A fájlrendszer feladata (13. rész)

Elérkeztünk következő nagy témakörünkhöz, a fájlrendszerhez. Ez nagyon bonyolult rendszer, aminek sok feladatot kell megoldania, ezért a mostani részben többnyire csak az alapfogalmakat járjuk körbe.

A fájlok és könyvtárak fogalma senkitől sem állhat távol, hiszen a fájlkezelés az operációs rendszereknek az a feladata, amit minden felhasználó először tanul meg. Ha azonban pontosan meg kellene határoznunk, hogy mik is azok a fájlok, illetve mi a feladata az operációs rendszer fájlkezelő „alrendszerének” (a továbbiakban: fájlrendszer), aligha adhatnánk pontos választ, ha csupán a felhasználó szemszögéből közelítenénk hozzá. Ezért első ízben ezt a témakört egy kissé „elvont” formában tárgyaljuk. Erre azért van szükség, hogyha majd a megvalósítást tárgyaljuk, érthetőbbek legyenek azok a szempontok, amelyeket egy jelenlegi korszerű operációs rendszer tervezésekor figyelembe kell venni.

A fájlrendszer tehát olyan valami, aminek igazából egyetlen feladata van: az adattárolás. Tekintsünk rá úgy, mint egy olyan dologra, amibe ha beleírunk valamilyen adatot, akkor azt megőrzi (elméletileg végtelen ideig, a rendszerösszeomlástól és egyéb katasztrófáktól függetlenül), és kérésre bármikor visszakaphatjuk tőle. Ezt úgy is megfogalmazhatjuk, hogy a beírt adat bármikor visszakereshető.

Miért van szükség adatot tárolni képes eszközre? Három különböző okból – ezek közül az első minden felhasználó számára ismert. Amikor egy folyamat befejezi futását, visszaadja az általa használt memóriarészt. Az itt tárolt adat így örökre elvész. Bizonyos alkalmazásoknak azonban szüksége lehet rá, hogy a memóriájuk tartalma ne vesszen a feledés homályába, hanem a futtatásuk után is megmaradjon. Gondoljunk csak az adatbázisrendszerekre: teljesen használhatatlan lenne egy olyan megoldás, aminél a rendszer minden újraindítása után az összes adatot újból be kellene táplálni.

A másik gond, ami megköveteli a fájlrendszer meglétét, az, hogy a számítógép memóriája véges (ugyanis határt szab neki a virtuális címtér). Ezért előfordulhat, hogy az operációs rendszer nem tud annyi memóriát adni egy-egy folyamatnak, amiben az egész általa használt adathalmaz elfér. Például egy bankszámlákat nyilvántartó rendszerben lehetetlen minden számlát és átutalási kérelmet a memóriában tartani.

A harmadik (ám nem kevésbé fontos) nehézség az, hogy ez idáig nem volt lehetőség arra, hogy több folyamat egy időben ugyanazzal az adattal dolgozhasson. Visszatérve a banki nyilvántartós példához: ha egy folyamat az összes bankszámla adatait a memóriában tartotta, ahhoz más nem férhetett hozzá, mivel a memóriája a többi folyamat számára elérhetetlen.

A fájlrendszer azonban lehetővé teszi a folyamatoktól teljesen független adattárolást. Ez azt jelenti, hogy az adatok megőrzését a legcsekélyebb mértékben sem befolyásolhatja a folyamatok indulása és befejeződése. (Például ha valamelyik folyamat létrehoz egy állományt, akkor az egészen addig ott is marad, amíg valaki le nem törli. A fájlrendszer számára azonban teljesen mindegy, hogy melyik állományt melyik folyamat hozta létre, e szempontból nem lehet megkülönböztetni a fájlokat egymástól.)

Ahhoz, hogy a fent leírtakat biztosítani tudjuk, a fájlrendszernek a következő három dolgot kell tudnia: először is meg kell oldania, hogy az adat ne veszhessen el, azután is megmaradjon, hogy a folyamat befejezte a futását vagy egy összeomlás miatt újra kellett indítani a rendszert. Fel kell készülnie arra is, hogy az itt tárolt adat mérete lehet, hogy igen nagy lesz. A harmadik és egyben utolsó feladat: semmiféle korlátozás se legyen arra nézve, hogy egy adatot egyszerre hány folyamat érhet el.

