Geometrie disků v BIOSu, DOSu a Linuxu CompactFlash média, CF čtečky na USB

Autor: František Ryšánek <rysanek@fccps.cz>

FCC Průmyslové systémy s.r.o.

Obsah

Úvodem

Autor nedávno řešil problém, jak zajistit klonování či automatickou instalaci systému DOS (FreeDOS) na CompactFlash média – na média o různé kapacitě, bez restartu počítače sloužícího pro hromadné instalace. Výsledkem několika dnů studia a experimentů je funkční bootovatelné CD, které zmíněný problém řeší.

Tento článek shrnuje zajímavé zapeklitosti, které bylo třeba cestou vyřešit.

Geometrie disků, tabulka rozdělení

Předpokládejme, že laskavý čtenář přibližně ví, co jsou to cylindry, hlavy a sektory pevného disku. Diskety, MFM/RLL disky a první IDE disky skutečně používaly svou fyzickou geometrii při komunikaci s BIOSem a se softwarem PC.

IDE disky velmi záhy začaly používat vnitřní překlad skutečné interní geometrie na jakousi externí, kterou prezentují BIOSu. Je to jednak proto, že se fyzická geometrie disku již nevešla do rozsahu datových typů C/H/S, které jsou pro ni BIOSem vyhrazeny, jednak proto, že se na stopách různě vzdálených od středu disku vzájemně liší počet sektorů či objem dat na stopu – takže si dnes výrobci IDE disků milosrdně nechávají složitosti skutečné geometrie pro sebe a navenek prezentují maškaru, kterou BIOS bez problémů stráví. Některé staré IDE disky (okolo 100 MB) umožňovaly jumperem vybrat jednu ze dvou „normálních“ geometrií – dnes se tento jumper již nevyskytuje.

Dosud se stále bavíme o geometrii, kterou IDE disk prezentuje ve svých režijních registrech – tedy mimo svůj prostor pro ukládání užitečných dat. Této geometrii se v BIOSu říká “normal”, v Linuxu “physical”.

Geometrie disku se používá jednak při komunikaci BIOSu „dolů“ s diskem, ale také při komunikaci BIOSu „nahoru“ s DOSem a jinými operačními systémy (a zavaděči), které jsou programovány pro CHS přístup k disku přes služby BIOSu.

Zběsilé tempo vývoje IDE disků a s ním spojený růst kapacity přinesly v historii několik okamžiků, kdy datové typy dosavadního rozhraní BIOS/DOS přestaly stačit na zachycení „normální“ geometrie disku. Kvůli zpětné kompatibilitě proto novější BIOSy zavedly své vlastní, softwarové translace. Asi nejlepším zdrojem informací o těchto meznících ve vývoji IDE disků je příslušná kapitola linuxového „Large disk HOWTO“ (viz odkazy v přehledu literatury).

Pokud v BIOSu konfigurujeme režim přístupu k disku výběrem z možností „Normal“, „LBA“ nebo „Large“, vybíráme softwarovou translaci geometrie, kterou má BIOS provádět při komunikaci s DOSem. O této translaci IDE disk nic neví a nijak se jí neúčastní, translaci provádějí služby BIOSu – na úrovni IDE protokolů mluví BIOS s diskem vždy pomocí „normální“ geometrie.

Pokud vše funguje jak má, tato „soft“ translace zvolená v BIOSu je Linuxem detekována a hlášena jako „logická“ geometrie. Pro uživatele Linuxu: viz `cat /proc/ide/hd<X>/geometry`.

Přesněji řečeno, IDE disky již delší dobu podporují alternativně také lineární adresaci logickým číslem sektoru (které je interně přímo přeloženo na skutečnou geometrii). Toto je režim, který je pro přístup k IDE disku standardně používán Linuxem a ostatně i moderními BIOSy (zapíná se patrně volbou „IDE block mode“). Navenek vůči DOSu se nicméně BIOS tváří zvolenou „soft“ translací

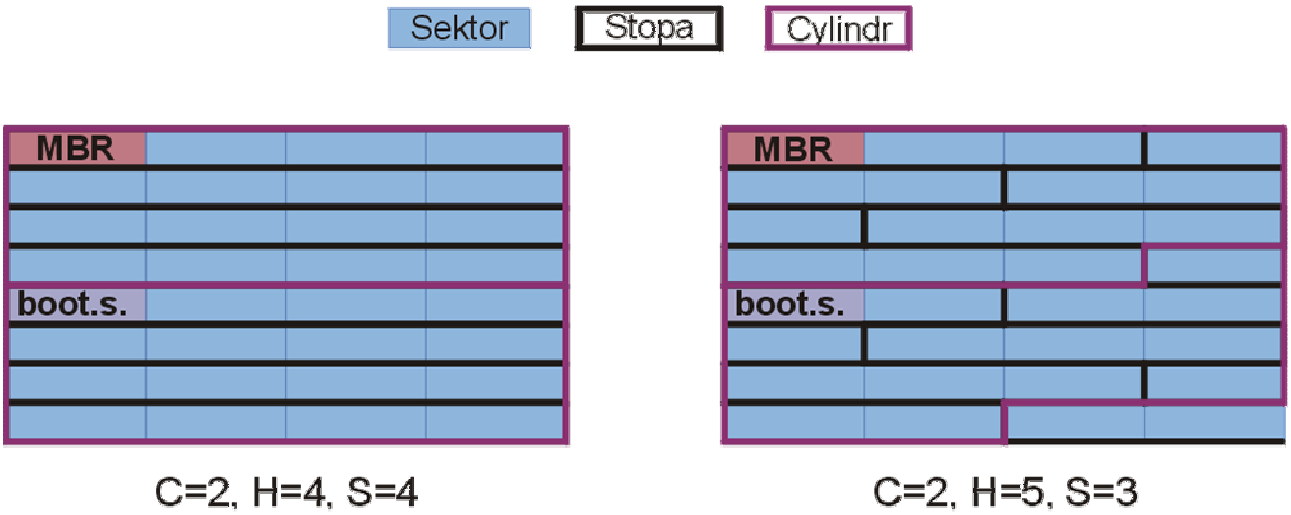
(viz výše). Popravdě řečeno, chová se tak konkrétní služba BIOSu (softwarový interrupt), kterou je DOS zvyklý používat. Již spoustu let je na vnějším rozhraní BIOSu k dispozici také služba pro přístup k disku s použitím lineární adresace...

