2008 samenvatting David Machiels

Inleiding

Het besturingssysteem is de softwarelaag tussen de toepassingssoftware en de uiteindelijke uitvoering van de machine-instructies.

Situering

Voor de gebruiker van een computersysteem is de toepassingslaag het belangrijkste onderdeel. Het computersysteem wordt slechts gebruikt omdat het deze software kan uitvoeren. Nochtans is de toepassingssoftware slechts een klein onderdeel van het volledige computersysteem.

Hardware

De hardware zorgt voor de uitvoering van bevelen, die door software d.m.v. machine-instructies worden gegeven. Deze instructies worden meestal niet rechtstreeks uitgevoerd door fysische onderdelen maar via de microprogrammatie of microcode: een verzameling van kleine programma’s in machinetaal die door de processoren begrepen worden en toelaten dat relatief ingewikkelde operaties via één machine-instructie kunnen worden uitgevoerd. Microcode is een voorbeeld van wat men firmware noemt: in ROM opgeslagen software. Microcode heef niet één specifieke taak, maar dient om het schrijven van programma’s in machinetaal te vereenvoudigen door vaak voorkomende taken aan te bieden in één oproep.

Evolutie

Zonder besturingssysteem

Programma’s werden in machinetaal geschreven, en de computer kon maar 1 programma inladen en dan uitvoeren. Pas als een programma afgelopen was kon het volgende geladen worden. Na een tijd stelde men vast dat de geschreven programma’s veel gelijkaardige routines bevatten, bijvoorbeeld voor in- en uitvoer. Hierdoor begonnen fabrikanten van computersystemen

bibliotheken met vaak gebruikte routines bij hun producten te leveren. Er moest een planning worden opgesteld om te bepalen wie de computer mocht gebruiken. Als een programma vroeger dan voorzien klaar was, stond het systeem stil tot de programmeur die het volgende tijdsblok gereserveerd had aankwam. Dat zo'n duur tijdens dergelijke "idle time" niet gebruikt werd wilde men natuurlijk zo veel mogelijk vermijden. Wanneer een programma langer moet rekenen dan voorzien ontstaat er ook een probleem. Bovendien bestaat de kans dat een programma door een programmeerfout vastloopt. Het computersysteem wordt dan ook niet gebruikt terwijl de programmeur de fout opspoort.

Een mens als besturingssysteem?

Om de planningsproblemen te ondervangen werd na verloop van tijd een menselijke operator ingeschakeld. Programmeurs brengen de uit te voeren programma's naar de operator. Het laden en starten van de verschillende programma's is nu de taak van de operator. De operator bezorgt de programmeur het resultaat van zijn programma, of eventueel de inhoud van het geheugen op het moment dat er een fout optrad. Dit noemt men de zogenaamde core dump.

Er kunnen prioriteiten toegekend worden. De operator kan hiermee rekening houden bij het bepalen van de volgorde waarin de programma's gestart worden. De operator kan gegevens bijhouden, bijvoorbeeld van de gebruikte computertijd per programma. Deze kan eventueel gefactureerd worden aan de gebruikers.

Computersystemen worden steeds sneller, maar de menselijke operatoren niet. Hierdoor worden zij meer en meer de vertragende factor.

Bovendien kunnen de door de fabrikant geleverde bibliotheken alsmaar meer en complexere taken vervullen. Zo wordt de vertraging van de menselijke operator gedeeltelijk weggewerkt. Als de bibliotheken ook nog proberen te voorkomen dat er misbruik wordt gemaakt van de beschikbare hulpmiddelen kunnen we stellen dat ze evolueren naar de eerste besturingssystemen.

De eerste besturingssystemen

We kunnen eigenlijk pas van een besturingssysteem spreken vanaf het moment dat de zogenaamde residente monitors gebruikt worden, omdat zo'n residente monitor instaat voor de controle van het systeem, en constant in het geheugen gehouden wordt.

De residente monitor zorgt ervoor dat een uit te voeren programma in het geheugen wordt geladen en gestart. De operator moet wel nog steeds zorgen voor het aanbieden van de uit te voeren programma's, maar de rest van het beheer is overgenomen door de residente monitor.

Uit de codebibliotheken die bij de computersystemen geleverd worden evolueren ook specifieke besturingsroutines voor randapparaten, device drivers genaamd. Door deze drivers wordt het geven van opdrachten aan randapparaten eenvoudig. Een programma moet gewoon de juiste functie van de driver aanroepen. Samen met de opkomst van de hogere programmeertalen zorgden de device drivers ervoor dat het voor veel meer geïnteresseerden mogelijk werd om computers te gebruiken.

De eerste besturingssystemen groeien uit tot eenvoudige batchbesturingssystemen. De term batch wijst op de gebruikswijze van dit soort systemen: men biedt een stel programma’s als een globaal pakket aan het besturingssysteem aan.

Multiprogrammatie en Time Sharing

Batchbesturingssystemen kennen een steeds groter wordend efficiëntieprobleem. Terwijl het programma dat ze aan het uitvoeren zijn wacht op het voltooien van in- of uitvoer, wordt de processor van het computersysteem niet gebruikt.

Multiprogrammatie zorgt voor de oplossing. Terwijl een programma moet wachten op een externe gebeurtenis laat het besturingssysteem de processor verder werken aan een ander programma. Een reeks programma’s is samen in het geheugen aanwezig en worden om de beurt uitgevoerd tot ze moeten wachten.

Wanneer een multiprogrammatiesysteem wordt uitgebreid met een spoolingsysteem (SPOOL: Simultaneous Peripheral Operation OnLine) dat een aantal jobs kan klaarhouden op een snel extern geheugen en de simultane uitvoer van verschillende jobs op een ordelijke manier kan afhandelen komt men tot een zeer krachtig besturingssysteem.

Er is echter nog een groot gebrek aan interactiviteit. Als het resultaat niet is wat men verwachtte, moet men het programma wijzigen en opnieuw laten uitvoeren. De schijnbaar simultane uitvoer van programma’s bij multiprogrammatie laat ook toe om verschillende gebruikers tegelijk op interactieve wijze met een computer te laten werken. Elke gebruiker beschikt dan over een in- en uitvoerapparaat, verbonden met de centrale verwerkingseenheid. Omdat de processor afwisselend gedurende korte tijd aan elke gebruiker wordt toegewezen krijgt elke gebruiker de indruk over de volledige capaciteiten van het systeem te beschikken. Deze vorm van multiprogrammatie wordt timesharing genoemd. Het belangrijkste verschil is dat er bij timesharing niet gewacht wordt tot een programma moet wachten om de processor aan een ander programma toe te kennen. Door dat de gebruiker op een time sharing systeem rechtstreeks met de computer kan communiceren, krijgt hij bijna onmiddellijk feedback van kleinere opdrachten. Door dat de gebruikers typisch niet tegelijkertijd pieken veroorzaken in het processorgebruik lijkt het systeem voor ieder van hen performant te werken. Voor het accepteren van invoer via een toetsenbord moet de processor zeer weinig werk verrichten, dus terwijl de ene gebruiker een opdracht typt, is er processorkracht over voor het inlezen van de toetsenborden van andere gebruikers, of het werken aan eerder gegeven opdrachten.

Besturingssystemen voor minicomputers

In de jaren ’60 kwamen de minicomputers op de markt.

Bij mainframes was het de gewoonte dat een fabrikant een besturingssysteem ontwikkelde telkens hij een nieuw model op de markt bracht. Dit was voor de fabrikant een zware inspanning, maar voor de gebruiker betekende het ook telkens grote conversieproblemen als er naar nieuwere en krachtigere computers overgeschakeld werd.

In 1964 lanceerde IBM voor het eerst een familie van computersystemen. De System/360 reeks van mainframes omvatte modellen met verschillende verwerkingscapaciteiten, maar ze waren allemaal gebaseerd op dezelfde architectuur. Alle modellen gebruikten dan ook hetzelfde besturingssysteem: OS/360. Omdat OS/360 voor allerlei soorten toepassingen moest gebruikt worden, werd het een zeer ingewikkeld besturingssysteem. Dit noemt men multimode-systemen.

