Large Disk HOWTO

Andries Brouwer <aeb@cwi.nl> diterjemahkan oleh Yaya Heryadi <yheryadi@necnusa.co.id> V1.1, 18 May 1998, terj. 11 Des 1998

Tulisan ini membahas masalah geometri dan batasan jumlah silinder 1024 pada disk.

Contents

1 Permasalahan

Seandainya Anda memiliki sebuah disk dengan jumlah silinder lebih dari 1024. Seterusnya, Anda juga memiliki sistem operasi yang menggunakan BIOS. Maka, Anda mempunyai sebuah masalah, karena interface BIOS biasa dengan disk I/O yaitu INT13 menggunakan sebuah field sepanjang 10-bit untuk menempatkan nomor silinder bagi I/O tersebut, sehingga silinder-silinder bernomor sama dengan atau lebih besar dari 1024 tidak akan dapat diakses.

Beruntung Linux tidak menggunakan BIOS sehingga masalah di atas tidak ditemui di dalam Linux. Kecuali dalam dua keadaan:

- Pada saat Anda mem-boot sistem, Linux belum berjalan dan tidak dapat melindungi Anda dari masalah BIOS di atas. Hal ini berakibat terhadap LILO atau loader sejenis.
- Setiap sistem operasi yang menggunakan sebuah disk perlu sama-sama menyetujui letak setiap partisi. Dengan kata lain, Jika Anda menggunakan Linux bersama dengan, misalnya DOS, dalam suatu disk maka kedua sistem operasi tadi harus memiliki cara yang sama dalam menginterpretasikan tabel partisi. Hal ini memberikan konsekuensi terhadap kernel Linux dan fdisk.

Berikut ini akan diberikan uraian yang lebih terinci mengenai hal-hal yang berkaitan dengan masalah di atas. Sebagai catatan: sebagai referensi penulis gunakan kernel versi 2.0.8. Versi-versi lain mungkin memiliki perbedaan meskipun sedikit.

2 Booting

Pada saat sistem di-boot, BIOS membaca sector 0 (dikenal dengan nama MBR - *Master Boot Record*) dari disk pertama (atau floppy), lalu menjalankan program yang terdapat di dalamnya - biasanya merupakan suatu bootstrap loader. Program bootstrap kecil yang ditemukan tadi umumnya tidak memiliki disk driver sendiri dan menggunakan servis-servis dari BIOS. Hal itu berarti kernel Linux hanya dapat di-boot jika keseluruhan kernel terdapat di dalam 1024 silinder pertama.

Masalah ini dapat diatasi dengan mudah: pastikan bahwa kernel (atau mungkin file-file lain yang digunakan saat bootup, misalnya file map LILO) terdapat di dalam sebuah partisi yang keseluruhannya berada di dalam 1024 silinder pertama dari suatu disk yang dapat diakses oleh BIOS - ini dapat berarti disk pertama atau kedua.

Hal penting lainnya adalah boot loader dan BIOS harus memiliki kesesuaian mengenai geometri dari disk. Penggunaan option 'linear' bagi LILO mungkin akan membantu hal ini.

3 Geometri dan Partisi dari Disk

Jika Anda memiliki beberapa sistem operasi dalam sebuah disk maka setiap sistem operasi menggunakan satu atau lebih partisi disk tersebut. Ketidak-sesuaian letak dari partisi-partisi tersebut dapat berakibat fatal.

Sebuah MBR berisi tabel partisi yang menjelaskan di mana letak partisi-partisi (primer) berada. Tabel tersebut memiliki 4 entri, seluruhnya untuk 4 partisi primer, masing-masing entri berupa struktur:

```
struct partition {
        Char    active;         /*  0x80: bootable, 0: not bootable */
        Char    begin[3];       /*  CHS untuk sector pertama  */
        Char    type;
        Char    end[3];         /*  CHS untuk sector terakhir  */
        Char    start           /*  32-bit nomor sector (dihitung mulai dari nomor 0)  */
        Char    length;         /*  32-bit nomor jumlah sector  */
} ;
```

(CHS singkatan dari *Cylinder/Head/Sector*).

