Séta a lemezes egységek körül

Folytassuk a beviteli-kiviteli eszközökkel való ismerkedést! Az előző részben a terminálokról szóltunk, most a lemezes egységek kerülnek sorra.

A lemezes egységeket a merev-, illetve hajlékonylemezes meghajtókkal az élen vesszük sorra. Most csupán a lemezegység és az operációs rendszer viszonyáról lesz szó, a fájlrendszerek egy későbbi rész témáját képezik, mivel az a Unix világában sokkal összetettebb, mint ahogy elsőre hinnénk. Talán akad még olyan olvasó, akiben halványan feldereng e sorozat kettővel ezelőtti része, amikor is a beviteli-kiviteli eszközök és az operációs rendszer viszonyáról próbáltunk általános képet felvázolni. Kísérletet tettünk ezek csoportosítására is. Megállapítottuk, hogy ugyan roppant sok, látszólag egymástól teljesen különböző B-K eszköz van forgalomban, az operációs rendszer szemszögéből nézve azonban alapvetően két osztályba sorolhatjuk őket, mégpedig a karakteres és a blokkalapú B-K eszközök csoportjába.

Az előző hónapban egy jellemzően karakteres eszközt vettünk alaposan szemügyre, mégpedig a terminálokat. A lemezek – amelyek mostani értekezésünk tárgyai – azonban kivétel nélkül blokk eszközök, tehát kezelésük teljesen eltérő szemléletet kíván meg az operációs rendszerektől, mint például a terminálok esetében.

De mit is értünk a „blokkeszköz” kifejezés alatt? Az ilyen típusú egység mindig adott méretű adatcsomagokkal, azaz blokkokkal dolgozik. Például a merevlemezről nem olvashatunk be csupán egyetlen bájtot, hanem az egész blokkot be kell olvasnunk (amelynek mérete általában 512 bájt és 32 kilobájt közé esik) a memóriába, majd ott megkeresni a számunkra érdekes részeket. Ugyanez a helyzet az írás esetében is.

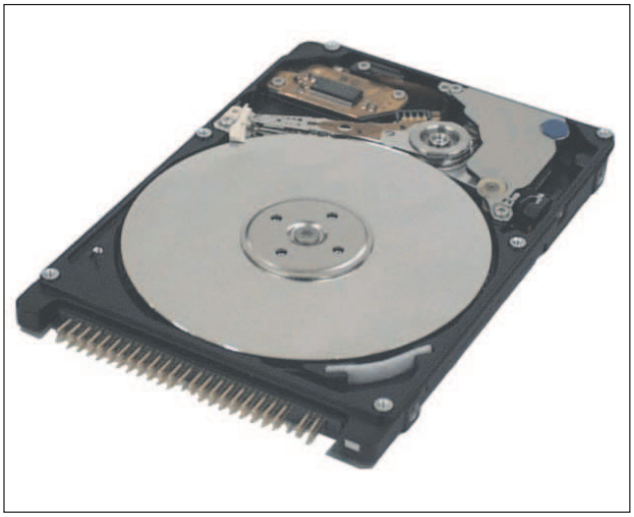
A blokkeszközökre jellemző az is , hogy a blokkok úgymond véletlen elérésűek. Ez azt jelenti, hogy bármelyik blokk külön címmel bír, és azokat egymástól teljesen függetlenül olvashatjuk, illetve írhatjuk újra.

(Bonyolítsuk a helyzetet azzal, hogy rámutatunk, a való életben mennyire nem használható ez a csoportosítás. Elmélkedjünk el egy kicsit a szalagos meghajtók esetén. Általában ezek is blokkokban tárolják az adatot, tehát blokkeszközök. Ám egy blokk megkereséséhez a meghajtónak az egész szalagot az elejére kell csévélnie (tekernie), majd egyenként végig kell mennie az összes blokkon, amíg el nem éri „úti célját”. Ez borzalmasan lassú művelet, ami már eleve kizárja, hogy valaki véletlen elérésű eszközként használhassa őket. Arról nem is beszélve, hogy egy meghatározott blokk újraírása általában nem lehetséges.)