De PDP's (Programmed Data Processor) van Digital Equipment waren een zeer succesvolle reeks van minicomputers. In de wetenschappelijke wereld raakt intussen stilaan UNIX bekend, een nieuw besturingssysteem dat niet door een computerconstructeur werd ontwikkeld en waar geen commerciële bedoelingen aan vastzaten. De belangrijkste nieuwigheid is dat het, mits enkele kleine aanpassingen, op elke hardware bruikbaar kon worden gemaakt en daarbij naar de gebruiker toe steeds dezelfde interface aanbood. Het werd niet ontwikkeld in een assembleertaal maar in een daarvoor ontwikkelde hogere programmeertaal genaamd C.

De PC en het Internet

Aan het eind van de jaren 70 dook de microcomputer op. Toen IBM zijn microcomputer lanceerde en hiervoor een nieuw besturingssysteem genaamd MS-DOS ontwikkelde, bleek dit zo'n succes te zijn dat de meeste andere constructeurs deze architectuur overnamen. Men kon nu toepassingssoftware ontwikkelen die op zowat 90% van de zich massaal verspreidende microcomputers kon worden gebruikt.

Er moest dringend werk worden gemaakt van de vereenvoudiging van de gebruikersinterface. Vandaar de ontwikkeling van GUI (Guided User Interface) waarmee op een symbolische manier bevelen konden worden gegeven aan de computer.

- In 1984 kwam de Apple Macintosh SE op de markt, de eerste computer met een GUI.
- In 1985 was er de eerste versie van Windows, met een grafische laag bovenop DOS.

De stijgende performantie van de microcomputer was ideaal voor multitasking: time sharing voor één gebruiker. De gebruiker kan verschillende opdrachten tegelijk laten uitvoeren en deze laten samenwerken. De processortijd werd door het systeem verdeeld over al de opdrachten.

Men introduceerde ook netwerken. Eerst LAN, dan WAN en uiteindelijk het Internet.

Voorbeelden van besturingssystemen

Aangezien besturingssystemen sinds OS/360 niet meer specifiek voor één bepaalde machine ontworpen worden, zijn ze nu meestal verbonden met één van de drie besproken types: mainframes, minicomputers of microcomputers.

IBM is nog steeds één van de grote spelers op de mainframe-markt, en er worden nog steeds mainframes ontwikkeld. Nu worden de IBM mainframes zSeries genoemd. De besturingssystemen zijn in de loop der jaren voor deze ontwikkeld met klinkende namen als MVS, VM en SE.

Ook minicomputers zijn nog steeds te verkrijgen en in gebruik. Men spreekt nu ook soms van midrange systems. IBM noemt ze iSeries.

Voor microcomputers waren er in de jaren ’80 drie families.

- Op IBM-compatibele PC’s had je de keuze tussen MS-DOS van Microsoft en OS/2 van IBM.
- Voor Macintosh was er System 7.

OS/2 is van de markt verdwenen en MS-DOS is geëvolueerd naar de verschillende versies van Windows. Windows was eerst slechts een grafische toevoeging aan MS-DOS, maar de recentere versies zijn volledige besturingssystemen. Voor Macintosh heb je tegenwoordig MacOS nodig.

Er zijn ook besturingssystemen die niet aan één van de drie categorieën gebonden zijn.

- Allereerst is er UNIX dat men op de meest diverse hardwaren aantreft, zij het in allerlei varianten.
- Op mainframes vind je AIX (IBM) als gast onder VM.
- Op VAX minicomputers werd Ultrix gebruikt
- Terwijl voor microcomputers SCO UNIX of HPUX beschikbaar is. Voor PC’s is er sinds 1991 ook het steeds populairdere GNU/Linux.

Functies van een besturingssysteem

Twee grote taken: het gebruiksgemak maximaliseren en de efficiëntie van het systeem maximaliseren.

- Toegang tot apparaten eenvoudig maken
- Informatie bewaren en overzichtelijk weergeven
- Operaties en gegevens beveiligen

Wat betreft de hardware:

- De onderlinge communicatie tussen de onderdelen organiseren
- De goede werking van de onderdelen bewaken
- Hulpmiddelen rechtvaardig verdelen

Merk op dat het besturingssysteem ook een programma is dat door de processor uitgevoerd moet worden. We kunnen stellen dat de tijd dat het besturingssysteem de processor gebruikt verloren gaat voor de gebruiker. Efficiëntie is dus een belangrijke vereiste voor het besturingssysteem, om de hoeveelheid overhead te beperken.

Een derde taak is het streven naar hardwareonafhankelijkheid. Als het besturingssysteem een uniforme interface aanbiedt aan de bovenliggende toepassingssoftware en de gebruiker kan de hardware aangepast worden zonder overgangsproblemen. De eerste stap naar hardwareonafhankelijkheid was de introductie van device drivers.

Onderdelen van een besturingssysteem

Apparaatbeheer

De toegang tot de randapparatuur moet door het besturingssysteem geregeld worden, om conflicten te vermijden. Als 2 programma’s tegelijkertijd naar de printer sturen zou men vreemde resultaten krijgen.

Geheugenbeheer

Programma’s mogen niet in mekaars geheugenindex schrijven of lezen. Het beschikbare werkgeheugen is bovendien beperkt en het besturingssysteem moet ook streven naar een efficiënt gebruik ervan.

Procesbeheer

Het verdelen van de beschikbare processortijd over de verschillende processen die uitgevoerd moeten worden. Hier wordt gezorgd voor multiprogrammatie en multitasking.

Communicatie

De communicatie tussen processen op het computersysteem. Maar ook voor eventuele communicatie met andere computersystemen via een netwerk.

Gebruikersinterface

Om het gebruik van de computer te vereenvoudigen moet het besturingssysteem ook zorgen voor een adequate gebruikersinterface. Dit kan een tekstinterface of een grafische omgeving zijn.

Basismechanismen van een besturingssysteem

Kernel- en gebruikerstoestand

Programma’s die deel uitmaken van het besturingssysteem worden uitgevoerd in kerneltoestand. Wanneer de processor zich in kerneltoestand bevindt, beschikt hij over al zijn mogelijkheden. Wanneer de processor niet in kerneltoestand is, maar in gebruikerstoestand, zijn bepaalde instructies niet toegelaten. Tussen de machine-instructies zijn er namelijk een aantal die men geprivilegieerde instructies noemt. Deze laatste kunnen enkel worden gebruikt in kerneltoestand en zijn daarvoor voorbehouden aan het besturingssysteem. Wanneer een programma probeert om in gebruikerstoestand een geprivilegieerde instructie uit te voeren zal een fatale fout optreden. Op die manier kunnen allerlei delicate operaties op een gegarandeerd veilige manier worden uitgeoefend. Wanneer het besturingssysteem een gewoon programma laat uitvoeren zal het de processor eerst naar gebruikerstoestand brengen alvorens het programma te starten. In de toestandsbeschrijving van de processor is één bit voorzien om de toestand aan te geven waarin deze zich bevindt: user- of kernel mode.

Interrupts

Het besturingssysteem komt slechts in actie na een signaal vanuit de hardware of na een vraag vanuit de hoger gelegen software. Men zegt dan ook dat een besturingssysteem gebeurtenisgestuurd of event driven is.

Interrupts

- Wanneer vanuit de hardware behoefte is aan een tussenkomst in het besturingssysteem treedt een interrupt of onderbreking op.
- Als de software de hulp van het besturingssysteem nodig heeft wordt er een software interrupt aangeroepen of system call.

Een interrupt is een elektrisch signaal naar de processor, of een gebeurtenis binnenin de processor zelf. Als de harde schijf bijvoorbeeld een leesopdracht voltooid heeft, zal de schijfbesturingseenheid dit via een interruptsignaal naar de processor aan het besturingssysteem melden.