Informasi di atas bersifat duplikasi: lokasi partisi ditunjukan tidak hanya oleh field-field begin (24 bit) dan end (24 bit), tetapi juga oleh start (32 bit) dan length (32 bit).

Linux hanya menggunakan field-field *start* dan *length* sehingga dapat menangani partisi yang berukuran tidak lebih dari 2^32 sector, atau 2 TB. Ukuran itu adalah 100 kali lebih besar dari disk yang tersedia saat ini, mungkin ukuran tersebut masih memadai untuk 8 tahun kedepan atau lebih.

Sayang sekali, INT13 call dari BIOS menginterpretasikan angka CHS yang terdiri dari 3 byte itu menjadi sbb: 10 bit bagi nomor silinder, 8 bit bagi nomor head, dan 6 bit untuk nomor sector dalam track. Sehingga nomor silinder dapat berupa angka diantara 0 - 1023, nomor head dapat berupa angka diantara 0 - 255, dan nomor sector dapat berupa angka diantara 1 - 63 (pemberian nomor sector dalam sebuah track dimulai dari 1 bukan 0). Dengan ke-24 bit tadi seseorang dapat mengakses ukuran sebesar 8455716864 byte (7.875 GB), 200 kali lebih besar dari disk yang tersedia pada tahun 1983.

Hal yang lebih disayangkan lagi, interface IDE standar memungkinkan adanya 256 sector/track, 65536 silinder dan 16 head. Dengan standar tersebut memungkinkan pengaksesan terhadap 2^37 = 137438953472 byte (128 GB), tetapi kalau digunakan bersama dengan pembatasan 63 sector dan 1024 silinder dari BIOS maka hanya sebesar 528482304 byte (504 MB) yang dapat diakses.

Ukuran di atas tentunya tidak memadai untuk kebutuhan disk saat ini, sehingga orang mencoba berbagai cara untuk mengatasi batasan di atas baik secara hardware maupun software.

4 Penterjemahan dan Disk Manager

Tidak seorangpun tertarik terhadap geometri sesungguhnya dari sebuah disk. Jumlah sector dalam setiap track sering bervariasi - jumlah sector per-track semakin besar dengan semakin dekatnya lokasi track tersebut ke sisi luar disk - sehingga jumlah sector per track yang real sebenarnya tidak ada. Bagi user, sebuah disk lebih baik dipandang sebagai sebuah array linier yang terdiri dari sector-sector yang memiliki nomor-nomor 0, 1, 2, 3, ... dst, dan merupakan tugas controller untuk mendapatkan lokasi sector sesungguhnnya di dalam disk.

Penomoran secara linier ini disebut LBA. Alamat linier dari suatu sector dengan posisi yang dinyatakan oleh (c,h,s) di dalam suatu disk dengan geometri (C,H,S) dinyatakan oleh: `c*H*S + h*S + (s-1)`. Semua SCSI controller dan beberapa IDE controller mengenal LBA.

Jika BIOS mengkonversi alamat dari bentuk (c,h,s) 24-bit kedalam bentuk LBA dan menyampaikannya kepada controller yang menggunakan LBA maka pengalamatan sector dapat dilakukan sampai ke ukuran 7.875 GB. Meskipun ukuran tersebut belum cukup bagi semua disk, tapi sudah dicapai suatu peningkatan. Perlu dicatat bahwa CHS yang digunakan BIOS tidak ada lagi kaitannya dengan realitas.