Starý software, programovaný pro jistou generaci služeb BIOSu, lze tedy ošidit/uspokojit konkrétní „soft" translací – ovšem s tím omezením, že třeba nevidí disk celý, ale jenom jeho určitou část od začátku (počínaje nultým sektorem), která se „vejde" do daného typu překládané geometrie. Což už může stačit například první úrovni zavaděče ke startu jádra operačního systému z malého prvního bootovacího oddílu, který se vejde do „viditelného okna" – a vlastní ovladače operačního systému, nezávislé na BIOSu, pak zpřístupní velký disk v celé jeho kráse.

Pokud se týče prezentované „normální" („fyzické") geometrie, moderní veliké IDE disky dávno ztratily veškerou fantazii. Standardem je 255 hlav a 63 sektorů na stopu, tj. maximální možné hodnoty, které připouští standard IDE a které zpracuje BIOS. V přepočtu 16065 sektorů na cylindr. Sektor má jednotnou velikost 512 bajtů. Konkrétní kapacita disku se pak promítá přímo úměrně do počtu cylindrů – parametr „C". (Připomeňme, že pojem „cluster" sem nepatří, protože se jedná o interní záležitost souborového systému FAT, tak jako je „block group" interní záležitostí UNIXových souborových systémů.)

SCSI disky jsou odedávna adresovány lineárně. Skutečnou geometrii u nich sice lze zjistit z některých režijních datových struktur (mimo datový prostor, diagnostickým softwarem), prakticky ale tyto hodnoty k ničemu nejsou a BIOS ani operační systémy je nepoužívají. Softwaru, který trvá na adresaci pomocí nějaké CHS geometrie (např. DOSu a bootloaderům) prezentuje BIOS řadiče nějakou fiktivní geometrii (translaci lineárního prostoru).

Lze shrnout, že moderní disky v zásadě interně a pokud možno i navenek pracují s lineární adresací. Dnešní disk je dokonale transparentní blokové médium o určité velikosti bloku=sektoru (až na výjimky 512 B) a o určitém počtu bloků – součin těchto dvou čísel dává celkovou kapacitu. Disk tedy poskytuje operačnímu systému spojitý, lineárně adresovatelný prostor pro ukládání dat. Ať již zvolíme jakoukoli translaci, máme přístup v zásadě vždy k celému disku. Pokud nám nepřetečou datové typy pro CHS, žádný sektor se pouhou změnou geometrie „neskryje do hyperprostoru" – pouze se přemalují hranice fiktivních stop a cylindrů na spojitém řetězu sektorů. Změna translace by tedy prakticky neměla hrát roli při použití moderního operačního systému, který používá lineární adresaci sektorů.



Vyvstává zde několik zajímavých otázek.

Pokud dnešní IDE disky i operační systémy podporují (a standardně používají) lineární adresaci, k čemu je nám vůbec geometrie? Proč se trápíme nějakými fiktivními cylindry, hlavami a sektory? K čemu ta konkrétní čísla vlastně jsou? Proč je hlásí Linux, a proč hlásí zvlášť fyzickou a logickou, když vlastně nepoužívá ani jednu z nich? Kde vlastně Linux zjistí „logickou“ geometrii disku?

Pes je zakopán v tabulce rozdělení disku.

Tabulka rozdělení disku popisuje rozložení oddílů pomocí adres v „geometrickém“ formátu – odkazuje na konkrétní cylindry, hlavy a sektory. Proto systém nainstalovaný na disku nenastartuje s jiným typem translace geometrie, než pod jakým byl instalován. Pokud po změně typu translace v BIOSu ještě smažeme nultý sektor disku a systém reinstalujeme (včetně nového rozdělení disku), není to problém. Tabulka rozdělení obsahuje jako dodatečnou informaci také lineární adresu (číslo sektoru) začátku každého oddílu a jeho velikost, ale jak se zdá, tato informace není brána v potaz.

Samotný BIOS se o tabulku rozdělení nezajímá – jeho úkolem při bootu je pouze nahrát a spustit kód zavaděče z nultého sektoru (tedy z MBR). Zato klasický DOSový zavaděč je změnou geometrie důkladně zmaten – nultý sektor bootovatelného oddílu bude hledat na nesprávném sektoru (viděno brýlemi lineární adresace).

S tabulkou v tomto formátu pracují samozřejmě také operační systémy - také ony potřebují znát jednoznačné začátky a konce oddílů, byť si je nejprve interně převedou na lineární adresaci. V konečném důsledku na správný formát tabulky rozdělení disku dohlížejí v prvé řadě klasické nástroje na její editaci, tj. fdisk a jeho příbuzní (sfdisk, cfdisk disk druid, partition magic apod.) – je to fdisk, kdo vám vynadá, když mu zadáte natvrdo geometrii odlišnou od té stávající, nebo když mu natvrdo přednesete svou představu rozdělení s přesností na sektory, a nedodržíte přitom základní zaokrouhlovací pravidla.

Proč je vlastně tabulka rozdělení na PC dodnes založena na CHS geometrii? Je to především kvůli zachování vzájemné kompatibility mezi operačními systémy, nebo chcete-li kvůli zpětné kompatibilitě s MS-DOSem. A možná také díky lenosti dominantního Microsoftu něco s tím konečně udělat. Ačkoli se v dnešní době může zdát, že si Microsoft může specifikovat prakticky co chce, kupodivu ho ještě nenapadlo udělat něco s dědictvím zvaným tabulka rozdělení. Všichni ostatní se snaží především držet zavedených standardů.

Takže se například dodnes dodržuje, že oddíly mají končit (a nejlépe i začínat) na hraně cylindrů. Pravda je, že první oddíl často začíná na hraně stop - na začátku první stopy, počítáno od nuly. Takže zůstává nepoužita pouze první stopa (nikoli cylindr), s výjimkou nultého sektoru = MBR, ve kterém je tabulka rozdělení a kód zavaděče. Moderní operační systémy na dodržování těchto zaokrouhlovacích pravidel také úplně striktně netrvají.