1. Schijfbesturingseenheid ontvangt een leesopdracht
2. Wanneer de leesopdracht voltooid is, stuurt de schijfbesturingseenheid een interrupt naar de interrupt controller
3. De interrupt controller geeft de interrupt door aan de processor. Wanneer er een interrupt met hogere prioriteit wordt afgehandeld is het mogelijk dat de schijfinterrupt hierop moet wachten.
4. Interrupt controller stuurt het volgnummer van de bron van de interrupt naar de processor, zodat die weet welk apparaat de interrupt veroorzaakte.
5. De processor zoekt het adres op van de juiste Interrupt Handler in de tabel met interrupt-vectoren.
6. De inhoud van bevelenteller, alle registers en de processorstatus worden op de stapel gezet.

7. Adres van Interrupt Handler wordt in de bevelenteller geplaatst en de processor wisselt naar kerneltoestand.
8. Sprong naar het adres van de Interrupt Handler en de handler wordt uitgevoerd. Deze code in de interrupt handler handelt het verzoek af. Deze handler is onderdeel van het besturingssysteem.
9. Als het werk erop zit herstelt de interrupt handler de bevelenteller, registers en processortoestand en schakelt de processor terug in gebruikerstoestand. Het programma dat in uitvoering was voor de interrupt wordt verder gezet.

Het gevolg van een interrupt is dus dat een stuk code uitgevoerd wordt, dat specifiek bestemd is voor de opgetreden interrupt: de Interrupt Handler.

Stappen 6 en 7 worden een context switch genoemd. Een context switch wijzigt de toestand van de processor zodat die een ander programma kan uitvoeren, en bovendien zo dat de toestand weer hersteld kan worden om verder te gaan met de uitvoering van het programma dat ervoor in uitvoering was.

Merk op dat dit door de processor zelf afgehandeld wordt, en niet door een stukje machinecode uit te voeren.

Externe interrupts komen van hardware buiten de processor:

- Klokinterrupts: De interne klok van de computer signaleert hiermee aan de processor dat een bepaalde tijdsperiode is verstreken
- I/O-interrupts: Wanneer een randapparaat een I/O-operatie is uitgevoerd meldt de besturingseenheid die het randapparaat bestuurt, op deze manier het einde van de I/O-operatie
- Keyboard interrupts: Telkens wanneer een toets op het klavier wordt ingedrukt heeft dit een interrupt tot gevolg, die tot doel heeft het bestemmende karakter te laten inlezen in een buffer zodat het programma waarvoor het bestemd is eraan kan.

Interne interrupts worden veroorzaakt door de processor en bestaan in twee groepen:

- Uitzonderingen of exceptions zijn het gevolg van een fout tijdens de uitvoering van een instructie zoals het delen van een getal door nul, het adresseren van een ongeldig adres of een poging om in gebruikerstoestand een geprivilegieerde opdracht uit te voeren.
- Traps of software interrupts zijn het gevolg van de uitvoering van speciaal daarvoor voorziene instructies in een programma. Als een programma bijvoorbeeld gegevens wil inlezen van op de harde schijf zal het dit via een software interrupt aan het besturingssysteem vragen.

Interrupts identificeren

Hoe wordt nu bepaald welke interrupt handler dient te worden uitgevoerd wanneer een interrupt optreedt? Bij exceptions wordt de interrupt door de processor zelf veroorzaakt, en kan de processor de inhoud van de conditiecode in zijn toestandsbeschrijving nagaan.

Wanneer een programma in uitvoering een trapinstructie doet wordt de nodige informatie meegegeven om de juiste interrupt handler te kiezen.

In 8086 assembler programma's voor MS-DOS wordt vaak "int 21h" opgeroepen. Deze software interrupt zorgt ervoor dat MS-DOS een interrupt handler uitvoert.

Bij hardware interrupts ligt de oorzaak buiten de processor, en moeten we op zoek naar de bron van de onderbreking. Dat kan op 3 algemene manieren:

- Als iedere controller een eigen interruptlijn heeft, bepaalt de lijn waarop het interruptsignaal verschijnt meteen welke handler er moet worden uitgevoerd.
- Als de controllers via één lijn met de processor verbonden zijn kan de controller d.m.v. een bit in één van zijn registers aangeven of hij een interrupt heeft verstuurd. Na de context switch wordt een routine uitgevoerd die aan software polling doet: achtereenvolgens elke controller controleren tot de afzender van de interrupt wordt gevonden.
- Bij de derde manier zijn alle controllers via één interruptlijn en één zogenaamde acknowledge-lijn met de processor verbonden. Zodra een hardware-interrupt optreedt zendt de processor via deze acknowledge-lijn een signaal naar de eerste controller. Indien deze de interrupt veroorzaakte stuurt hij een identificatiecode naar de processor; in het andere geval geeft hij het signaal door naar de volgende controller, die hierop op dezelfde manier reageert. Deze werkwijze, daisy chaining genoemd, is erg snel omdat ze volledig via de hardware verloopt.

Prioriteiten voor interrupts

Wanneer twee of meer interrupts tegelijkertijd optreden, moet er gekozen worden welke interrupt eerst afgehandeld wordt. Vaak wordt gekeken naar de vereiste interrupt latency. Dit is de maximale tijd waarbinnen de interruptafhandeling begonnen moet zijn.

Hoe we de prioriteit van interrupts kunnen bepalen hangt af van de gebruikte identificatiemethode.

- Bij gebruik van meerdere interruptlijnen zal de processor zelf moeten beslissen aan welk signaal hij eerst aandacht zal besteden.
- Bij software polling zal de volgorde waarin de controllers worden nagekeken bepalend zijn voor de prioriteit.
- Bij daisy chaining is het de volgorde waarin de controllers zijn aangesloten op de acknowledgelijn die dit bepaald. Een duidelijk nadeel is dat een verandering in prioriteit een hardwarematige ingreep vereist.

Een tweede vraag die we ons moeten stellen is of we nieuwe interrupts willen toelaten tijdens het afhandelen van een interrupt. Vaak wordt een vlag voorzien waarmee alle interrupts tijdelijk geblokkeerd worden. Andere systemen accepteren alleen interrupts met een hogere prioriteit.

System Calls

System calls of systemaanroepen leggen een link tussen de overige software en het besturingssysteem. Bij alle activiteiten uitgevoerd in kerneltoestand is immers zeer frequent de tussenkomst nodig van het besturingssysteem om operaties te kunnen uitvoeren die alleen in kernel-toestand mogelijk zijn.

Het uitvoeren van een system call

Essentieel bij een system call is het overschakelen van de processor naar kerneltoestand. Dit gebeurt altijd via een trapinstructie, waardoor wordt vermeden dat na het instellen van de kerneltoestand het gebruikersprogramma de controle zou kunnen behouden. De trapinstructie zorgt er voordat de gepaste systeemroutine wordt gestart. Daartoe wordt als operand voor de trapinstructie een code meegegeven, die in een register wordt geplaatst.

Omdat de trap functioneert als een interrupt zal een interrupt handler worden gestart in kernel mode, die de meegegeven code zal vertalen naar het beginadres van een specifieke systeemroutine. Aan de hand van de code wordt tevens bepaald hoeveel bytes aan informatie als parameters aan de systeemroutine moeten worden doorgegeven. Deze bytes worden dan klaargezet, waarna de systeemroutine kan worden opgeroepen. Zodra de opdracht voltooid is wordt een eventueel resultaat op een afgesproken plaats gedeponeerd, waarna de processor wordt teruggeschakeld naar user mode en tenslotte terugkeert naar het opgeroepen programma.

Beschikbare system calls

- Opstarten en beëindigen van programma’s
- Laten wachten van programma’s op een gebeurtenis
- Toekennen en vrijgeven van geheugen
- Bestandsbeheer
- Besturing en beheer van apparaten
- Uitwisselen van informatie
- Communicatie

Voorbeeld

In Linux wordt de read-system call genaamd read (file.desc, buffer, nbytes) opgeroepen.