Ez a csoportosítás ugyan nem tökéletes, azonban mindig jó támpontot ad egy operációs rendszer B-K eszközkezelőjének megtervezéséhez, habár sokféle típusú lemez létezik, amelyeket egy operációs rendszernek tudnia kell kezelni. A Unixokban a ramlemezeket, a hajlékony- és merevlemez-meghajtókat, valamint a CD-ROM és SCSI egységeket általában különálló meghajtó program kezeli. Még mielőtt megnéznénk őket külön-külön, ejtsünk pár szót magáról az eszközkezelő felépítéséről.

Itt is az a jól bevált módszer érvényesül, hogy minél kevesebb rész tudjon bármi kézzelfoghatót magáról az eszközről. Ezért egy jól megtervezett operációs rendszerben a következő kialakítás a legcélszerűbb: legyen egy erősen eszközfüggő rész, ami ugyan nagyon „buta”, mert csak a legalapvetőbb szolgáltatások elvégzésére alkalmas, viszont mindent tud az adott eszköz programozásáról. A felette elhelyezkedő rész elől azonban már rejtve van az eszköz silány világa, csak az eszközfüggő rész segítségével érheti el az adott egységet. Ide azok az eljárások kerülnek, amelyek az összes eszköztípus esetében azonosak.

A Linux- és a hozzá hasonló folyamatstrukturált operációs

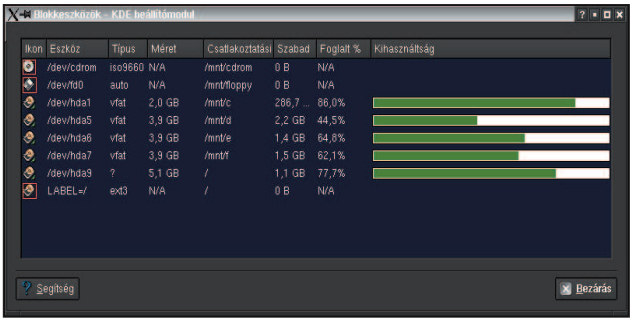


rendszerekben az eszközkezelők is külön folyamatok, mélyen a rendszermagban futnak, így számukra egyrészt a gép egyes részei elérhetőek, másrészt közös memóriaterületen osztoznak. Ez utóbbi azért célravezető, mert az eszközfüggetlen részek bármilyen típusú eszköz esetében szinte ugyanazokat az eljárásokat használják, így elég csak egyszer betölteni őket a memóriába. Az eszközfüggő részek egymástól teljesen eltérőek, ám méretük aránylag nem számottevő.

Az eszközfüggő rész tehát meglehetősen „buta”, csupán a legalapvetőbb műveletek elvégzésére képes. Mit is értünk alapvető művelet alatt? Például egy megadott blokk beolvasását, írását. Egy merevlemez-kezelő az állományokról vagy a könyvtárakról semmit sem tud, ami rendjén is van, hiszen erről gondoskodik majd a felsőbb területeken (régiókban) lakó fájlrendszerkezelő. Az is igaz azonban, hogy a fájlrendszerkezelő számára a lemez felépítése rejtett, tehát csak logikai blokkcímekkel képes dolgozni, míg az eszközfüggő rész kizárólag „fizikaiakkal”. Az eszközfüggetlen részek a lemezen található blokkokat lineáris sorozatnak tekintik, ami például a merevlemez esetében nem felel meg a valóságnak, mivel az cilinderekre és pályavonalakra van osztva.