Je tu ještě jedna zajímavá oblast, přinejmenším v Linuxu: zjištění/stanovení logické geometrie konkrétního detekovaného disku.

Linux pracuje s hardwarem disků a řadičů již od startu jádra (včetně inicializace hardwaru) přímo, bez pomoci BIOSu. Protože je však třeba při montování filesystémů přesně určit počátek a konec (resp. velikost) diskových oddílů, potřebuje Linux napřed zjistit geometrii, pro kterou byla vytvářena tabulka rozdelení – tj. „logickou“ geometrii. Tato může být stejná nebo jiná než geometrie fyzická. (Fyzická geometrie v zásadě nehraje roli, Linux na nízké úrovni beztak používá lineární přístup.)

Jak tedy Linux zjišťuje „logickou“ geometrii? Nabízejí se dvě možnosti: dotazem na služby BIOSu, nebo ze samotné tabulky rozdělení (která je vždy v nultém sektoru).

Ačkoli kód inicializačních rutin IDE v Linuxovém jádře naznačuje možnost zeptat se na logickou geometrii BIOSu, pokud podrobně prozkoumáme nižší patra ovladačů, zjistíme, že tato možnost není implementována. Prakticky se vždy používá možnost druhá, tj. zjištění logické geometrie z tabulky rozdělení, kterou Linux najde v nultém sektoru disku.

Což není až tak jednoduché, jak to na první pohled vypadá.

Tabulka rozdělení totiž neobsahuje explicitní informaci, že byla vytvořena „s takovou a takovou C/H/S geometrií“. Tuto informaci lze odvodit pouze nepřímo, z hodnot C/H/S jednotlivých oddílů. Odvozovací algoritmus bere v úvahu určitá pravidla o přípustných hodnotách počtu cylindrů, hlav a sektorů pro některé běžné translace, plus pravidla o začátku a konci oddílů na hraně cylindrů, a iterativně tuto heuristiku uplatňuje, dokud nedojde k použitelné geometrii.

Pokud Linux touto heuristikou nedojde k použitelné geometrii, přidrží se poslední známé platné geometrie (např. po vnucené úpravě tabulky rozdělení programem fdisk, který na závěr volá ioctl BLKRRPART pro znovunačtení tabulky rozdělení). Případně, pokud jde o první inicializaci disku při startu počítače (takže předchozí geometrie není známa), použije Linux geometrii fyzickou – nezávisle na translaci nastavené pro daný disk natvrdo v BIOSu. Fyzická geometrie se použije také v případě, kdy je nultý sektor prázdný (vynulovaný) – např. když je disk „factory clean“. Datové typy používané Linuxem pro zachycení geometrie nemají problém zpracovat jakoukoli fyzickou geometrii (hlášenou hardwarem IDE disku).

Lze tedy shrnout, že pokud je na disku tabulka rozdělení, která byla např. vytvořena v DOSu pod nějakou BIOSovou translací, linux načte a použije logickou geometrii z tabulky rozdělení. Pokud je nultý sektor vynulovaný nebo tabulka rozdělení nedává smysl, Linux použije poslední známou logickou geometrii, nebo (po restartu systému) zkopíruje fyzickou geometrii do geometrie logické.

U SCSI disků může nastat ještě jedna eventualita. SCSI disky totiž neznají pojem „fyzická geometrie“.

Na vyšší úrovni SCSI ovladačů se opět počítá s možností zjistit logickou geometrii z BIOSu. Implementace příslušné detekční funkce (metody) je svěřena do kompetence nízkoúrovňového hardwarově specifického ovladače konkrétního řadiče a až na výjimky tato funkce není implementována.

Druhou možností je, stejně jako u IDE disků, zjistit geometrii z tabulky rozdělení. Pozor, u SCSI disků je volána odlišná implementace této metody (ovladače pro IDE disky a pro SCSI disky mají každý svou odvozovací funkci).

Pokud je nultý sektor prázdný nebo tabulka rozdělení obsahuje nesmysly (připomeňme, že fyzická geometrie neexistuje), uplatní se jako třetí instance algoritmus, který si nějakou použitelnou geometrii „vycucá z prstu“. Toto je tedy již třetí funkce v rámci diskových ovladačů Linuxu, která používá heuristiky v oblasti diskových geometrií.

SCSI ovladače (na rozdíl od IDE ovladačů) dokonce neudržují pojem geometrie v persistentních paměťových strukturách, které se pro jednotlivé disky vytvářejí – potažmo neexistuje SCSI analogie „proc entries“ /proc/ide/hd<X>/geometry a při inicializaci oddílů či při spuštění fdisku (ioctl HDIO_GETGEO_BIG) se geometrie pokaždé znovu odvozuje výše uvedeným postupem.

Z výše uvedeného plyne, že pokud vynulujeme bootsektor na SCSI disku, nebo do něj zapíšeme nesmyslnou tabulku rozdělení, po příštím znovunačtení tabulky rozdělení (ioctl BLKRRPART) se linux nepřidrží dosavadní platné geometrie, ale vycucá si z prstu nějakou jinou.

Novější BIOSy vlastně také umí v omezené míře používat heuristiky na způsob Linuxu. Pokud v BIOSu nastavíme u typu translace volbu „auto“ (tj. nenastavíme natvrdo konkrétní translaci), prohlédne si BIOS při startu tabulku rozdělení, a pokud tato odpovídá některé podporované translaci, automaticky ji použije. Pokud tabulka rozdělení neodpovídá žádné známé translaci, použije se obvykle „normální“ geometrie - takže pokud se má z disku bootovat, bootloader zhavaruje.

Jistá část dnešních IDE a SCSI disků není prodávána „factory clean“, ale již s vytvořenou tabulkou rozdělení a souborovým systémem FAT (nebo FAT32) – což v mnoha případech nemusí vyhovovat, např. pokud chceme jiný souborový systém, více oddílů, nebo pokud chceme disk použít v diskovém poli (nepotřebuje tabulku rozdělení). Také při hrátkách s fdiskem a souborovými systémy nebo při boji s viry se lze snadno dostat do situace, kdy je nanejvýš vhodné disk úplně vymazat, tj. především vynulovat nultý sektor. Tím vezme za své obsah MBR – kód zavaděče a zavaděčových virů, a také tabulka rozdělení. Takže se ztratí také nepřímá informace o translaci, se kterou byl disk dosud používán. Takže u IDE disku zbyde pouze informace o fyzické geometrii, u SCSI disku ani ta ne.

