Large Disk HOWTO

Andries Brouwer, `aeb@cwi.nl` Vertaald door: Ellen Bokhorst, `bokkie@nl.linux.org` v2.2m, 15 febr 2000

Alles over diskgeometrie en de 1024 cylinder limiet voor disks. Zie *www.win.tue.nl* voor de meest recente versie van deze tekst.

1 Het probleem

Veronderstel dat je een disk hebt met meer dan 1024 cylinders. Veronderstel bovendien dat je een besturingssysteem hebt dat gebruik maakt van de oude INT13 BIOS interface naar disk I/O. Dan heb je een probleem, omdat deze interface een 10-bit veld voor de cylinder gebruikt, waarop de I/O wordt gedaan, dus dat cylinders 1024 en daarboven niet toegankelijk zijn.

Gelukkig maakt Linux geen gebruik van de BIOS, dus is er geen probleem.

Behalve voor twee zaken:

(1) Als je je systeem boot, draait Linux nog niet en kan 't je de BIOS problemen niet besparen. Dit heeft een aantal consequenties voor LILO en vergelijkbare bootloaders.

(2) Het is nodig voor alle besturingssystemen die een disk gebruiken om het er over eens te zijn waar de partities zich bevinden. Met andere woorden, als je zowel Linux gebruikt en, laten we zeggen, DOS op één disk, dan moeten beiden de partitietabel op dezelfde wijze interpreteren. Dit heeft een aantal consequenties voor de Linux kernel en voor `fdisk`.

Hieronder staat een nogal gedetailleerde beschrijving van alle relevante details. Merk op dat ik kernel versie 2.0.8 bron als referentie gebruikte. Andere versies zouden een beetje kunnen verschillen.

2 Samenvatting

Je hebt een nieuwe grote disk. Wat nu? Aan de software kant: gebruik `fdisk` (of, nog beter, `cfdisk`) om partities aan te maken en vervolgens `mke2fs` om een bestandssysteem aan te maken, en `mount` het nieuwe bestandssysteem dan om het aan de grote bestandshiërarchie te koppelen.

Ongeveer een jaar geleden kon ik schrijven: Je hoeft deze HOWTO niet te lezen aangezien er tegenwoordig *geen* problemen zijn met grote harddisks. De overgrote meerderheid van ogenschijnlijke problemen wordt veroorzaakt door mensen die denken dat er een probleem zou kunnen zijn en die een diskmanager installeren, of in de expert mode van `fdisk` gaan, of de diskgeometries expliciet aan LILO of op de kernel opdrachtregel opgeven.

Kenmerkende probleemgebieden zijn echter: (i) oude hardware, (ii) verscheidene besturingssystemen op dezelfde disk, en soms (iii) het booten.

Tegenwoordig is de situatie wat slechter. Misschien dat 2.3.21 en later weer voor alle disks goed zal zijn.

Advies:

Voor grote SCSI disks: Linux heeft ze vanaf het allereerste begin ondersteund. Geen actie vereist.

Voor grote IDE disks (van meer dan 8.4 GB): zorg dat je aan een recente stabiele kernel komt (2.0.34 of later). Meestal zal alles nu prima in orde zijn, vooral als je zo verstandig bent de BIOS niet om diskvertalingen zoals LBA en dergelijke te vragen.

Zie 12.1 (IDE problemen met 34+ GB disks) hieronder voor zeer grote IDE-disks (meer dan 33.8 GB).

Als LILO tijdens het opstarten blijft hangen, geef dan ook `5.1 (linear)` op in het configuratiebestand `/etc/lilo.conf`.

Het kan zijn dat er geometrie problemen zijn die kunnen worden opgelost door een expliciete geometrie op te geven aan de kernel/LILO/fdisk.

Als je een oude `fdisk` hebt en het geeft een waarschuwing over 6 (overlappende) partities: negeer dan de waarschuwingen, of gebruik `cfdisk` om te controleren dat alles in orde is.

Zie de *Linux HPT366 HOWTO* voor HPT366.

Als je denkt dat er iets mis is met de grootte van je disk, zorg er dan voor dat je binaire en decimale 3 (eenheden), niet met elkaar verwart en realiseer je dat de vrije ruimte die `df` op een lege disk rapporteert vanwege administratieve overhead een paar procent kleiner is dan de partitiegrootte.

Als je nu nog steeds denkt dat er problemen zijn of gewoon nieuwsgierig bent, lees dan verder.

3 Eenheden en Grootte

Een kilobyte (kB) is 1000 bytes. Een megabyte (MB) is 1000 kB. Een gigabyte (GB) is 1000 MB. Een terabyte (TB) is 1000 GB. Dit is de *SI norm*. Er zijn echter mensen die 1 MB=1024000 bytes gebruiken en het hebben over 1.44 MB diskettes, en mensen die denken dat 1 MB=1048576 bytes. Hier volg ik de *recente standaard* en schrijf Ki, Mi, Gi, Ti voor de binaire eenheden, dus dat deze diskettes 1440 KiB (1.47 MB, 1.41 MiB) zijn, 1 MiB is 1048576 bytes (1.05 MB), 1 GiB is 1073741824 bytes (1.07 GB) en 1 TiB is 1099511627776 bytes (1.1 TB).

Geheel correct volgen fabrikanten van harddisks de SI norm en maken ze gebruik van decimale eenheden. De Linux bootmeldingen en een aantal fdisk-type programma's gebruiken echter de symbolen MB en GB voor binaire of gemengde binaire-decimale eenheden. Bereken dus eerst de werkelijke grootte in decimale eenheden (of gewoon in bytes) voordat je denkt dat je disk kleiner is dan toegezegd toen je hem kocht.

Betreft terminologie en afkortingen voor binaire eenheden, heeft *Knuth* een alternatief *voorstel*, namelijk het gebruiken van KKB, MMB, GGB, TTB, PPB, EEB, ZZB, YYB en om deze te noemen *large kilobyte*, *large megabyte*, ... *large yottabyte*. Hij schrijft: 'Merk op dat het verdubbelen van de letter zowel betekent binary-ness en large-ness.' Dit is een goed voorstel - 'large gigabyte' klinkt beter dan 'gibibyte'. Voor ons doel is echter het belangrijkst er de nadruk op te leggen dat een megabyte precies 1000000 bytes heeft, en dat er één of andere, andere term en afkorting nodig is als je iets anders bedoelt.

3.1 Sectorgrootte

In de huidige tekst is een sector 512 bytes. Dit is bijna altijd waar, maar bijvoorbeeld bepaalde MO disks gebruiken een sectorgrootte van 2048 bytes en moeten alle capaciteiten hieronder met vier worden vermenigvuldigd. (Als je `fdisk` met dergelijke disks gebruikt, zorg er dan voor dat je versie 2.9i of later hebt en geeft de optie '-b 2048' op.)

3.2 Diskgrootte

Een disk met C cylinders, H heads en S sectoren per spoor heeft in totaal C*H*S, en kan C*H*S*512 bytes opslaan. Als er op het disklabel bijvoorbeeld staat C/H/S=4092/16/63 dan bestaat de disk uit 4092*16*63=4124736 sectoren, en kan 4124736 *512=2111864832 bytes (2.11 GB) bewaren. Er is een industrie-akkoord om disks groter dan 8.4 GB C/H/S=16383/16/63 te geven, en de diskgrootte kan niet langer van de C/H/S waarden die door de disk worden aangegeven, worden afgelezen.

4 Disk Toegang

Om iets van disk te kunnen lezen of naar disk te kunnen schrijven, moeten we een positie op de disk aangeven, door bijvoorbeeld het geven van een sector- of bloknummer. Als de disk een SCSI-disk is, dan gaat dit sectornummer direct in de SCSI-opdracht en wordt door de disk begrepen. Als de disk een IDE-disk is, die van LBA gebruik maakt, dan geldt precies hetzelfde. Maar als het een oude, RLL of MFM of IDE-disk van voor de LBA-tijd is, dan verwacht de diskhardware een drietal (cylinder, head, sector) om de gewenste plek op de disk aan te duiden.

De overeenkomst tussen de lineaire nummering en deze 3D notatie is als volgt: voor een disk met C cylinders, H heads en S sectoren/spoor is de positie (c,h,s) in 3D of CHS notatie hetzelfde als positie $c*H*S + h*S + (s-1)$ in lineaire of LBA notatie. (De min één is omdat sectoren traditioneel worden geteld vanaf 1, niet 0, in deze 3D notatie.)

