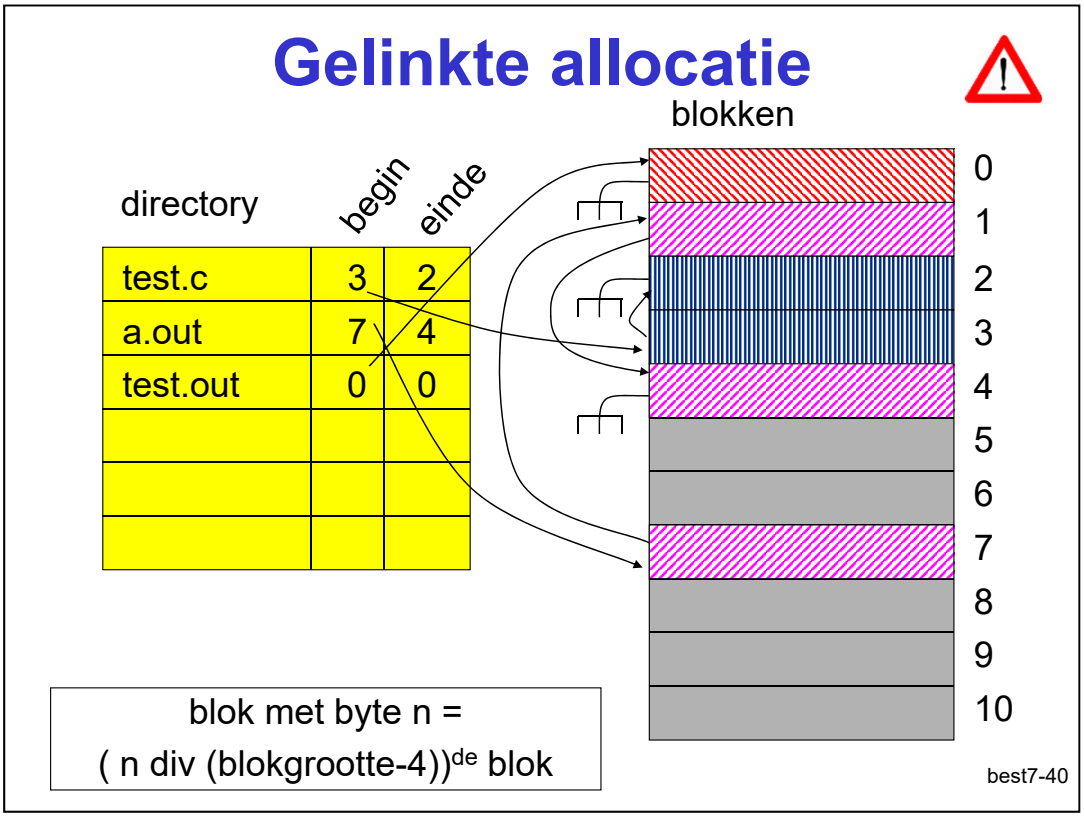
Contigue allocatie is de eenvoudigste allocatiemethode. Een bestand van n blokken zal n opeenvolgende blokken op de schijf innemen. De verplaatsingen van de lees/schrijfkop zullen hierdoor minimaal zijn: bij het sequentieel lezen zal de kop enkel op het einde van het laatste spoor van een cilinder moeten veranderen naar een volgende cilinder en ook bij directe toegang zullen de verplaatsingen van de kop minimaal zijn. Contigue allocatie is hierdoor de **snelste allocatiemethode**.

Contigue allocatie heeft echter ook nadelen. Vooreerst is er het probleem van de **externe fragmentatie**. Doordat het schijfgeheugen dat gealloceerd moet worden contigu moet zijn, zal men na verloop van tijd fragmentatie krijgen. Compactering kan hier een oplossing brengen, maar meestal zal het bestandssysteem tijdens het compacteren minder bruikbaar zijn. Dit is niet steeds aanvaardbaar.

Ten tweede is er het probleem dat men nog vóór de creatie van een bestand moet weten **hoe groot het zal worden**. Schat men de werkelijke grootte te hoog, dan krijgt men een aanzienlijke interne fragmentatie, schat men het te klein, dan zal het programma dat het bestand aanmaakt afgebroken worden. Dit laatste kan vermeden worden door in dat geval het bestand te kopiëren naar een grotere vrije ruimte en daar de uitvoering verder te zetten. Het hoeft geen betoog dat dit de doorlooptijd van een proces negatief zal beïnvloeden. Langzaam groeiende bestanden zoals log-bestanden zijn vrij moeilijk efficiënt te implementeren.

Sommige systemen (b.v. Veritas File System) laten daarom toe om de contigue allocatie stuksgewijs te ondersteunen. Men begint dan met een bestand met een gegeven grootte, en indien het te klein zou blijken te zijn, kan men een bijkomende uitbreiding toevoegen. De grootte van deze uitbreiding moet ook op voorhand vastgelegd worden. Ofschoon deze oplossing efficiënter is dan het volledig kopiëren van het bestand, heeft ze ook te

lijden onder het probleem van de externe fragmentatie.

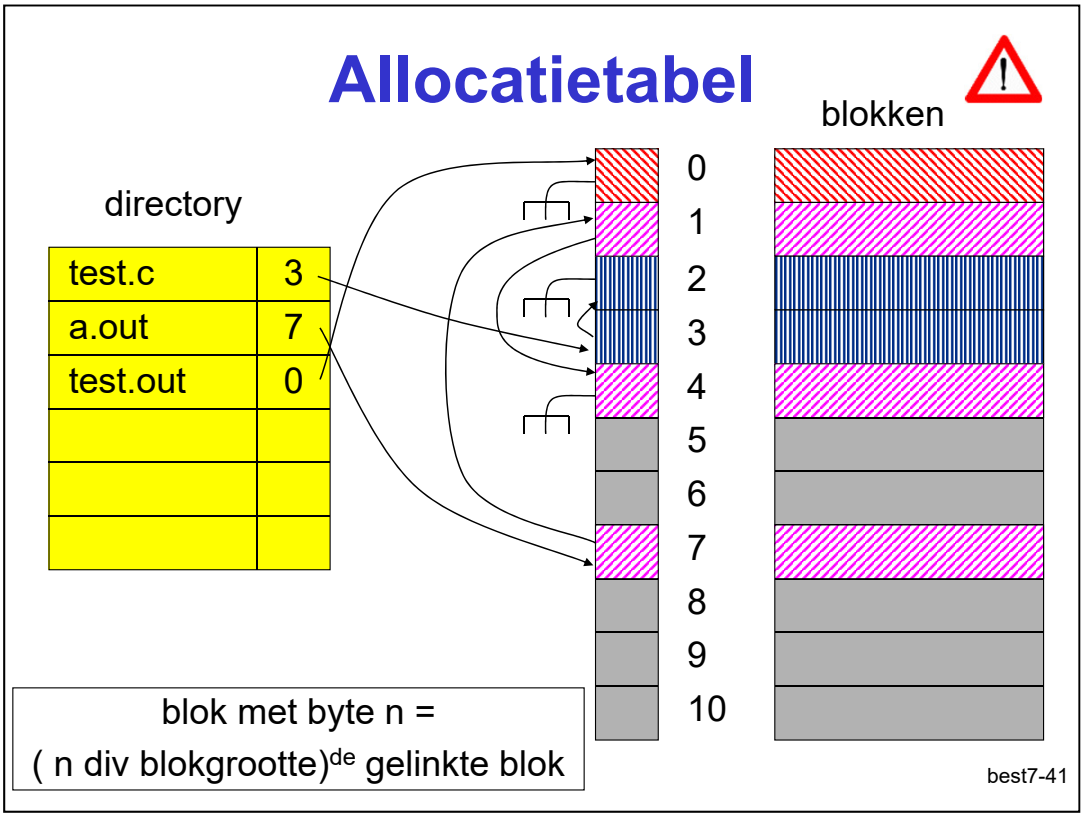


In de plaats van alle blokken contigu op de schijf op te slaan worden de blokken op de schijf bij gelinkte allocatie met elkaar gelinkt. Zolang er vrije blokken zijn, kan een bestand blijven groeien. Ofschoon deze methode het probleem van de externe fragmentatie effectief oplost, heeft ze ook haar problemen. Vooreerst wordt directe toegang nagenoeg onmogelijk omdat steeds de lijst van blokken moet afgelopen worden. Ten tweede kan ook het sequentieel lezen van een bestand zeer traag worden omdat er per nieuw in te lezen blok in principe een verplaatsing van de kop nodig kan zijn. Ten derde zullen de blokken nu een beetje kleiner zijn omdat de wijzer naar het volgende blok ook moet opgenomen worden. Tenslotte is gelinkte allocatie ook niet zeer betrouwbaar omdat van zodra er één blok corrupt wordt de rest van het bestand verloren gaat.

Om het sequentieel lezen te versnellen kan men bij de allocatie van de blokken er trachten voor te zorgen dat ze toch zoveel mogelijk sequentieel op de schijf staan zodat de bewegingen van de lees/schrijfkop beperkt worden. De contiguïteit en dus de snelheid kan verbeterd worden door regelmatig te compacteren.

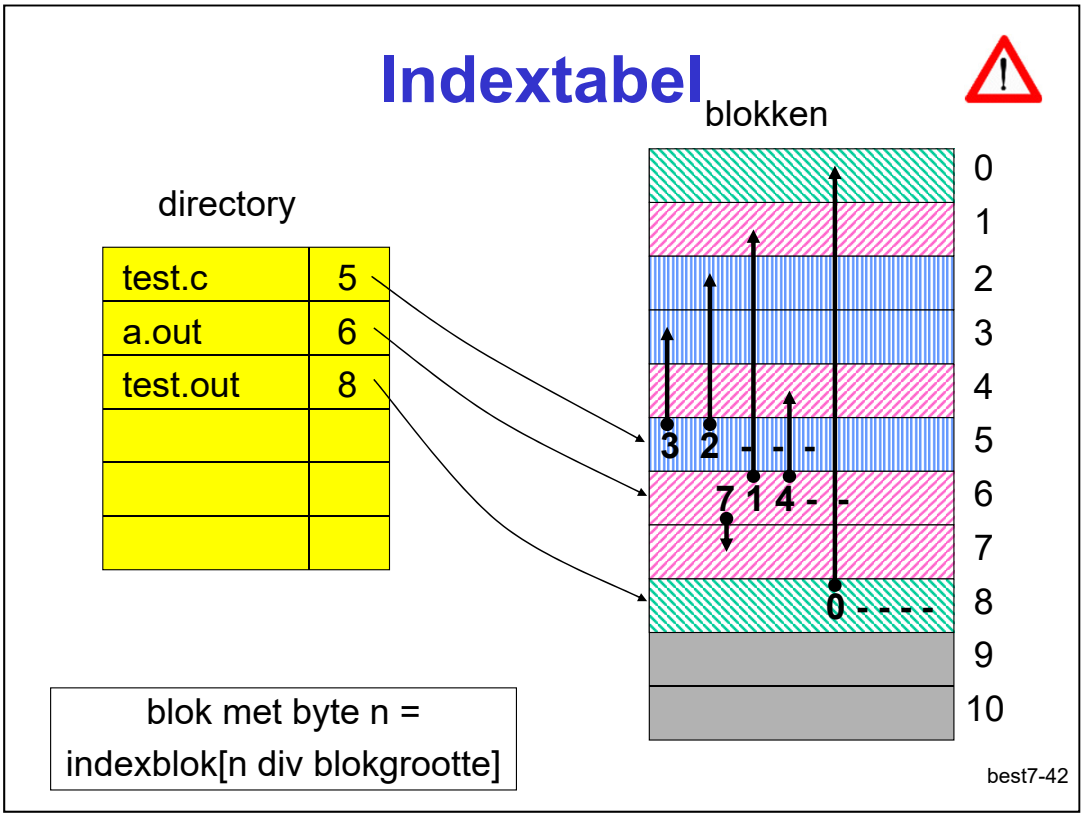
Het derde probleem kan minder erg gemaakt worden door de blokgrootte te vergroten. Gezien er slechts één wijzer per blok opgeslagen wordt, verkleint hierdoor de overhead voor de wijzers.

De trage directe toegang is onvermijdelijk bij gelinkte allocatie.



Een aantal van de nadelen van gelinkte allocatie kunnen minder erg gemaakt worden door alle wijzers bij te houden in afzonderlijke blokken, de zogenaamde FAT of file allocation table. Deze tabel bevat één wijzer per gegevensblok. In de plaats van de wijzers fysiek in de blokken te schrijven worden ze nu in de FAT geschreven. Dit heeft als voordelen (i) dat men om een bestand af te lopen slechts een paar blokken van de schijf moet lezen (de FAT), (ii) dat er geen wijzers meer in de gegevensblokken moeten opgeslagen worden en de gegevensblokken dus hun originele capaciteit behouden, (iii) dat men de FAT meer dan eens kan opslaan op de schijf om beperkte schijfdefecten te kunnen opvangen. Vrije blokken kunnen teruggevonden worden door een speciale wijzer in de FAT te schrijven (b.v. 0). MS-DOS en OS/2 maken gebruik van een FAT.