Az első feladatot egyszerűen megoldhatjuk olyan módon, hogy az adatot például merevlemezen tároljuk. A merevlemez azonban buta eszköz, saját maga nem képes a tárolt adatok rendszerezésére. Ez az operációs rendszer feladata, ami tárolási egységeket, úgynevezett fájlokat vezet be, és a kezelésükről is gondoskodik. Az operációs rendszernek azt a részét, ami az állományokat kezeli, gondoskodik azok hozzáférhetőségéről és védelméről, fájlrendszernek nevezzük. (A továbbiakban a „fájl” és az „állomány” kifejezések ugyanazt a fogalmat takarják).

A memóriakezelés esetében a felhasználók elől a legtöbb folyamat rejtve maradt. A felhasználók számára érdektelen volt, hogy a virtuális memória éppen lapozott-e vagy szakaszolt. A fájlrendszer esetében más a helyzet, ezzel a felhasználó már közvetlenül „érintkezik”. Ezért a fájlrendszereket két különböző szempont alapján szokás tanulmányozni. Az első eset, amikor a felhasználó szemszögéből vizsgálódunk. Ilyenkor az számít, hogy miként végezhetünk különböző műveleteket a fájlrendszerrel, milyen szabályokat kell betartanunk az állományok névválasztásánál, milyen tulajdonságokkal bírnak az állományok stb. Ezek olyan dolgok, amelyeket a felhasználóknak ismerniük kell, és minden rendszerben másmilyenek.

A másik vizsgálati szempont a megvalósítás. Ebben az esetben azt elemezzük, hogy az állományok miként vannak tárolva a lemezen, hogyan tarthatjuk nyilván az üres blokkokat, milyen módon épülnek fel a könyvtári bejegyzések stb.

Minket természetesen az utóbbi szempont érdekel, ehhez azonban szükség van arra, hogyha csak felületesen is, de „felhasználói szemmel” is megnézzük a fájlrendszer szolgáltatásait. A következő fogalmak lehet, hogy számos olvasó számára már ismertek lesznek, mindenesetre fontosnak tartjuk, hogy kitérjünk rájuk, mivel a megvalósítás tárgyalásakor gyakran elő fognak bukkani.

Fájlnév, fájlszerkezet, fájltípus

A fájlok azért vannak, hogy a felhasználók elől el legyenek rejtve a számukra érdektelen részletek, például az, hogy az adataik fizikailag hol is tárolódnak. Inkább egy elvont eszközt adunk a kezébe, amit könnyen megjegyezhető névvel láthat el, és egyszerűen végezhet vele különböző műveleteket.

A fájlnévre vonatkozó megkötések minden operációs rendszerben mások, de a legtöbb rendszerben lehetőség nyílik a fájlnév kibővítésére, úgynevezett kiterjesztés hozzáadásával

(amelyet . – ponttal – elválasztva írhatunk a fájlnévhez). Léteznek olyan rendszerek, amelyekben komoly jelentőséggel bírnak ezek a kiterjesztések, de akad olyan is, amiben kizárólag csak a felhasználó „tájékoztatására” vezették be, ez ugyanis utal a fájl tartalmára. (A DOS/Windows rendszerekben például a futtatható bináris programok csak ***.EXE*** kiterjesztéssel bírhatnak, különben nem indíthatók el. A Unix esetében a futtatható fájloknak nincsen kiterjesztésük, de ez csak a „szokás hatalma”, valójában semmi sem tiltja, hogy a futtatható binárisoknak kiterjesztést adjunk.)

A fájlszerkezet azt jelenti, hogy az állományban lévő tartalom milyen szerkezetbe rendeződik. Ebből háromféle létezik; az első a legáltalánosabb, a Unix és a Windows rendszerek is ezt használják. Itt a struktúra maga a strukturálatlanság. Az operációs rendszer számára minden állomány csupán bájtok sorozata, egyáltalán nem foglalkozik annak mélyebb „jelentéstartalmával”. Ez ugyan kezdetleges megoldásnak tűnhet, de valójában ez biztosítja a legnagyobb rugalmasságot, hiszen teljesen a felhasználóra van bízva, hogy az állományokban mit és milyen módon tárol.