Dus om een zeer oude niet-SCSI disk te benaderen hebben we z'n *geometrie* nodig, dat wil zeggen, de waarden van C, H en S.

4.1 BIOS Disk Toegang en de 1024 cylinder limiet

Linux maakt geen gebruik van de BIOS, maar een aantal andere systemen doen dit wel. De BIOS die dateert van voor de LBA tijd, biedt disk I/O routines die (c,h,s) als invoer hebben. (Preciezer: In `AH` staat de functie die moet worden uitgevoerd, `CH` bestaat uit de lage 8 bits van het cylindernummer, in `CL` de bits 7-6 staan de hoge twee bits van het cylindernummer en in bits 5-0 het sectornummer, `DH` is het head nummer, en `DL` is het drive nummer (80h of 81h). Hiermee wordt een deel van de lay-out van de partitietabel verduidelijkt.)

Op deze manier hebben we CHS in drie bytes gecodeerd, met 10 bits voor het cylindernummer, 8 bits voor het head nummer, en 6 bits voor het spoorsectornummer (genummerd 1-63). Hieruit volgt dat cylindernummers voor kunnen komen in het bereik van 0 tot 1023 en dat er niet meer dan 1024 cylinders BIOS-adresseerbaar zijn.

De DOS en Windows software wijzigde niet toen IDE disks met LBA ondersteuning werden geïntroduceerd, dus DOS en Windows hadden nog steeds een diskgeometrie nodig, zelfs toen dit niet langer nodig was voor de feitelijke disk I/O, maar alleen maar om met de BIOS te kunnen communiceren. Dit betekent weer dat Linux de geometrie nodig heeft in die situaties, zelfs met een moderne disk, waar communicatie met de BIOS of met andere besturingssystemen is vereist.

Deze kwestie duurde tot ongeveer 4 jaar geleden en toen verschenen er disks op de markt die niet met de INT13 functies konden worden geadresseerd (omdat de 10+8+6=24 bits voor (c,h,s) niet meer dan 8.5 GB kunnen adresseren) en er werd een nieuwe BIOS interface ontworpen: de zogenoemde Extended INT13 functies, waarin DS:SI verwijst naar een 16-byte Disk Adres Packet dat een 8-byte beginnnend absoluut bloknummer bevat.

De Microsoft wereld beweegt zich heel langzaam in de richting van het gebruiken van deze Extended INT13 functies. Waarschijnlijk zal over een paar jaar geen modern systeem op moderne hardware het concept 'diskgeometrie' nog nodig hebben.

4.2 Historie van BIOS en IDE limieten

ATA Specificatie (voor IDE disks) - de 137 GB limiet

Voor een maximale totale capaciteit van 267386880 sectoren (van elk 512 bytes), dat wil zeggen 136902082560 bytes (137 GB), ten hoogste 65536 cylinders (genummerd 0-65535), 16 heads (genummerd 0-15), 255 sectoren/spoor (genummerd 1-255). Dit is net nog geen probleem (in 1999) maar dat zal het over een aantal jaren wel zijn.

BIOS Int 13 - de 8.5 GB limiet

Voor een maximum totale capaciteit van 8455716864 bytes (8.5 GB) op z'n hoogst 1024 cylinders (genummerd 0-1023), 256 heads (genummerd 0-255), 63 sectoren/spoor (genummerd 1-63). Dit is tegenwoordig een serieuze beperking. Het betekent dat DOS de huidige grote disks niet kan gebruiken.

De 528 MB limiet

Als dezelfde waarden voor c,h,s voor de BIOS Int 13 call en voor de IDE disk I/O worden gebruikt, dan combineren beiden beperkingen, en kan men maximaal 1024 cylinders, 16 heads, 63 sectoren/spoor gebruiken, voor een maximale capaciteit van 528482304 bytes (528 MB), de schandelijke 504 MiB limiet voor DOS met een oude BIOS. Dit begon een probleem te worden rond 1993, en mensen namen hun toevlucht tot allerlei soorten bedotterij, zowel in hardware (LBA) als in firmware (BIOS vertalingen), en in software (diskmanagers). Het concept 'vertaling' werd uitgevonden (1994): een BIOS zou een geometrie kunnen gebruiken bij het communiceren met de drive, en een andere nepgeometrie bij het communiceren met DOS, en tussen die twee vertalen.

De 2.1 GB limiet (April 1996)

Een aantal oudere BIOSsen wijzen slechts 12 bits toe aan het veld in CMOS RAM waarin het aantal cylinders wordt aangegeven. De consequentie hiervan is, dat dit aantal maximaal 4095 kan zijn, en er slechts 4095*16*63*512=2113413120 bytes toegankelijk zijn. Als je een grotere disk hebt, heeft dit als effect dat het systeem tijdens het booten blijft hangen. Dit zorgde ervoor dat disks met een geometrie van 4092/16/63 nogal populair waren. En tegenwoordig komen er nog steeds vele grote diskdrives met een jumper om ze in te stellen als 4092/16/63. Zie ook *over2gb.htm*. *Andere BIOSsen* zouden niet blijven hangen, maar detecteren slechts een veel kleinere disk, zoals 429 MB in plaats van 2.5 GB.

De 3.2 GB limiet

In de Phoenix 4.03 en 4.04 BIOS firmware zat een bug die veroorzaakte dat het systeem vastliep in de CMOS setup bij drives met een capaciteit groter dan 3277 MB. Zie *over3gb.htm*.

De 4.2 GB limiet (Feb 1997)

Eenvoudige BIOS vertaling (ECHS=Extended CHS, soms 'Large disk ondersteuning' of gewoon 'Large' genoemd) werkt door het aantal heads dat aan DOS wordt getoond herhalend te verdubbelen en het aantal cylinders te halveren totdat het aantal cylinders maximaal 1024 is. Nu kunnen DOS en Windows 95 niet met 256 heads omgaan, en in het algemene geval dat de disk 16 heads rapporteert, betekent dit dat dit eenvoudige mechanisme alleen werkt tot 8192*16*63*512=4227858432 bytes (met een nep geometrie van 1024 cylinders, 128 heads, 63 sectoren/spoor). Merk op dat ECHS het aantal sectoren per spoor niet wijzigt, dus als dit niet gelijk is aan 63, zal de limiet lager zijn. Zie *over4gb.htm*.

De 7.9 GB limit

Iets slimmere BIOSsen voorkomen het voorgaande probleem door eerst het aantal heads aan te passen tot 15 ('gereviseerde ECHS'), zodat een nepgeometrie met 240 heads kan worden verkregen, goed voor 1024 *240*63*512=7927234560 bytes.

De 8.4 GB limit

Als laatste, als de BIOS alles doet om deze vertaling tot een succes te maken, en 255 heads en 63 sectoren/spoor gebruikt ('geassisteerde LBA' of gewoon 'LBA') kan het 1024*255*63*512=8422686720 bytes bereiken, iets minder dan de eerdere 8.5 GB limiet omdat de geometries met 256 heads moeten worden vermeden. (Deze vertaling zal voor het aantal heads de eerste waarde H in de reeks 16, 32, 64, 128, 255 gebruiken waarvan de totale diskcapaciteit past in 1024*H*63*512, en berekent het aantal cylinders C als totale capaciteit gedeeld door (H*63*512).)

De 33.8 GB limiet (augustus 1999)

> De volgende horde komt met een grootte van meer dan 33.8 GB. Het probleem is dat met de standaard 16 heads en 63 sectors/track dit overeenkomt met een aantal cylinders van meer dan 65535, wat niet in een short past. De meeste tegenwoordig bestaande BIOSsen kunnen met dergelijke disks niet omgaan. (Zie, b.v., *Asus upgrades* voor nieuwe flash images die wel werken). Linux kernels ouder dan 2.2.14 / 2.3.21 hebben een patch nodig. Zie 12.1 (IDE problemen met 34+ GB disks) hieronder.

Zie, *Breaking the Barriers*, en met meer details, *IDE Hard Drive Capacity Barriers* voor een andere bespreking over dit onderwerp.

Van harddrives van meer dan 8.4 GB wordt verondersteld dat ze hun geometrie als 16383/16/63 rapporteren. Dit betekent in feite dat de 'geometrie' verouderd is, en de totale diskgrootte niet langer vanuit de geometrie berekend kan worden.