1. Oproepend programma zet de waarden van de parameters op de stapel.
2. Read functie wordt aangeroepen
3. Functie zet system call nummer in het juiste register en voert een trapinstructie uit
4. Besturingssysteem zoekt juiste routine op voor de system call
5. Besturingssysteem start systeemroutine
6. Systeemroutine zet resultaat op voorziene plaats, verlaat kernel mode en geeft controle terug aan read-functie
7. Read-functie ontvangt de gegevens, zet ze op hun plaats en geeft controle terug aan oproepend programma
8. Uitvoer programma gaat verder

Stappen 3 tem 6 worden uitgevoerd in kernel mode.

Bootproces

Basic Input/Output System (BIOS)

Het eerste programma dat de processor uitvoert is het Basic Input/Output System, vaak afgekort tot BIOS. Het zorgt dat er communicatie mogelijk is tussen enkele onderdelen van het computersysteem, in de eerste plaats tussen secundaire opslagapparaten en het werkgeheugen. Het Basic Input/Output System dankt zijn naam aan het feit dat het naast deze communicatie ook zorgt voor uitvoer op het scherm, en de mogelijkheid tot invoer via het toetsenbord.

Aangezien net de BIOS ervoor moet zorgen dat de opslagapparaten bruikbaar worden kan de BIOS-code niet op een dergelijk apparaat worden opgeslagen. Daarom wordt deze in een ROM-chip op het moederbord bewaard. Tegenwoordig wordt hiervoor EEPROM gebruikt. BIOS is een voorbeeld van firmware.

Omdat een processor altijd de opdracht uit het werkgeheugen inleest waarvan het adres in de bevelenteller staat, wordt de inhoud van de BIOS-chip beschikbaar gesteld via een deel van het adresbereik van het werkgeheugen. Hiervoor is een deel van het Upper Memory gereserveerd: van F0000h tot FFFFFh. In sommige systemen blijft dit deel van het werkgeheugen leeg en wordt van de ROM-chip gelezen. Andere systemen kopiëren de inhoud van de chip naar dit gedeelte van het werkgeheugen omdat RAM sneller is dan ROM. Dit noemt men ROM shadowing.

Wanneer een processor geactiveerd wordt zal hij altijd beginnen met de instructie op adres FFFF0h. Hier staat dan de jump instructie naar de eigenlijke opstartcode.

Power-On Self Test

De BIOS-opstartcode begint met het testen van de vitale systeemonderdelen. Het geheel van deze test wordt Power-On-Self-Test of POST genoemd. De aanwezigheid of correcte werking van bijvoorbeeld processor, werkgeheugen en toetsenbord worden getest. De feedback gebeurt d.m.v. geluidsignalen die we beeps noemen. Bij problemen kan men aan het aantal beeps en hun duur horen wat de oorzaak is.

Opstartvolgorde

Als de POST succesvol voltooid is zal de BIOS-code beginnen met het laden van het besturingssysteem. Wanneer een computersysteem meerdere opslagapparaten bevat moet in de BIOS aangegeven zijn vanop welk apparaat een besturingssysteem geladen moet worden. Het besturingssysteem kan op een harde schijf staan, maar ook op een diskette, CD-ROM of USB-opslagapparaat. In welke volgorde op alle aanwezige apparaten gezocht moet worden staat in de

zogenaamde opstartvolgorde of boot sequence. Omdat deze instellingen gewijzigd moeten kunnen worden kunnen ze niet samen met de code van de BIOS bewaard worden op de ROM-chip. Voor de veranderlijke instellingen is er een afzonderlijke CMOS-chip voorzien, die door een batterij onder spanning gehouden wordt wanneer het systeem uitgeschakeld is. Zo gaan de instellingen niet verloren.

Partities

Als in de opstartvolgorde wordt verwezen naar de harde schijf is de vraag weer hoe we het besturingssysteem terugvinden op zo'n schijf. De code in de BIOS zal een sprong uitvoeren naar de opstartcode opgeslagen in de eerste sector van de schijf, de Master Boot Record. Deze code wordt vaak Initial Program Loader (IPL) genoemd.

Een harde schijf kan logisch ingedeeld worden in partities. Hierdoor kunnen we meerdere besturingssystemen op één schijf bewaren, of kunnen we op één schijf verschillende bestandssystemen gebruiken.

In de partitietabel wordt het begin- en eindpunt en het type van iedere partitie opgeslagen. Op het PC-platform vind je de partitietabel in de MBR, achter de opstartcode. Er is dan nog plaats voor 4 maal een partitiebeschrijving en een magic number van 2 bytes. Dit magic number is een controlegetal en moet altijd als waarde 0xAA55 hebben, anders wordt de MBR als ongeldig beschouwd.

Elk van de 4 elementen van de partitietabel beschrijft een partitie, en bevat de volgende velden:

- Active geeft aan of deze partitie moet gebruikt worden om van te booten of niet (Windows)
- Start cylinder / kop / sector adres van de eerste sector van de partitie
- Type indicatie van het gebruikte bestandssysteem
- End cylinder / kop / sector adres van de laatste sector van de paritie
- Lengte lengte van de partitie

De partities die in de partitietabel gedefinieerd worden noemen we primitieve partities. Om meer partities toe te laten introduceerde men logische partities. Er is geen plaats om de partitietabel uit te breiden, dus wordt een primaire partitie onderverdeeld in logische partities. Een primaire partitie die verder onderverdeeld wordt noemen we extended partitie.

In de partitietabel geven de types 0x05 of 0x0f een extended partitie aan. Het start-veld bevat dan het adres van de eerste sector van de eerste logische partitie. Iedere logische partitie verwijst naar de volgende logische partitie. Als limiet voor het aantal logische partities wordt vaak 24 opgegeven. De gelinkte lijst kan langer zijn maar aangezien DOS letters vanaf C toekent kan de lijst niet langer dan 24 partities lang zijn.

Een logische partitie bevat een Extended MBR met een eigen partitietabel. Zo'n EMBR bevat een partitietabel die de structuur van de tabel in de MBR volgt, maar slechts 2 partities beschrijft: één logische en een nieuwe extended partitie. Deze extended partitie bevat ook weer zo'n tabel die

één logische en één extended partitie beschrijft, enz. De taak van de opstartcode in de MBR is om in de partitietabel de actieve partitie op te sporen. De eerste sector van deze partitie heeft Partition Boot Sector en bevat ook weer een stuk opstartcode. De code in de MBR zal de code in de PBS inladen en dan een sprong naar deze code uitvoeren. Het is deze code die uiteindelijk het besturingssysteem op de partitie zal inladen en starten.

Bestandssystemen

Computersystemen hebben er naast gegevensverwerking een taak bij gekregen: gegevensopslag. Het onderdeel van het besturingssysteem dat instaat voor permanente opslag is het bestandssysteem. De eisen die men hieraan stelt zijn:

- Persistente gegevensopslag
- Eenvoudige organisatie en logische ordening van gegevens
- Een zo snel mogelijke gegevenstoegang
- Zo goed mogelijk gebruik maken van de beschikbare capaciteit
- Het beveiligen van de gegevens (geen toegang tot gegevens v/e andere gebruiker)

Fysische en logische gegevensordening

Fysische ordening: Sectoren

Eén sector is de kleinst adresseerbare eenheid voor gegevensopslag op een harde schijf. Hoe de gegevens over de sectoren van de harde schijf verdeeld zijn noemen we de fysische ordening van gegevens.

Logische ordening: Bestanden

Een gebruiker wil een logische gegevensordening, die er voor zorgt dat gegevens gemakkelijk terug te vinden zijn. Een bestand is een persistent opgeslagen logische verzameling gerelateerde gegevens.

Het bestandssysteem vertaalt het logische beeld van de gebruiker en de bijbehorende operaties in de fysische schijftoegangen. Zo wordt een bestand altijd voorgesteld als een aaneengesloten reeks gegevens, zelfs al is dit in de fysische ordening niet zo is. Het opzetten van de nodige gegevensstructuren voor de fysische ordening van een bestandssysteem op een partitie noemt men formatteren.