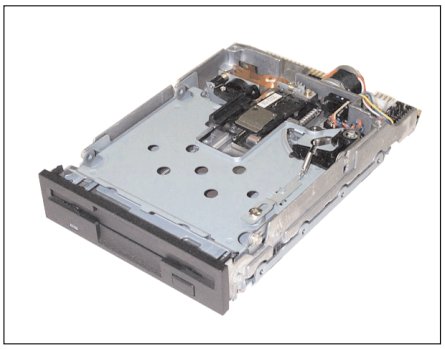
A Unixokban az eszközfüggő rész általában hat utasítást ismer, ezek pedig: `OPEN`, `CLOSE`, `READ`, `WRITE`, `IOCTL`, `SCATTERED_IO`. Az `OPEN` az adott eszköz rendelkezésre állását ellenőrzi, a `CLOSE` pedig a még az átmeneti tárban

lévő adatok azonnali kiírását biztosítja.
Az előző részben már találkozhattunk az átmeneti tár fogalmával. Amikor az eszközfüggő részt megkérjük, hogy írjon ki egy blokkot, lehet, hogy a lemezmeghajtó éppen valami mással van elfoglalva, vagy hatékonysági megfontolásokból (lásd később) először egy ideiglenes tárba, az átmeneti tárba rakja azt. Ezután visszaküld egy üzenetet, hogy a feladatot sikerült végrehajtania, habár maga a kiírás még nem történt meg, de majd arra is sort kerít unalmas perceiben (ezért is veszélyesek a váratlan áramszünetek, vagy amikor a rendszer rendeltetésszerű leállítása helyett egyszerűen csak „kirúgjuk” gépünket az áramellátásból, mert azok az adatok, amelyek csak az átmeneti tárban vannak, még nem íródtak ki, és a gép áram nélkül maradásakor örökre elvesznek).
Az `IOCTL` utasítások eredetileg az eszköz valamely műveletjellemzőjének megváltoztatására hívatottak. A leggyakrabban emlegetett példa a soros kapu (bár az nem blokkeszköz) átviteli sebességének és párosságának (ami a hibaellenőrzésben játszik szerepet) a megváltoztatása. Az `IOCTL` műveletekre blokkeszközök esetén viszonylag ritkán van szükség. A merevlemezek esetében például a lemezfelosztási (partition table) tábla megváltoztatása tekinthető ilyen műveletnek.
A `SCATTERED_IO` lehetőséget biztosít arra, hogy egyszerre több blokk írását, illetve olvasását is elvégezhessük (ellentétben a `READ`-del és a `WRITE`-tal, ahol egy utasítással csak egyetlen blokkhoz férhetünk hozzá). Ennek értelme szintén a hatékonyság növelésében rejlik, mivel a lemezkezelőnek lehetősége van dönteni, hogy a kért blokkokat például milyen sorrendben olvassa be.
Az eszközfüggetlen rész is közönséges folyamat (noha a rendszer szintjén fut), de csak az eszközfüggő részekkel tart kapcsolatot, az eszközzel közvetlenül nem. Egyik legfontosabb feladata a fájlrendszertől érkező utasítások végrehajtása, illetve olyan formára hozása, amelyet a nála alacsonyabb szinten lévő részek is megértenek. Ezek után térjünk is rá magukra a lemezekre. Először az úgynevezett ramlemezt említenénk, ami nem más, mint amikor a memóriának egy szeletét elkülönítjük, és úgy használjuk, mintha hagyományos lemez lenne. Erre a szolgáltatásra leginkább a merevlemez nélküli számítógépeknél, illetve a rendszer telepítésekor lehet szükségünk.
A ramlemez-meghajtó működése nem nevezhető bonyolultnak. Két utasítást ismer: a blokk írását és a blokk olvasását. Mindkét esetben annyi csupán a dolga, hogy kiszámolja az adott blokk helyét a fizikai memóriában (a ramlemez kezdeti címéhez hozzáadja azt), majd elvégzi az adatok mozgatását.
A Unixban a ramlemezek négy fajtája lelhető fel. Az első a */dev/ram* eszközfájlhoz rendelt. Ez tulajdonképpen a hagyományos ramlemez, aminek a méretét és kezdőpontját szabadon meghatározhatjuk. A */dev/null* eszköz is biztosan ismerős lehet a tapasztaltabb felhasználók számára, de azt kevesen gondolnák, hogy ennek a kezelése is a ramlemez-meghajtóra hárul. Természetesen az is igaz, hogy nincs túl sok gond vele, mivel az ide irányított adatokat egyszerűen csak el kell dobni. Ez az eszköz akkor tehet jó szolgálatokat, amikor például egy parancsnak nem vagyunk kíváncsiak a végeredményére. Ilyenkor annyi a feladatunk, hogy a kimenetét a */dev/null*-ba kell irányítanunk.
Van még két, a ramlemezekkel szorosan összefüggő eszköz, a */dev/mem* és a */dev/kmem*. Ezek használatáról a felhasználói szinten dolgozó alkalmazások nem is álmodhatnak, mivel ezek magának a rendszer memóriájának a közvetlen elérését teszik lehetővé. Az első az egész fizikai memóriához való hozzáférést teszi lehetővé (a 0. bájttól kezdve, ahol az ütemező tárgyalásakor bemutatott megszakításvektor-tábla kezdődik). A */dev/kmem* nullpontja a rendszermag adatmemóriájának kezdete. Érdemes megjegyeznünk, hogy a */dev/mem* lefedi a */dev/kmem*-et, azaz a */dev/mem*-en keresztül is elérhetjük a rendszermag adatmemóriáját, feltéve, ha pontosan tudjuk, hol is kezdődik.