5 Booten

Als het systeem wordt opgestart, leest de BIOS sector 0 in (bekend als de MBR - de Master Boot Record) vanaf de eerste disk (of vanaf diskette of CDROM), en springt naar de daar gevonden code - meestal een bootstrap loader. Deze kleine bootstrap programma's die daar worden gevonden hebben kenmerkend geen eigen diskdrivers en gebruiken BIOS services. Dit betekent dat een Linux kernel slechts kan worden geboot als het zich volledig binnen de eerste 1024 cylinders bevindt.

Dit probleem kan makkelijk worden opgelost: zorg ervoor dat de kernel (en misschien andere bestanden die tijdens het opstarten worden gebruikt, zoals LILO map bestanden) zich op een partitie bevinden die zich volledig binnen de eerste 1024 cylinders bevindt van een disk waartoe de BIOS toegang heeft - waarschijnlijk betekent dit de eerste of tweede disk.

Dus: maak een kleine partitie aan, laten we zeggen ter grootte van 10 MB, zodanig dat er ruimte is voor een handjevol kernels, ervoor zorgend dat het zich volledig bevindt binnen de eerste 1024 cylinders van de eerste of tweede disk. Mount het op `/boot` zodat LILO zijn zaken hier neer zal zetten.

5.1 LILO en de 'linear' optie

Een ander punt is dat de boot loader en de BIOS het eens moeten zijn over de diskgeometrie. LILO ondervraagt de kernel naar de geometrie, maar meer en meer auteurs van diskdrivers volgen de slechte gewoonte een geometrie vanuit de partitietabel af te leiden, in plaats van LILO te vertellen wat de BIOS zal gebruiken. Dus vaak is de geometrie, waarin door de kernel is voorzien, waardeloos. In dergelijke gevallen helpt het om aan LILO de '`linear`' optie mee te geven. Het effect hiervan is dat LILO geen geometrie informatie nodig heeft op het moment dat de bootloader zich installeert (het bewaart lineaire adressen in de maps) maar doet de conversie van lineaire adressen niet tijdens het booten. Waarom is dit niet de standaard? Er is één nadeel: met de 'linear' optie, is LILO niet langer bekend met cylindernummers, en kan je daarom niet waarschuwen als een deel van de kernel boven de 1024 cylinder limiet is opgeslagen, en zou je kunnen eindigen met een systeem dat niet boot.

5.2 Een LILO bug

Met LILO versies lager dan v21 is er nog een ander nadeel: de adresconversie die tijdens de systeemstart wordt uitgevoerd bevat een bug: als c*H gelijk is aan 65536 of meer, vindt in de berekening overflow plaats. Bij H groter dan 64 zorgt dit voor een strictere limiet voor c dan de bekende $c < 1024$; met bijvoorbeeld H=255 en een oude LILO moet men een $c < 258$ hebben (c=cylinder waar de kernel-image voorkomt, H=nummer van diskkoppen).

5.3 1024 cylinders is niet gelijk aan 1024 cylinders

Tim Williams schrijft: 'Ik had mijn Linux partitie binnen de eerste 1024 cylinders en nog steeds wilde het niet booten. Toen ik het eerst onder de 1 GB verplaatste werkte het.' Hoe kan dat? Dit was een SCSI disk met AHA2940UW controller die of H=64, S=32 gebruikt (dat wil zeggen, cylinders van 1 MiB = 1.05 MB), of H=255, S=63 (dat wil zeggen, cylinders van 8.2 MB), afhankelijk van setup opties in firmware en in de BIOS. Geen twijfel dat de BIOS van het eerste uitging, zodat de 1024 cylinder limiet werd gevonden op 1 GiB, terwijl Linux de laatste gebruikte en LILO dacht dat deze limiet op 8.4 GB zat.

6 Disk geometrie, partities en 'overlap'

Als er zich verscheidene besturingssystemen op je disks bevinden, dan gebruikt ieder besturingssysteem één of meer diskpartities. Een meningsverschil over waar deze partities zich bevinden kan rampzalige gevolgen hebben.

In de MBR bevindt zich een *partitietabel* die beschrijft waar de (primaire) partities zijn. Er zijn 4 tabelingangen, voor 4 primaire partities, en elk ziet er ongeveer zo uit

```
struct partition {
        char active;    /* 0x80: bootable, 0: niet bootable */
        char begin[3];  /* CHS voor eerste sector */
        char type;
        char end[3];    /* CHS voor laatste sector */
        int start;      /* 32 bit sector nummer (tellend vanaf 0) */
        int length;     /* 32 bit aantal sectoren */
};
```

(waar CHS staat voor Cylinder/Head/Sector).

Dit is overtollige informatie: de lokatie van een partitie wordt zowel door de 24-bit `begin` en `eind` velden gegeven, als door de 32-bit `start` en `lengte` velden.

Linux gebruikt alleen de `start` en `lengte` velden, en kan daarom slechts partities van niet meer dan 2^32 sectoren beheren, dat wil zeggen, partities van maximaal 2 TiB. Dat is zestig keer groter dan de disks die heden ten dage beschikbaar zijn, dus misschien zal het voor ongeveer de volgende zeven jaar voldoende zijn. (Dus partities kunnen erg groot zijn, maar er is een serieuze beperking dat een bestand in een ext2 bestandssysteem op hardware met 32-bit integers niet groter kan zijn dan 2 GiB.)

DOS gebruikt de `begin` en `eind` velden, en gebruikt de BIOS INT13 call om de disk te benaderen, en kan daarom slechts disks van niet meer dan 8.4 GB beheren, zelfs met een vertalende BIOS. (Partities kunnen niet groter zijn dan 2.1 GB vanwege beperkingen van het FAT16 bestandssysteeem.) Hetzelfde geldt voor Windows 3.11 en WfWG en Windows NT 3.*.

Windows 95 geeft ondersteuning voor Extended INT13 interface, en gebruikt speciale partitietypes (c, e, f in plaats van b, 6, 5) om aan te geven dat een partitie op deze manier zou moeten worden benaderd. Als deze partitietypes worden gebruikt, bevatten de `begin` en `eind` velden dummy informatie (1023/255/63). Windows 95 OSR2 introduceert het FAT32 bestandssysteem (partitie type b of c), die partities ter grootte van maximaal 2 TiB toestaat.

Wat is de nonsens die je van `fdisk` krijgt over 'overlappende' partities, als er in feite niets mis is? Er is iets 'mis': als je kijkt naar de `begin` en `eind` velden van dergelijke partities, zoals DOS doet, overlappen ze. (En dat kan niet worden gecorrigeerd, omdat deze velden geen cylindernummers boven 1024 op kunnen slaan - er zal altijd 'overlap' zijn zodra je meer dan 1024 cylinders hebt.) Als je echter kijkt naar de `start` en `lengte` velden, zoals Linux doet, en zoals Windows 95 doet in het geval van partities type c, e of f, dan is alles goed. Dus negeer deze waarschuwingen als `cfdisk` tevreden is en je een alleen-Linux disk hebt. Wees voorzichtig als de disk met DOS wordt gedeeld. Gebruik de opdrachten `cfdisk -Ps /dev/hdx` en `cfdisk -Pt /dev/hdx` om de partitietabel van `/dev/hdx` te bekijken.

7 Vertaling en Disk Managers

Disk geometrie (met heads, cylinders en sporen) is iets uit de tijd van MFM en RLL. In die dagen correspondeerde het met een fysieke werkelijkheid. Tegenwoordig, met IDE of SCSI, is niemand meer geïnteresseerd in wat de 'werkelijke' geometrie van een disk is. Inderdaad, het aantal sectoren per spoor is variabel - er zijn dichtbij de buitenste rand van de disk meer sectoren per spoor - dus er is geen 'werkelijk' aantal sectoren per spoor. Geheel tegengesteld: de IDE opdracht INITIALIZE DRIVE PARAMETERS (91h) dient om de disk te vertellen hoeveel heads en sectoren per spoor het tegenwoordig verondersteld is te hebben. Het is heel normaal om een grote moderne disk te zien met 2 heads die 15 of 16 heads aan de BIOS rapporteert, terwijl de BIOS weer 255 heads aan de gebruikerssoftware kan rapporteren.

Voor de gebruiker is het 't beste een disk in acht te nemen als gewoon een lineaire array met sectoren genummerd 0,1, ..., en het aan de controller over te laten om uit te zoeken waar een gegeven sector zich op de disk bevindt. Deze lineaire nummering wordt LBA genoemd.

Het begripsplaatje is nu als volgt. DOS, of een bepaalde boot loader, praat tegen de BIOS, door gebruik te maken van (c,h,s) notatie. De BIOS converteert (c,h,s) naar LBA notatie door gebruik te maken van de nep geometrie die de gebruiker gebruikt. Als de disk LBA accepteert dan wordt deze waarde gebruikt voor disk I/O. Anders wordt het terug geconverteerd naar (c',h',s') de geometrie gebruikend die de disk heden gebruikt, en die voor disk I/O wordt gebruikt.