Bestanden

Een bestand is een verzameling gerelateerde gegevens, die door een gebruiker of applicatie op een bepaalde manier geïnterpreteerd worden. Het bestandssysteem verzamelt alle gegevens over een bestand in de bestandsbeschrijving of file descriptor.

- De naam
- De adressen van de gebruikte sectoren

De lengte van de naam kan beperkt zijn door de beschikbare ruimte in de file descriptor. Welke karakters toegelaten zijn hangt af van de gebruikte codering (ASCII of Unicode) en het vermijden van tekens die een speciale betekenis hebben in het systeem. Ook de hoofdlettergevoeligheid kan van belang zijn. Sommige systemen leggen bovendien een vaste structuur op de naam, bijvoorbeeld de “8+3”-structuur van MS-DOS: elke bestandsnaam kan 8 tekens lang zijn en een extensie van 3 tekens hebben.

Naast de bestandsnaam worden in de file descriptor allerlei andere bestandskenmerken bijgehouden.

- De grootte van het bestand
- De tijdstippen waarop het bestand is aangemaakt of gewijzigd
- De eigenaar van het bestand
- Op welke manier toegang tot het bestand wordt geregeld

Om de plaats te kunnen terugvinden waar de gegevens op het extern geheugenmedium worden bijgehouden moet er ook een plaatsaanduiding bijgehouden worden. Het kan ook interessant zijn om te weten of het bestand in gebruik is, en of er een kopie bestaat.

Directories

Om een overzicht mogelijk te maken in de massale hoeveelheden informatie is het noodzakelijk dat bij elkaar horende bestanden gegroepeerd worden in directories. Directories zijn een toevoeging aan het logische beeld dat de gebruiker van de opgeslagen gegevens krijgt.

Er is ook een fysische gegevensstructuur op de schijf, die de bestandsbeschrijvingen van de bestanden bevat. Structuur van het bestandssysteem

Lineaire directory

De meest eenvoudige organisatievorm. De bestandsbeschrijvingen van alle in het bestandssysteem aanwezige bestanden worden in één enkele directory opgenomen. Nadelen:

- Indien het bestandssysteem een groot aantal bestanden bevat zullen de basisbewerkingen veel tijd vragen.
- Een dergelijke lange lijst van bestanden is uiterst onoverzichtelijk.
- Aan elk bestand wordt een unieke naam gegeven.

Voordeel:

- Bij schijfgeheugens kan de directory op de meest gunstige plaats van de schijf worden gezet, zodat snelle toegang kan worden verkregen.

Lineaire directories zijn slechts bruikbaar in kleine bestandssystemen in omgevingen met één gebruiker.

Hiërarchisch bestandssysteem

Hierbij wordt de volledige verzameling bestanden opgesplitst in een aantal kleine groepen, die elk hun eigen directory hebben. Een directory bevat dus bestanden of andere directories. De directories waarnaar een directory verwijst noemen we subdirectories.

Het is niet meer nodig dat elk bestand een unieke naam heeft. Bestanden in verschillende directories kunnen dezelfde naam hebben. Nadelen:

- Men moet aangeven waar het bestand zich bevindt in de hiërarchische structuur.
- Sommige bewerken worden complexer. Het opzoeken van een bestand waarvan alleen de naam bekend is vereist nu het recursief doorlopen van de volledige directorystructuur.
- Ook het schrappen stelt een probleem: wat gebeurt er met de bestanden en subdirectories waarnaar vanuit deze directory wordt verwezen? Indien deze mee moeten worden vernietigd moet dit eveneens op een recursieve manier gebeuren.

Pad: de opsomming van alle directories die moeten doorlopen worden vanaf de hoofddirectory om de directory waarin de file wordt beschreven te bereiken. Working directory: de directory waarin de gebruiker zich bevindt Relatief pad: Een pad dat vertrek van de huidige werkdirectory Absoluut pad: Een pad dat vertrekt vanuit de wortel van de boomstructuur

Boomstructuur

De meest eenvoudige hiërarchische structuur. Naar ieder bestand in het systeem leidt er juist één uniek pad. We gebruiken de term boomstructuur omdat deze structuur zich vertakt als een boom.

Algemene structuur

Wanneer we toch meer dan één pad naar een bestand toelaten krijgen we een algemene structuur. Dit betekent dat een ebstand een element van twee verschillende directories kan zijn, of dat een directory een subdirectory is van twee of meer directories.

De algemene structuur laat toe dat er lussen gecreëerd worden in het bestandssysteem. Iedere recursieve operatie op het bestandssysteem, zoals het zoeken naar een bestand, zal oneindig doorgaan in zo’n lus als het recursief algoritme niet detecteert dat het een bepaalde directory al heeft behandeld.

Een bijkomend probleem betreft het achterblijven van onbereikbare bestanden in het systeem na het verwijderen van de directory. Een directory moet steeds bereikbaar zijn vanuit de wortel. Als mogelijke oplossing voor dit probleem zouden we kunnen beslissen om altijd recursief alle subdirectories te verwijderen, zelfs al lijken ze nog bereikbaar. Maar dan stuiten we op een tweede probleem. Als we alle onderliggende directories recursief verwijderen, zullen we uiteindelijk een directory verwijderen die hoger in de hiërarchische structuur ligt dan de directory die we oorspronkelijk wilden verwijderen. Omwille van deze problemen worden dergelijke lussen in het bestandssysteem zelden toegelaten. Men krijgt dan een acyclische structuur.

Acyclische structuur

De acyclische structuur is nog steeds algemener dan de boomstructuur, maar er mogen geen lussen voorkoen. Lussen vermijden kan op twee manieren. We kunnen bij iedere operatie die een lus zou kunnen veroorzaken een controle uitvoeren. De operatie wordt enkel toegelaten als er geen lus ontstaat. Operaties in het bestandssysteem worden dan wel complexer en dus trager.

Een eenvoudigere oplossing is het verbieden van meervoudige verwijzingen naar directories. Wanneer enkel bestanden vanuit twee directories kunnen benaderd worden is het onmogelijk een lus te creëren.

Aaneengesloten bestanden

Vanaf een bepaald adres worden alle bytes van de file achtereenvolgens op de schijf gezet, in een reeks op elkaar volgende sectoren. We noemen dit een aaneengesloten bestand. Er ontstaat een belangrijk probleem: Er is voldoende grote vrije ruimte nodig om het bestand een plaats te kunnen geven. Naarmate het gebruik van de schijf voortduurt zal er door het verwijderen van bestanden een steeds groter aantal afzonderlijke vrije ruimten ontstaan. De omvang van de grootste vrije ruimte zal daarbij ook steeds kleiner worden. Men zegt dan dat er fragmentatie ontstaat.

Om een geschikte locatie te kiezen moeten we in de eerste plaats de grootte van het bestand kennen. Aangezien een bestand kan groeien naarmate de tijd vordert, is de grootte niet gekend op het moment dat het bestand aangemaakt wordt. De enige oplossing is om op de een of andere manier een schatting te maken van de uiteindelijke grootte.

Er zijn 3 algoritmes om een element van de tabel te kiezen wanneer een nieuw bestand bewaard moet worden:

- Het First Fit algoritme kiest de eerste vrije ruimte die voldoende groot is voor het bestand.
- Het Best Fit algoritme kiest de kleinste vrije ruimte die groot genoeg is voor het bestand.
- Het Worst Fit algoritme kiest de grootste beschikbare vrije ruimte.

Het belangrijkste voordeel van het werken met aaneengesloten bestanden is ongetwijfeld de hoge snelheid waarmee deze bestanden kunnen worden verwerkt. Door het gebruik van op elkaar volgende sectoren zal een bestand zich meestal binnen één cilinder van de schijf bevinden en moet de arm van de schijf zich niet bewegen.