A */dev/ram*, a */dev/mem* és a */dev/kmem* eszközkezelői megegyeznek, csak míg az első egy, a felhasználó által meghatározható memóriatartományra vonatkozik, addig a másik kettő magára a fizikai memóriára. Természetesen az utóbbihoz a közönséges felhasználók nem férhetnek hozzá, mivel ezzel könnyedén kárt lehet okozni, de az ügyesebbek fel is törhetik a rendszert – leginkább a különböző hibakereső programok számára tehet jó szolgálatokat.
De térjünk is rá a valódi fizikai lemezekre, azaz a merev- és hajlékonylemez-egységekre. Még mielőtt mélyebben belemerülnénk az operációs rendszer lemezkezelésének rejtelmeibe, nem árt, ha szólunk pár szót az eszközök felépítéséről is.
Maga a lemezegység cilinderekbe szervezett, amelyek pályavonalakból állnak, a pályavonalak pedig szektorokból. Blokkeszközről lévén szó, minden szektor ugyanannyi bájtból áll, és a pályavonalankénti szektorok száma is megegyezik. Ez elgondolkodtató, mivel a lemez peremén lévő pályavonalak nyilván hosszabbak, mint a bentebb lévők, tehát több szektor is elfér rajtuk. Az IDE-eszközök esetében bizony több szektor található a külsőbb területeken, mint a belsőkben, ám erről az operációs rendszer nem tud semmit, mivel az előbbi kezeléséről maga az IDE-vezérlő gondoskodik.
A ramlemez-meghajtóval szemben a valódi lemezeknél sajnos számolnunk kell azzal is, hogy egy művelet elvégzése bizony időbe telik. Míg a memóriába való írás, illetve az onnan történő olvasás gyakorlatilag azonnal, egy órajel alatt végbemegy, addig a lemezmeghajtó esetében a fejet először a megfelelő cilinder fölé kell „állítani” (positioning), majd a lemezt úgy kell forgatnunk, hogy a megfelelő szektorok kerüljenek a fej alá. A tényleges beolvasás, illetve írás csak ez után kezdődhet meg.
A lemezműveletek tehát borzalmasan lassúak. Szerencsére nem maga a tényleges írás, illetve olvasás veszi ez az idő nagy részét, hanem a fej célba állítása. Ezért ha az operációs rendszer arra törekszik, hogy a beérkező lemezműveletek végrehajtásánál egyfajta sorrendet állítson fel közöttük, mégpedig úgy, hogy a fejet a lehető legkisebb úton kelljen mozgatni, akkor lényeges hatékonyságbeli növekedést érhetünk el. Ennek fényében nézzük meg, miként is működhet egy valamirevaló operációs rendszer lemezkezelő programja. Egy lemezműveletre való felkérés bármikor érkezhet, akkor is, amikor a merevlemez épp javában dolgozik az előző műveleten. Ezért először gondoskodnia kell a kérelmek tárolásáról. Ezt egy egyszerű láncolt lista formában teszi meg, vagyis az