Merk op dat er wat verwarring is over het gebruik van de uitdrukking 'LBA': Als een term die diskcapaciteiten beschrijft betekent het 'Lineaire Blok Adressering' (als tegengestelde van CHS Adressering). Als een term in de BIOS setup, beschrijft het een vertalingsschema die soms 'ondersteunde LBA' wordt genoemd - zie hierboven onder '4.2 (De 8.4 GB limiet)'.

Iets vergelijkbaars werkt als de firmware geen LBA spreekt maar de BIOS bekend is met vertaling. (In de setup wordt dit vaak aangegeven met 'Large'.) Nu zal de BIOS een geometrie (C,H,S) presenteren aan het besturingssysteem en (C',H',S') gebruiken als het met de diskcontrolleer communiceert. Meestal S = S', C = C'/N en H = H'*N, waar N de kleinste kracht van de twee is zal verzekeren dat C' <= 1024 (dus dat de minste capaciteit wordt verspild door het naar beneden afronden van C' = C/N). Dit geeft weer toegang tot 8.4 GB (7.8 GiB).

(De derde setup optie is gewoonlijk 'Normal', waarbij van geen vertaling sprake is.)

Als een BIOS niet bekend is met 'Large' of 'LBA', dan zijn er software oplossingen voorhanden. Disk Managers zoals OnTrack of EZ-Drive vervangen de BIOS disk handling routines door hun eigen routines. Vaak wordt dit bewerkstelligd door de diskmanager-code in de MBR en opeenvolgende sectoren te plaatsen (OnTrack noemt deze code DDO: Dynamic Drive Overlay), zodanig dat het voor enig ander besturingssysteem wordt opgestart. Daarom kan men problemen tegenkomen als men vanaf een diskette opstart als er een Disk Manager is geïnstalleerd.

Het effect is min of meer hetzelfde als met een vertalende BIOS - maar vooral als er verschillende besturingssystemen vanaf dezelfde disk worden gedraaid, kunnen diskmanagers voor heel wat problemen zorgen.

Linux geeft sinds versie 1.3.14 ondersteuning voor OnTrack Disk Manager en EZ-Drive sinds versie 1.3.29. Wat meer details worden hieronder gegeven.

8 Kernel disk vertaling voor IDE disks

Als de Linux kernel de aanwezigheid van een bepaalde diskmanager op een IDE disk detecteert, zal het proberen de disk opnieuw op dezelfde manier in te delen als deze diskmanager zou hebben gedaan, zodat Linux dezelfde diskpartitionering ziet als bijvoorbeeld DOS met OnTrack of EZ-Drive. Er zal echter geen herindeling plaatsvinden als er een geometrie op de opdrachtregel werd opgegeven - dus een '`hd=`*cyls*`,`*heads*`,`*secs*' opdrachtregeloptie zou heel goed compatibiliteit met een diskmanager kunnen ruïneren.

Als dit je treft, en je weet iemand die een nieuwe kernel voor je kan compileren, zoek dan naar het bestand `linux/drivers/block/ide.c` en verwijder in de routing `ide_xlate_1024()` de test `if (drive->forced_geom) {...;return 0; }`.

De herindeling wordt gedaan door het uitproberen van 4, 8, 16, 32, 64, 128, 255 heads (waarbij H*C constant wordt gehouden) totdat of C <= 1024 of H = 255.

De details zijn als volgt - subsectie titelkoppen bestaan uit de tekst die in de corresponderende bootmeldingen verschijnen. Hier en elders in de tekst worden partitietypes in hexidecimaal gegeven.

8.1 EZD

EZ-Drive wordt gedetecteerd door het feit dat de eerste primaire partitie type 55 heeft. De geometrie wordt heringedeeld zoals hierboven beschreven, en de partitietabel is vanaf sector 0 opzijgezet - in plaats daarvan wordt de partitietabel vanaf sector 1 ingelezen. Diskbloknummers zijn niet gewijzigd, maar schrijfopdrachten naar sector 0 worden doorgestuurd naar sector 1. Dit gedrag kan worden gewijzigd door de kernel te hercompileren met `#define FAKE_FDISK_FOR_EZDRIVE 0` in `ide.c`.

8.2 DM6:DDO

OnTrack DiskManager (op de eerste disk) wordt gedetecteerd door het feit dat de eerste primaire partitie type 54 heeft. De geometrie is heringedeeld als hierboven beschreven en de volledige disk is met 63 sectoren verschoven (zodat de oude sector 63 sector 0 wordt). Daarna wordt een nieuwe MBR (met partitietabel) vanaf de nieuwe sector 0 ingelezen. Natuurlijk is deze verschuiving om ruimte voor de DDO te maken - daarom is er geen verschuiving op andere disks.

8.3 DM6:AUX

OnTrack DiskManager (op andere disks) wordt gedetecteerd door het feit dat de eerste primaire partitie het type 51 of 53 heeft. De geometrie is heringedeeld als hierboven beschreven.

8.4 DM6:MBR

Een oudere versie van OnTrack DiskManager wordt niet gedetecteerd door het partitietype maar door een kenmerk. (Test of de offset die in bytes 2 en 3 van de MBR wordt gevonden, niet meer is dan 430, en of de short die op deze offset wordt gevonden gelijk is aan 0x55AA, en wordt gevolgd door een even byte.) Weer wordt de geometrie heringedeeld als hierboven.

8.5 PTBL

Als laatste is er een test dat een vertaling van de `start` en `eind` waarden van de primaire partities probeert af te leiden: Als een bepaalde partitie een start- en eind sectornummer van respectievelijk 1 en 63 heeft, en end heads 31, 63, 127 of 254 dan, aangezien het gebruikelijk is om partities op een cylindergrens te beëindigen, en aangezien de IDE interface bovendien maximaal 16 heads gebruikt, wordt verondersteld dat er een BIOS vertaling actief is, en de geometrie is heringedeeld dat het respectievelijk 32, 64, 128 of 255 heads gebruikt. Er heeft echter geen herindeling plaatsgevonden als het huidige idee van de geometrie reeds 63 sectoren per spoor heeft en tenminste zoveel heads (aangezien dit waarschijnlijk betekent dat er reeds een herindeling heeft plaatsgevonden).

9 Consequenties

Wat betekent dit allemaal? Voor Linux gebruikers slechts één ding: dat ze ervoor moeten zorgen dat LILO en `fdisk` de juiste geometrie gebruiken waar met 'juist' voor `fdisk` de gebruikte geometrie wordt bedoeld die door andere besturingssystemen op dezelfde disk worden gebruikt, en voor LILO de geometrie die tijdens het opstarten een succesvolle interactie met de BIOS activeert. (Gewoonlijk komen deze twee overeen.)

Hoe weet `fdisk` van de geometrie? Het vraagt het de kernel, door gebruik te maken van de `HDIO_GETGEO` ioctl. Maar de gebruiker kan de geometrie interactief of op de opdrachtregel overschrijven.

Hoe weet LILO van de geometrie? Het vraagt het de kernel, door gebruik te maken van de `HDIO_GETGEO` ioctl. Maar de gebruiker kan de geometrie overschrijven door gebruik te maken van de '`disk=`' optie in `/etc/lilo.conf` (zie lilo.conf(5)). Men kan ook de `linear` optie aan LILO opgeven, en het zal LBA adressen in zijn mapbestand opslaan in plaats van CHS adressen, en de te gebruiken geometrie tijdens het booten uitzoeken (door de INT 13 Functie 8 gebruiken om te vragen naar de geometrie).

Hoe weet de kernel wat te antwoorden? Als eerste kan de gebruiker een expliciete geometrie hebben opgegeven met een '`hda=`*cyls*`,`*heads*`,`*secs*' kernel opdrachtregeloptie (zie bootparam(7)), misschien handmatig, of door de bootloader te vragen een dergelijke optie aan de kernel te leveren. En anders zal de kernel ernaar raden, waarschijnlijk door waarden te verkrijgen van de BIOS of de hardware.