V takové situaci (nebo lépe předem) je třeba si rozmyslet všechny důsledky, které má tato „ztráta logické geometrie“ pro software, který hodláme s diskem nadále používat. Chceme-li disk dále

používat jako bootovatelný s konkrétním BIOSem a operačním systémem, musíme v BIOSu nastavit translaci, která bude vyhovovat i operačnímu systému, přinejmenším pro potřeby bootu. Novou tabulku rozdělení, která bude tuto translaci reflektovat, lze poté vytvořit DOSovým fdiskem (který použije BIOSovou translaci), nebo i Linuxovým fdiskem a spol. za předpokladu, že nastavíme natvrdo požadovanou geometrii, kterou jsme opsali v BIOSu – u klasického interaktivního fdisku se toto zařídí v „expertním“ režimu, např. sfdisku lze domluvit příslušnými parametry na příkazové řádce.

CompactFlash

CompactFlash média mohou fungovat ve třech různých režimech – asi nejvíce se používá emulace IDE disku a přímý paměťově mapovaný přístup. CF karta zvolí konkrétní režim při své inicializaci, podle konfigurace určitých pinů v CF konektoru.

Pokud je CF karta připojena pasivní redukcí na IDE kanál, bude se vždy inicializovat jako IDE disk. CF sloty integrované na průmyslových procesorových deskách na bázi PC fungují bez výjimky v IDE režimu – CF karta je při startu detekována jako IDE disk, jakožto IDE disk vykazuje určitou fyzickou geometrii, BIOS na ni umí uplatnit jednu nebo dvě „soft“ translace (LBA a LARGE), prostor CF IDE disku lze rozdělit fdiskem, v oddílech vytvářet souborové systémy, nainstalovat operační systém a z disku bootovat.

Samotná CF paměť je pochopitelně lineárně adresovatelná (pomineme interní složitosti s rozloženým blokovým mazáním a zápisem). Zcela určitě nemá ve skutečnosti žádné cylindry a hlavy. Pokud se však CF karta tváří jako IDE disk, součástí emulace je, že hlásí určitou geometrii - hovoříme tedy opět o “normální” geometrii v terminologii PC BIOSu nebo o “fyzické” geometrii v terminologii Linuxu.

U malých CF médií je situace s geometrií analogická jako u prvních IDE disků, tak trochu se opakuje historie – fyzická („normální“) geometrie CF karty je natolik „malá“, že ani pro nejstarší software není třeba ji BIOSem překládat. Nicméně BIOS možnost translace přesto nabízí a každá translace dává jinou logickou geometrii. Patrně díky maličké kapacitě se u CF karet dosud nevyskytuje typický „společný jmenovatel“ velkých točivých disků, parametry 255H/63S (což by způsobovalo plýtvaní prostorem na nevyužité nulté stopě/cylindru).

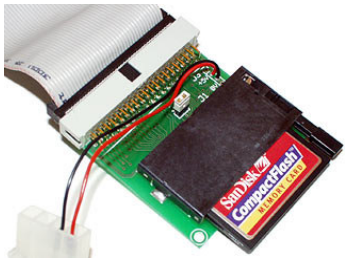
„Paměťové mapovaný“ režim používají některá zařízení z oblasti spotřební elektroniky a snad také USB CF mechaniky pro počítače. V tomto režimu není informace o fyzické geometrii dostupná – nezdá se však, že by to u zmíněných zařízení vadilo, protože spokojeně používají DOSovou tabulku rozdělení a FAT filesystém, které jsou na CF médiích z výroby vytvořeny. Pochopitelně – obsah datového prostoru vypadá v obou režimech stejně.

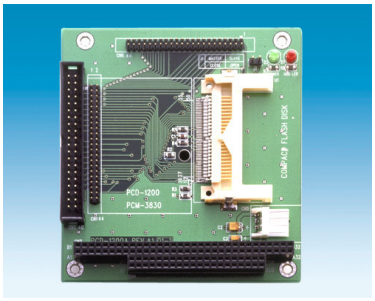
Možnosti připojení CF karty k počítači

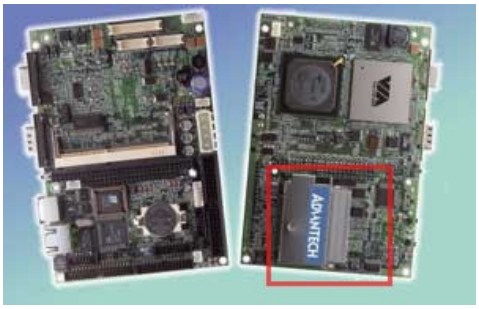
Jak již výše zmíněno, CF kartu lze připojit do počítače buď pasivní redukcí na IDE kanál, nebo pomocí USB čtečky.

Přímé připojení na IDE kanál

Pasivní CF=>IDE redukce se vyskytují jako samostatné desky či „dvojité konektory“, nebo jsou v průmyslových počítačích na bázi PC přímo integrovány na motherboardu (a interně připojeny na jeden kanál IDE řadiče).







S IDE režimem CF karet je ta potíž, že nelze provést hot-swap – po hot-swapu CF karta nereaguje na IDE transakce operačního systému. (Inicializuje se do paměťově mapovaného režimu?) Nepomáhá ani pokyn operačnímu systému k re-inicializaci IDE zařízení. Pokud chceme CF kartu vyměnit, lze to provést jedině při vypnutém počítači nebo s tvrdým resetem.

Výhodou přímého připojení na IDE kanál je to, že lze s CF kartou pracovat naprosto transparentně jako s IDE diskem – pokud pomineme možné problémy s rychlostí, kapacitou a ztrátami sektorů častým přepisem, lze na CF kartu v tomto režimu nainstalovat prakticky jakýkoli operační systém pro platformu PC.