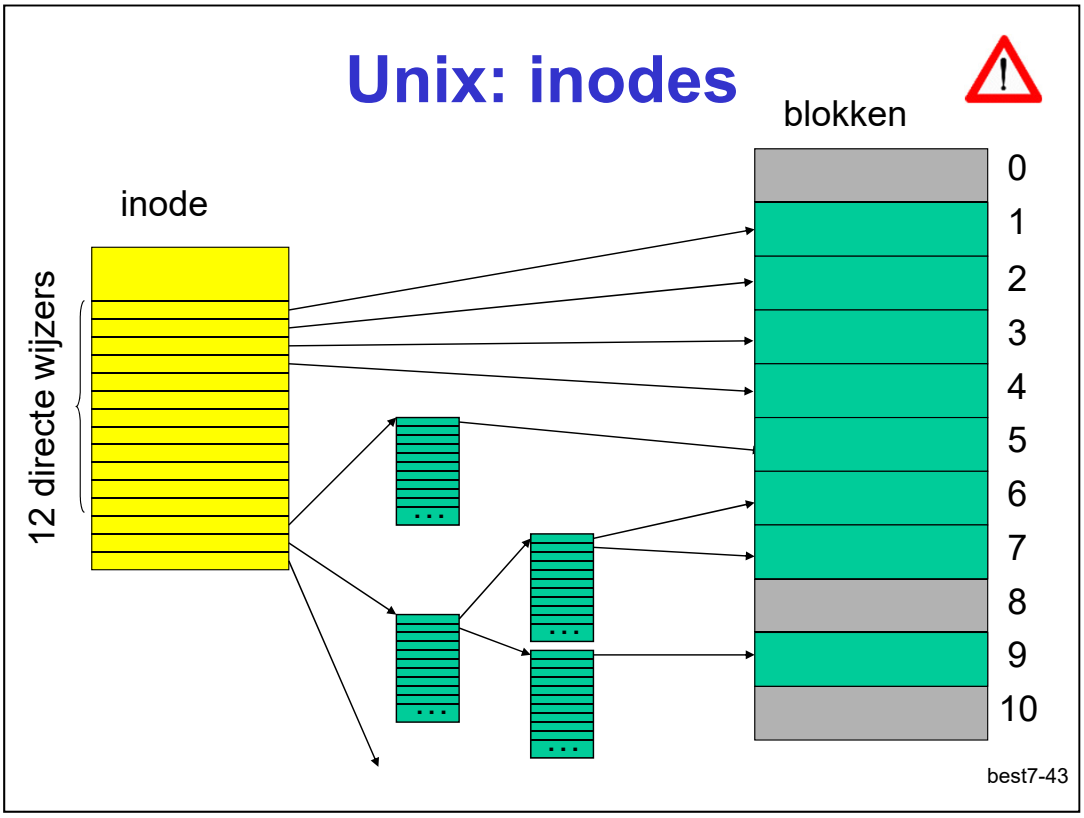
Meestal bewaart men een kopie van de FAT (of delen ervan) in het geheugen om snel een blok op de schijf terug te kunnen vinden. Indien de FAT niet in het geheugen bijgehouden wordt, zal de toegangstijd uiteraard aanzienlijk toenemen. Het integraal in het geheugen houden van de FAT is echter niet zo eenvoudig voor grote schijven. Indien er gewerkt wordt met blokken van 8 KiB zal men voor een schijf van 80 GiB toch 40 MiB FAT ruimte moeten alloceren. Bovendien zal het gebruik van de FAT voor zeer grote bestanden toch traag blijven. Een bestand van 8 GiB heeft meer dan 1 miljoen wijzers die moeten afgelopen worden om de laatste blokken van het bestand te bereiken. Dit zal miljoenen instructies vergen en milliseconden duren (met veel cachemissers omwille van de slechte lokaliteit van de FAT-gegevens).



Gelinkte allocatie of allocatietabellen zijn een afdoende oplossing voor het probleem van de externe fragmentatie, maar laten geen efficiënte directe toegang tot een bestand toe omdat de links sequentieel moeten afgelopen worden. Geïndexeerde allocatie laat dit wel toe. Het idee is dat alle wijzers naar de gealloceerde blokken opgenomen worden in een indexblok. Hiermee lost men eigenlijk twee problemen op: het verbetert de lokaliteit van de wijzers die horen bij 1 bestand, en de wijzertabel kan nu rechtstreeks geïndexeerd worden..

Om niet beperkt te zijn tot bestanden met een te kleine maximale lengte kan men ofwel de indexblokken linken, ofwel verschillende niveaus van indexblokken creëren. Het topniveau verwijst dan naar indexblokken die op hun beurt verwijzen naar de gegevensblokken. Gecombineerde systemen komen ook voor.

Geïndexeerde allocatie maakt doorgaans minder efficiënt gebruik van de schijfruimte dan gelinkte allocatie. De meeste indexblokken zullen immers maar gedeeltelijk gevuld zijn. Een blok van 8 KiB biedt plaats aan 2048 wijzers terwijl bestanden tot 1 MiB maximaal 128 wijzers nodig hebben. De meeste bestanden zullen minder dan 10 van de 2048 wijzers gebruiken. Voor kleine bestanden is de overhead van de indexblokken dan ook zeer groot.



In Unix maakt men gebruik van een gecombineerd systeem waarbij een vast aantal indices opgenomen wordt als attributen in de zogenaamde inode (index node). Naast dit beperkt aantal directe indices (bv. 12) die zullen volstaan voor de bestanden kleiner dan 96 KiB, is er nog een wijzer naar een indexblok van niveau 1, en ook nog een wijzer naar een indexblok van niveau 2 (en zelfs naar niveau 3). De indexblokken hebben dezelfde grootte als de datablokken.

Inodes combineren eigenlijk het beste van twee werelden. Voor het grote aantal kleine bestanden is er enkel de overhead van de eerste 12 wijzers en zullen er geen indextabellen gealloceerd worden. Dit is prima omdat in de praktijk toch blijkt dat minder dan 5% van alle bestanden groter zijn dan 96 KiB en enkel deze een indexblok nodig hebben. Alle andere kunnen het stellen zonder afzonderlijk indexblok. De toegang is bovendien snel. Voor de grotere bestanden zal de directe toegang iets trager worden door de tussenkomst van de indextabellen (die uiteraard wel in het geheugen kunnen bijgehouden worden). Deze oplossing laat toe om (i) snel directe toegang te hebben tot kleine bestanden, redelijk snel tot de middelgrote bestanden, en aanvaardbaar voor de echt grote bestanden waarbij de grootte van de bestanden niet beperkt wordt door de implementatie van het bestandssysteem, maar door de grootte van de schijf.

Merk wel op dat men bij het gebruik van 2 niveaus van indexering meer dan 32 GiB kan alloceren en dat dit in de praktijk zal vereisen dat de bestandswijzer (die de adressen van de bytes in het bestand bijhoudt) meer dan 32 bit groot is.

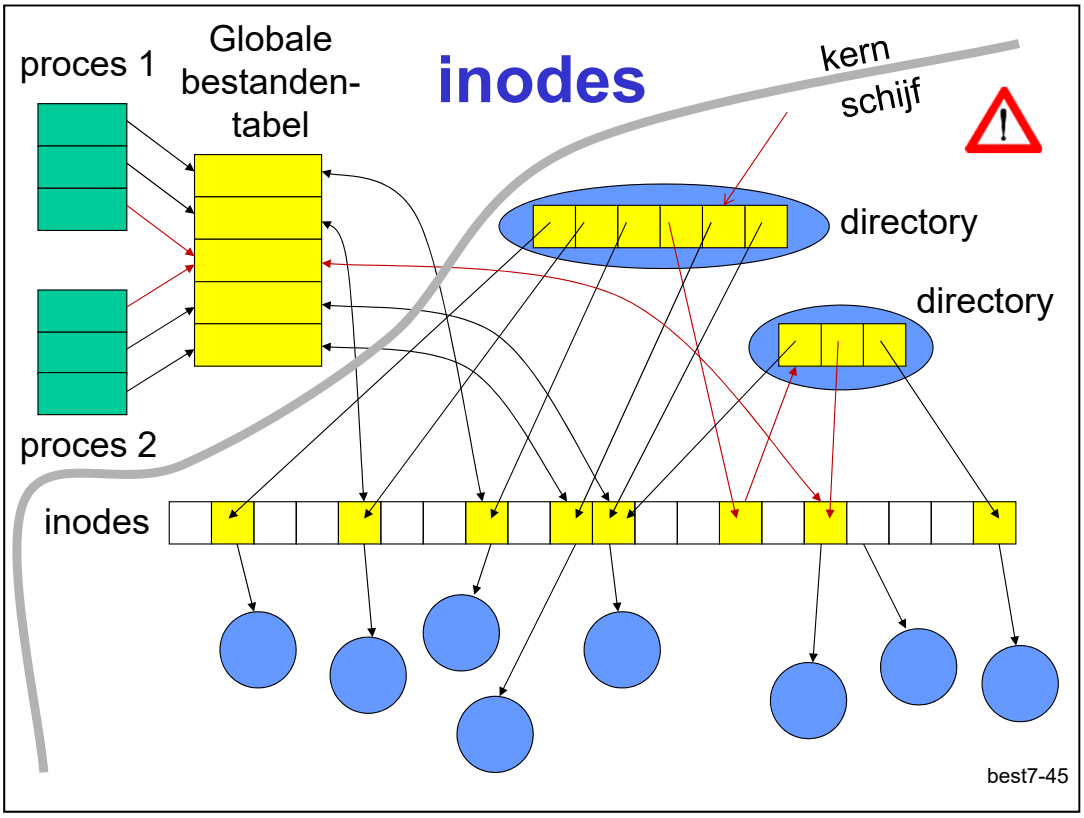
Unix Inode

Offset	Size	Name	Description
0x0	__le16	i_mode	
0x2	__le16	i_uid	Lower 16-bits of Owner UID.
0x4	__le32	i_size_lo	Lower 32-bits of size in bytes.
0x8	__le32	i_atime	Last access time, in seconds since the epoch.
0xC	__le32	i_ctime	Last inode change time, in seconds since the epoch.
0x10	__le32	i_mtime	Last data modification time, in seconds since the epoch.
0x14	__le32	i_dtime	Deletion Time, in seconds since the epoch.
0x18	__le16	i_gid	Lower 16-bits of GID.
0x1A	__le16	i_links_count	Hard link count.
0x1C	__le32	i_blocks_lo	
0x20	__le32	i_flags	
0x24	4 bytes	Union osd1:	
0x28	60 bytes	i_block[15] (12+1+1+1)	Block map or extent tree. See the section "The Contents of inode.i_block".
0x64	__le32	i_generation	File version (for NFS).
0x68	__le32	i_file_acl_lo	Lower 32-bits of extended attribute block.
0x6C	__le32	i_size_high / i_dir_acl	Upper 32-bits of file size.
0x70	__le32	i_obso_faddr	(Obsolete) fragment address.
0x74	12 bytes	Union osd2:	
0x80	__le16	i_extra_isize	Size of this inode - 128.
0x82	__le16	i_checksum_hi	Upper 16-bits of the inode checksum.
0x84	__le32	i_ctime_extra	Extra change time bits. This provides sub-second precision.
0x88	__le32	i_mtime_extra	Extra modification time bits. This provides sub-second precision.
0x8C	__le32	i_atime_extra	Extra access time bits. This provides sub-second precision.
0x90	__le32	i_crtime	File creation time, in seconds since the epoch.
0x94	__le32	i_crtime_extra	Extra file creation time bits. This provides sub-second precision.
0x98	__le32	i_version_hi	Upper 32-bits for version number.

best7-44

Deze tabel geeft een idee van de informatie die voorkomt in de inode. Merk op dat de naam van het bestand hier niet in voorkomt (deze is specifiek voor de directory waarin een bestand opgenomen werd; een bestand dat in twee directory's voorkomt kan twee verschillende namen hebben, en kan op twee verschillende manieren beschermd zijn).

De inode van een bestand in Linux kan opgevraagd worden met “ls –i bestandsnaam”.



Op deze afbeelding wordt schematisch weergegeven hoe het complete bestandssysteem van Unix werkt. directory's zijn bestanden met daarin de informatie over de bestanden zoals ze voorkomen in de directory (b.v. de naam). Verder bevat het directoryelement ook nog een verwijzing naar een inode die de overige informatie van het bestand bijhoudt (grootte, ogenblik laatste gebruik, plaats op de schijf, enz.). Merk op dat een subdirectory ook een bestand is.

Bij het openen van een bestand wordt het bestand opgezocht in de globale bestandentabel. Indien het daar nog niet in voorkomt, wordt het bestand opgezocht in de directorystructuur en wordt de inode gekopieerd naar de globale bestandentabel. Vanuit de lokale bestandentabel per proces wordt er dan een verwijzing gemaakt naar de globale tabel. Eenmaal het proces beschikt over een verwijzing naar de inode wordt de directorystructuur niet meer gebruikt. Deze wordt enkel gebruikt om de inode van een bestand op te zoeken. Bij het terug sluiten van het bestand wordt de informatie uit de inode op de schijf uiteraard terug geactualiseerd (b.v. nieuwe bestandslengte, tijdstip van laatste gebruik).