Hal yang sama berlaku pula jika BIOS dapat melakukan konversi namun controller tidak mengenal LBA. (Saat Setup, kasus ini sering ditandai dengan istilah 'Large') . Untuk kasus ini, BIOS akan memberikan geometri (C', H', S') kepada sistem operasi, dan menggunakan geometri (C,H,S) sewaktu berkomunikasi dengan controller. Pada umumnya `S = S', C' = C`N, dan H' = H*N/ di mana N adalah bilangan pangkat dua terkecil yang memenuhi kondisi C' <= 1024 (sehingga kapasitas terkecil yang dibuang melalui pembulatan kebawah terhadap nilai C' = C/N menjadi minimal). Sekali lagi, hal tersebut memungkinkan akses sampai 7.875 GB.

Jika BIOS tidak memahami istilah 'Large' atau LBA maka digunakan solusi secara software. Disk Manager seperti **OnTrack** atau **EZ-Drive** akan menggantikan peran-peran routine BIOS dalam menangani disk. Dalam melakukan peran tadi seringkali program disk manager ditempatkan di dalam MBR dan sector-sector berikutnya (OnTrack menyebut program ini dengan *DDO: Dynamic Drive Overlay*) sehingga disk manager dapat di-boot sebelum hal yang sama dilakukan terhadap sistem operasi. Itulah sebabnya mengapa seseorang dapat menemui masalah saat melakukan booting dari floppy dan saat itu disk manager sudah di-install.

Meskipun efek penggunaan disk manager kurang lebih sama dengan penggunaan penterjemahan BIOS namun disk manager dapat menimbulkan banyak permasalahan, khususnya apabila beberapa sistem operasi dijalankan pada sebuah disk.

Linux men-support OnTrack mulai Linux versi 1.3.14, dan men-support EZDrive mulai Linux versi 1.3.29. Beberapa rincian lainnya diberikan berikut ini.

5 Penterjemahan Disk oleh Kernel pada disk IDE

Jika kernel Linux mendeteksi adanya disk manager pada suatu disk IDE, Linux akan berusaha melakukan pemetaan ulang (remap) terhadap disk tersebut melalui cara yang telah dilakukan oleh disk manager, misalnya DOS dengan OnTrack atau EZDrive. Namun, pemetaan ulang tidak akan dilakukan jika geometri telah dispesifikasikan melalui baris perintah, sehingga opsi baris perintah seperti "hd = cyls, heads, secs" akan menghapuskan kompatibilitas dengan disk manager.

Pemetaan ulang dilakukan dengan mencoba angka-angka 4, 8, 16, 32, 64, 128, 255 bagi nomor head (dengan membuat bilangan H*C konstan) sampai dipenuhi salah satu, yaitu : C <= 1024 atau H = 255.

Secara terinci akan diberikan di dalam sub-bab berikut. Judul dari sub-bab berikut menunjukan text yang ditampilkan pada pesan saat disk manager bersangkutan di-boot. Di dalam tulisan ini tipe partisi dinyatakan di dalam bilangan hexadesimal.

5.1 EZD

Disk manager EZ-Drive dapat dideteksi melalui partisi primer (primary partition) pertamanya yang selalu memiliki tipe 55. Remap dari geometri dilakukan dengan cara yang dijelaskan di atas, dan tabel partisi dari sektor 0 dihilangkan - sebagai penggantinya tabel partisi baru dibaca dari sektor 1. Nomor-nomor block disk tidak berubah, tetapi penulisan terhadap sector 0 akan dialihkan ke sector 1. Perilaku ini dapat dirubah dengan meng-compile kernel dengan penambahan perintah berikut di dalam file ide.c:

```
#define                 FAKE_FDISK_FOR_EZDRIVE  0
```

5.2 DM6:DDO

Disk manager OnTrack dapat dideteksi melalui partisi primer pertamanya yang selalu bertipe 54. Remap dari geometri dilakukan dengan cara yang telah dijelaskan di atas, dan seluruh disk digeser (shift) sejauh 63 sector (sehingga sector 63 menjadi sector 0). Kemudian, MBR baru dibaca dari sector 0 yang baru. Penggeseran ini untuk memberi tempat bagi DDO - itulah sebabnya mengapa tidak ada pergeseran pada disk lainnya.