Ez az elv ma már teljesen hétköznapinak számít, de létezett olyan operációs rendszer (például a CP/M egy ősi változata), ami gyökeresen eltért ettől a szemlélettől. Ebben minden állománynak jól meghatározott szerkezete volt, mégpedig rögzített méretű (128 bájtos) rekordok sorozataként tekintett a fájlokra, sőt maguknak a rekordoknak is előre meghatározott felépítésük volt. Ez egy kicsit furcsának tűnhet, de valójában akkoriban nagyon célszerű volt egy olyan gépen, ami lyukkártyákkal és sornyomtatókkal dolgozott.

A rekordszerkezet ma sem halt ki teljesen, csak továbbfejlődött. A rekordok hosszának már nem kell állandónak lennie, és az egész faszerkezetbe szervezhető. Egyetlen megkötés, hogy – az adatbázisokhoz hasonlóan – minden rekordnak tartalmaznia kell egy kulcsmezőt. Fontos, hogy a rekordok fájlon belüli elhelyezkedését is az operációs rendszer intézi, tehát a felhasználónak nincs beleszólása, hogy melyik rekord hova kerüljön. Az ehhez hasonló rekordszerkezettel elsősorban adatfeldolgozásra használt nagygépes rendszereknél találkozhatunk. Például a VMS is támogat valami ehhez hasonlót.

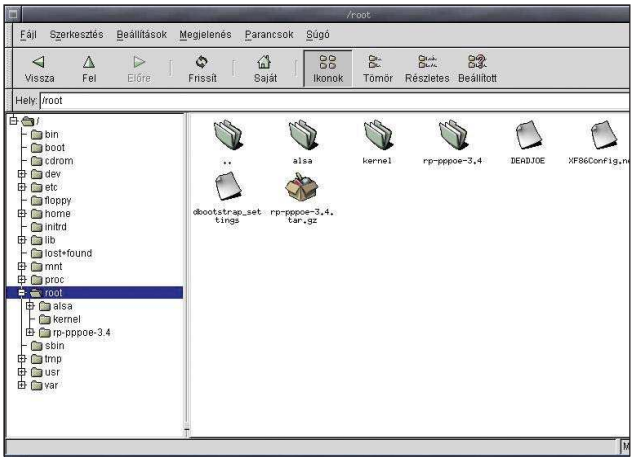
A fájlszerkezet tehát arra ad választ, hogy az adott állományban az adat milyen szerkezetben van tárolva. Azt azonban, hogy az állományban milyen jellegű adatra lelhetünk, a fájltípus mondja meg, amit szintén több különböző csoportra bonthatunk.

A fájltípusok első csoportja az egyszerű felhasználói állományok, amik a felhasználók (illetve az alkalmazások) adatait tartalmazzák. A második csoportba a rendszerfájlok tartoznak, amik a fájlrendszer számára hasznos adatot hordoznak. Ilyenek (általában) a könyvtárak is, amik a felhasználóknak „mutatott” hiearchikus szerkezetet teszik lehetővé. A Unixok egy másik fájltípust is támogatnak, mégpedig az eszközfájlt. Valójában ez két különböző típus gyűjtőneve: a karakterspecifikusé, amelyek a karakteres eszközök B-K műveleteit „vezetik ki” a fájlrendszerhez, és a blokkspecifikusé, amik a blokkos (lemezes) eszközök kezelésére szolgálnak.

Az eszköz- és rendszerállományokra csak a megvalósítás tárgyalásakor térnénk ki részletesebben, de az egyszerű felhasználói állományokról most kell szólnunk. Ezt a csoportot további két osztályra bonthatjuk: azokra, amelyek tartalma a felhasználó számára érthető, és azokra, amelyeké nem érthető. Kicsit szakszerűbben megfogalmazva: a szövegesekre és a binárisokra.

A szöveges állományokra az a jellemző, hogy tartalmukat mindenféle átalakítás nélkül kiírathatjuk a képernyőre vagy átküldhetjük a nyomtatónak. Fontos, hogy a szöveges állományok minden rendszerben egy kicsit másképp épülnek fel. Például a DOS esetében minden fájl végén egy úgynevezett „kocsi vissza” jel volt, de a rendszerek között különbség lehet például az „új sor” jelekben is. (A rendszer szöveges állományokhoz való „viszonyában” is komoly különbségek vannak. Például a Unix nagy hangsúlyt fektet a szöveges állományok magas szintű kezelhetőségére, míg egy DOS parancssorban csak a legegyszerűbb feladatokat végezhetjük el.)