Het aantal inodes wordt vastgelegd bij de formattering van de schijf (zie verder). Dit aantal legt meteen ook het maximaal aantal bestanden van de schijf vast. Enkel de inodes van de open bestanden worden bijhouden in het geheugen (in tegenstelling met de volledige FAT).

ijle bestanden

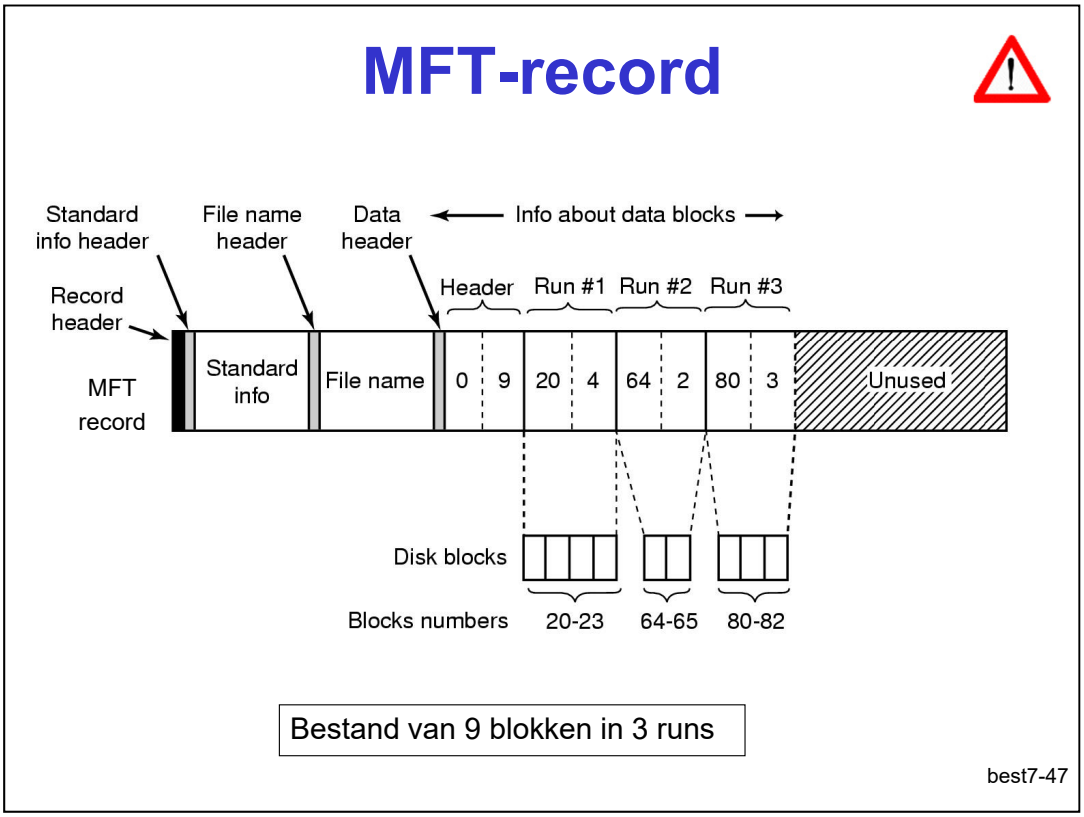
```
main()
{
   int h = open("test.dat", O_CREAT|O_WRONLY, 0777);
   lseek(h, 100000, SEEK_SET);
   write(h, "Hallo", 5);
   close(h);
}
```

```
% ls -al test.dat
-rwx--x--x    1 kdb      users      100005 Nov 15 20:13 test.dat
% du –b test.dat
8192  test.dat
```

best7-46

De bestandsorganisatie van Unix laat toe om zgn. ijle bestanden aan te maken (sparse files). Dit zijn bestanden waarvan niet alle gegevensblokken gealloceerd werden. In de afbeelding wordt een bestand aangemaakt met 5 bytes op adres 100000. Op de andere plaatsen in het bestand wordt er niet geschreven. Uiteindelijk rapporteert de directorylisting een bestand van 100005 bytes – dit is inderdaad het adres van de laatste byte in het bestand.

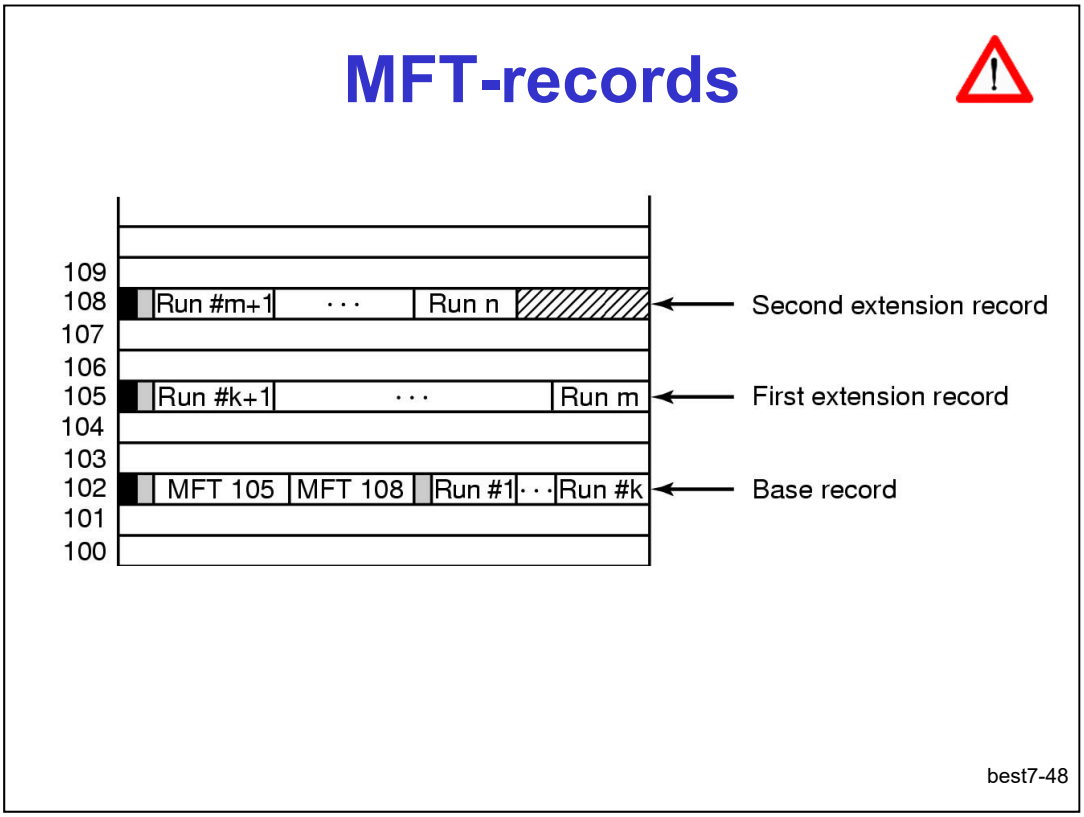
Als we met het commando ‘du –b’ de effectieve ingenomen ruimte opvragen, dan blijkt dit maar 8192 bytes te zijn, of 1 blok. Blijkbaar werden alle tussenliggende blokken niet gealloceerd. Het commando ‘du’ maakt blijkbaar ook geen vermelding van het indexblok (eigenlijk werden er 2 blokken gealloceerd). Als men het bestand opent, en men leest vanaf het begin, dan retourneert het bestand 100 000 nulbytes gevolgd door de tekenrij Hallo.



De informatie over bestanden wordt in NTFS bijgehouden in de master file table (MFT), dit is een systeembestand in een NTFS-partitie met records van 1 KiB. Per bestand of directory wordt er minstens 1 MFT record gebruikt om de attributen ervan op te slaan. Een MFT kan maximaal 2^{48} records bevatten. Een MFT-record bestaat uit een opeenvolging van attribuut-waarde paren (inclusief het attribuut data!). Indien mogelijk worden attribuut-waarde paren in de MFT-record opgeslagen (residente attributen). Indien ze te groot zijn (b.v. voor de data), dan wordt er een wijzer opgenomen naar een andere plaats op de schijf (niet-residente attributen). De eerste 16 bestanden in de MFT worden gereserveerd voor speciale bestanden (b.v. $LogFile, $Volume, $Bitmap, $BadClus, enz.).

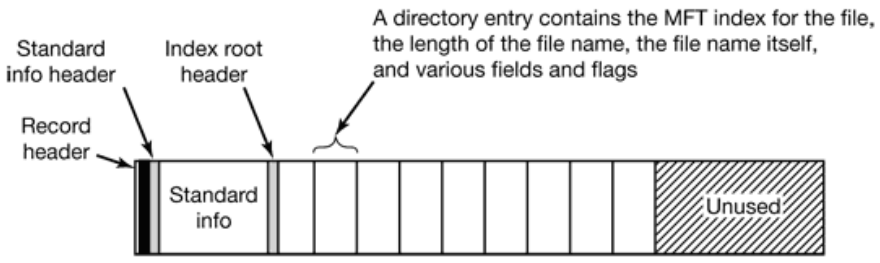
Het data attribuut bevat de inhoud van het bestand. Indien het bestand slechts enkele honderden bytes groot is, kan de inhoud ervan in de MFT zelf opgeslagen worden. Men spreekt dan van een ‘immediate file’.

Indien de hoeveelheid data te groot is, worden de data opgeslagen in afzonderlijke blokken op de schijf. De data header wordt dan gevolgd door een omschrijving van die blokken. Dit gebeurt door het opgeven van een header die het aantal blokken van het bestand aangeeft (hier van 0 tot en met 8, 9 geeft het eerste vrije blok aan), gevolgd door een aantal runs van blokken, hier 4 blokken beginnend bij blok 20, gevolgd door 2 blokken beginnend bij blok 64 en eindigend met 3 blokken beginnend bij blok 80. Desgewenst kan er dan een nieuwe header volgen b.v. (20,30) die een gat laat van blok 9 tot blok 19 en vanaf 20 tot 29 opnieuw blokken alloceert. Deze header wordt dan opnieuw gevolgd door de runs voor dit stuk van het bestand. Op die manier worden ijle bestanden ondersteund.



Het kan voorkomen dat de totale hoeveelheid informatie niet in 1 MFT record past. Dan kunnen er bijkomende MFT records gealloceerd worden zoals getoond in de afbeelding. Indien er zoveel bijkomende MFT records nodig zijn dat hun verwijzingen niet in de basis MFT-record passen dan moet de verwijzingen niet-resident opgeslagen worden waardoor er een onbeperkt aantal MFT-records kan gespecificeerd worden.

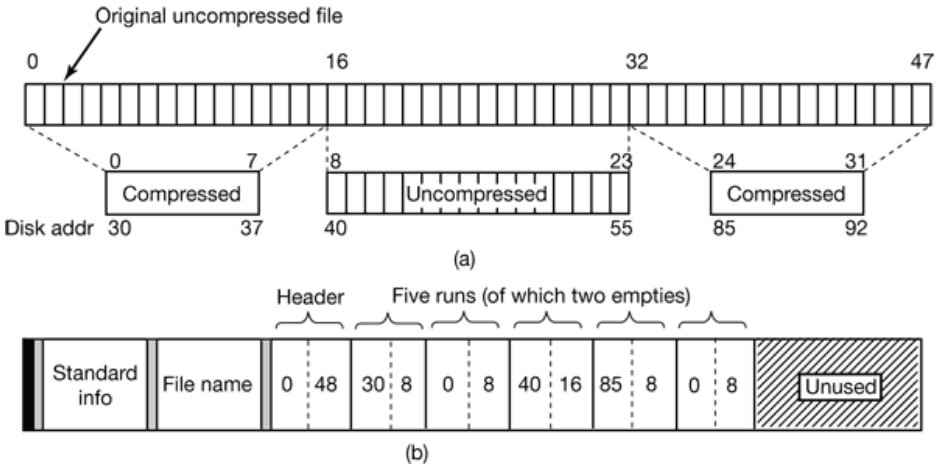
Directory



best7-49

Directory's worden ook opgeslagen in de MFT-records. De directory-elementen worden gewoon sequentieel in de MFT-record geplaatst (variabele lengte als gevolg van de variabele lengte van de bestandsnamen). Het directory-element verwijst naar het bestand door het opnemen van de corresponderende MFT-index. Voor grote directory's wordt er gebruik gemaakt van een intelligentere indexeringsmethode (B+boom).

Compressie



best7-50

NTFS kan ook bestanden comprimeren. Hiervoor verdeelt het het bestand in stukken van 16 logische blokken en probeert ieder van die stukken afzonderlijk te comprimeren. Indien de compressie een kleiner stuk oplevert, dan wordt dat opgeslagen, indien niet, dan wordt de ongecomprimeerde data opgeslagen. De header blijft altijd de oorspronkelijke omvang aangeven. De gecomprimeerde stukken worden nu steeds in twee delen opgeslagen: één of meer runs met de gecomprimeerde data en een fictieve run (die start met blok 0) om de totale omvang van de runs terug op 16 blokken te brengen. Bij het lezen van een byte uit een bestand moet eerst het gecomprimeerde stuk waartoe het behoort gedecomprimeerd worden, en pas dan kan de byte op de juiste plaats gelezen worden. Het is dus niet nodig om het volledige bestand te decomprimeren. De grootte van 16 blokken is een compromis. Met meer blokken kan de compressie efficiënter zijn, maar de kost om random te lezen wordt groter (omdat er grotere stukken moeten gedecomprimeerd worden).

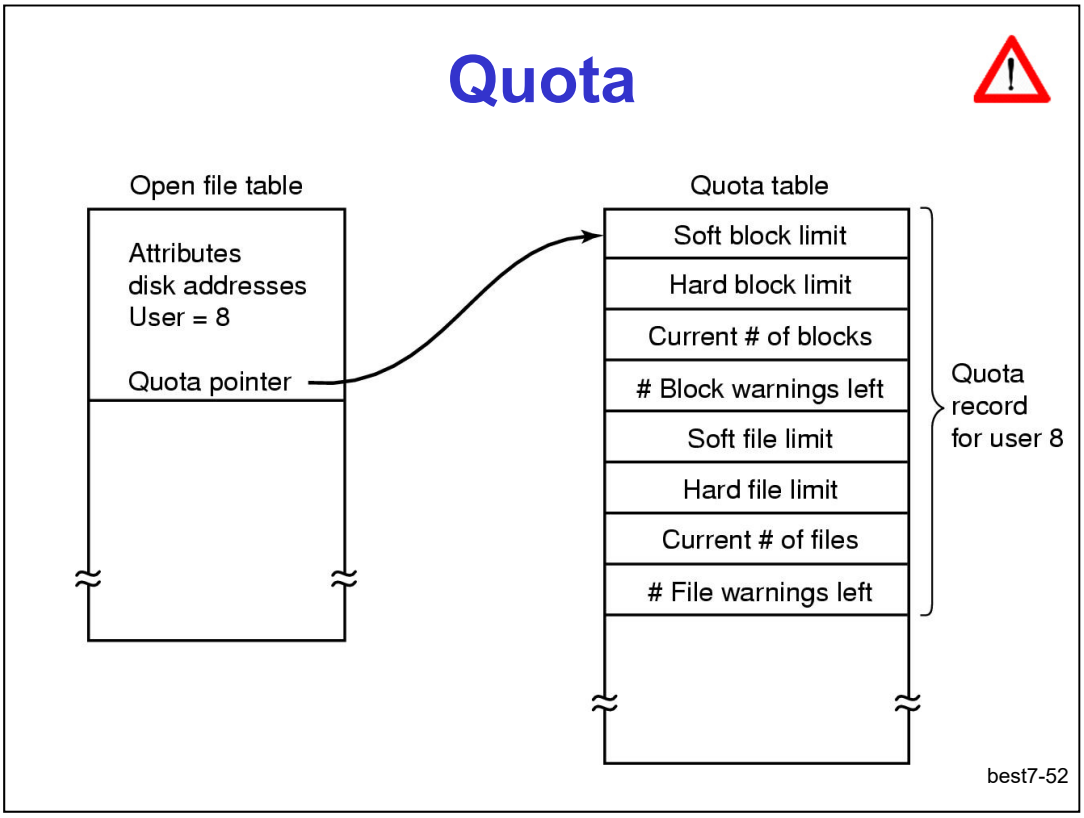
NTFS Streams

```
c:\>echo Hallo > test.txt
c:\>dir
15/11/2004  19:46     8 test.txt
c:\>more test.txt
Hallo
c:\>echo Nevenboodschap >test.txt:boodschap
c:\>dir
15/11/2004  19:46     8 test.txt
C:\>more test.txt
Hallo
C:\>more test.txt:boodschap
Nevenboodschap
```