5.3 DM6:AUX

Disk manager OnTrack (di dalam disk lainnya) dapat dideteksi melalui partisi primer pertamanya yang bertipe 51 atau 53. Geometri di-remap dengan cara yang dijelaskan sebelumnya.

5.4 DM6:MBR

Versi lama dari disk manager OnTrack dapat dideteksi bukan melalui tipe partisi melainkan melalui signature-nya. (Dengan melakukan pengujian apakah offset yang ditemukan pada byte ke-2 dan ke-3 dari MBR tidak melebihi angka 430, dan short yang ditemukan pada offset ini sama dengan 0x55AA, dan diikuti oleh byte ganjil). Remap geometri dilakukan seperti yang telah diuraikan di atas.

5.5 PTBL

Terakhir, ada sebuah uji (test) yang mencoba melakukan deduksi mengenai penterjemahan berdasarkan nilai-nilai start dan end dari partisi primer: Jika sebuah partisi memiliki start dan end silinder kurang dari 256; nomor start-sector = 1 dan nomor end-sector = 63; dan end-head = 31, 63 atau 127, dan karena sebuah partisi umumnya diakhiri dalam batas-batas silinder (cylinder boundary) demikian pula interface IDE menggunakan paling banyak 16 head, maka dapat disimpulkan bahwa suatu penterjemahan BIOS telah aktif dan selanjutnya geometri di-remap untuk menggunakan head bernomor 32, 64 atau 128. (Mungkin disini ada kekurangan, dan `genhd.c` seharusnya tidak melakukan pengujian terhadap 2 bit order tertinggi dari nomor silinder?) Namun demikian, remapping tidak dapat dilakukan jika geometri saat ini telah memiliki 63 sector per-track serta head yang setidaknya berjumlah sama (karena mungkin berarti telah dilakukan suatu remapping).

6 Konsekuensi

Apakah arti semua ini? Bagi para user Linux hanya satu hal: mereka harus memastikan bahwa LILO dan fdisk menggunakan geometri yang benar, di mana arti kata 'benar' bagi fdisk adalah geometri yang digunakan harus sama dengan geometri yang digunakan oleh sistem operasi lain pada disk tersebut, dan arti kata 'benar' bagi LILO adalah geometri tersebut memungkinkan dilakukannya interaksi dengan BIOS pada saat boot. (Biasanya kedua hal tadi dipenuhi secara bersamaan).

Bagaimana fdisk dapat mengetahui geometri? fdisk akan bertanya kepada kernel menggunakan `HDIO_GETGEO` ioctl. Namun user dapat mengatasi (override) geometri baik secara interaktif maupun melalui baris perintah.

Bagaimana LILO dapat mengetahui geometri? LILO akan bertanya kepada kernel menggunakan `HDIO_GETGEO` ioctl. Namun user dapat mengatsi geometri menggunakan opsi "disk = ". Seseorang dapat pula memberikan opsi linier terhadap LILO yang menyebabkan LILO akan menyimpan alamat LBA kedalam file map menggantikan alamat CHS, serta menemukan geometri yang akan digunakannya saat boot (menggunakan INT13 Function 8 untuk menanyakan geometri drive).

Bagaimana kernel mengetahui jawaban di atas? Terlebih dahulu, user mungkin telah memberikan spesifikasi sebuah geometri secara eksplisit melalui opsi baris perintah "hd = cyls, head, sect"; kalau tidak, maka kernel akan bertanya kepada hardware.

6.1 IDE secara terinci

Baiklah akan penulis jelaskan lebih lanjut. Driver IDE memiliki 4 sumber informasi mengenai geometri. Pertama (G_user) spesifikasi dari user melalui baris perintah. Kedua (G_bios) adalah Tabel Parameter Fixed Disk (hanya untuk disk pertama dan kedua) yang dibaca pada saat startup sistem sebelum berpindah ke mode 32-bit. Ketiga (G_phys) dan Keempat (G_log) berasal dari controller IDE sebagai respon terhadap perintah IDENTIFY, masing-masing merupakan geometri "physical" dan "current logical".