CF/USB čtečky

CF čtečky na USB sběrnici mají tu výraznou výhodu, že podporují výměnu média bez restartu počítače (hot swap). Těžko říci, zda je tato možnost důkazem, že CF čtečky na USB používají paměťově mapovaný režim. V každém případě, pokud za chodu vytáhnete CF kartu a zasunete jinou, při příštím přístupu na tuto kartu Linux automaticky provede znovunačtení tabulky rozdělení (a detekci geometrie). Pochopitelně zde platí podobná pravidla pro ruční montování/odmontování souborových systémů, jako u disket – což je ovšem velmi přijatelné omezení.

V Linuxu jsou CF čtečky na USB vidět jako SCSI harddisky s výměnným médiem. Možnost vidět jedinou CF kartu střídavě jako IDE nebo SCSI disk je jistě úžasně vzrušující, ale má také své stinné stránky.

Detekce geometrie „out of band“ není USB/SCSI ovladači implementována – pokud tedy na CF kartě smažeme nultý sektor (například protože chceme rázně zatočit s původním obsahem CF disku včetně MBR), CF karta okamžitě přejde na geometrii, kterou si „vycucá z prstu“ příslušný heuristický algoritmus ve SCSI subsystému. Dlužno podotknout, že tato geometrie je obvykle odlišná od všeho, co navrhuje běžný PC BIOS pro tutéž CF kartu v roli IDE disku.

V této situaci nezbývá, než nějak jinak zjistit „fyzickou“ geometrii prezentovanou CF kartou v IDE režimu (či nějakou translaci, která se líbí BIOSu) a tuto zadat natvrdo linuxovému fdisku při vytváření nové tabulky rozdělení.

Lepší přístup je ten, že pokud se chystáme CF kartu v USB čtečce vynulovat, napřed si zjistíme její geometrii ze staré tabulky rozdělení (obvykle odpovídá fyzické geometrii, ale translace kompatibilní s BIOSem také nevadí), poté ji teprve vynulujeme a nakonec výše zjištěné údaje zadáme natvrdo fdisku.

Zásadní nevýhodou CF čteček na USB může být pro některé uživatele skutečnost, že pro USB nejsou ovladače v DOSu a že starší BIOSy nedokáží z USB bootovat. Potažmo nejde DOS nainstalovat na CF kartu přímo v USB čtečce. Pokud je třeba množit větší počet bootovatelných DOSových CF médií, nelze využít výraznou přednost USB CF čteček, kterou je možnost výměny CF karty bez restartu počítače (hot-swap).

CF/USB čtečky se dnes vyskytují často v podobě „interních“ kombinovaných čteček, např. „6 v jednom“, které se montují do 3.5“ pozice v počítačové skříni místo disketové jednotky – propoj na USB sběrnici vede uvnitř skříně. Jednotlivé sloty takové čtečky pro různá média se v Linuxu tváří jako několik SCSI zařízení – CF slot bývá mezi prvními, tedy /dev/sda nebo /dev/sdb. Některé sloty (vyšší SCSI zařízení) bývá někdy třeba napřed zprovoznit zápisem do několika větví „proc filesystému“ – samozřejmě pouze pokud je potřebujeme používat.

Na okraj – kombinace USB/IDE konvertoru a pasivní IDE/CF redukce

Svého času, nemaje USB čtečky, použil autor i tuto frankensteinovskou kombinaci. Výsledek byl zajímavý – zařízení bylo v linuxu vidět zcela korektně jako USB Mass Storage device, tj. jako emulované SCSI zařízení, ale nefungoval hot-swap samotné CF karty v pasivní IDE/CF redukci. K hot-swap výměně CF karty bylo třeba napřed vypnout USB/IDE konvertor (což v kernelu vyvolá USB device detach, a USB je hot-swappable), vyměnit kartu, a opět zapnout USB/IDE konvertor. Což je rozdíl oproti nativní USB/CF čtečce, kde funguje hot-swap samotné karty, aniž by se prováděl hot-swap celé čtečky na USB (nenastává USB detach příslušného zařízení, je ošetřena pouhá výměna média).

Instalace DOSu v USB čtečce

Konkrétní zákazník řešil problém, jak co nejsnáze hromadně instalovat DOS a jednoúčelovou aplikaci na CF karty, které používá ve svých průmyslových počítačích. Vzhledem k tomu, že MS-DOS se již neprodává, zákazník používá FreeDOS. Dosud prováděl instalace ručně v cílovém zařízení, s DOSem zdlouhavě startovaným z diskety, s nutností restartovat po každé instalaci.

Částečným řešením je jistě klonovat CF karty v USB čtečce pod Linuxem. Toto řešení ovšem z několika důvodů není optimální – zápis image (kompletního obsahu média) trvá oproti kopírování několika souborů poměrně dlouho, navíc lze klonovat pouze přesně identická média – o stejné velikosti, od stejného výrobce, se stejnou geometrií. Situace, kdy má tento zákazník na skladě velkou zásobu karet jedné velikosti ovšem není příliš běžná a navíc někteří výrobci (resp. jejich místní

importéři) dodávají pod jednou nálepkou několik různých druhů CF karet - se stejnou kapacitou, ale s jinou geometrií (protože s jinými vnitřnostmi).

Zatím nejlepším řešením je toto:

- Klonovací systém je založen na Linuxu a běží z bootovatelného CD
- CF disky se připojují prostřednictvím USB/CF čtečky – funguje hot-swap
- Instalace DOSu je řešena částečně prostředky Linuxu, částečně nativními diskovými utilitami FreeDOSu, který je spouštěn v emulátoru (dosemu)

Protože nové verze dosemu již neumožňují zpřístupnit celý disk včetně MBR (ve starších verzích dosemu vlastnost „wholedisk"), ale pouze konkrétní partition (včetně boot sektoru), je nezbytné provést rozdělení disku linuxovým fdiskem (přesněji v našem případě sfdiskem, protože ho lze volat ze skriptu).

Poté se nastartuje dosemu (bootuje FreeDOS z image 1.44" systémové diskety), kterému se jako disk D: namapuje budoucí fyzický bootovatelný oddíl – z CF karty, která je vidět jako SCSI disk. Dosemu na příkazové řádce obdrží dva příkazy ke spuštění: jméno dávky, která provede příkazy FORMAT a SYS, a nakonec příkaz EXITEMU – takže se na systémový oddíl zavede systém a dosemu se automaticky ukončí. Nakonec se do MBR zavede lilo (protože FreeDOS běžící pod DOSEMU nemá k MBR přístup).