A bináris állományok tartalma azonban az ember számára értelmezhetetlen, általában ugyanis mindig valamilyen belső szerkezetet tartalmaznak. A bináris állományok egyik különleges fajtája a végrehajtható bináris fájl.



A végrehajtható bináris programokat az operációs rendszer olvassa be, betölti a memóriába és elindítja őket. Ehhez az szükséges, hogy az adott fájl megfelelő felépítésű legyen.

A végrehajtható binárisok felépítése operációs rendszerenként különböző lehet, de abban megegyeznek, hogy az első pár bájt azonosításra szolgál. Például az MS-DOS esetében az EXE-k mindig „MZ”-vel, Windowsnál „NE”-vel kezdődnek. (Igazából a windowsos programok egy kis DOS-os programmal kezdődnek, ami nem csinál mást, mint kiír egy hibaüzenetet, és kilép. A windowsos program csak ezután következik.) A régi unixos binárisokban (amelyek ***a.out*** néven híresültek el) pedig egy úgynevezett mágikus szám található a fájl elején. Ezekre azért van szükség, hogy egyrészt az operációs rendszer felismerhesse a bináris programokat, másrészt a felhasználót is védi, nehogy véletlenül nem bináris programot akarjon futtatni. Minden operációs rendszernek fel kell tehát ismernie futtatható állományait. Egyes operációs rendszerek azonban más típusú állományok felismerésére is képesek. Például a Windows-rendszerek a fájlok kiterjesztéséből következtetnek a típusára, és ha egy állományra kétszer kattintunk, önműködően elindul a hozzárendelt alkalmazás. (Ám a Windows kutyafüle a TOPS-20 nevű operációs rendszerhez képest. Ha elindítottunk egy alkalmazást, az operációs rendszer ellenőrizte, hogy a bináris létrehozásának időpontja régebbi-e, mint a forrásban történt utolsó változás. Ha igen, akkor újból lefordította az adott programot.)

Felmerül a kérdés, hogy mennyire célszerű, ha egy operációs rendszer ilyen mértékben foglalkozik a fájltípusokkal. A kezdő felhasználók számára biztosan nagy segítség, mivel kisebb az esélye, hogy valami hibát követnek el (például rossz alkalmazással nyitnak meg egy állományt). A tapasztalt felhasználókat azonban idegesítheti, mivel sokkal bonyolultabban tehet meg olyan dolgokat, amelyek eltérnek a „hétköznapi” műveletektől.

Fájlelérés

A fájlszerkezet és a fájltípus mellett fontos fogalom még a fájlelérés. Ez tulajdonképpen azt határozza meg, hogy milyen módon olvashatunk (illetve írhatunk) az állományokból.
Az első operációs rendszerek fájlelérése szekvenciális volt, azaz a bájtokat (vagy rekordokat) kizárólag egymás után olvashattuk ki a fájlból. Lehet, hogy nekünk csak az 5. bájtra volt szükségünk, de akkor is előbb ki kellett olvasnunk az első négyet. A fájl első bájtjára azonban bármikor visszaugorhattunk, tehát újraolvasásra mindig volt lehetőségünk.
Ezekben az ősi rendszerekben azért nem nyílt lehetőség a fájlon belüli pozícionálásra, mert az adatok mágnesszalagon voltak tárolva, amelyek csak lineárisan írhatók és olvashatók. A mágneslemezek megjelenésével viszont ez a helyzet megváltozott, itt már csupán egyetlen meghatározott blokk beolvasására, illetve újraírására volt lehetőség. Ezért az operációs rendszerekben is megjelentek a véletlen elérésű fájlok. A véletlen elérésű állományok esetében létezik egy úgynevezett `SEEK` művelet, ennek segítségével megmondhatjuk, hogy melyik pozíciótól kezdődően szeretnénk olvasni az adott fájl tartalmát.

Tulajdonságok

Azt már tudjuk, hogy minden állománynak van neve és adattartalma. A fájlrendszer azonban mást is tárol az adott állományokról, például a létrehozásának a dátumát, a tulajdonost, a jogosultságait (ez mondja meg, hogy az adott állománnyal ki mit csinálhat); de léteznek olyan rendszerek is, amelyekben jelszót is rendelhetünk a fájlhoz. Ezeket az adatokat nevezzük az állomány tulajdonságainak (attribútum). Minél több tulajdonság adható meg egy fájlnak, annál több lehetőség rejlik a fájlrendszerben. (Például a Unixban a védelmet elég kevés tulajdonság írja le, csupán írási, olvasási és futtatási jogosultság adható. Más rendszerekben, például a Windows NT-ben vagy a VMS-ben ennél finomabb „hangolásra” is lehetőség nyílik.) Viszont az is igaz, hogy minél több a tulajdonság, annál bonyolultabb a fájlkezelés.