best7-51

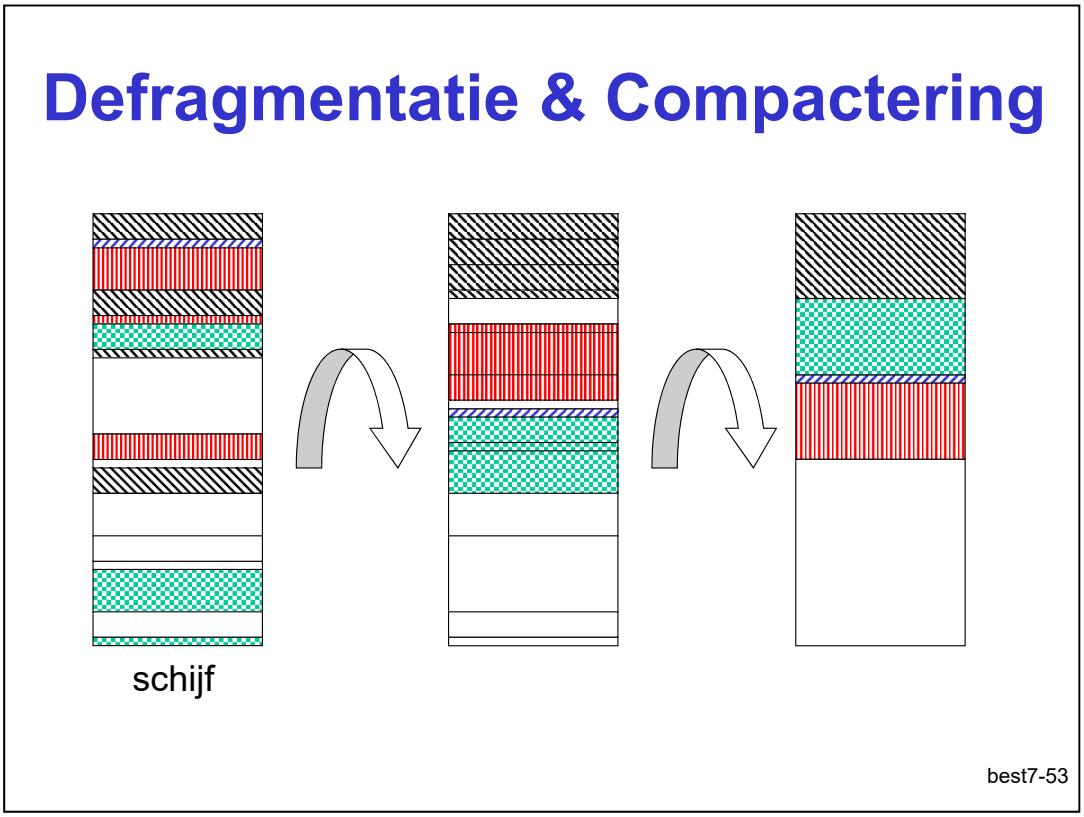
Aan een NTFS bestand kunnen naar willekeur bijkomende data streams toegevoegd worden. Dit werd oorspronkelijk zo beslist om compatibiliteitsreden met het HFS, het bestandssysteem van MacOS. Daar worden de bijkomende data streams b.v. gebruikt om iconen met programma's te associëren, enz. In de afbeelding wordt een bestand test.txt aangemaakt. De inhoud van het bestand wordt door NTFS opgeslagen als het regulier data attribuut. In dit geval is het voldoende klein om het in de MFT-record op te slaan.

Eenmaal het bestand aangemaakt is, wordt er een bijkomende data stream gecreëerd. Deze bijkomende data stream krijgt als naam 'boodschap'. Zoals blijkt uit de afbeelding wordt het bestand hierdoor niet groter (omdat de gerapporteerde grootte steeds op de reguliere data stream slaat), blijkt de inhoud van het bestand dezelfde, maar kan men de bijkomende data stream wel terug opvragen. Dit is een zeer krachtig mechanisme om bijkomende informatie met een bestand te associëren (zoals de auteur). Daarnaast is het ook een nachtmerrie voor de beveiliging van systemen. Malware kan zich immers onopvallend verbergen in een data stream.



Om te vermijden dat 1 gebruiker een onredelijk deel van de schijfruimte inneemt, kan men een quotasysteem implementeren. Men maakt een onderscheid tussen zachte en harde quota. Bij het overschrijden van het zachte maximum, krijgt men een waarschuwing bij het inloggen, bij het overschrijden van het harde maximum, zal het alloceren van bijkomende blokken falen.

Om toe te laten dat gebruikers tijdelijk meer plaats innemen, kan men het quotasysteem b.v. uitschakelen voor de tmp-directory.



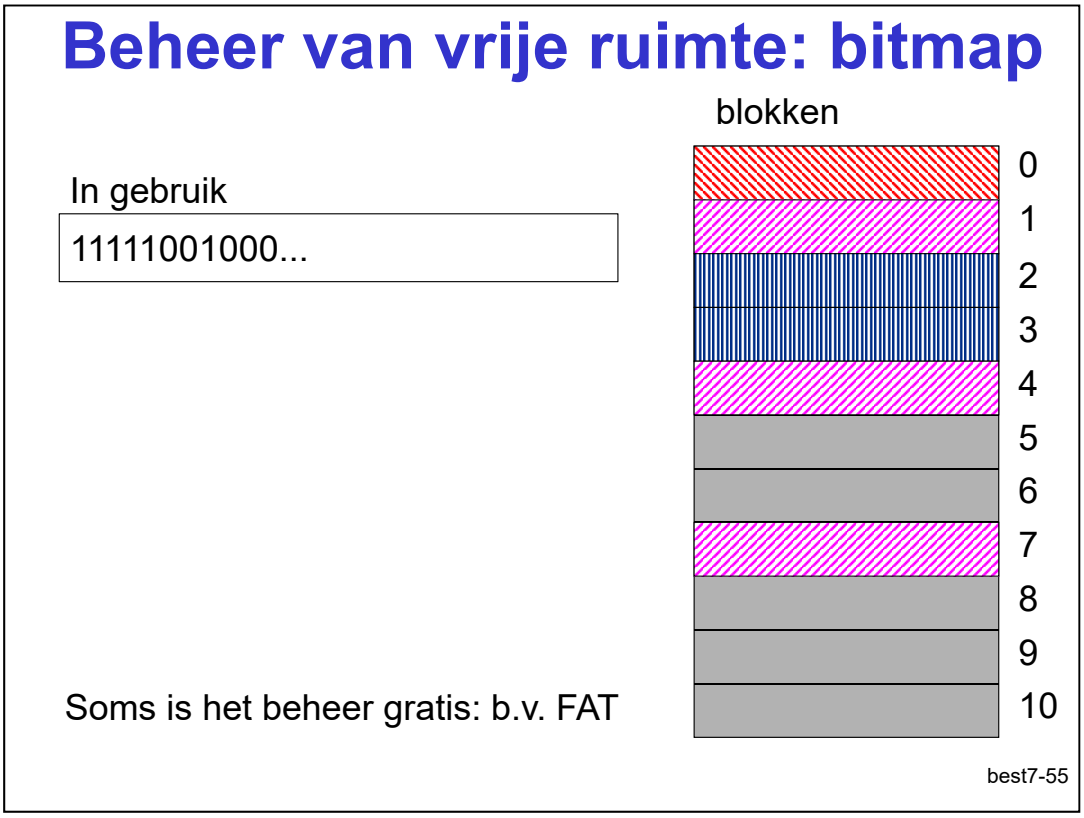
Welke allocatiemethode men ook gebruikt, na verloop van tijd zal de vrije ruimte geen aaneengesloten blok meer vormen, maar verspreid geraken over de volledige schijf. Het gevolg van deze versnippering zal zijn dat bestanden opgebouwd zullen worden uit reeksen van blokken die zich op verschillende plaatsen op de schijf bevinden (behalve bij contigue allocatie, maar daar wordt de vrije ruimte dan weer versnipperd). Doordat de kop van de schijf zich verschillende keren zal moeten verplaatsen om een dergelijk bestand te lezen, zal de prestatie van het bestandssysteem sterk dalen. Dit probleem wordt fragmentatie van de schijf genoemd.

Het probleem kan opgelost worden door de bestanden op de schijf te herschikken. Hierbij zullen alle bestanden eerst contigu gemaakt worden en wordt vervolgens ook de vrije ruimte contigu gemaakt. Deze techniek heeft zijn aanhangers en zijn tegenstanders. Aanhangers claimen dat de fragmentatie van de bestanden weggewerkt wordt hetgeen nuttig is voor de prestatie. Tegenstanders argumenteren dat door de compactie de volgorde van bestanden gewijzigd wordt, en dat dit een negatieve impact kan hebben op de prestatie. Indien men een applicatiepakket installeert op een nagenoeg lege schijf, is de kans groot dat alle bestanden van het pakket na elkaar op de schijf zullen komen te staan. Bij het opstarten van het pakket kunnen de bestanden dan één na één ingelezen worden, zonder dat de kop zich veel moet verplaatsen. Na compactering kunnen deze bestanden verspreid geraken over heel de schijf waardoor er tijd verloren wordt bij het springen van het ene bestand naar het andere.

Overzicht

- Logische bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruimte
  - Partities
  - Reservekopieën
- Optimalisaties

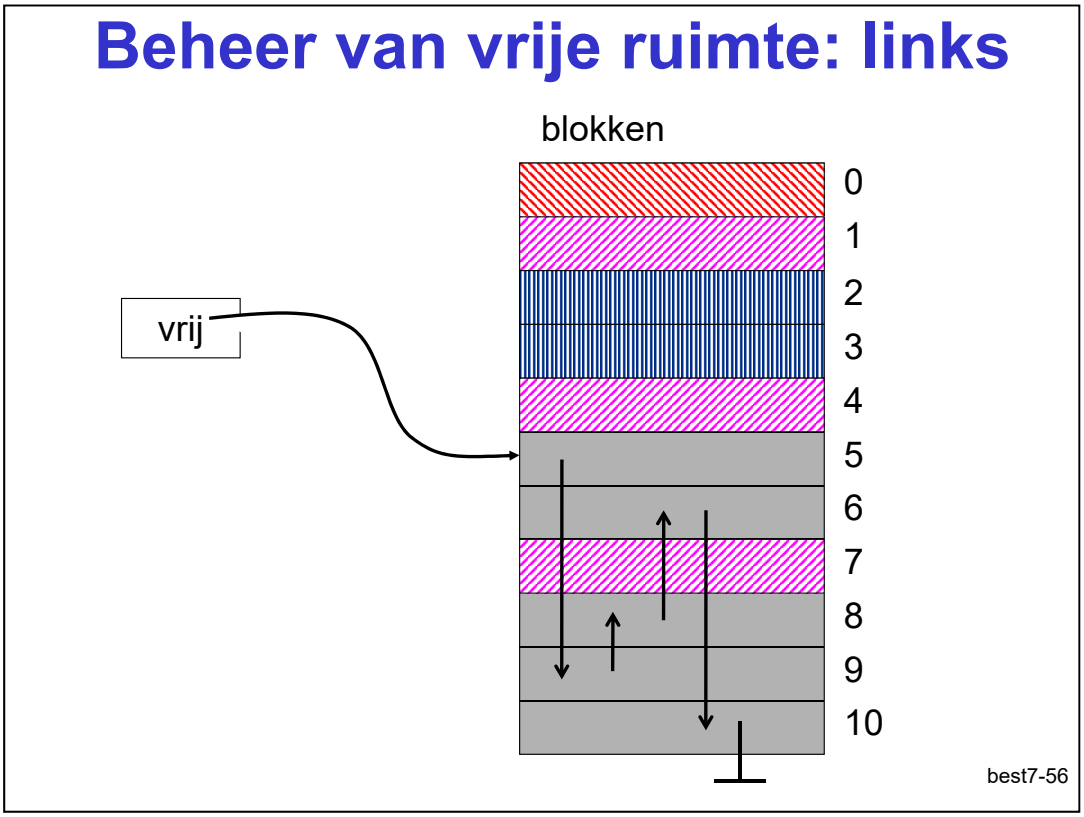
best7-54



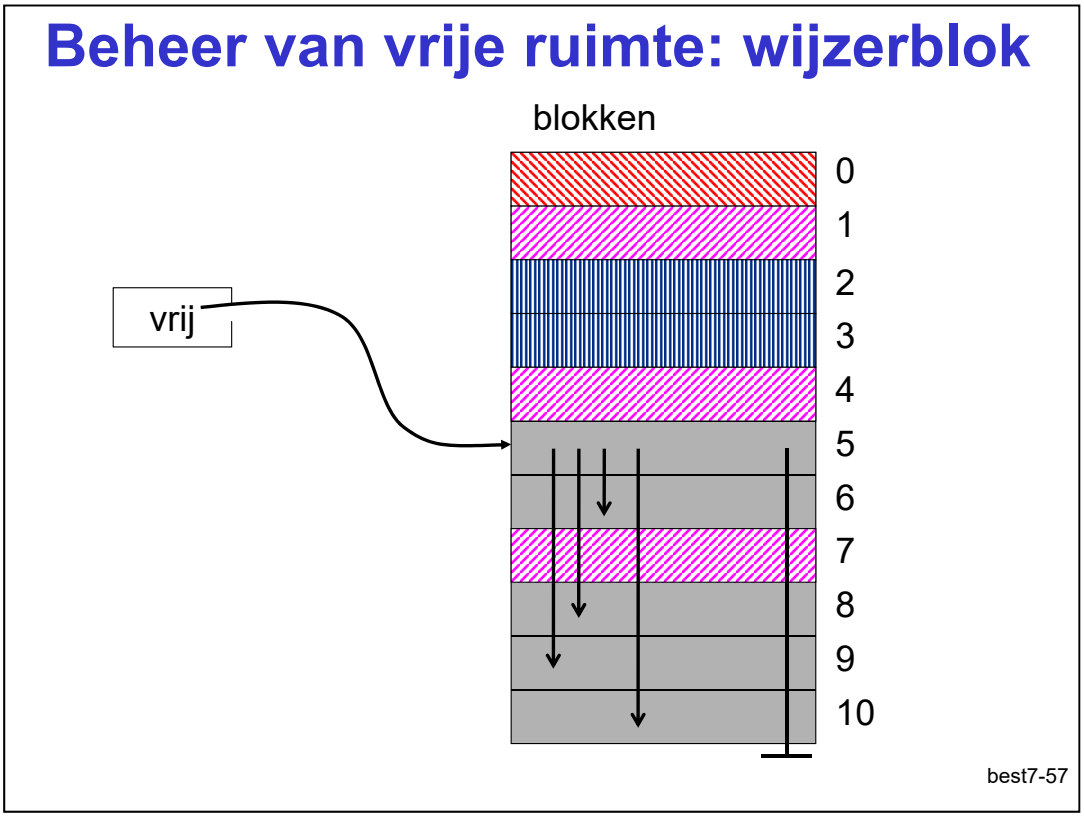
De vrije ruimte op een schijf moet op een efficiënte manier kunnen bijgehouden worden om snel een vrij blok te kunnen terugvinden. Een aantal gebruikelijke methoden wordt hier besproken.