Di sisi lain, sebuah driver memerlukan dua nilai bagi geometri, yaitu: G_fdisk berasal dari `HDIO_GETGEO` ioctl, dan G_used yang digunakan sesungguhnya untuk melakukan I/O.

G_fdisk dan G_used kedua-duanya akan diberi nilai inisial dengan nilai dari: G_user, atau dengan G_bios jika informasi ini tersedia di CMOS, dan terakhir dengan G_phys jika semua yang disebut sebelumnya tidak tersedia. G-used akan diset ke G_log jika nilai G-log dapat diterima (reasonable); sebaliknya, jika G_used tidak dapat diterima (unreasonable) sedangkan G_phys dapat diterima maka G_used diset ke G_phys. Dalam hal ini arti istilah "dapat diterima" adalah nomor head yang dimilikinya berada diantara angka 1-16.

Dengan perkataan lain: baris perintah dapat mengatasi (overriden) BIOS dan akan menentukan apa yang dilihat fdisk, tetapi jika baris perintah memberikan spesifikasi terjemahan geometri (dengan head lebih dari 16) maka spesifikasi bagi keperluan I/O kernel akan diatasi oleh output dari perintah IDENTIFY.

6.2 SCSI secara terinci

Situasi pada SCSI agak sedikit berbeda karena perintah SCSI telah menggunakan penomoran block secara logika sehingga "geometri" tidak berguna untuk I/O aktual. Namun, format dari tabel partisi masih tetap sama, sehingga fdisk harus menemukan suatu geometri dan menggunakan `HDIO_GETGEO` - fdisk sungguh tidak membedakan antara disk IDE dengan SCSI. Seperti yang dapat dilihat pada deskripsi dibawah ini, setiap driver menggunakan geometri yang berbeda-beda. Betul-betul keadaan yang kacau.

Jika Anda tidak menggunakan DOS atau sejenisnya, hindarilah setiap seting bagi partisi extended, jika memungkinkan gunakanlah hanya 64 head, 32 sector per-track (agar lebih baik dan mudah: 1 MB per-silinder), agar tidak menimbulkan masalah saat memindahkan disk tersebut dari satu controller ke controller lainnya. Beberapa driver SCSI (ah152x, pas16, ppa, qlogicfas, qlogicisp) sangat kaku dengan DOS kompatibilitas sehingga tidak mengizinkan sistem Linux menggunakan lebih dari 8GB. Ini merupakan suatu bug.

Apakah geometri yang sesungguhnya? Jawaban paling mudah adalah: Tidak ada. Jika memang ada, Anda mungkin tidak ingin mengetahuinya, dan sangat pasti tidak akan pernah memberikannya kepada fdisk, LILO, atau kernel. Geometri riil hanya merupakan keperluan controller SCSI dan disk. Penulis ulangi: hanya orang-orang yang aneh saja yang akan memberikan geometri riil dari disk SCSI kepada fdisk/LILO/kernel.

Namun jika Anda betul-betul ingin tahu, Anda dapat mencari jawaban dari disk itu sendiri. Perintah READ CAPACITY merupakan perintah penting untuk mendapatkan ukuran total dari disk. Perintah MODE SENSE: di dalam "Rigid Disk Drive Geometry Page" (page 04), akan memberikan jumlah silinder dan head (informasi ini tidak dapat diubah), di dalam "Format Page" (page 03) akan memberikan jumlah byte per-sector dan sector per-track. Angka yang terakhir ini umumnya tergantung kepada notch, sedangkan jumlah sector per-track bervariasi - track yang lebih luar akan memiliki jumlah sector lebih besar dibandingkan track yang lebih dalam. Banyak sekali hal-hal rinci dan kompleks, dan jelas sekali bahwa tidak seorangpun (bahkan mungkin tidak juga sistem operasi) ingin menggunakan informasi ini. Selanjutnya, selama kita hanya berkepentingan dengan fdisk dan LILO, seseorang mendapat jawaban seperti C/H/S=4476/27/171 -

nilai-nilai tersebut tidak dapat digunakan bagi fdisk karena tabel partisi hanya mneyediakan tempat 10 resp. 8 resp. 6 bit bagi C/H/S.