Het is (sinds Linux 2.1.79) mogelijk de ideeën van de kernel over de geometrie te wijzigen door gebruik te maken van het `/proc` bestandssysteem. Bijvoorbeeld

```
# sfdisk -g /dev/hdc
/dev/hdc: 4441 cylinders, 255 heads, 63 sectors/track
# cd /proc/ide/ide1/hdc
# echo bios_cyl:17418 bios_head:128 bios_sect:32 > settings
# sfdisk -g /dev/hdc
/dev/hdc: 17418 cylinders, 128 heads, 32 sectors/track
#
```

9.1 Berekenen van LILO parameters

Soms is het handig een bepaalde geometrie te forceren door op de kernel opdrachtregel '`hda=`*cyls*`,`*heads*`,`*secs*' toe te voegen. Men wil bijna altijd *secs*=63, en het doel van deze toevoeging is de *heads* te specificeren. (Tegenwoordig zijn redelijke waarden *heads*=16 en *heads*=255.) Wat zou men voor *cyls* op moeten geven? Precies dat aantal dat de juiste totale capaciteit van C*H*S sectoren oplevert. Voor een drive met bijvoorbeeld 71346240 sectoren (36529274880 bytes) zou men C kunnen berekenen als 71346240/(255*63)=4441 (bijvoorbeeld met behulp van het programma `bc`), en als bootparamter `hdc=4441,255,63` op kunnen geven. Hoe weet men wat de juiste totale capaciteit is? Bijvoorbeeld,

```
# hdparm -g /dev/hdc | grep sectors
 geometry     = 4441/255/63, sectors = 71346240, start = 0
# hdparm -i /dev/hdc | grep LBAsects
 CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=71346240
```

geeft twee manieren waarop het totaal aantal sectoren van 71346240 kan worden gevonden. De kerneluitvoer

```
# dmesg | grep hdc
...
hdc: Maxtor 93652U8, 34837MB w/2048kB Cache, CHS=70780/16/63
 hdc: [PTBL] [4441/255/63] hdc1 hdc2 hdc3! hdc4 < hdc5 > ...
```

vertelt ons over (op z'n minst) 34837*2048=71346176 en over (op z'n minst) 70780*16*63=71346240 sectoren. In dit geval blijkt de tweede waarde exact correct te zijn, maar in het algemeen zijn beiden mogelijk naar beneden afgerond. Dit is een goede manier om de diskgrootte te benaderen wanneer `hdparm` niet beschikbaar is. Geef nooit een te grote waarde op voor *cyls*! In het geval van SCSI-disks wordt het precieze aantal sectoren gegeven in de kernel bootmeldingen:

```
SCSI device sda: hdwr sector= 512 bytes. Sectors= 17755792 [8669 MB] [8.7 GB]
```

(en MB, GB zijn naar afgerond, niet naar beneden afgerond, en 'binary').

10 Details

10.1 IDE details - de zeven geometries

De IDE driver heeft vijf bronnen met informatie over de geometrie. De eerste (G_user) is datgene wat door de gebruiker op de opdrachtregel is gespecificeerd. De tweede (G_bios) is de BIOS Fixed Disk Parameter Tabel (voor alleen de eerste en de tweede disk) die bij het opstarten van het systeem wordt ingelezen, voor de overschakeling naar 32-bit mode. De derde (G_phys) en vierde (G_log) worden door de IDE controller geretourneerd als een reactie op de opdracht IDENTIFY - dit zijn de 'fysieke' en 'huidig logische' geometries.

Aan de andere kant heeft de driver twee waarden nodig voor de geometrie: aan de ene kant G_fdisk, geretourneerd door een `HDIO_GETGEO` ioctl, en aan de andere kant G_used, die eigenlijk wordt gebruikt voor het doen van I/O. Beiden, G_fdisk en G_used zijn geïnitialiseerd als G_user als gegeven, aan G_bios als deze informatie overeenkomstig CMOS aanwezig is, en anders aan G_phys. Als G_log er redelijk uitziet dan wordt G_used daarop ingesteld Anders, als G_used redelijk is en G_phys ziet er redelijk dat wordt G_used ingesteld naar G_phys. Hier betekent 'redelijk' dat het aantal heads zich bevindt in het bereik 1-16.

Met andere woorden: de opdrachtregel overschrijft de BIOS en zal vaststellen wat `fdisk` ziet, maar als het een vertaalde geometrie specificeert (met meer dan 16 heads), dan zal de vertaalde geometrie voor kernel I/O door de uitvoer van de opdracht IDENTIFY worden overschreven.

Merk op dat G_bios nogal onbetrouwbaar is: voor systemen die vanaf SCSI booten kunnen de eerste en tweede disk heel goed SCSI-disks zijn, en de geometrie die de BIOS voor sda rapporteert wordt door de kernel voor hda gebruikt. Bovendien worden disks die niet in de BIOS Setup staan vermeld niet door de BIOS gezien. Dit betekent b.v. dat in een alleen-IDE systeeem waar hdb niet in de Setup is opgegeven, de geometries die door de BIOS voor de eerste en tweede disk worden gerapporteerd, voor hda en hdc zullen gelden.

10.2 SCSI details

De situatie voor SCSI is iets anders, aangezien de SCSI opdrachten reeds logische bloknummers gebruiken, dus een 'geometrie' is volledig irrelevant voor de eigenlijke I/O. Het formaat van de partitietabel is echter nog steeds hetzelfde, dus `fdisk` moet een bepaalde geometrie uitvinden en gebruikt ook hier `HDIO_GETGEO` - inderdaad, `fdisk` maakt geen onderscheid tussen IDE en SCSI disks. Zoals men in de hieronderstaande gedetailleerde beschrijving kan zien, vinden de diverse drivers ieder een iets andere geometrie uit. Inderdaad, één grote rommel.

Als je geen DOS of iets dergelijks gebruikt, voorkom dan alle uitgebreide vertalingsinstellingen en gebruik gewoon 64 heads, 32 sectoren per spoor (voor een mooie, gepaste 1 MiB per cylinder) als dit mogelijk is, zodat er geen problemen ontstaan als je de disk van de ene naar de andere controller verplaatst. Een aantal SCSI diskdrivers (aha152x, pas16, ppa, qlogicfas, qlogicisp) zijn zo nerveus over DOS compatibiliteit dat ze het niet toestaan dat een alleen-Linux systeem van meer dan ongeveer 8 GiB gebruiken. Dit is een bug.

Wat is de werkelijke geometrie? Het makkelijkste antwoord is dat er zoiets niet is. En als dit wel zo zou zijn, zou je het niet willen weten, en zeker NOOIT, OOIT `fdisk` of LILO of de kernel erover vertellen. Het is zuiver een zaak tussen de SCSI controller en de disk. Laat me dat herhalen: alleen domme mensen vertellen `fdisk`/LILO/kernel over de ware SCSI disk geometrie.

Maar als je nieuwsgierig bent en aandringt, zou je het disk zelf kunnen vragen. Er is de belangrijke opdracht READ CAPACITY dat de totale grootte van de disk zal geven, en je hebt de MODE SENSE opdracht, dat in de Rigid Disk Drive Geometry Page (page 04) het aantal cylinders en heads geeft (dit is informatie die niet kan worden gewijzigd), en geeft in de Format Page (page 03) het aantal bytes per sector en sectoren per spoor. Dit laatste nummer is typisch afhankelijk van de inkeping, en het aantal sectoren per spoor varieert - de buitenste sporen hebben meer sectoren dan de binnenste sporen. Het Linux programma `scsiinfo` zal je deze informatie geven. Er zijn veel details en complicaties en het is duidelijk dat niemand (waarschijnlijk zelfs het besturingssysteem niet) deze informatie wil gebruiken. Bovendien, zolang als we slechts zijn geïnteresserd in `fdisk` en LILO, krijgt men typisch antwoorden als C/H/S=4476/27/171 - waarden die niet kunnen worden gebruikt door `fdisk` omdat de partitietabel slechts 10 resp. 8 resp. 6 bits voor C/H/S reserveert.

Waar haalt de kernel `HDIO_GETGEO` dan zijn informatie vandaan? Of van de SCSI controller, of door het maken van een ontwikkelde gissing. Een aantal drivers schijnen te denken dat we de 'werkelijkheid' willen weten, maar natuurlijk willen we slechts weten wat DOS of OS/2 FDISK (of Adaptec AFDISK, enz) zal gebruiken.

Merk op dat Linux `fdisk` de nummers H en S van heads en sectoren per spoor nodig heeft om LBA sectornummers te vertalen naar c/h/s adressen, maar het aantal C van cylinders speelt geen rol in deze conversie. Een aantal drivers gebruiken (C,H,S) = (1023,255 ,63) om te signaleren dat de drive capaciteit tenminste 1023*255*63 sectoren is. Dit is ongelukkig, aangezien het de werkelijke grootte niet bekend maakt, en zal de limiet van gebruikers van de meeste `fdisk` versies tot ongeveer 8 GiB van hun disks beperken - tegenwoordig een echte beperking.