Instalace Lila má několik háčků. Zaprvé je třeba v lilo.conf zadat, že to, co se teď jmenuje /dev/sda (CF karta), bude v cílovém systému BIOS vidět jako první IDE disk – k tomu slouží poměrně známá volba

```
disk=/dev/sda
     bios=0x80
```

Zadruhé je tu ten problém, že lilo potřebuje jako svou „boot partition" na cílovém disku oddíl typu ext2, na kterém najde dva stupně svého zavaděče (boot.b a chain.b). Takže naše DOSová bootovací CF karta bude muset mít dva oddíly: hlavní FAT oddíl s MS-DOSem, a ještě miniaturní pomocný bootovací oddíl se souborovým systémem ext2. Na tento pomocný oddíl s přehledem stačí cca 80 kB místa, obvykle se tedy vejde do jednoho až dvou cylindrů a zabere několik promile diskového prostoru na CF kartě. Samotné zavaděče zabírají cca 30 kB, něco spolkne režie souborového systému Ext2.

Další podmínkou je, že velký DOSový oddíl musí být na disku první, takže maličký pomocný oddíl ext2 bude umístěn na konci. Důvodem je to, že pokud umístíme ext2 oddíl jako první, nedostaneme se na velký DOSový oddíl přes CF čtečku z Windows. (Boot FreeDOSu na IDE kanálu normálně funguje.) Windows patrně u CF karet, resp. možná obecně u USB Mass Storage médií, nepočítají s tím, že by na disku bylo více oddílů, z nichž některé nebudou známého typu. Jinak řečeno, při zasunutí CF karty do USB čtečky se Windows snaží namontovat první primární oddíl CF karty a očekávají, že bude typu FAT. Pokud něco není v pořádku, prohlásí, že disk není naformátován a žádají o povolení disk „zformátovat" podle svého gusta.

Po tom všem je již hračkou namontovat pod Linuxem hotový FAT oddíl (/dev/sda1) a hlavní harddisk „klonovacího" počítače a nakopírovat na CF kartu připravenou DOSovou aplikaci. Alternativně se dá na CF kartu dostat přes USB z Windows.

Asi nejsložitějším problémem bylo vyřešit automatizované přerozdělení disku. Protože se autorovi nepodařilo přinutit k poslušnosti inteligentnější fukce sfdisku, je tento problém nakonec vyřešen několika funkcemi v Perlu, které implementují logiku diskové geometrie, tabulky rozdělení, vyhledávání úseků volného místa (nakonec nepoužito) a komunikaci se sfdiskem. Tělo perlového skriptu tvoří heuristika této konkrétní aplikace: vytvoř malý oddíl na konci disku zarovnaný na cylindry, pak vytvoř velký oddíl, který bude začínat na první stopě a končit na posledním dosud volném cylindru; nakonec zapiš novou tabulku rozdělení na CF kartu.

Poznámka o verzi jádra

Výše uvedený popis jaderných algoritmů v oblasti práce s tabulkou rozdělení a USB platí pro Linux ve verzích 2.2 a 2.4. V Linuxu 2.6 prodělaly IDE, SCSI a USB subsystémy výraznou přestavbu a autor tohoto článku dosud neměl příležitost tuto oblast zkoumat.

Literatura

Zdrojů bylo mnoho, hlavně manuálové stránky použitých programů a zdrojové kódy jádra.

Tabulka rozdělení disku je popsána např. na (nezávislé) stránce
http://www.ata-atapi.com/hiwtab.htm

Popis zapojení CF konektoru:
http://www.howell1964.freeserve.co.uk/parts/cf_pinout.htm
(Oficiální stránka http://www.compactflash.org/ je bez hesla dost k ničemu.)

Jako další čtení lze doporučit především Linux Large Disk HOWTO - kapitoly o geometrii a o „historii maximálních adresovatelných kapacit“
http://www.win.tue.nl/~aeb/linux/Large-Disk-4.html
http://www.win.tue.nl/~aeb/linux/Large-Disk-3.html
Alternativní umístění na stránkách Linux Documentation Project (neaktuální verze)
http://www.tldp.org/HOWTO/Large-Disk-HOWTO-4.html

Přílohy - vybrané zdrojové kódy

Základní skript, realizující vytváření bootovatelného systému na CF kartě, je napsán v shellu. Pomocný skript pro rozdělení disku je napsán v Perlu (má lepší možnosti práce s dynamickými daty). Autor dále napsal dvě utilitky pro práci s diskem: utilita „geom“ slouží ke zjištění geometrie IDE a SCSI disků (volá ioctl HDIO_GETGEO_BIG), utilita reset_disk explicitně vyvolá znovunačtení tabulky rozdělení (volá ioctl BLKRRPART).