Bitmap. Er wordt een bitrij bijgehouden waarbij elk bitje een blok voorstelt. Een vrij blok kan dan voorgesteld worden door 0, en een blok dat in gebruik is door een 1. Voor grote schijven kunnen de bitmaps een aanzienlijke ruimte in het geheugen innemen. Ze moeten regelmatig op de schijf bewaard worden als beveiliging. Na een systeemcrash zal deze lijst opnieuw moeten opgebouwd worden uitgaande van de bestanden die op de schijf teruggevonden worden.

Bij FAT herkent men de vrije blokken aan het feit dat ze niet gelinkt zijn met een ander blok. In dat geval is er geen extra ruimte nodig voor het beheer van de vrije ruimte.

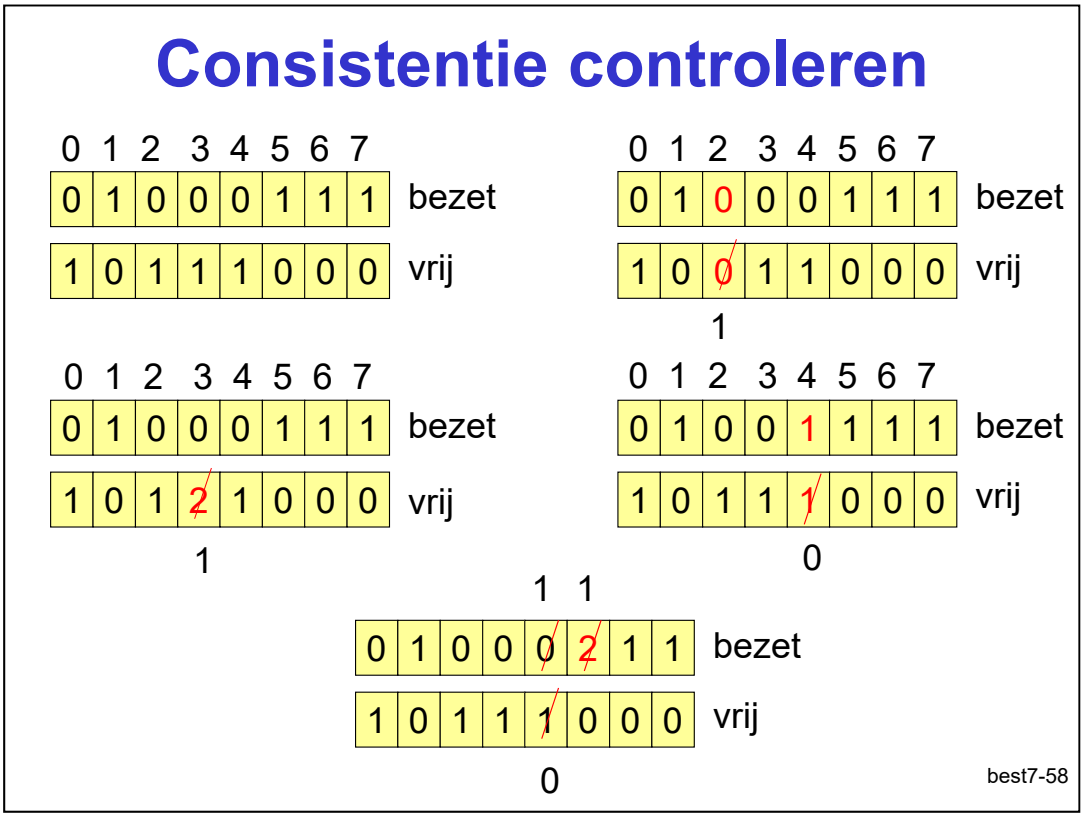


Gelinkte lijst. Gezien de vrije blokken toch niet gebruikt worden, kan elk blok gebruikt worden om ze met elkaar te linken. De kop van de lijst wijst steeds naar een vrij blok indien er een aanwezig is. Nadeel van deze methode is dat indien er bijvoorbeeld 100 blokken moeten gealloceerd worden, er ook 100 schijftoegangen nodig zijn om de gelinkte lijst af te lopen.



Wijzerblok. Net zoals bij de geïndexeerde allocatie kunnen de vrije blokken ook in indexblokken opgenomen worden. In dit geval zal men meestal kiezen voor gelinkte indexblokken i.p.v. voor verschillende niveaus. Het voordeel van deze methode is dat men verschillende blokken ineens kan alloceren als men i.p.v. individuele vrije blokken, sequenties van vrije blokken bijhoudt. Dit werkt de versnippering van de blokken tegen en is bovendien efficiënter.

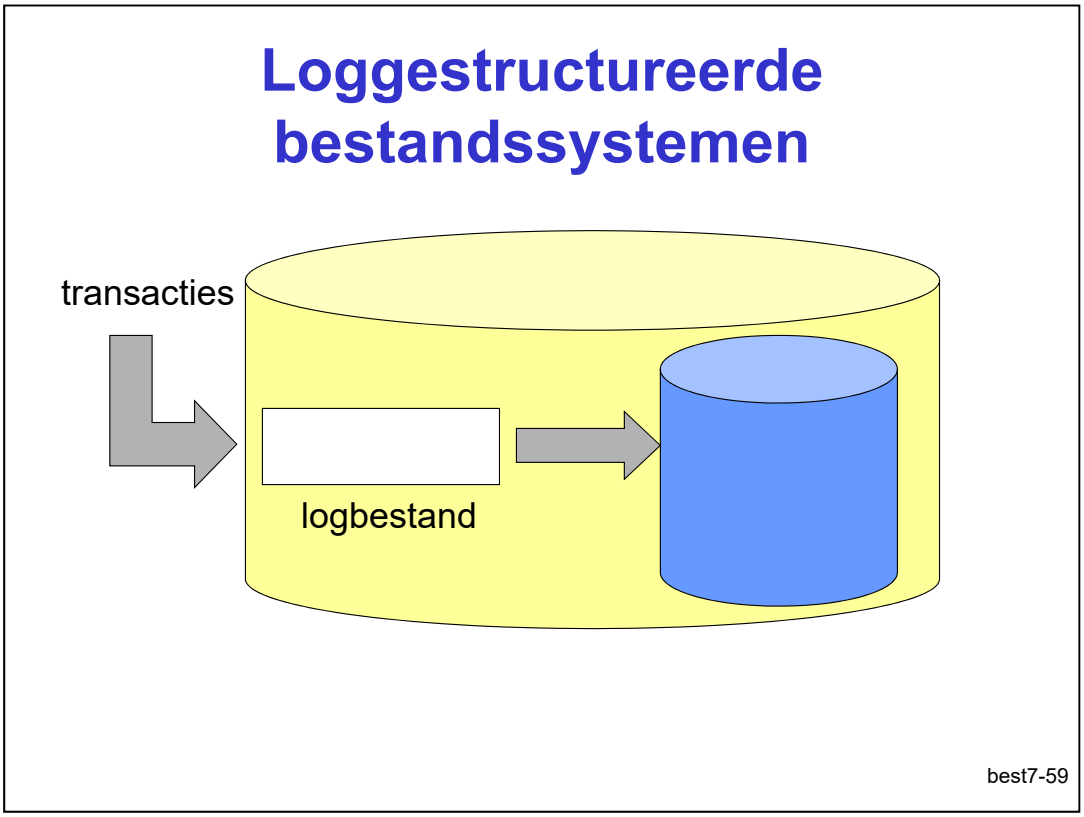
De allocatiemethode die in MS-DOS en OS/2 gebruikt wordt, is vrij van een aantal van deze problemen. Elk blok heeft slechts één cel in de FAT die ofwel in gebruik ofwel vrij is. Er is dus geen nood aan een afzonderlijk beheer van de vrije ruimte.



Merk op dat de lijst met vrije blokken in feite redundante informatie bevat. Het is dan ook belangrijk om te garanderen dat de informatie in de lijst met vrije blokken consistent is met de informatie die opgeslagen is in de directory's (resp. inodes). Hiervoor overloopt men alle directory's en de lijst met vrije blokken en telt men hoeveel keer de blokken voorkomen.

1. Indien bepaalde blokken noch bezet noch vrij zijn, dan neemt men ze op in de lijst met vrije blokken.
2. Als een blok 2 x voorkomt in de vrije lijst (kan niet voorkomen bij bitmaps), dan is de lijst met vrije blokken corrupt, en moet het vrije blok 1x verwijderd worden uit de lijst met vrije blokken. Zoniet lopen we het gevaar hetzelfde blok op te nemen in twee verschillende bestanden.
3. Als een blok zowel vrij als bezet is, is de lijst met vrije blokken ook corrupt, en moet het blok uit de lijst met vrije blokken gehaald worden.
4. Als een blok in 2 bestanden voorkomt, is er ook een ernstig probleem. In dat geval is het best om het blok te dupliceren en elk bestand zijn eigen versie te geven. Merk op dat deze situatie niet het gevolg kan zijn van een bestand dat in 2 directory's voorkomt in Unix. Dan is er maar 1 inode, en dus maar 1 bestand.

Ofschoon men in FAT niet voor afzonderlijk beheer van vrije ruimte hoeft te zorgen, kunnen er zich wel problemen voordoen. In de praktijk komt het voor dat een bestand gewist wordt uit de directory, maar dat de blokken in de FAT nog niet vrijgegeven werden. De programma's chkdsk en scandisk zullen deze gevallen opsporen en corrigeren. Uiteraard kan het als gevolg van schijffouten voorkomen dat bepaalde bestanden verkeerde wijzers bevatten. Ook dan moet de volledige directorystructuur overlopen worden en alles wat niet tot de directorystructuur behoort moet vrijgemaakt worden.



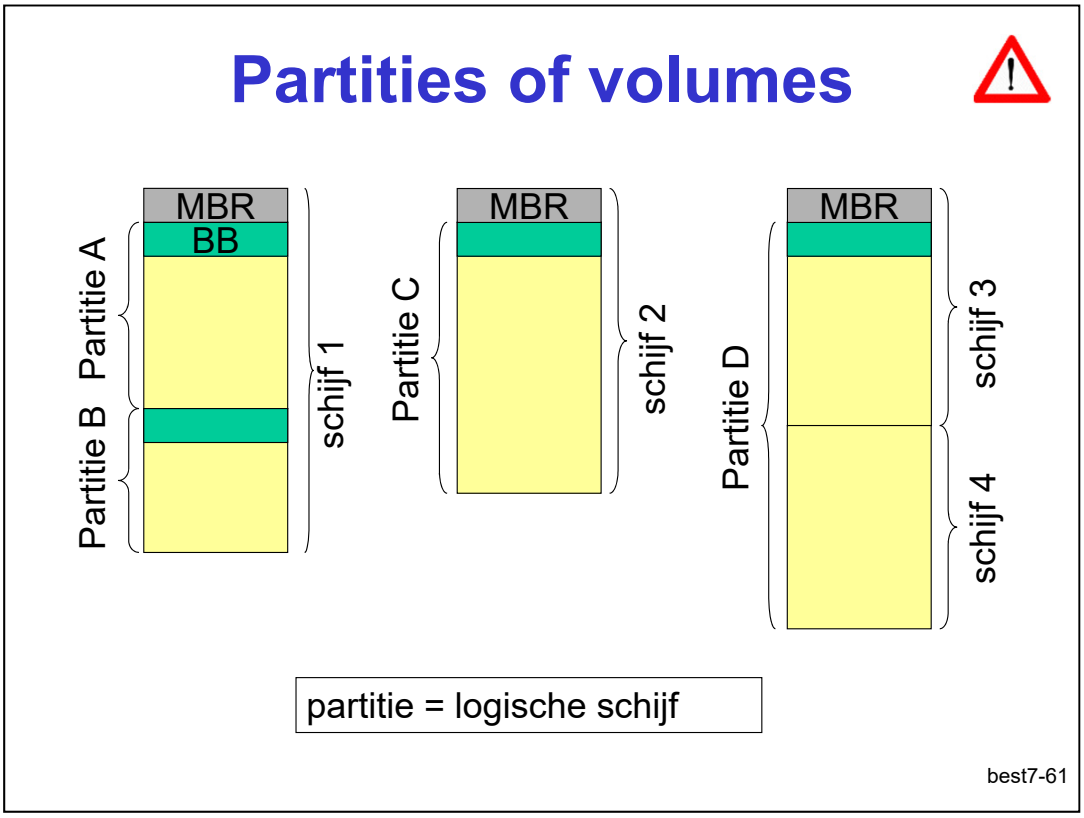
Loggestructureerde bestandssystemen (Log structured file systems of journaling file systems) proberen inconsistenties in het bestandssysteem te vermijden door alle aanpassingen in het bestandssysteem als (atomaire) transactie door te voeren.