In de beschrijving hieronder, duidt de M de totale diskcapaciteit aan, en C, H, S het aantal cylinders, heads en sectoren per spoor. Het volstaat om H, S te geven als we C beschouwen als gedefinieerd door M / (H*S).

Standaard, H=64, S=32.

aha1740, dtc, g_NCR5380, t128, wd7000:

H=64, S=32.

aha152x, pas16, ppa, qlogicfas, qlogicisp:

H=64, S=32 tenzij C > 1024, in welk geval H=255, S=63, C = min(1023, M/(H*S)). (Dus C is afgekapt, en H*S*C is geen benaderde waarde van de diskcapaciteit M. Dit zal de meeste versies van `fdisk` in de war brengen.) De `ppa.c` code gebruikt M+1 in plaats van M en zegt dat te wijten aan een bug in `sd.c` M is uit door 1.

advansys:

H=64, S=32 tenzij C > 1024 en bovendien de '> 1 GB' optie in de BIOS is geactiveerd, in welk geval H=255, S=63.

aha1542:

Vraag de controller welke van de twee mogelijke vertaalschema's in gebruik is, en gebruik H=255, S=63 of H=64, S=32. In het eerste geval krijg je een bootmelding äha1542.c: Using extended bios translation".

aic7xxx:

H=64, S=32 tenzij C > 1024, en bovendien of de ëxtended"boot parameter was opgegeven, of de 'extended' bit in de SEEPROM van de BIOS werd ingesteld, in welk geval H=255, S=63. In Linux 2.0.36 zou deze extended vertaling altijd ingesteld moeten zijn voor het geval geen SEEPROM werd gevonden, maar in Linux 2.2.6 wordt als er geen SEEPROM is gevonden, extended vertaling alleen ingesteld als de gebruiker erom vraagt door deze bootparameter te gebruiken (terwijl als een SEEPROM wordt gevonden, de bootparameter wordt genegeerd). Dit kan betekenen dat een setup die onder 2.0.36 werkt niet boot met 2.2.6 (en het 'linear' sleutelwoord voor LILO vereist, of de 'aic7xxx=extended' kernel boot parameter).

buslogic:

H=64, S=32 tenzij C >= 1024, en bovendien extended vertaling werd geactiveerd op de controller, in welk geval als M < 2^22 dan H=128, S=32; anders H=255, S=63. Echter na het maken van deze keuze voor (C,H,S), wordt de partitietabel ingelezen, en als voor één van de drie mogelijkheden (H,S) = (64,32), (128,32), (255,63) de waarde endH=H-1 ergens wordt gezien dan wordt dat paar (H,S) gebruikt, en wordt de bootmelding "Adopting Geometry from Partition Tableäfgedrukt.

fdomain:

Zoek de geometrie informatie op in de BIOS Drive Parameter Tabel, of lees de partitietabel in en gebruik H=endH+1, S=endS voor de eerste partitie, op voorwaarde dat het niet leeg is, of gebruik H=64, S=32 voor M < 2^21 (1 GiB), H=128, S=63 voor M < 63*2^17 (3.9 GiB) en anders H=255, S=63.

in2000:

Gebruik de eerste van (H,S) = (64,32), (64,63), (128,63), (255,63) dat zal C <= 1024 maken. Kap in het laatste geval C af bij 1023.

seagate:

Lees C,H,S vanaf de disk. (Griezels!) Als C of S te groot is, zet dan S=17, H=2 en verdubbel H tot C <= 1024. Dit betekent dat H op 0 zal worden ingesteld als M > 128*1024*17 (1.1 GiB). Dit is een bug.

ultrastor and u14_34f:

Een van drie afbeeldingen ((H,S) = (16,63), (64,32), (64,63)) wordt gebruikt afhankelijk van de controller afbeeldingsmode.

Als de driver de geometrie niet specificeert, vallen we terug op een ontwikkelde gissing door de partitietabel te gebruiken, of de totale diskcapaciteit te gebruiken.

Kijk naar de partitietabel. Aangezien volgens afspraak partities eindigen op een cylindergrens, kunnen we, gegeven `end = (endC,endH,endS)` voor enige partitie, gewoon zetten H = `endH+1` en S = `endS`. (Ter herinnering dat sectoren worden geteld vanaf 1.) Preciezer, het volgende wordt gedaan. Als er een niet-lege partitie is, neem de partitie met de grootste `beginC`. Voor die partitie, kijk naar `end+1`, berekend door het toevoegen van `start` en `lengte` en door ervan uit te gaan dat deze partitie op een cylindergrens eindigt. Als beide waarden overeenkomen, of als `endC` = 1023 en `start+lengte` een integrale veelvoud van `(endH+1)*endS` is, ga er dan vanuit dat deze partitie echt was aangepast op een cylindergrens en zet H = `endH+1` en S = `endS`. Als dit niet werkt, of omdat er geen partities zijn, of omdat ze vreemde groottes hebben, kijk dan slechts naar de diskcapaciteit M. Algorithme: zet H = M/(62*1024) (naar boven afgerond), S = M/(1024*H) (naar boven afgerond), C = M/(H*S) (naar beneden afgerond). Dit heeft als effect het produceren van een (C,H,S) met C ten hoogste 1024 en S ten hoogste 62.

11 De Linux IDE 8 GiB limiet

De Linux IDE driver krijgt de geometrie en capaciteit van een disk (een veel andere zaken) door een ATA IDENTIFY verzoek te gebruiken. Tot voor kort zou de driver het niet geloven als de geretourneerde waarde van lba_capacity meer zou zijn dan 10% groter dan de capaciteit berekend door C*H*S. Echter, door een industrie overeenkomst retourneren grote IDE disks (met meer dan 16514064 sectoren) C=16383, H=16, S=63, voor een totaal van 16514064 sectoren (7.8 GB) onafhandelijk van hun eigenlijke grootte, maar geven hun eigenlijke grootte in lba_capacity.

Recente Linux kernels (2.0.34, 2.1.90) zijn hiermee bekend en doen het juist. Als je een oudere Linux kernel hebt en niet wilt upgraden, en deze kernel ziet slechts 8 GiB van een veel grotere disk, probeer dan de routine `lba_capacity_is_ok` in `/usr/src/linux/drivers/block/ide.c` te wijzingen in iets als

```
static int lba_capacity_is_ok (struct hd_driveid *id) {
        id->cyls = id->lba_capacity / (id->heads * id->sectors);
        return 1;
}
```

Zie 2.1.90 voor een omzichtiger patch.

11.1 BIOS complicaties

Zoals net aangegeven, retourneren grote disks de geometrie C=16383, H=16, S=63 onafhankelijk van de werkelijke grootte, terwijl de werkelijke grootte wordt geretourneerd in de waarde van LBAcapacity. Een aantal BIOSsen herkennen dit niet, en vertalen deze 16383/16/63 in iets met minder cylinders en meer heads, bijvoorbeeld 1024/255/63 of 1027/255/63. Dus, de kernel moet niet alleen de enkele geometrie 16383/16/63 herkennen, maar ook alle BIOS-verminkte versies ervan. Sinds 2.2.2 wordt dit correct gedaan (door het overnemen van het BIOS idee van H en S, en te berekenen C = capacity/(H*S)). Gewoonlijk wordt dit probleem opgelost door de disk in te stellen op Normal in de BIOS setup (of nog beter, op None, het in het geheel niet vermelden in de BIOS). Gebruik kernelbootparameters, als dat niet mogelijk is omdat je ervan moet booten of het ook met DOS/Windows gebruikt en upgraden naar 2.2.2 of later geen optie is.

Als een BIOS 16320/16/63 rapporteert, dan is dit meestal gedaan om na de vertaling 1024/255/63 te verkrijgen.

Hier is een extra probleem. Als de disk met behulp van een diskvertaling werd gepartioneerd, dan is het mogelijk dat de kernel tijdens de systeemstart in de partitietabel gebruikt ziet, en rapporteert `hda: [PTBL] [1027/255/63]`. Dit is niet goed, omdat de disk nu slechts 8.4 GB is. Dit werd in 2.3.21 hersteld. Nogmaals, kernelbootparameters zullen hierbij van hulp zijn.