Een ander voordeel is de mogelijkheid om niet alleen sequentiële, maar ook directe toegang tot een bepaald deel van het bestand te krijgen. Als we het startadres s van het bestand en de blokgrootte b kennen dan kan op eenvoudige wijze het adres a worden berekend van het blok dat de byte op positie B binnen het bestand bevat: A = s + B/b Ernstige nadelen:

- Een eerste moeilijkheid is de verplichting een schatting te geven van de grootte van elk nieuw bestand. Regelmatig zal bij het schrijven naar een ebstand blijken dat de voorziene ruimte niet volstaat.
- Dan kan het programma afgebroken worden om met een betere schatting te worden hernomen of kan het programma onderbroken worden om het bestaande deel van het bestand naar een grotere vrije ruimte te kopiëren.
- Fragmentatie zorgt ervoor dat het bijna onmogelijk is de capaciteit van de schijf volledig te gebruiken. In dat geval zal reorganisatie van de gegevens op de schijf noodzakelijk zijn. Dit defragmenteren kan eventueel ook zonder tweede harde schijf, maar is een tijdrovend proces.

Niet-aaneengesloten bestanden

De oplossing voor dit probleem kan gevonden worden in het verdelen van het bestand in kleinere stukken. Het gevolg is natuurlijk dat delen van eenzelfde bestand over de schijf zullen worden verspreid. De lees/schrijfkop moet zich vaak verplaatsen, wat erg tijdrovend is.

Blokken

Het bestandssysteem groepeert een aantal sectoren in een blok, ook soms cluster genoemd.

- Een verdeling in grotere eenheden zorgt ervoor dat het bestand in minder delen gesplitst wordt. Hierdoor wordt het bestand minder sterk verspreid dus moet de leeskop minder verplaatsingen maken.
- Bovendien zal de adresinformatie die we nodig hebben om de verschillende delen van het bestand terug te vinden, kleiner zijn omdat het aantal delen kleiner is.

Kleinere eenheden hebben dan weer als voordeel dat het plaatsverlies aan interne fragmentatie kleiner zal zijn. Bij het verdelen van een bestand over een aantal blokken zal het laatste blok zelden volledig gevuld zijn. Het laatste stuk van dit laatste blok is onbruikbaar om andere gegevens op te slaan, en gaat dus verloren.

Wanneer we bestanden opsplitsen moet het bestandssysteem natuurlijk ook bijhouden waarde verschillende delen van het bestand opgeslagen zijn.

Bestanden als gelinkte lijsten

De file descriptor bevat het adres van het eerste blok van het bestand, en elk blok bevat een verwijzing of pointer naar het volgende blok. Het laatste blok van het bestand bevat een speciale pointer die aangeeft dat de lijst stopt, soms nul-pointer genoemd (vaak 999). De vrije blokken

worden ook d.m.v. een gelinkte lijst bijgehouden. Ieder vrij blok bevat het adres van het volgende vrije blok. Voordeel:

- Adresinformatie neemt weinig plaats in beslag.

Nadeel:

- De vaststelling dat directe toegang hier niet mogelijk is. De enige manier om bij een bepaald deel van een bestand te komen is immers de ketting van verwijzingen te volgen tot bij de gezochte plaats. De toegangssnelheid is erg laag.
- Het verwijderen van een bestand verloopt erg traag omdat de ketting van alle blokken moet doorlopen worden om de nul-pointer te vervangen door een verwijzing naar het eerste blok van de lijst met vrije blokken.

Door in de bestandsbeschrijving ook een pointer naar het laatste blok bij te houden kunnen we het verwijderen van een bestand versnellen.

Wanneer er iets misloopt met een verwijzing gaat een deel van het bestand verloren, of kan het bestand onterecht verwijzen naar een deel van een ander bestand. Om dergelijke problemen te vermijden wordt soms gewerkt met een dubbel gelinkte lijst, waarin ieder blok ook verwijst naar het vorige blok in de lijst. Een foutieve verwijzing kan hersteld worden door de lijst van achter naar voor te doorlopen.

Bestanden met indexblokken

Dit probleem kan worden opgelost door de adressen van de verschillende onderdelen van een bestand te verzamelen in een speciaal daarvoor voorbehouden blok op de schijf: een indexblok. In de file descriptor staat dan de plaatsaanduiding van dit indexblok.

Iedere bestandsbeschrijving in de directory bevat het adres van een indexblok.

Elk van deze blokken bevat de adressen van de blokken met de eigenlijke gegevens van het bestand. Ook de vrije blokken worden bijgehouden in een indexblok. Indien één blok niet volstaat kan men een gelinkte lijst van indexblokken creëren of kan men een hiërarchische index-structuur opbouwen. Hierbij verwijst het eerste indexblok niet naar delen van het bestand, maar naar indexblokken die de adressen van de blokken met gegevens bevatten. Voordeel:

- De toegangssnelheid. Wanneer een bestand geopend wordt kan het indexblok volledig in het geheugen geladen worden. Zelfs bij seriële toegang is er sprake van een snelheidswinst omdat er niet moet gewacht worden tot het bestandssysteem het adres van het volgende blok uit het vorige blok gelezen heeft.

Nadeel:

- De indexblokken nemen schijfruimte in. Dit komt omdat er per bestand één blok extra nodig is: het indexblok. Dit plaatsverlies is vooral merkbaar wanneer er veel kleine bestanden zijn.

Daarom wordt in de bestandsbeschrijving vaak plaats voorzien voor een klein aantal adressen van blokken van het bestand. Als de hoeveelheid benodigde blokken onder dit aantal blijft, moet er geen indexblok gebruikt worden.

De toegangssnelheid kan verder geoptimaliseerd worden door extents te gebruiken. Een extent is een verzameling opeenvolgende blokken op de schijf. Hierdoor zullen grotere delen van een file op een zelfde plaats van de schijf terecht komen, zodat het lezen of schrijven minder onderbroken hoeft te worden voor een verplaatsing van de leeskop. Het adres van het eerste blok en de lengte van de extent zijn voldoende.

Bestanden beschrijven in een file map

We beschrijven de inhoud van de schijf in een aparte tabel. Deze tabel, de file map, wordt voor ieder blok van de schijf aangegeven of het vrij is, en indien niet tot welk bestand het behoort.

De eenvoudigste versie van een file map is een bitvector. Met ieder blok komt een bit overeen: 1 betekent dat het blok in gebruik is, 0 dat het blok vrij is. Vergeleken met indexblokken zorgt zo'n bitvector van vrije blokken voor minder plaatsverlies. De vaste lengte van de bitvector (1 bit blok) is wel nadeling als er weinig vrije blokken zijn. Dan ziet de bitvector er zo uit: 111...111101111..111.

Met elk blok komt een element van de tabel overeen. De elementen, behorend bij de blokken die samen een bestand vormen, bevatten pointers zodat ze een gelinkte lijst vormen. Het laatste element in de lijst bevat een code om het einde van het bestand aan te geven. De plaatsaanduiding in de file descriptor bestaat uit de index van het element dat overeenstemt met het eerste blok in de file.

Voordeel:

- De mogelijkheid om de volledige beschrijving van een schijf in het centrale geheugen te brengen. We werken eigenlijk weer met een gelinkte lijst, maar nu is de volledige lijst van adressen al beschikbaar in het werkgeheugen.

Men kan er ook op een eenvoudige wijze voor zorgen dat een bestand zo goed mogelijk gegroepeerd blijft: de combinatie van de beschrijving met de adresgegevens van het bestand in één tabel laat toe bijkomende ruimte voor het bestand te zoeken.

Directories (2)

Een directory is een verzameling bestandsbeschrijvingen. Er zijn verschillende fysische structuren mogelijk om deze bestandsbeschrijvingen op een schijf op te slaan: een gewone tabel, een gesorteerde tabel, een hashtabel of een gelinkte lijst.

Tabel

Wanneer we deze bestandsbeschrijvingen in een tabel bijhouden, zal vooral het opzoeken van een bestandsbeschrijving vrij veel tijd vragen. We moeten de tabel doorlopen tot we het gezochte element vinden. Het toevoegen van een file descriptor kan zeer snel gebeuren, aangezien die gewoon achteraan kan bijgeplaatst worden.