Daartoe worden de acties als transacties naar een logbestand geschreven. De transactie is pas definitief nadat deze in het logbestand opgenomen werd. Op dat ogenblik hoeft het bestandssysteem nog niet aangepast te zijn. De transacties worden asynchroon uitgevoerd op de schijf. Pas nadat het bestandssysteem aangepast is, wordt de transactie verwijderd uit het logbestand. Indien een bestandssysteem om de één of andere reden afgebroken wordt, zal een consistente toestand gemakkelijk gereconstrueerd kunnen worden. De transacties in het logbestand kunnen alsnog uitgevoerd worden en er zullen geen transacties verloren gaan.

Overzicht

- Logische bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruimte
  - Partities
  - Reservekopieën
- Optimalisaties

best7-60

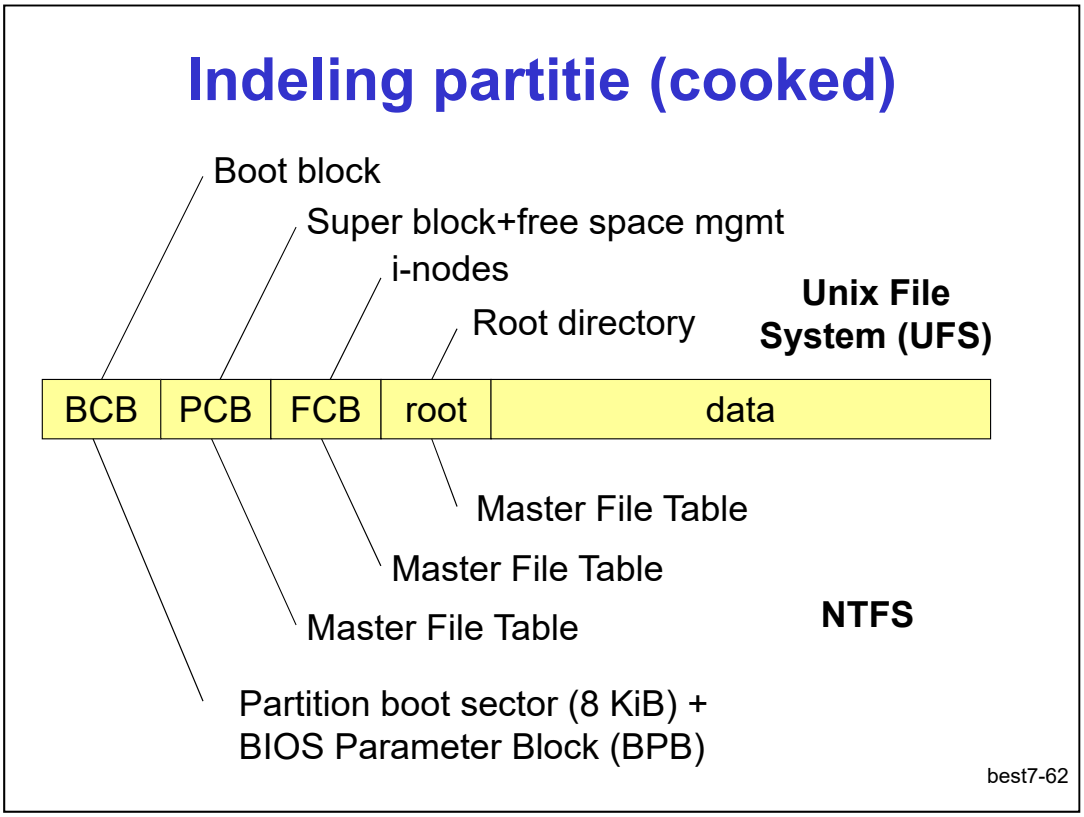


Een computersysteem kan over één of meer fysieke schijven beschikken. Het besturingssysteem zal echter nooit rechtstreeks gebruik maken van deze fysieke schijven, maar wel van logische schijven, ook partities of volumes genoemd. Door de ontkoppeling van een logische en een fysieke schijf ontstaan er heel wat combinatiemogelijkheden.

- Een fysieke schijf kan b.v. ingedeeld worden in verschillende logische schrijven. Onder windows heeft men al eens de gewoonte om de software op de partitie C: te zetten en de gegevensbestanden op partitie D:. De partities C: en D: zullen dan deel uitmaken van dezelfde fysieke schijf. Het feit dat alle gegevens op de D:-partitie staan maakt het maken van een reservekopie b.v. een stuk eenvoudiger. Indien men op één fysieke schijf verschillende logische partities installeert, kan men b.v. per partitie een ander besturingssysteem installeren. Door bij het opstarten de partitie te specificeren kan men de computer b.v. in Windows of in Linux laten starten.
- Een partitie kan uiteraard volledig samenvallen met een fysieke schijf. In dat geval is er geen onderscheid tussen de twee.
- Tenslotte is het ook mogelijk om verschillende fysieke schijven samen te voegen tot 1 **logisch volume**. Hierbij worden alle blokken van een aantal schijven gewoon samengevoegd tot een virtuele partitie en wordt er een bestandssysteem geïnstalleerd in de virtuele partitie. Het defect gaan van 1 partitie heeft dan uiteraard belangrijke consequenties. Voordelen van logische volumes zijn: eenvoudiger beheersbaar (slechts 1 bestandssysteem), sneller (door de parallelle toegang tot de verschillende schijven) en de mogelijkheid om bestanden op te slaan die groter zijn dan 1 partitie. Raid systemen (zie verder) zijn een belangrijk voorbeeld van deze techniek.

Het begin van de fysieke schijf begint steeds met een zgn. Master Boot Record, dit is het eerste blok van de schijf met daarin informatie over de partities van de schijf (de partitietabel), en welke partitie

de actieve partitie is – m.a.w. vanaf welke partitie de computer moet opgestart worden.



Een 'cooked' partitie bestaat uit de volgende onderdelen:

BCB (Boot Control Block), dit zijn de eerste sectoren (b.v. 8 KiB) met code en data die nodig zijn om te achterhalen welk bestandssysteem er gebruikt wordt in de partitie en om het besturingssysteem binnen de partitie te vinden en het inladen ervan op te starten.

PCB (Partition Control Block), dit bevat informatie over de toestand van het bestandssysteem: het aantal vrije blokken, het aantal vrije FCB's, enz. In UFS zit deze informatie in het super block en in het free space management block, in NTFS wordt deze informatie bijgehouden in een aantal van de 16 systeembestanden van NTFS die opgeslagen liggen in de MFT.

FCB (File Control Block), dit zijn bestandscontroleblokken waarin de meta-informatie per bestand bijgehouden wordt. In UFS is dit de inodetabel, in NTFS zijn dit de records van de MFT.

Root, dit is het startpunt van de directorystructuur van de partitie. In UFS is dit de root directory waarvan het begin een vaste plaats heeft in de layout van de partitie. In NTFS bevindt de root directory zich in element 5 van de MFT.

De **datablokken** worden gebruikt om de gegevens van de bestanden in op te slaan.

Een 'raw' partitie heeft geen dergelijke indeling. De indeling zal door een applicatieprogramma zelf aangebracht worden (b.v. Databanken, swap-ruimte, ...).

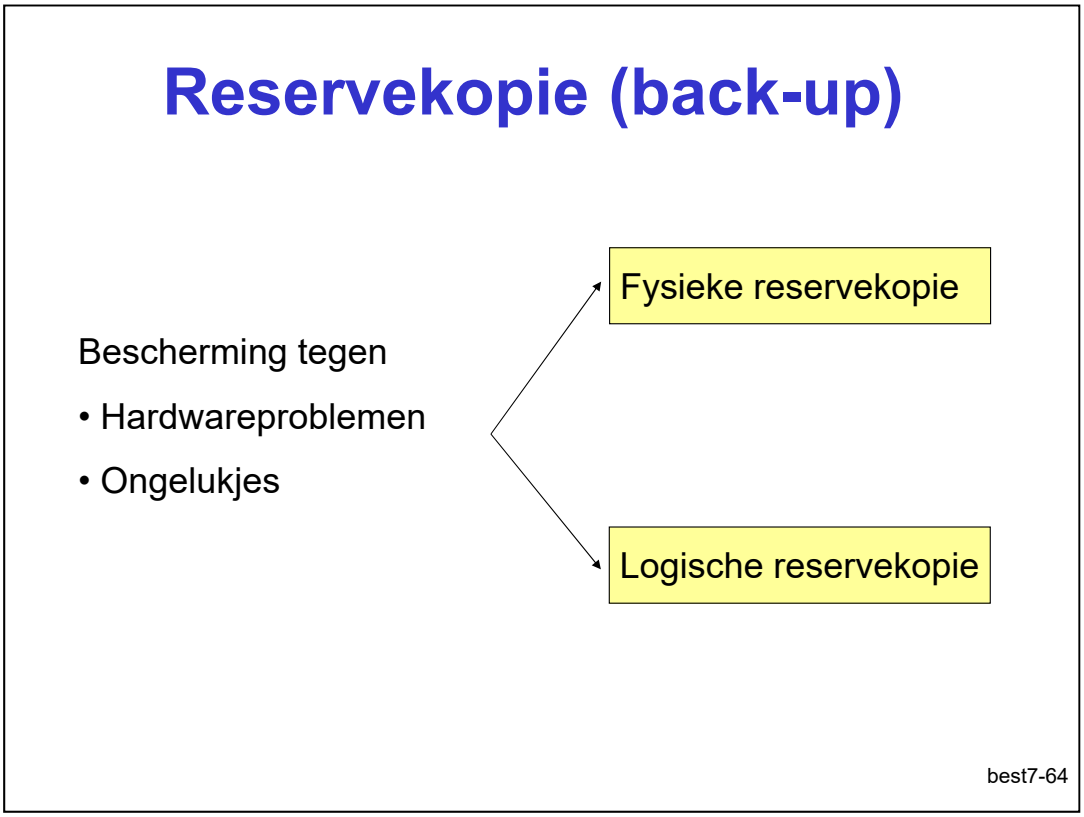
Bij het FAT bestandssysteem heeft men een boot sector, gevolgd door 2 kopieën van de

FAT, en finaal de root directory. Door zijn eenvoud wordt FAT nog heel veel gebruik in kleine systemen zonder veel beveiligingseisen (b.v. USB-sleutels).

Overzicht

- Logische bestandssysteem
  - Gegevensbestanden
  - directory's
  - Bestandssystemen
- Bestandsorganisatie
  - directory's
  - Gegevensbestanden
  - Vrije ruimte
  - Partities
  - Reservekopieën
- Optimalisaties

best7-63

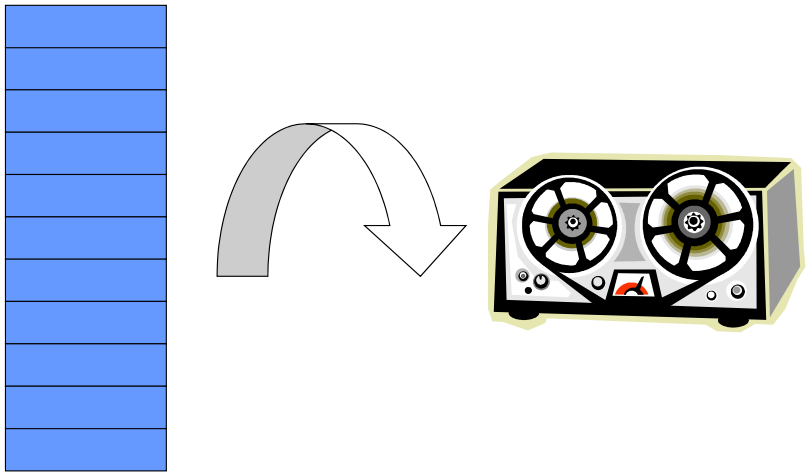


Een goed beheer van schijfgeheugen vereist ook het maken van regelmatige reservekopieën (back-up) van de inhoud van de schijf. Dit is nodig om twee redenen: vooreerst is een schijf inherent onbetrouwbaar en kan er zich op elk ogenblik een probleem voordoen waardoor bestanden beschadigd worden, en ten tweede, en minstens even belangrijk kunnen er fouten gemaakt worden door programma's of gebruikers waardoor er gegevens verloren gaan. De reservekopie kan dan gebruikt worden om de gegevens in hun oorspronkelijke staat te herstellen.

Door dagelijks kopieën te nemen kan men garanderen dat ten hoogste het werk van één dag verloren kan gaan. Er zijn twee soorten van kopieën: volledige kopieën en incrementele kopieën. Een volledige kopie bevat alle bestanden van de schijf, een incrementele kopie bevat enkel die bestanden die sinds de laatste kopie gewijzigd zijn geweest. Normaal gezien zal men op regelmatige tijdstippen (bv. wekelijks) een volledige kopie nemen, en dan dagelijks een incrementele kopie. In dit geval zullen er dus 7 kopieën zijn. Om te vermijden dat er problemen zouden ontstaan met foutief opgenomen kopieën zal men gebruikte banden niet meteen overschrijven, maar eerst een paar weken bewaren. Na een maand kunnen de oudste gegevens dan b.v. opnieuw overschreven worden.

Technisch gezien zijn er twee soorten van kopieën die men kan nemen: fysieke kopieën waarbij elk blok van de schijf gekopieerd wordt, en logische kopieën waarbij enkel de gebruikte bestanden en directory's gekopieerd worden.