format.sh

```
#!/bin/bash

#TARGET_DISK="/dev/hdc"
#TARGET_DISK="/dev/sda"
TARGET_DISK="/dev/sdb"

# Enter a string formatted as "C/H/S" as the first argument ($1)
# to force a particular geometry.

# We might be working on hot-swappable devices.
# Re-init the device just in case.
echo "Re-detecting disk, to accomodate a potential recent hot-swap:"
./reset_disk $TARGET_DISK

# This is much too "magical" (=undocumented) and inflexible.
#sfdisk $TARGET_DISK <<EOF
#,50,83
#,,6
#;
#;
#EOF

# This way we know exactly what sfdisk is doing.
./fdisk.pl $TARGET_DISK $1

PART1=`echo "$TARGET_DISK""1"`
PART2=`echo "$TARGET_DISK""2"`

dd if=/dev/zero of=$PART1 bs=512 count=10
dd if=/dev/zero of=$PART2


### fun with DOSEMU ###

# prevent dosemu from asking silly questions on first startup
if [ ! -d "/.dosemu" ]; then
   THIS_DIR=`pwd`
   cd /
   tar xvzf /usr/dosemu-private.tgz
   cd $THIS_DIR
fi

# prepare a special config file for dosemu
# Unfortunately, „<<HERE" files require temporary storage in the current directory,
# which is read-only on a CD. Hence, we have to echo line by line...
mv /etc/dosemu/dosemu.conf /etc/dosemu/dosemu.bak
echo "\$_hdimage = \"/usr/share/dosemu/freedos $PART1\"" >>/etc/dosemu/dosemu.conf
echo "\$_vbootfloppy = \"/usr/bowling/freedos-floppy-1440.img\"" >>/etc/dosemu/dosemu.conf
echo "\$_xms = (1024)" >>/etc/dosemu/dosemu.conf
echo "\$_ems = (1024)" >>/etc/dosemu/dosemu.conf
echo "\$_dpmi = (0x0400)" >>/etc/dosemu/dosemu.conf
echo "\$_video = \"vga\"" >>/etc/dosemu/dosemu.conf
echo "\$_console = (1)" >>/etc/dosemu/dosemu.conf
echo "\$_graphics = (1)" >>/etc/dosemu/dosemu.conf
echo "\$_chipset = \"plainvga\"" >>/etc/dosemu/dosemu.conf
echo "\$_mouse = \"mousesystems\"" >>/etc/dosemu/dosemu.conf
echo "\$_mouse_dev = \"/dev/gpmdata\"" >>/etc/dosemu/dosemu.conf
echo "\$_speaker = \"native\"" >>/etc/dosemu/dosemu.conf

# this does a FORMAT and SYS on $TARGET_DISK, partition 1
dosemu.bin '-I keystroke "\rFMT.BAT\r"'

mv /etc/dosemu/dosemu.bak /etc/dosemu/dosemu.conf

### done with DOSEMU ###
```

```
# this creates and EXT2 filesystem on the tiny helper partition for Lilo
mkfs.ext2 $PART2

# generate a lilo.conf
if [ -e "/var/lilo.conf" ]; then
   rm /var/lilo.conf
fi

#echo "prompt"            >/var/lilo.conf
#echo "timeout=30"        >>/var/lilo.conf
echo "default=DOS"       >/var/lilo.conf
echo ""                  >>/var/lilo.conf
echo "disk=$TARGET_DISK" >>/var/lilo.conf
echo "        bios=0x80" >>/var/lilo.conf
echo ""                  >>/var/lilo.conf
echo "boot=$TARGET_DISK" >>/var/lilo.conf
echo ""                  >>/var/lilo.conf
echo "other=$PART1"      >>/var/lilo.conf
echo "        label=DOS" >>/var/lilo.conf

# copy the bootloaders for Lilo to the target disk
mount $PART2 /mnt
cp /boot/* /mnt
umount $PART2
# and run lilo with our generated config file
mount $PART2 /boot
lilo -C /var/lilo.conf
umount $PART2

usleep 100
perl -e 'print "\a";'
usleep 200000
perl -e 'print "\a";'
usleep 200000
perl -e 'print "\a";'
```