Gesorteerde tabel

Om het opzoeken in een tabel te versnellen kan de tabel gesorteerd worden. Het is dan mogelijk om het binair zoeken toe te passen: bekijk het middelste element, en dan weet je of het gezochte element ervoor of erachter ligt. Zo kan je onmiddellijk de helft van de tabel negeren, en op dezelfde manier verder zoeken in de overgebleven helft. Doordat we de tabel gesorteerd moeten houden zal het toevoegen van de file descriptors natuurlijk trager verlopen. Dit zoeken kan ook binair, maar dan moeten alle achterliggende elementen nog opgeschoven worden om plaats te maken voor de nieuwe bestandsbeschrijving. Bij het verwijderen moeten we de lege plaats opvullen door alle volgende elementen een plaats naar voor te schuiven.

Hashtabel

De positie in de tabel wordt berekend met behulp van de hashfunctie. In dit geval zal de gekozen hashfunctie de bestandsnaam transformeren naar een geheel getal. De bestandsbeschrijving zal dan op deze positie in de hashtabel bewaard worden. Wanneer we een bestandsbeschrijving zoeken, bereken we de hashwaarde van de bestandsnaam, en we kennen onmiddellijk de positie waarop de gezochte file descriptor staat. Een mogelijk probleem zijn collisions of botsingen. Wanneer de hashfunctie voor twee verschillende bestandsnamen dezelfde positie teruggeeft.

Hoe groter de hashtabel, hoe meer posities en hoe meer mogelijke uitkomsten van de hashfunctie, en hoe minder kans op een botsing. Het nadeel is natuurlijk dat er meer plaats verloren gaat voor de grotere tabel. In gunstige omstandigheden levert de hashfunctie dus onmiddellijk de plaats van een op te zoeken file descriptor. Ook het toevoegen of verwijderen van file descriptors kan zeer snel gebeuren.

Gelinkte lijst

In de tabel verwijst ieder element naar de positie van het volgende. We moeten dan alleen de startpositie kennen. Bijvoorbeeld: index 1 is de startpositie voor de directory, index 2 is de startpositie voor de vrije plaatsen in de tabel. Een verwijzing naar 0 geeft het einde van de lijst aan.

Wat betreft het opzoeken van een element is deze structuur te vergelijken met de klassieke tabel. Het moet sequentieel gebeuren.

Bij het toevoegen of verwijderen kunnen we gewoon de verwijzingen aanpassen, en alle elementen kunnen op hun plaats blijven staan. Het geringe plaatsverlies door de verwijzing naar de volgende bestandsbeschrijving levert dus snelheidswinst bij deze operaties.

Voorbeeld 1: FAT

Het FAT-bestandsysteem is het oorspronkelijke bestandssysteem van het MS-DOS besturingssysteem.

File Allocation Table

FAT staat voor File Allocation Table. Deze FAT is een file map met daarin de nodige informative over ieder blok van de schijf. Een FAT-partitie bevat een boot sector, 2 kopieën van de FAT, en een datagebied dat de directories en bestanden bevat.

Wanneer een blok onderdeel van een bestand is bevat het overeenkomstige FAT-element het adres van het volgende blok van het bestand. Het laatste blok van een bestand wordt aangeduid met de End Of File-code OxFFFF, en een vrij blok met 0x0000.

Het getal in de naam van deze systemen duidt aan hoeveel bits er gebruikt worden voor ieder element in de FAT. In een FAT12-systeem bevat ieder element van de FAT dus 12 bits.

Het aantal bits dat een FAT-element bevat bepaalt welke adressen er gebruikt worden voor de blokken. In een FAT-12 tabel zijn er voor ieder element 212 of 4096 mogelijkheden. We stellen dus vast dat de maximale adresseerbare schijfcapaciteit bepaald wordt door het beschikbaar aantal bits voor het adres, maar dat ook de blokgrootte een belangrijke rol speelt.

De totale grootte van de FAT-tabel speelt ook een belangrijke rol, aangezien we omwille van de performantie de volledige tabel in het werkgeheugen willen houden.

FAT12:

- De grootte van de tabel is dan 212 x 12 bits, dus 49152 bits of 6 KB.

Voor FAT32 wordt dit 232 x 32 bits, wat overeenkomt met 16GB. Door de blokgrootte te vergroten kunnen we dus oftewel zorgen voor een grotere adresseerbare schijfcapaciteit, of voor een kleinere FAT-tabel.

Partitiegrootte Blokgrootte 33 tot 64MB 512B 65 tot 128 MB 1KB 129 tot 256MB 2KB 257MB tot 8GB 4KB 8GB tot 16 GB 8KB 16GB tot 32GB 16KB Meer dan 33GB Niet ondersteund.

Voor meer dan 32GB is NTFS vereist.

Directories

De logische structuur van het FAT-besturingssysteem is een boomstructuur. Fysisch wordt een directory voorgesteld als een gewone tabel. Deze tabel wordt op dezelfde manier opgeslagen op de schijf als een gewoon bestand: blokken waarop de directory-tabel staan worden via een gelinkte lijst in de FAT bijgehouden.

In FAT12 en FAT16 staat de rootdirectory onmiddellijk na de 2 kopieën van de FAT. In FAT32 is dit niet meer verplicht. In de directory-tabel neemt iedere bestandsbeschrijving 32 bytes in.

- Naam
- Extensie
- Attributen
- Tijdstip creatie
- Hoogste bytes startadres
- Startadres
- Grootte

Het attributenveld verdient nog wat aandacht. Elk van de 8 bits komt overeen met een bepaald attribuut, en duidt aan of dit attribuut op het element van toepassing is. Eén van de attributen is het “directory”-attribuut. Als hier 1 staat beschrijft dit element een subdirectory, anders een bestand. Voor ieder bestand zijn er de read-only, hidden, system en archive attributen.

Het Volume Label wordt normaal slechts voor één speciaal element van de root-directory gebruikt. Hiervan is handig gebruik gemaakt om in Windows de mogelijkheid te voorzien om langere bestandsnamen te gebruiken in FAT-besturingssystemen.

De lange naam wordt verspreid over bepaalde delen van meerdere tabelelementen. Het volumeattribuut van deze elementen wordt op 1 gezet, zodat MS-DOS ze negeert.

Voorbeeld 2: NTFS

NTFS staat voor New Technology File System. NTFS bestandsnamen kunnen 255 tekens lang zijn, en de tekens worden m.b.v. unicode bewaard.

Master File Table

NTFS maakt gebruik van een Master File Table (MFT) In deze tabel wordt voor ieder bestand 1 bestandsbeschrijving bijgehouden. Deze beschrijving bevat naast de bestandsnaam, de grootte en de toegangsdata informatie over de toegangsrechten tot het bestand.

De MFT bevat één record per bestand, en niet één record per blok op de schijf. Hierdoor is de grootte van het MFT variabel. Standaard wordt 12,5% van de partitie voorzien voor de MFT, en 87,5% voor data. Wanneer het datagebied echter vol is, en de MFT neemt niet de voorziene 12,5% in, kan deze ruimte ook voor data worden gebruikt.

Header | Standard Information | File Name | Datagebied | … | Security Descriptor

Een record in de MFT bevat attributen. Welke overige attributen gebruikt worden staat niet vast in de specificatie van de NTFS. De gebruikte attributen op de partitie worden in het systeembestand $AttrDef bijgehouden.

De standaardinformatie omvat de grootte van het bestand, de toegangsdatums en andere eigenschappen. De toegangsrechten worden bewaard in de security descriptor. Omdat attributen geen vaste lengte hebben is er een hoofding nodig die de structuur van het record beschrijft.

Wanneer de inhoud van een attribuut te groot wordt om in het MFT-record te bewaren wordt het attribuut non-resident gemaakt. Dit betekent dat de inhoud van het attribuut in vrije blokken in het datagebied bewaard wordt, en dat het MFT-record een verwijzing naar deze blokken bevat.

Het data-attribuut in een MFT-record bevat verwijzingen naar blokken in het datagebied. Om de toegangssnelheid te verhogen en de hoeveelheid adresinformatie te beperken wordt gebruik gemaakt van extents, die in de context van NTFS vaak data runs of streams genoemd worden.