összes új elvégzendő műveletet hozzáfűzi a lista végéhez, az elvégzetteket pedig kitörli onnan.
A lemezkezelés hatékonyságának kulcsa az, hogy a lemezkezelő milyen elv alapján hajtja végre a listában szereplő utasításokat. Könnyű belátni, hogy nem lenne túlzottan célravezető, ha a kéréseken beérkezésük sorrendje alapján kezdene el dolgozni, mivel lehet, hogy egymástól teljesen különböző cilindereken lévő szektorokkal kéne foglalkoznia, ami ebben az esetben azzal járna, hogy a fejnek kilométereket kéne oda-vissza rohangálnia.
Ezért e nehézség megoldására az egyik legismertebb módszer az úgynevezett liftes algoritmus, ami – mint neve is utal rá – sok szempontból hasonlít a magasabb épületekben fellelhető felvonószerkezetek programjára. Csak itt a lift az olvasófej, az emeletek a cilinderek, a beérkező kérések pedig az egy-egy emeletről érkező hívásoknak felelnek meg.
A liftek általában a következő algoritmus követik: miközben fel-le száguldoznak, a kérések a felhasználóktól (illetve a ház lakóitól) véletlenszerűen érkeznek a különböző emeletekről.



A lift azonban mindig csak egy irányba mozog, amíg ugyanabba az irányba van kérés. Az irányváltoztatás csak akkor következik be, amikor az eredeti irányban lévő kérések már elfogytak.
Sebességnövekedést érhetünk el úgynevezett pályavonalankénti raktározással is. Mint már említettük, a legnagyobb időt a fej cilinderek közötti mozgatása veszi el, ehhez képest a szektorok beolvasása, illetve átmozgatása a memóriába szinte pillanatok műve. Sőt mi több, amikor a lemezt elforgatjuk a fej alatt, hogy a megfelelő szektort kiolvashassuk, a fejnek alkalma nyílik a pályavonalon található többi szektor beolvasására is. Ezért az a bevált módszer, hogy az adott pályavonalon található összes szektort beolvassuk. (A jelenlegi vezérlők többsége ezt már saját maga is megcsinálja, a beolvasott adatokat pedig a saját belső átmeneti tárába helyezik, ahonnan majd később átkerülhetnek a memóriába).
Az utolsó dolog, amire a lemezkezelőnek figyelnie kell, a hibakezelés. A ramlemezek esetében kizárólag programozási hibáról beszélhetünk, például egy nem létező blokkot szeretnénk elérni. Ebben az esetben sok mindent nem lehet tenni, ki kell javítani a lemezkezelő programot. A valódi lemezeknél azonban már külső tényezők is közrejátszhatnak, ha például porszemcse kerül a fej és a lemez közé, vagy csütörtököt mond a vezérlőelektronika.
Egy hiba észlelésénél mindig el kell dönteni, mit tegyünk a kialakult helyzettel. Vannak olyan hibák, amelyek csak valamely ideiglenes ok miatt állnak fent. Ilyen lehet a lemezfelületre került porszemcse. Ilyenkor általában az „idő majd mindent elrendez” elve érvényesül, azaz az adott művelet egy későbbi megismétlésével a hiba megszűnik.
De nem mindig. Ha az adott blokkon fizikai sérülés keletkezett, hiába olvassuk a végtelenségig, nem jutunk előbbre. Ilyenkor a „bad sector” esete forog fenn, tehát tartózkodnunk kell a további használatáról. Régebbi lemezek esetén ez a feladat a lemezkezelőre hárult. Neki kellett számon tartania azt egy különleges állományban, hogy mely blokkok hibásak, és azokat foglaltnak tekintette.