Fysieke reservekopie

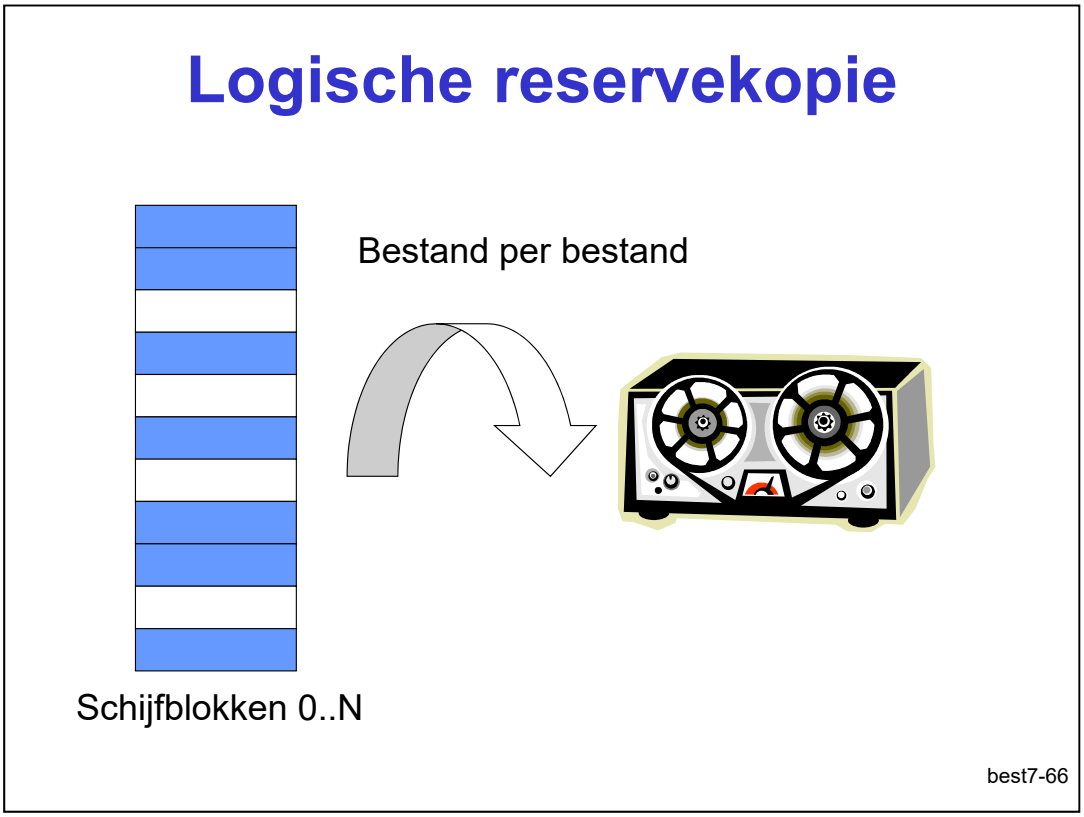


best7-65

Bij een fysieke reservekopie beschouwt men een schijf als een opeenvolging van sectoren – zonder zich te bekommeren om de inhoud van die sectoren. Deze sectoren kunnen efficiënt naar de tape gekopieerd worden. Men kan cilinder na cilinder kopiëren waardoor het aantal verplaatsingen van de schijfkop minimaal is. Bovendien hoeft men zich geen zorgen te maken over het type van bestandssysteem of over de inhoud van de sectoren. Op het laagste niveau wordt alles vertaald in sectoren.

Deze manier van werken heeft echter ook nadelen:

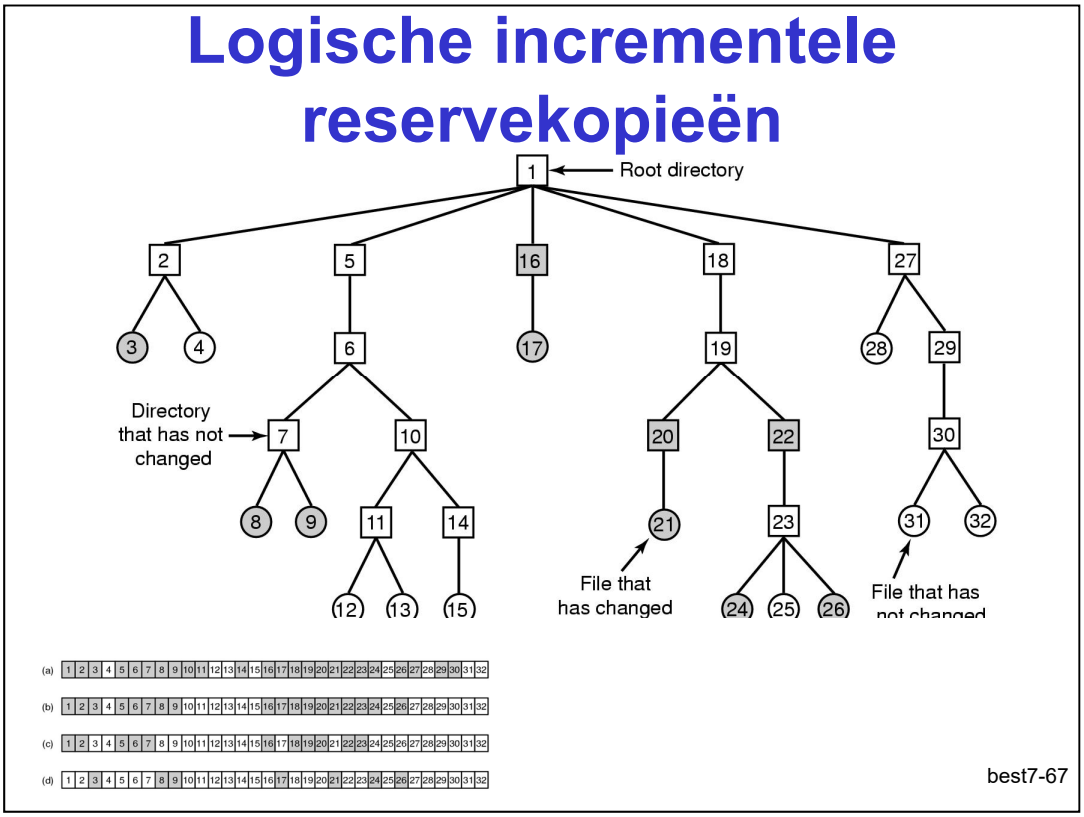
1. Het herstellen van 1 bestand is niet eenvoudig. Men moet eigenlijk eerst de schijf herstellen om dan het bestand te kunnen zoeken.
2. Het nemen van incrementele reservekopieën is met deze methode niet mogelijk.
3. Men moet steeds een complete partitie kopiëren. Het overslaan van bepaalde directory's (b.v. een web-cache) is niet mogelijk.



Een logische reservekopie neemt een kopie van het bestandssysteem, d.w.z. directory per directory en bestand per bestand. Doordat de bestanden in een directorystructuur verspreid kunnen liggen over de gehele schijf kan dit proces zeer lang duren.

Om de tijd nodig om een kopie te nemen te beperken neemt men in de praktijk incrementele reservekopieën waarbij enkel die bestanden die gewijzigd zijn sinds de vorige volledige reservekopie, of sinds de vorige incrementele reservekopie gekopieerd worden.

Uiteraard is het bij een logische reservekopie mogelijk de kopie te beperken tot een gedeelte van de directorystructuur, bepaalde stukken over te slaan, enz.



Bij een logische reservekopie moet men erop letten om niet alleen de gewijzigde gegevensbestanden, maar ook de directory's waartoe ze behoren te bewaren. Dit is nodig om een gegevensbestand te kunnen herstellen waarvan de directory waartoe het behoorde inmiddels weggelaten werd. De directory moet in dat geval ook gereconstrueerd worden (met inbegrip van al zijn attributen).

Het algoritme dat in Unix gebruikt wordt, werkt als volgt. Eerst worden alle inodes gemarkeerd die directory's bevatten, of die gewijzigde gegevensbestanden bevatten. In tweede instantie worden alle directory's weggelaten die geen gewijzigde gegevensbestanden bevatten (b.v. 10). Vervolgens worden de inodes van de directory's en van de gegevensbestanden gescheiden. De directory's worden eerst op de tape geschreven en nadien de gegevensbestanden. Op die manier zullen bij het terugplaatsen van een gegevensbestand eerst de vereiste directory's gecreëerd worden.

Verder zijn er nog een aantal speciale gevallen.

1. Bestanden die gelinkt zijn in twee of meer directory's mogen slechts 1x gekopieerd worden, en de links moeten bij het terugplaatsen van een bestand opnieuw gecreëerd worden.
2. Ijle bestanden moeten als ijle bestanden bewaard en hersteld kunnen worden. Dit impliceert dat de reservekopiesoftware toegang moet hebben tot de organisatie van de bestanden. Het gewoon kunnen lezen van een bestand vertelt immers niets over het al dan niet ijl zijn van het bestand.

Overzicht

- Logische bestandssysteem
- Bestandsorganisatie
- Optimalisaties
  - Vergrendelen bestanden
  - Variabele blokgrootte
  - Beperkte dynamische informatie
  - RAM-disk
  - Virtuele schijf
  - Disk caches
  - Free-behind - Read-ahead
  - Compressie

best7-68

De basisfunctionaliteit van een bestandssysteem kan verbeterd worden door het toepassen van een aantal optimalisaties.