Voor kleine bestanden wordt de inhoud van het bestand zelf binnen het data-attribuut in het MFT-record opgeslagen. Zo gauw het bestand is opgezocht is directe toegang mogelijk zonder schijftoegangen. Dit gebeurt typisch voor bestanden kleiner dan 1,5KB.

Wanneer een bestand zo groot wordt dat de verschillende data runs niet meer binnen het data-attribuut kunnen beschreven worden is het mogelijk om het data-attribuut zelf buiten het MFT-record op te slaan. Door deze flexibele structuur is er bij NTFS geen sprake van een maximale bestandsgrootte.

Directories

De logische structuur van NTFS is een boomstructuur, net als bij FAT. Voor directories wordt ook een record in de MFT aangemaakt. Een directory-record heeft extra attributen die verwijzingen bevatten naar de bestanden en subdirectories. Net als bij gewone bestanden worden deze gegevens voor een kleinere directory in het record bijgehouden.

De fysische structuur van de inhoud van een directory wordt bijgehouden m.b.v. B-bomen. Ingewikkelde shit die niet in de cursus staat.

Second Extended Filesystem

Het Extended Filesystem wordt beschouwd als het standaard-bestandssysteem van Linux.

Inodes

Net zoals Linux zelf is het Extended Filesystem, en dus ook de opvolgers ervan, gebaseerd op Unix. De blokken van de schijf worden ingedeeld in 4 soorten:

- Bootblok
- Superblok
- Inodeblokken
- Datablokken

In het bootblok staat bootstrap code die het besturingssysteem moet laden en opstarten. Het heeft geen specifieke functie in het bestandssysteem. Het superblok bevat globale gegevens over het bestandssysteem en enkele instellingen.

Met ieder bestand in ext2 wordt een bestandsstructuur geassocieerd, die inode wordt genoemd. Hierin vinden we alle metadata en de verwijzingen naar de datablokken waarin de inhoud van het bestand opgeslagen is. Achter het superblok staat een lijst met inodeblokken, soms ook I-list genoemd. De overige blokken in het bestandssysteem worden gebruikt om de inhoud van bestanden in weg te schrijven.

Een inode bevat dus allerlei gegevens, waaronder de eigenaar van het bestand en de toegangsrechten. Merk op dat de bestandsnaam niet in de inode wordt bewaard. De bestandsnamen vinden we in de directories, samen met een verwijzing naar de inode.

Een ext2-bestandsnaam kan tot 255 tekens bevatten. Zoals gezegd is een directory een bestand met daarin een lijst va bestandsnamen gekoppeld aan een inode. Het inode-nummer wordt opgeslagen, gevolgd door 2 variabelen die respectievelijk de lengte van de entry en de naam aangeven. Als laatste wordt de eigenlijke bestandsnaam opgeslagen. Doordat de lengte van de bestandsnaam gekend is kan er naar het volgende element gesprongen worden.

Bestanden kunnen in ext2 niet-aaneengesloten gealloceerd worden. In de inode is ruimte voorzien om de adressen van 13 blokken te bewaren. De eerste 10 zijn directe blokadressen, die verwijzen naar datablokken. Het 11de adres verwijst naar een indexblok, waarin de adressen van datablokken opgeslagen kunnen worden. Het 12de en 13de adres in de inode zijn respectievelijk dubbel en driedubbel indirect. Het 11de adres verwijs dus naar een datablok met daarin adressen van indexblokken.

Het grote voordeel van deze structuur is dat de adresinformatie voor kleine bestanden volledig in het werkgeheugen beschikbaar is. Met bestanden die maximaal 10 datablokken in beslag nemen kan dus erg snel gewerkt worden. De maximale bestandsgrootte in bytes wordt gegeven door:

- $Smax = [12 + (b/4) + (b/4)^2 + (b/4)^3]b$

Vrije blokken en inodes

In het superblok worden tabellen bijgehouden met de adressen van een aantal vrije datablokken en een aantal vrije inodes. In deze tabellen is echter niet voldoende plaats om alle vrije elementen te bewaren. Wanneer bijvoorbeeld een bestand moet worden uitgebreid, wordt een adres van een vrij blok uit de tabel in het superblok genomen. Bij het verwijderen van een bestand worden de adressen van de vrijgekomen blokken in de tabel ingeschreven. Indien de tabel leeg is op het ogenblik dat een blok aan de file moet worden toegekend zal de tabel uit het eerste blok in de gelinkte lijst naar het superblok worden gekopieerd. Het hierdoor vrijgekomen blok wordt aan het bestand toegevoegd. Als de tabel helemaal gevuld is met adressen, en er wordt een blok vrijgegeven, dan wordt de tabel uit het superblok naar het vrijgekomen blok gekopieerd, waarna dit als eerste blok in de gelinkte lijst wordt opgenomen.

Voor vrije inodes wordt een gelijkaardige methode gebruikt. In het superblok staat een tabel voor 100 nummers van vrije inodes. Wanneer een nieuw bestand wordt aangemaakt vindt men hier het nummer van een beschikbare inode. Bij het verwijderen van een bestand wordt het nummer van de vrijgekomen inode toegevoegd aan de tabel.

Het veld voor het bestandstype staat op 0 als de inode niet in gebruik is. Wanneer de tabel in het superblok leeg is, kan de kernel de i-list doorlopen en de nummers van de 100 eerste vrije inodes in de tabel plaatsen.

Om het zoeken naar vrije inodes zo efficiënt mogelijk te laten verlopen, wordt de positie van de laatst gevonden vrije inode apart bijgehouden, om de volgende zoektocht vanop die positie te kunnen starten. Wanneer er een inode vrijkomt door het verwijderen van een bestand zijn er 2 mogelijkheden. Als de positie van deze inode voor de opgeslagen positie komt wordt de opgeslagen positie erdoor vervangen. Als de positie erna komt moet er niets gebeuren, want dan wordt de nieuwe vrije inode sowieso gevonden bij één van de volgende zoektochten. Harde en zachte links

Omdat de bestandsbeschrijving niet in de directory bewaard wordt, kan men op een eenvoudige en effectieve manier eenzelfde bestand via verschillende paden aanspreken. Men laat in dit geval 2 bestanden verwijzen naar eenzelfde inode, en dus ook naar hetzelfde bestand. Dit wordt een hard link genoemd.

Voor symbolische links wordt verwezen naar een ander pad, niet naar een inode.

Bij het verwijderen mogen de inode en datablokken pas verwijderd worden als er geen enkel link meer is. Daarom wordt in een inode een teller met het aantal links bijgehouden. Bij symbolic links is dit niet nodig. Wanneer het bestand waarnaar een symbolic link verwijst verwijderd wordt, blijft de link bestaan. Deze wordt dan wees of orphan genoemd.

Bijzonderheden

Ext 2 heeft enkele specifieke eigenschappen.

Het aantal inodes wordt bepaald bij het aanmaken van het bestandssysteem. Het kan gebeuren dat je geen nieuw bestand kan aanmaken, hoewel er toch datablokken beschikbaar zijn. Een ext2 bestandssysteem heeft dus een maximum aantal bestanden. Standaard wordt er één inode aangemaakt voor iedere 4096 bytes beschikbare schijfruimte. Als de gemiddelde grootte van je bestanden kleiner is dan 4 kB, kan je dus snel met een tekort aan inodes kampen.

Bij het creëren van het bestandssysteem wordt een deel (standaard 5%) toegekend aan de beheerder of root. Zo kan een gebruiker niet alles volschrijven. Een laatste bijzonder bestandstype is de zogenaamde device file. Deze bestanden bevatten geen data, maar stellen een systeemapparaat voor. Zo kan men communiceren met een apparaat door te lezen of schrijven van of naar de bijhorende device file.

Revision #1
Created 16 June 2021 20:39:50 by Jasper G.
Updated 3 December 2021 22:13:08 by Jasper G.