A mai merevlemezek vezérlői már nagyon okosak, saját maguk tartják nyilván a hibás blokkokat. A lemezeken található néhány úgynevezett tartalék pályavonal. Ha a vezérlő talál egy hibás blokkot, akkor az egész pályavonalat egy tartalékkal helyettesíti. Tehát az összes, a hibás szektort tartalmazó pályavonalra vonatkozó kérés valójában a kijelölt tartalék pályavonalon fog végbemenni. Mivel erről maga a lemezegységen található elektronika gondoskodik, az operációs rendszer erről semmit sem tud, sőt, még azt sem, hogy a lemez valahol hibás blokkot tartalmaz. Ez kényelmi szempontból valóban hasznos, ám könnyen felborulhat például a liftes algoritmus által nyújtott sebességnövekedés (mondjuk abban az esetben, amikor a hibás blokk jó messze esik a tartalék pályavonalakat tartalmazó cilinderektől).
Időnként úgynevezett keresési hibák is előfordulnak, ilyenkor a fej nem a megfelelő cilinder fölé kerül. A fejet mozgató kis motort is a vezérlő irányítja, ami pontosan tudja, hogy a fej épp hol tartózkodik. Amikor utasítást kap egy blokk kiolvasására, áramimpulzusokat küld a motornak, hogy lökdösse arrébb a fejet ennyi cilinderrel és ebben az irányban. A merevlemezek esetében a vezérlő ezt észlelni tudja, és kijavítja a hibát, tehát a merevlemez-kezelő megint mentesül a munkától. A hajlékonylemez-kezelőnek rosszabb sors jutott: ő nem számíthat ilyen jellegű segítségre a hajlékonylemez-vezérlőtől.
Igazából a hajlékonylemez vezérlője sokkal fejletlenebb a merevlemezekénél, ami azzal a furcsa következménnyel jár, hogy hiába egyszerűbb felépítésű a hajlékonylemez, a kezelőprogramja sokkal bonyolultabb, mint a merevlemezé. Ha keresési hiba lép fel, akkor ahelyett, hogy a vezérlő önállóan megoldaná, a lemezkezelőnek fel kell szólítania a lemezmeghajtót, hogy állítsa be újra a fejet, azaz tolja oldalra, amíg csak bírja, és a pillanatnyi cilindert nyilvántartó változóját állítsa 0-ra. Ezután ismét kísérletet lehet tenni a szükséges cilinder megkeresésére. Akadnak más olyan gondok is, amelyek a merevlemezek kezelésénél nem kell számítanunk, de erről inkább a következő részben szólnánk részletesebben.
A legkellemetlenebb hiba, ami előfordulhat, a vezérlőprogram nem megfelelő működése. Tulajdonképpen ez is egy közönséges program, tehát előfordulhatnak benne hibák. Például bizonyos szerencsétlen véletlenek közrejátszásából fakadóan elképzelhető, hogy végtelen ciklusba kerül vagy hasonló hiba lép fel. Ilyen esetben sincs minden veszve, mert megoldható a vezérlőprogram újraindítása, de elképzelhető, hogy ez sem segít. Ekkor jön a már jól ismert magpánik (kernel panic) felirat. Szerencsére ilyesmi nagyon ritkán fordul elő.
A továbbiakban folytatni fogjuk a lemezekkel való ismerkedést, még gyakorlatiasabb megközelítésben. A következő alkalommal kicsit részletesebben bemutatjuk a DMA-t, ejtünk pár szót az LBA-ról, továbbá a lemezkezelők működését elemezgetjük egy picit. Ha ezzel megvagyunk, egy egészen egyedi bevitelikiviteli eszköz következik, az óra, ami fontosabb szerepet játszik az operációs rendszer életében, mint gondolnánk.

Garzó András
(garzoand@interware.hu) körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.