fdisk.pl

```
#!/usr/bin/perl

# check the input argument - determine the disk device to be manipulated
if ($ARGV[0] =~ /.+/)      # if argument #1 is not empty, it should be a disk device node
{
   $disk = $ARGV[0];
   print "Using disk $disk\n";
}
else
{
   print "Usage:  ./fdisk.pl </dev/disk_device> [C/H/S]\n";
   print "No disk specified - exiting.\n";
   exit 1;
}
my $disk_bare_name = $disk;
$disk_bare_name =~ s/^\/dev\///;


# check that this disk exists
if (! (open(TEST_DISK, "<$disk")))
{
   print "Failed to open that disk. Exiting\n";
   exit(1);
}
else
{
   close(TEST_DISK);
}


# read disk geometry
if (open(GEOM, "./geom $disk |"))
{
   $_ = <GEOM>;

   if (! (/^[0-9]+\/[0-9]+\/[0-9]+\/[0-9]+$/))
   {
      print "Could not read disk geometry. Geom said:\n";
      print;
      print "Exiting\n";
      close(GEOM);
      exit(1);
   }

   /^[0-9]+/;           # cylinders - the first sequence of digits
   $cyls = $&;
   s/^[0-9]+\///;       # discard the "cylinders" section

   /^[0-9]+/;           # heads (tracks per cylinder) - now the first sequence of digits
   $heads = $&;
   s/^[0-9]+\///;       # discard the "heads" section

   /^[0-9]+/;           # sectors per track - now the first sequence of digits
   $sectors = $&;
   s/^[0-9]+\///;       # discard the "sectors" section

   /^[0-9]+/;           # Linear/LBA capacity (in sectors) - the last sequence of digits
   $LBA_capacity = $&;

   close(GEOM);
}
else
{
   print "Could not read disk geometry. Exiting\n";
   exit(1);
}
```

```
# comply to forced geometry, if asked for that

$sfdisk_forced_geom = "";

if ($ARGV[1] =~ /.+/)     # if argument #2 is not empty, it should be the forced geometry
{
   $forced_geom = $ARGV[1];

   if (! ($forced_geom =~ /^[0-9]+\/[0-9]+\/[0-9]+$/))
   {
      print "Obtained incorrectly formatted forced geometry string:\n";
      print "$forced_geom    ( should be in the form of C/H/S )\n";
      print "Ignored, sticking to geometry obtained by ioctl(HDIO_GETGEO_BIG)\n";
   }
   else
   {

      $forced_geom =~ /^[0-9]+/;          # cylinders - the first sequence of digits
      $cyls = $&;
      $forced_geom =~ s/^[0-9]+\///;      # discard the "cylinders" section

      $forced_geom =~ /^[0-9]+/;        # heads (tracks per cylinder) - now the first sequence of digits
      $heads = $&;
      $forced_geom =~ s/^[0-9]+\///;      # discard the "heads" section

      $forced_geom =~ /^[0-9]+/;          # sectors per track - now the first sequence of digits
      $sectors = $&;

      $sfdisk_forced_geom = "-C$cyls -H$heads -S$sectors";

      #    $LBA_capacity is unambiguous -> makes no sense to force a different one
   }
}

$capacity = $LBA_capacity / 2;    # => capacity in kB

print "C: $cyls, H: $heads, S: $sectors, Cap: $LBA_capacity sectors (per 512 B).\n";
print "Total disk capacity: $capacity kB\n";

#my $align = $sectors;            # aligned to track boundary
my $align = $sectors * $heads;   # aligned to cylinder boundary



# Try to run sfdisk, look for verbose error messages.
# No actual processing yet.
open(TEST_DEV, "sfdisk -l $disk 2>&1 |" );
my $dev_err = 0;
while (<TEST_DEV>)
{
   if (/error/) { $dev_err = 1; }
}
close(TEST_DEV);
if ($dev_err != 0)
{
   print "ERROR: Can't open $disk, sfdisk -l signals errors. Exiting\n\n";
   exit 1;
}
print "\n";



@part_table = ();

init_empty_part_table();
#read_part_table();
```

```
my $tmp_end = align_end($LBA_capacity - 1);
# We need 80 kB of space, times 2 (sector size is 512 B)
# We're using align_end() instead of align_start(), in order to round down.
$part_table[1]->{"START"} = align_end($tmp_end - (80 * 2)) + 1;
$part_table[1]->{"SIZE"} = $tmp_end - $part_table[1]->{"START"} + 1;
$part_table[1]->{"ID"} = "83";
$part_table[1]->{"BOOT"} = "N";

$part_table[0]->{"START"} = $sectors;
$part_table[0]->{"SIZE"} = $part_table[1]->{"START"} - $part_table[0]->{"START"};
$part_table[0]->{"ID"} = "6";
$part_table[0]->{"BOOT"} = "Y";

print_partitions(*STDOUT);

if (open(SFDISK_OUT,"|sfdisk $sfdisk_forced_geom $disk >/dev/null"))
{
   print "Writing partition table...\n";
   print_partitions(*SFDISK_OUT);
   close(SFDISK_OUT);
}
else
{
   print "Couldn't run sfdisk to write partition table!\n";
   exit 1;
}


print "\n";

exit 0;



sub init_empty_part_table
{
   my $part_num;
   for ($part_num = 0; $part_num < 4; $part_num++)
   {
      $part_table[$part_num]->{"START"} = 0;
      $part_table[$part_num]->{"SIZE"} = 0;
      $part_table[$part_num]->{"ID"} = 0;
      $part_table[$part_num]->{"BOOT"} = "N";
   }
}


sub print_partitions
{
   local *Part_Stream = shift;
   my $part_num = 0;

   print Part_Stream "unit: sectors\n\n";

   for ($part_num = 0; $part_num <= $#part_table; $part_num++)
   {
      print_partition(*Part_Stream, $part_num);
   }
}



sub print_partition
{
   local *Stream = shift;
   my $part_num = shift;

   print Stream $disk, $part_num + 1, " : ",
         "start= ", $part_table[$part_num]->{"START"},
         ", size= ", $part_table[$part_num]->{"SIZE"},
```

```
         ", Id= ", $part_table[$part_num]->{"ID"};
   if ($part_table[$part_num]->{"BOOT"} =~ /^Y$/)
   {
      print Stream ", bootable";
   }
   print Stream "\n";
}



sub align_start
{
   my $original = shift;
   my $result = $original;

   my $rounded_down = int($original / $align) * $align;
   if ($original > $rounded_down)
   {
      $result = ( int($original / $align) + 1 ) * $align;
   }

   return($result);
}



sub align_end
{
   my $original = shift;
   my $result = $original;
   $original++;            # to compensate an off_by_one issue

   my $rounded_down = int($original / $align) * $align;
   if ($original > $rounded_down)
   {
      $result = $rounded_down;
      $result--;           # to compensate an off_by_one issue
   }

   return($result);
}
```

geom.c

```
#include <stdio.h>          // general I/O
#include <fcntl.h>          // options for open()
#define O_LARGEFILE    0100000
#include <sys/ioctl.h>      // provides ioctl()
#include <linux/fs.h>       // BLKGETSIZE
#include <linux/hdreg.h>    // HDIO_GETGEO_BIG

int main(int argc, char** argv, char** env)
{
 int fd = -1;
 int err = 0;
 struct hd_big_geometry geom;
 unsigned int size = 0;
 char* filename = NULL;

   if (argc > 1)
   {
      filename = argv[1];
   }
   else
   {
      printf("Usage example: geom /dev/hda\n");
      printf("Output:  Cylinders/Heads/Sectors_per_track/Capacity   (in sectors)\n");
      exit(1);
   }


   fd = open(filename, O_RDONLY|O_LARGEFILE);
   if (fd <= 0)
   {
      printf("Unable to open %s. Exiting\n", filename);
      exit(1);
   }

   err = ioctl(3, BLKGETSIZE, &size);
   if (err)
   {
      printf("Error getting capacity using BLKGETSIZE\n");
      exit(1);
   }

   geom.cylinders = 0;
   geom.heads = 0;
   geom.sectors = 0;
   err = ioctl(3, HDIO_GETGEO_BIG, &geom);
   if (err)
   {
      printf("Error getting capacity using HDIO_GETGEO_BIG\n");
      exit(1);
   }

   printf("%u/%u/%u/%u\n",geom.cylinders, geom.heads, geom.sectors, size);
   close(fd);

   exit(0);
}
```

reset_disk.c

```
#include <stdio.h>          // general I/O
#include <fcntl.h>          // options for open()
#define O_LARGEFILE    0100000
#include <sys/ioctl.h>      // provides ioctl()
#include <linux/fs.h>       // BLKRRPART

int main(int argc, char** argv, char** env)
{
 int fd = -1;
 int err = 0;
 char* filename = NULL;

   if (argc > 1)
   {
      filename = argv[1];
   }
   else
   {
      printf("Usage example: reset_disk /dev/hda\n");
      printf("Calls the BLKRRPART ioctl() to make kernel re-read and re-interpret partition table.\n");
      printf("This only works if no devices (filesystems) are currently mounted off that disk.\n");
      exit(1);
   }

   fd = open(filename, O_RDONLY|O_LARGEFILE);
   if (fd <= 0)
   {
      printf("Unable to open %s. Exiting\n", filename);
      exit(1);
   }

   err = ioctl(3, BLKRRPART, NULL);
   if (err)
   {
      printf("Error re-reading partition tables using the BLKRRPART ioctl().\n");
      exit(1);
   }

   close(fd);

   exit(0);
}
```