11.2 Jumpers die het aantal heads selecteren

Op veel disks komen jumpers voor waarmee het mogelijk is dat je een keuze maakt uit een 15-head of een 16-head geometrie. De standaardinstellingen zullen je een 16-head disk geven. Soms adresseren beide geometries hetzelfde aantal sectoren, soms is de 15-head versie kleiner. Er kan een goede reden zijn voor deze setup: Petri Kaukasoina schrijft: 'Een 10.1 Gig IBM Deskstar 16 GP (model IBM-DTTA-351010) was standaard via jumper ingesteld op 16 heads maar deze oude PC (met AMI BIOS) bootte niet en ik moest de jumper instellen voor 15 heads. hdparm -i zegt RawCHS=16383/15/63 en LBAsects=19807200. Ik gebruik 20960/15/63 om de volledige capaciteit te verkrijgen.' Voor de jumper instellingen, zie *http://www.storage.ibm.com/techsup/hddtech/hddtech.htm*.

11.3 Jumpers die de totale capaciteit afkappen

Veel disks hebben jumpers die je toestaan om de disk kleiner te doen lijken dan hij is. Een dom ding om te doen en waarschijnlijk wil geen enkele Linux-gebruiker dit ooit gebruiken, maar een aantal BIOS'sen lopen vast op grote disks. De gebruikelijke oplossing is om de disk volledig uit de BIOS-setup achterwege te laten. Maar dit is alleen uitvoerbaar als de disk niet je bootdisk is.

De eerste serieuze limiet was de 4096 cylinder limiet (dat wil zeggen, met 16 heads en 63 sectoren/spoor, 2.11 GB). Een Fujitsu MPB3032ATU 3.24 GB disk heeft bijvoorbeeld een standaardgeometrie van 6704/15/63, maar kan worden gejumperd dat het een 4092/16/63 lijkt, en het rapporteert vervolgens een LBA-capaciteit van 4124736 sectoren, zodat het besturingssysteem niet kan raden dat het in werkelijkheid groter is. In een dergelijke situatie (met een BIOS dat crasht als het hoort hoe groot de disk in werkelijkheid is, waardoor de jumper is vereist) heeft men bootparameters nodig om Linux de grootte van de disk te vertellen.

Dat is jammer. De meeste disks kunnen nu zo worden gejumperd dat ze als een 2 GB disk verschijnen en dan een vastgestelde geometrie als 4092/16/63 of 4096/16/63 rapporteren, maar nog steeds de volledige LBA-capaciteit rapporteren. Dergelijke disks werken goed, en ze gebruiken onder Linux ongeacht de jumperinstellingen de volledige capaciteit.

Een recentere beperking is 12.1 (de 33.8 GB limiet). Linux heeft nog steeds een patch nodig om met IDE-disks groter dan dit om te kunnen gaan. Een aantal disks groter dan deze limiet kunnen zodanig worden gejumperd dat het een 33.8 GB disk lijkt. De IBM Deskstar 37.5 GB (DPTA-353750) met 73261440 sectoren (corresponderend met 72680/16/63, of 4560/255/63) kan bijvoorbeeld zo worden gejumperd dat het een 33.8 GB disk lijkt, en het rapporteert dan net als iedere grote disk een geometrie van 65531/16/63, maar een LBA-capaciteit van 66055248 (overeenkomstig met 65531/16/63 of 4111/255/633). Helaas schijnt de jumper te effectief te zijn - het heeft niet alleen invloed op wat de drive aan het systeem rapporteert, maar het heeft ook invloed op de feitelijke I/O: Petr Soucek rapporteert dat bepaalde bootparameters voor deze disk niet helpen - met de aanwezige jumper geeft iedere benadering tot sector 66055248 of meer een I/O-fout. Dus op een moederbord met Award 4.51PG BIOS, kan men deze disk niet als bootdisk en gebruiken en ook de volledige capaciteit niet benutten. Zie ook *de BIOS 33.* GB limiet*.

Een andere grote disk is de 40 GB Maxtor. Mensen rapporteren dat met een oude BIOS en een dergelijke disk, de BIOS tijdens de systeemstart zal blijven hangen, zelfs wanneer de disk uit de CMOS-instellingen is verwijderd, en in een aantal gevallen zelfs wanneer de J46 jumper die de capaciteit beperkt tot 32 GB aanwezig is. In dergelijke gevallen is de beste oplossing aan een BIOS-upgrade te komen. Maxtor voorziet ook in een utility (htmlurl url="http://www.maxtor.com/technology/tecnotes/20012.html"name="JUMPON.EXE"> waarmee de grootte van de disk volledig wordt verborgen en in combinatie met MaXBlast software kan worden gebruikt. Ik heb geen informatie over of (en hoe) een disk behandeld met JUMPON.EXE tot zijn volledige capaciteit onder Linux kan worden gebruikt.

12 De Linux 65535 cylinder limiet

De `HDIO_GETGEO` ioctl retourneert verkort het aantal cylinders. Dit betekent dat als je meer dan 65535 cylinders hebt, het aantal wordt afgekapt, en (voor een typische SCSI setup met 1 MiB cylinders) kan een 80 GiB disk verschijnen als een 16 GiB disk. Zodra men het probleem herkent, is het makkelijk op te lossen.

12.1 IDE problemen met 34+ GB disks

Drives groter dan 33.8 GB zullen met kernels ouder dan 2.3.21 niet werken. De details zijn als volgt. Veronderstel dat je een nieuwe IBM-DPTA-373420 disk met een capaciteit van 66835440 sectoren (34.2 GB) kocht. Kernels van voor 2.3.21 zullen je laten weten dat de grootte 769*16*63 = 775152 sectoren (0.4 GB) is, wat een beetje teleurstellend is. En het opgeven van opdrachtregelparameters hdc=4160,255,63 helpt helemaal niet - deze worden gewoon genegeerd. Wat is er aan de hand? De routine idedisk_setup() haalt de geometrie op die door de disk wordt gerapporteerd (en dat is 16383/16/63) en overschrijft wat de gebruiker op de opdrachtregel specificeerde, zodat de gebruikersdata alleen voor de BIOS geometrie wordt gebruikt. De routine current_capacity() of idedisk_capacity() berekent opnieuw het cylindernummer als 66835440/(16*63)=66305, maar aangezien dit in een short is opgeslagen wordt het 769. Aangezien lba_capacity_is_ok() id->cyls verwijderde, zal iedere volgende aanroep ernaar false retourneren, zodat de diskcapaciteit 769*16*63 wordt. Voor verscheidene kernels is een patch beschikbaar. Een patch voor 2.0.38 is te vinden op *ftp.kernel.org*. Een patch voor 2.2.12 is te vinden op *www.uwsg.indiana.edu*. De 2.2.14 kernels ondersteunen deze disks. In de 2.3.* kernelseries is er sinds 2.3.21 ondersteuning voor deze disks. Met kan het probleem via de hardware ook öplossen"door 11.3 (een jumper te gebruiken) om de grootte tot 33.8 af te kappen. In veel gevallen zal een 4.2 (BIOS upgrade) vereist zijn als men van de disk wil booten.

13 Extended en logische partities

6 (Hierboven) zagen we de structuur van de MBR (sector 0): boot loader code gevolgd door 4 partitietabel-ingangen van elk 16 bytes, gevolgd door een AA55 kenmerk. Partitietabel ingangen van type 5 of F of 85 (hex) hebben een speciale betekenis: ze beschrijven *extended* partities: ruimte die verder is gepartitioneerd in *logische* partities. (Dus een extended partitie is slechts een box, het kan zelf niet worden gebruikt, men gebruikt de logische partities daarbinnen in.) Slechts de lokatie van de eerste sector van een extended partitie is belangrijk. Deze eerste sector bevat een partitietabel met vier ingangen: één logische partitie, één extended partitie en twee ongebruikte. Op deze manier krijgt men een keten met partitietabel sectoren, verspreid over de disk, waar de eerste de drie primaire partities beschrijft en de extended partitie en iedere volgende partitietabel sector beschrijft een logische partitie en de lokatie van de volgende partitietabel sector.

Het is belangrijk dat je dit begrijpt: Als mensen tijdens het partitioneren van een disk iets stoms doen, willen ze weten: Zijn mijn gegevens er nog steeds? En het antwoord is meestal: Ja. Maar als logische partities werden aangemaakt, dan zijn de partitietabel sectoren die deze gegevens beschreven aan het begin van deze logische partities geschreven, en gegevens die zich daar bevonden zijn verloren gegaan.