Vergrendelen bestanden

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static void main(String arsg[]) throws IOException {
        RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
        // get the channel for the file
        FileChannel ch = raf.getChannel();
        // this locks the first half of the file - exclusive
                FileLock exclusiveLock = ch.lock(0, raf.length()/2, false);
                /** Now modify the data . . . */
                exclusiveLock.release();
        // this locks the second half of the file - shared
                FileLock sharedLock = ch.lock(raf.length()/2+1, raf.length(), true);
                /** Now read the data . . . */
                sharedLock.release();
    }  catch (java.io.IOException ioe)  { } finally { }
}
```

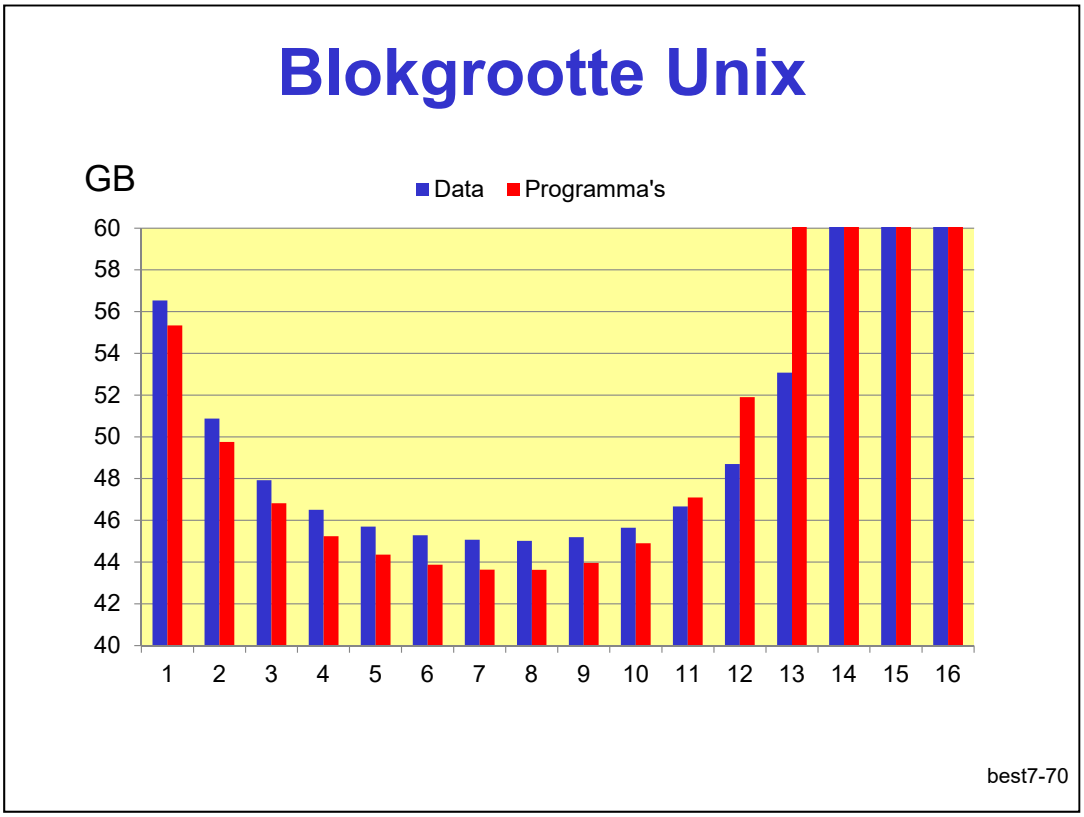
Mandatory locks: hard afgedwongen

Advisory locks: kunnen genegeerd worden

best7-69

Een bestandssysteem kan zorgen voor exclusieve toegang tot bestanden of tot delen ervan. Mandatory locks worden door het bestandssysteem afgedwongen, dit wil zeggen dat geen enkele applicatie in staat is om toegang te krijgen tot vergrendelde bestanden. Advisory locks daarentegen kunnen wel genegeerd worden.

Als men om de lock vraagt, dan blokeeert deze totdat de vraag kan gehonoreerd worden. Verschillende vragen naar een shared lock kunnnen simultaan gehonoreerd worden, van zodra er een exclusieve lock gehonoreerd werd, moeten alle andere aanvragen wachten. Dit is vergelijkbaar met lees-schrijf-synchronisatie.



Bestanden worden niet gealloceerd in sectoren, maar in blokken of clusters. Een dergelijk blok zal minstens één sector moeten bevatten (sectoren kunnen variëren tussen 256 bytes en 1024 bytes). Vaak zal een blok 4 tot 8 KiB zijn terwijl een sector typisch 512 bytes is. Hoe groter de blokken, des te efficiënter de gegevensoverdracht, maar ook des te groter de interne fragmentatie gezien een blok de kleinste hoeveelheid schijfruimte is die gealloceerd kan worden. De blokgrootte kan bij de generatie van het bestandensysteem gekozen worden en zal afhangen van de kenmerken van de bestanden op de schijf. Indien men louter met zeer grote bestanden werkt, kan men de voorkeur geven aan vrij grote blokken. Indien de meerderheid van de bestanden klein is (paar KiB), dan kan het interessanter zijn om een kleinere blokgrootte te kiezen. Eenmaal men een blokgrootte gekozen heeft is de kleinste hoeveelheid informatie die met de schijf kan uitgewisseld worden het blok. Voor PCs vindt men als blokgrootte voor harde schijven bv. 16 sectoren (8 KiB), en voor zeer oude floppy's 1 sector (512 bytes).

Voor de databestanden en codebestanden van een typische Windows machine geeft dit het bovenstaande beeld. Zoals blijkt valt het met de fragmentatie wel mee tot een blokgrootte van 8 KiB. Daarboven neemt de fragmentatie sterk toe. Voor de kleine blokken neemt de ruimte toe omdat er een toenemend aantal indexblokken nodig is.

Anderzijds heeft een grotere blokgrootte een positieve invloed op de transfersnelheid van de bestanden. Het inlezen van een klein of een groot blok neemt in de praktijk ongeveer evenveel tijd in beslag.

Variabele blokgrootte

- Kleine blokken voor kleine bestanden
- Grote blokken voor grote bestanden

best7-71

Grote blokken hebben als voordeel dat er minder wijzers nodig zijn om een bestand aan te maken. Anderzijds hebben ze als nadeel dat er heel wat schijfruimte ongebruikt kan blijven. Een blokgrootte van 32 KiB kan een enorme impact hebben op een bestandssysteem waar de meeste bestanden b.v. kleiner zijn dan 8 KiB.

In BSD Unix kan men 2 soorten blokgroottes gebruiken: de standaard grootte van b.v. 8 KiB, en een kleinere blokgrootte van b.v. 1 KiB. Een bestand zonder indirecte blokken kan ofwel uitsluitend kleine blokken gebruiken ofwel enkel de laatste wijzer laten wijzen naar een klein blok. Dit laat toe om de interne fragmentatie bij kleine bestandsgroottes weg te werken.

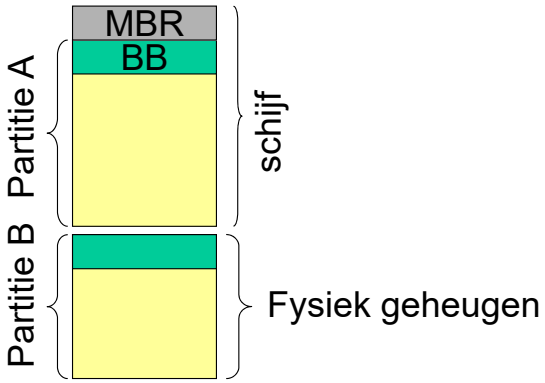
Beperkte dynamische informatie

- Voornamelijk toegangstijdstippen

best7-72

Het bijhouden van het tijdstip waarop een bestand voor het laatst gebruikt werd zal een schrijfoperatie veroorzaken bij elke toegang tot het bestand (ook indien er alleen maar gelezen wordt). Indien het niet strikt nodig is om de leesoperaties op die manier bij te houden, is het beter om dit in het belang van de prestatie niet te doen.

RAM-disk / tempfs



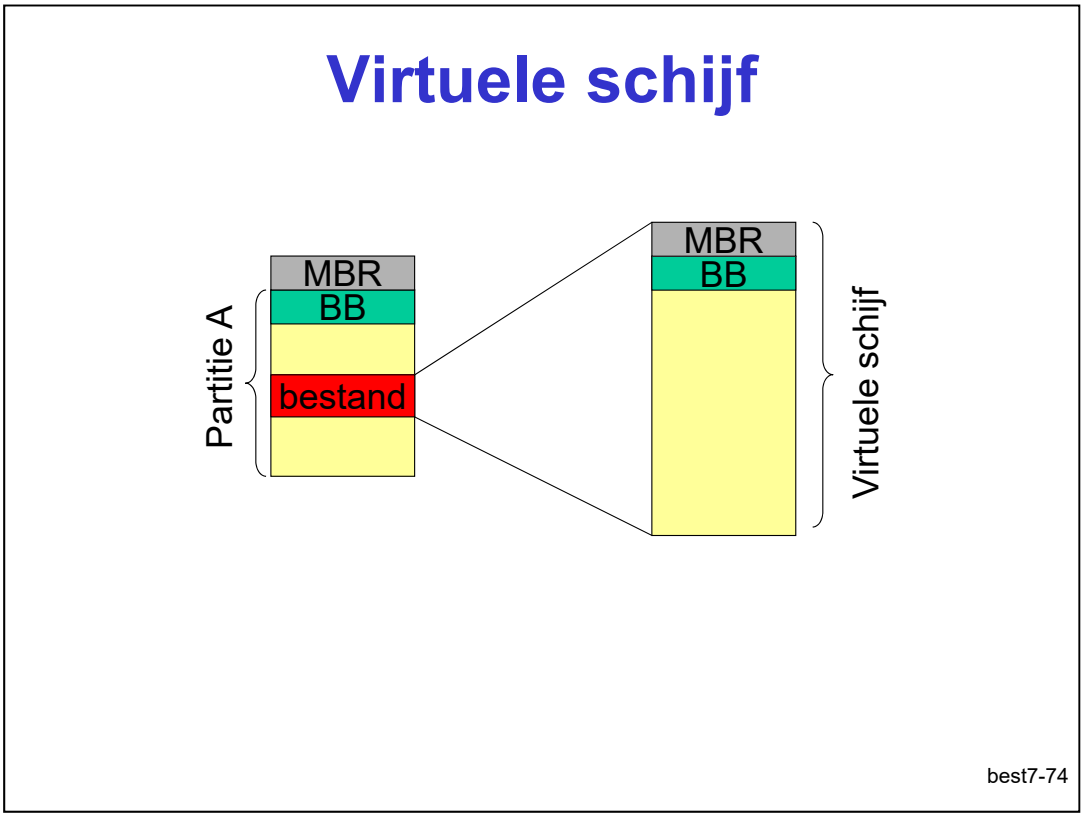
best7-73

Een bestandssysteem wordt doorgaans geassocieerd met een magneetschijf. Het kan echter geïnstalleerd worden in alle types van geheugen: CD-ROM, tapes, USB-sleutels, en ook in het RAMgeheugen.

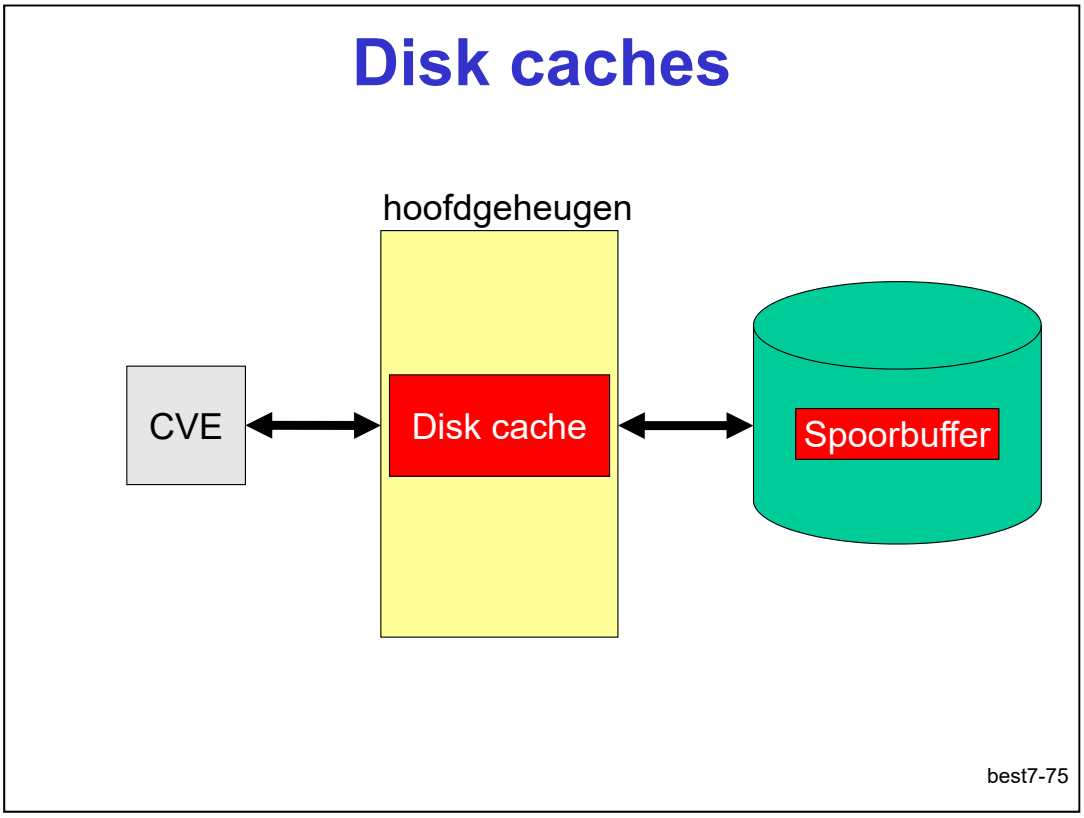
Om een bestandssysteem te installeren in het geheugen (een zgn. RAM-disk) volstaat het het geheugen te initialiseren als een partitie en het fysieke bestandssysteem de aanvragen naar de betrokken partitie te laten beantwoorden door het geheugen.

Een RAM-disk is hierdoor uiteraard vele malen sneller dan een fysieke schijf. Anderzijds wordt het effect van een RAM-disk wel grotendeels tenietgedaan door de uitgebreide disk caches die men in hedendaagse systemen vindt.

Een nadeel van de RAM-disk is dat de gegevens niet persistent zijn en dat men dus niet mag vergeten om zijn gegevens terug op de fysieke schijf te schrijven. Voor het bijhouden van werkbestanden is het echter een prima oplossing.



Een bestandssysteem kan ook aangebracht worden in een gewoon databestand. Op die manier slaat men dan alle informatie over een volledig bestandssysteem op in één (groot) bestand. Virtuele machines maken meestal gebruik van bestanden ipv afzonderlijke partities om hun informatie op te slaan. Het maken van een kopie van het bestand staat dan gelijk met het maken van een backup van de virtuele schijf.



Toegang tot de schijf is inherent traag. Om de traagheid van de schijf wat te verbergen kan men schijfblokken in een cache opnemen. Dit kan op verschillende plaatsen gebeuren. In de schijfregelaar zelf zal men doorgaans een track buffer bijhouden waarin een volledig spoor van de schijf gelezen wordt. Uit deze buffer worden dan blokken naar het geheugen gestuurd, zonder hiervoor de schijfkop zelf nog te moeten aansturen (zie volgende les).

In het geheugen zelf kan men de blokken bewaren in een disk cache (block buffer). Door zoveel mogelijk blokken van de schijf in het geheugen te bewaren kan men vermijden dat ze meer dan eens moeten gelezen worden. Solaris en Linux stellen al het ongebruikte RAMgeheugen ter beschikking als disk cache (zowel voor bestands-IO als voor paginering).

Free-behind

- Bij sequentieel lezen is het beter om een gelezen blok vrij te geven bij het inlezen van het volgende blok.

Read-ahead

- Bij sequentieel lezen een aantal extra sequentiële blokken inlezen.

best7-76

Free-behind --- Read-ahead zijn twee methodes die naast LRU gebruikt kunnen worden om de disk cache beter te beheren bij sequentiële toegang.

Free-behind zal een blok vrijgeven van zodra een volgend blok aangevraagd wordt, en read-ahead zal ervoor zorgen dat er steeds enkele blokken extra ingelezen worden in een disk cache om de wachttijden bij het effectief opvragen van gegevens uit dat blok te beperken.

Dit werkt enkel bij sequentieel lezen, en vervuilt de cache bij random access. Het besturingssysteem kan sequentieel gedrag wel vrij gemakkelijk onderscheiden van directe toegang door het aantal seek() en read()/write() systeemoproepen te vergelijken.

Schijfcompressie

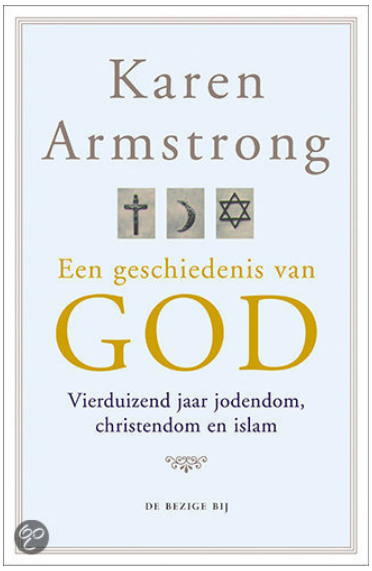
- Verhoogt de capaciteit van de schijf
- Verhoogt de effectieve bandbreedte naar de schijf

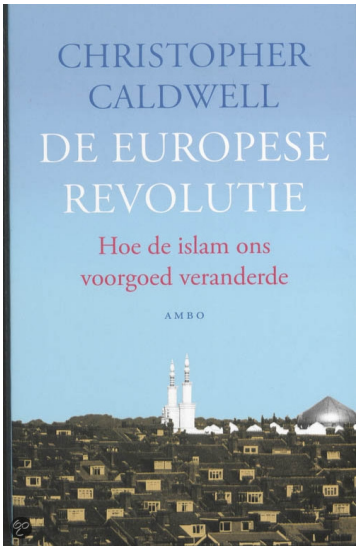
best7-77

Om de schijfcapaciteit beter te benutten kan men de gegevens gecomprimeerd op de schijf plaatsen. Afhankelijk van het soort van gegevens dat men wil opslaan kan de hoeveelheid die men op deze manier kan opslaan een veelvoud zijn van de originele schijfcapaciteit. Bovendien kan de bandbreedte naar de schijf hierdoor in sommige gevallen vergroot worden omdat er minder gegevensuitwisseling zal zijn met de schijf terwijl het comprimeren en decomprimeren vrij snel kan gaan. Deze compressie is totaal transparant voor de gebruiker.

Het nadeel van compressie is dat de reconstructie van de bestanden bij een ernstige fout in het bestandssysteem zeer moeilijk is. Zelfs bij het defect gaan van 1 blok kunnen er verschillende bestanden aangetast worden.

NTFS ondersteunt compressie.


Karen
Armstrong
Een geschiedenis van
GOD
Vierduizend jaar jodendom,
christendom en islam
DE BEZIGE BIJ


CHRISTOPHER
CALDWELL
DE EUROPESE
REVOLUTIE
Hoe de islam ons
voorgoed veranderde
AMBO

best7-78