Lalu, dari manakah `HDIO_GETGEO` kernel mendapatkan informasi? Informasi tersebut dapat berasal dari controller SCSI atau merupakan hasil pendugaan cerdas (educated guess). Beberapa driver nampaknya mengira kita ingin mengetahui hal yang sesungguhnya, padahal yang ingin kita ketahui hanyalah apakah yang akan digunakan oleh FDISK dari DOS atau OS/2 (atau AFDISK dari adaptec, dst) itu.

Perlu dicatat bahwa fdisk Linux membutuhkan angka-angka jumlah head (H) dan sector per-track (S) untuk membuat konversi LBA menjadi alamat c/h/s, tetapi C (jumlah silinder) tidak berperan dalam konversi ini. Beberapa driver menggunakan (C,H,S) = (1023, 255, 63) untuk memberi tanda bahwa kapasitas drive sekurang-kurangnya berjumlah 1023*255*63 sector. Hal ini disayangkan, karena hal tersebut tidak menunjukan ukuran sesungguhnya, dan akan membatasi pemakai dari kebanyakan versi fdisk hanya sampai 8 GB dari disk mereka - sangat membatasi untuk kebutuhan saat ini.

Di dalam keterangan dibawah ini, M menunjukkan kapasitas total dari disk; C, H, S masing-masing menunjukkan jumlah silinder, jumlah head, dan jumlah sector per-track. Jika C didefinisikan sebagai M/(H*S), nilai tersebut sudah cukup untuk mendapatkan nilai-nilai H dan S.

Secara default, H=64, S=32

aha1740, dtc, g_NCR5380, t128, wd7000: H=64, S=32

aha152x, pas16, ppa, qlogicfas, qlogicisp: H=64, S=32; kecuali apabila C > 1024 maka H=255, S=63, C=min(1023, M/(H*S)). (Sehingga nilai C dipotong, dan H*S*C tidak merupakan pendekatan bagi kapasitas disk M. Hal ini akan membingungkan hampir semua versi dari fdisk). File ppa.c menggunakan M+1 sebagai pengganti M dan menyatakan bahwa sebagai akibat bug di dalam sd.c maka M berbeda 1).

advansys: H=64, S=32; kecuali jika C > 1024 dan opsi "> 1 GB" pada BIOS di-enable maka H=255, S=63

aha1542 Controller akan diminta untuk menentukan skema penterjemahan yang akan digunakan: apakah H=255, S=63 ataukah H=64, S=32. Dalam kasus pertama, akan muncul pesan boot "aha1542.c: Using extended bios translation".

aic7xxx: H=64, S=32; kecuali jika C > 1024, dan apakah parameter "extended" boot telah diberikan atau "extended" bit di dalam SEEPROM telah diset, dalam hal ini H=255, S=63

buslogic: H=64, S=32; kecuali jika C >= 1024 dan penterjemahan "extended" dari controller sudah di-enable, dalam kasus ini jika M < 2^22 dipenuhi maka H=128, S=32, jika tidak dipenuhi maka H=255, S=63. Sesudah membuat pilihan mengenai (C,H,S), tabel partisi dibaca dan jika nilai endH=H-1 ditemui pada salah-satu pasangan (H,S) dari ketiga kemungkinan pasangan berikut: (64,32), (128, 32), (255, 63) maka pasangan (H,S) tersebut digunakan, dan pesan boot menampilkan "Adopting Geometry from Partition Table".