Het programma sfdisk zal de volledige keten laten zien. B.v.,

```
# sfdisk -l -x /dev/hda

Disk /dev/hda: 16 heads, 63 sectors, 33483 cylinders
Units = cylinders of 516096 bytes, blocks of 1024 bytes, counting from 0

   Device Boot Start     End   #cyls   #blocks   Id  System
/dev/hda1          0+    101     102-    51376+  83  Linux
/dev/hda2        102    2133    2032   1024128   83  Linux
/dev/hda3       2134   33482   31349  15799896    5  Extended
/dev/hda4          0       -       0         0    0  Empty

/dev/hda5       2134+   6197    4064-  2048224+  83  Linux
    -           6198   10261    4064   2048256    5  Extended
    -           2134    2133       0         0    0  Empty
    -           2134    2133       0         0    0  Empty
```

```
/dev/hda6        6198+  10261    4064-   2048224+  83  Linux
    -           10262  16357    6096    3072384    5  Extended
    -            6198   6197       0          0    0  Empty
    -            6198   6197       0          0    0  Empty
...
/dev/hda10      30581+  33482    2902-   1462576+  83  Linux
    -           30581  30580       0          0    0  Empty
    -           30581  30580       0          0    0  Empty
    -           30581  30580       0          0    0  Empty

#
```

Het is mogelijk slechte partitietabellen te construeren. Veel kernels geraken in een loop als een bepaalde extended partitie naar zichzelf terug verwijst of naar een eerdere partitie in de keten. Het is mogelijk twee extended partities te hebben in één van deze partitietabel sectoren zodat de partitietabel keten zich splitst. (Dit kan bijvoorbeeld gebeuren met een fdisk die zowel 5, F, als 85 niet als een extended partitie herkent, en een 5 naast een F aanmaakt.) Geen enkel standaard fdisk type programma kan een dergelijke situatie aan, en er is wat handwerk voor vereist om het te repareren. De Linux kernel zal een splitsing accepteren op het buitenste niveau. Dat wil zeggen dat je twee ketens van logische partities kunt hebben. Soms is dit handig - men kan bijvoorbeeld type 5 gebruiken en dat deze zichtbaar is voor DOS en het andere type 85, onzichtbaar voor DOS, zodat DOS FDISK niet zal crashen vanwege logische partities voorbij cylinder 1024. Meestal heeft men voor het aanmaken van een dergelijke setup `sfdisk` nodig.

14 Probleem oplossing

Veel mensen denken dat ze problemen hebben, terwijl er in feite niets mis is. Of ze denken dat de problemen die ze hebben, te wijten zijn aan de diskgeometrie, terwijl de diskgeometrie er in feite niets mee heeft te maken. Het kan zijn dat het bovenstaande gecompliceerd klinkt, maar diskgeometrie afhandeling is uiterst eenvoudig: doe in het geheel niets en alles is in orde; of geef LILO wellicht het sleutelwoord 'linear' als het bij het booten niet verder komt dan 'LI'. Bekijk de kernel bootmeldingen, en denk er aan: hoe meer je met geometries knoeit (het specificeren van heads en cylinders aan LILO en fdisk en op de kernel opdrachtregel) hoe minder waarschijnlijker het is dat het zal werken. Globaal genomen is standaard alles in orde.

En denk eraan: nergens in Linux wordt diskgeometrie gebruikt, dus je kunt tijdens het draaien van Linux geen probleem hebben die door diskgeometrie wordt veroorzaakt. Inderdaad, diskgeometrie wordt alleen door LILO en door fdisk gebruikt. Dus als LILO er niet in slaagt de kernel te booten, kan dat een geometrie probleem zijn. Als andere besturingssystemen de partitietabel niet begrijpen, kan dat een geometrieprobleem zijn. Anders niets. In het bijzonder, als mount niet schijnt te werken, maak je dan nooit zorgen over de diskgeometrie - het probleem ligt ergens anders.

14.1 Probleem: Mijn IDE-disk krijgt de verkeerde geometrie als ik vanaf SCSI boot

Het is heel goed mogelijk dat een disk de verkeerde geometrie krijgt. De Linux kernel ondervraagt de BIOS over hd0 en hd1 (de BIOS drives genummerd 80H en 81H) en gaat ervan uit dat deze data voor hda en hdb is. Maar op een systeem dat vanaf SCSI boot, kunnen de eerste twee disks net zo goed SCSI disks zijn, en kan het dus gebeuren dat de vijfde disk, wat de eerste IDE disk hda is, een geometrie krijgt toegewezen die aan sda toebehoort. Dergelijke zaken worden gemakkelijk opgelost door boot parameters 'hda=C,H,S' op te geven voor de bestemde nummers C, H en S, of bij het booten of in /etc/lilo.conf.

14.2 Geen probleem: Identieke disks hebben verschillende geometries?

'Ik heb twee identieke 10 GB IBM disks. Echter, fdisk geeft verschillende groottes voor ze op. Kijk:

```
# fdisk -l /dev/hdb
Disk /dev/hdb: 255 heads, 63 sectors, 1232 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot  Start      End   Blocks   Id  System
/dev/hdb1           1     1232  9896008+  83  Linux native
# fdisk -l /dev/hdd
Disk /dev/hdd: 16 heads, 63 sectors, 19650 cylinders
Units = cylinders of 1008 * 512 bytes

   Device Boot  Start      End   Blocks   Id  System
/dev/hdd1           1    19650  9903568+  83  Linux native
```

Hoe komt dit?'

Wat is hier aan de hand? Als eerste zijn deze drives in werkelijkheid 10gig: hdb heeft een grootte van 255*63*1232*512 = 10133544960, en hdd heeft een grootte van 16*63*19650*512 = 10141286400, dus er is niets mis en de kernel ziet beiden als 10.1 GB. Waarom het verschil in grootte? Dat is omdat de kernel zijn data voor de eerste twee IDE disks vanuit de BIOS krijgt, en de BIOS heeft hdb heringedeeld alsof het 255 heads heeft (en 16*19650/255=1232 cylinders). De afronding naar beneden kost hier bijna 8 MB.

Als je hdd op dezelfde manier zou willen herindelen, geef dan de kernel boot parameters 'hdd=1232,255,63' op.

14.3 Geen probleem: fdisk ziet meer ruimte dan df?

fdisk laat je weten hoeveel blokken er op de disk voorkomen. Als je een bestandssysteem op de disk aanmaakt, laten we zeggen met mke2fs, dan heeft dit bestandssysteem wat ruimte nodig voor adminstratie - kenmerkend iets als 4% van de grootte van het bestandssysteem, meer als je vraagt om een boel inodes gedurende mke2fs. Bijvoorbeeld:

```
# sfdisk -s /dev/hda9
4095976
# mke2fs -i 1024 /dev/hda9
mke2fs 1.12, 9-Jul-98 for EXT2 FS 0.5b, 95/08/09
...
204798 blocks (5.00%) reserved for the super user
...
# mount /dev/hda9 /ergens
# df /ergens
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/hda9            3574475      13  3369664      0%   /mnt
# df -i /ergens
Filesystem           Inodes   IUsed   IFree  %IUsed Mounted on
/dev/hda9            4096000      11 4095989     0%  /mnt
#
```

We hebben een partitie met 4095976 blokken, maken er een ext2 bestandssysteem op aan, mounten het ergens en bemerken dat het slechts 3574475 blokken heeft - 521501 blokken (12%) zijn verloren gegaan aan inodes en andere administratie. Merk op dat het verschil tussen het totaal 3574475 en de 3369664 beschikbaar voor de gebruiker de 13 blokken zijn die in gebruik zijn plus de 204798 blokken die voor root zijn gereserveerd. Dit laatste nummer kan worden gewijzigd door tune2fs. Deze '-i 1024' is alleen redelijk voor news spools en dergelijke, met heel veel kleine bestanden. De standaard zou zijn:

```
# mke2fs /dev/hda9
# mount /dev/hda9 /somewhere
# df /somewhere
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/hda9            3958475      13  3753664      0%   /mnt
# df -i /somewhere
Filesystem           Inodes   IUsed   IFree  %IUsed Mounted on
/dev/hda9            1024000      11 1023989     0%  /mnt
#
```

Nu worden er slechts 137501 blokken (3.3%) voor inodes gebruikt, zodat we 384 MB meer dan voorheen hebben. (blijkbaar, neemt iedere inode 128 bytes in beslag.) Aan de andere kant, kunnen er op dit bestandssysteem maximaal 1024000 bestanden voorkomen (meer dan genoeg), tegen 4096000 (te veel) daarvoor.