fdomain: Cari informasi geometri di dalam Drive Parameter Table dari BIOS, atau bacalah tabel partisi dan gunakan H=endH+1, S=endS untuk partisi pertama apabila partisi tersebut tidak kosong; atau gunakan H=64, S=32 untuk M < 2^21 (1 GB), H=128, S=63 untuk M < 63*2^17 (3.9 GB), atau H=255, S=63 untuk yang lainnya.

in2000: Gunakan (H,S) pertama dari urutan: (64,32), (64,63), (128,63), (255,63) yang memenuhi C <=1024. Dalam kasus terakhir, C dipotong menjadi 1023.

seagate: Membaca C,H,S dari disk. (Horror!) Jika C atau S terlalu besar maka tetapkan S=17, H=2 dan kalikan H dengan 2 sampai tercapai C <=1024. Ini berarti H=0 jika M > 128*1024*17 (1.1 GB). Ini merupakan bug.

ultrastor dan **u14_34f**: Salah satu mapping berikut: ((H,S) = (16,63), (64,32), (64,63)) digunakan tergantung dari mode mapping controller-nya.

Jika driver tidak memberikan spesifikasi geometri, kita akan kembali melakukan pendugaan menggunakan tabel partisi atau menggunakan kapasitas total dari disk.

Perhatikan kembali tabel partisi. Berdasarkan konvensi (convention), ujung sebuah partisi terletak pada batas silinder, jika diketahui end = (endC, endH, endS) untuk setiap partisi maka kita dapat menetapkan H=endH+1 dan S=endS. (Ingatlah bahwa penomoran sector mulai dari nomor 1.) Lebih tepat lagi, dilakukan hal berikut ini. Jika tidak terdapat partisi kosong, ambillah partisi yang memiliki nilai beginC terbesar. Dari partisi tersebut lihatlah isi field end+1, jumlahkan isi field-field start dan length, serta dilakukan asumsi bahwa partisi tadi berakhir pada batas silinder. Jika end+1 dan start+length memiliki nilai yang sama; atau jika endC=1023 serta hasil penjumlahn start+length merupakan multiple integral dari (endH+1)*endS maka dapat ditetapkan bahwa H=endH+1 dan S=endS juga dengan asumsi bahwa batas partisi ini terletak pada batas silinder. Jika cara ini tidak berhasil disebabkan karena tidak adanya partisi atau ukuran partisinya agak asing, maka perhatikan saja ukuran kapasitas disk M. Dengan algoritma sbb:

```
        H = M/(62*1024)         (dibulatkan keatas),
        S = M/(1024*H)          (dibulatkan keatas),
        C = M /(H*S)            (dibulatkan kebawah).
```

Cara ini menghasilkan (C,H,S) dengan C < 1024 dan S < 62.

7 Batasan 8GB IDE dalam Linux

Driver IDE dalam Linux mendapatkan geometri dan kapasitas disk (dan banyak informasi lainnya) menggunakan perintah ATA IDENTIFY. Sampai saat ini, driver tersebut tidak akan menggunakan nilai lba_capacity yang diberikan oleh perintah di atas apabila nilai tersebut 10% lebih besar dari kapasitas yang dihitung dari C*H*S. namun demikian, disk Quantum Bigfoot yang berukuran 12 GB saat ini memberikan C=16383, H=16, S=63 dari jumlah total sector 16514064 (7.8 GB) tetapi memberikan lba_capacity 23547888 sector (11.2 GB, yaitu C=23361).

Kernel Linux (2.0.34pre14, 2.1.90) mengetahui akan hal ini dan melakukan hal yang benar. Jika Anda memiliki versi lama kernel Linux tetapi Anda tidak ingin melakukan upgrade, sedangkan kernel Anda hanya mengenal 8 GB dari disk yang kapasitas sebenarnya lebih besar dari itu, maka cobalah merubah routine `lba_capacity_is_ok` di dalam file `/usr/src/linux/drivers/block/ide.c` sehingga berisi sbb:

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

Untuk patch yang lebih baik, gunakanlah versi 2.1.90.