Könyvtárszerkezet

Ma már mindenki számára teljesen természetes, hogy állományainkat úgynevezett könyvtárakba szervezhetjük. Ezáltal lehetőségünk nyílik adataink rendszerezésére, így eszményi esetben könnyedén megtalálhatjuk a keresett adatot.
A könyvtárak azonban nem voltak mindig ennyire alapvetőek. A régi CP/M operációs rendszernél csupán egyetlen könyvtár létezett, ami az összes állományt tartalmazta. Ez az elgondolás nem lehetett célravezető egy többfelhasználós rendszer esetében, főleg, ha a felhasználók megegyező nevű állományokat akartak létrehozni. Ezután bevezették, hogy minden felhasználóhoz külön könyvtárat rendeltek. Ám azon felhasználók számára, akik nagyszámú állománnyal rendelkeztek, ez se volt túl célszerű megoldás. Végül létrejött a ma is használt hiearchikus szerkezet, tehát a könyvtárak maguk is tartalmazhatnak további könyvtárakat, amelyek szintén további könyvtárakat rejthetnek magukban és így tovább.
A faszerkezetű könyvtárszerkezetnél megjelenik az útvonal fogalma, azaz innentől kezdve az állományoknak már nemcsak nevük, hanem „helyük” is van a fájlrendszerben. Az útvonal megadására hagyományosan két mód kínálkozik: az első az úgynevezett abszolút útvonal, amikor is a hierarchia legtetején álló (gyökér) könyvtárhoz viszonyítva adjuk meg egy állomány helyét (például */home/user/alma.txt*). A másik az úgynevezett relatív címzés, ahol a munkakönyvtárhoz viszonyítunk, amit a felhasználó jelöl ki.
Sok rendszerben a könyvtárak maguk is állományok, amelyek a bennük található fájlok könyvtári bejegyzéseit tartalmazzák (lásd később).

Megvalósítás: a fájlok

A fájl tehát egy elvont adattárolási módszer. Annak nyilvántartása, hogy egy állomány tartalma pontosan hol is helyezkedik el a merevlemezen, a fájlrendszer feladata. Ennek módja is teljesen rendszerfüggő – a továbbiakban megnézünk pár gyakrabban előforduló megoldást.
Ezek közül a legegyszerűbb a folytonos helyfoglalás, ami a régi nagygépes rendszerekben volt elterjedt (még most akad olyan hely, ahol használják). Itt minden állomány kizárólag egymást követő blokkok sorozatán tárolható. Ha a lemez 1 kilobájtos blokkokba szervezett, akkor például egy 100 kilobájt méretű állományt csak 100 egymás mellett elhelyezkedő blokkon tárolhatunk.
A módszernek két behozhatatlan előnye van a többi módszerrel szemben. Egyrészt hihetetlenül könnyű megvalósítani, hiszen minden állományhoz egy kezdőértéket kell rendelni, ami megmondja, hogy hányadik blokktól kezdődik az adott fájl. Másrészt ez a valaha létezett leghatékonyabb fájlrendszer, mivel az olvasófejet csak egyszer kell pozícionálnunk, utána folyamatosan olvashatjuk az adatokat, egészen a fájl végéig.
Sajnos e két jó tulajdonság mellé két rossz is társul, de ezek az előnyök mellett teljesen eltörpülnek. Ahhoz, hogy ez megvalósítható legyen, minden állománynak létezik egy legnagyobb mérete, aminek átlépésére nincs lehetőség. (Azokban a bizonyos nagygépes rendszerekben ezt a legnagyobb méretet minden állomány létrehozásakor meg kellett adni. A fájl mérete természetesen kevesebb is lehetett, de több soha.) Másik üneprontó tulajdonsága, hogy rendkívül hajlamos a töredezésre. Töredezettség (fragmentáció) alatt azt értjük, amikor a gyakran használt blokkok nem egybefüggő részt alkotnak, hanem össze-vissza találhatók a lemezen, kisebb-nagyobb szabad blokkokkal szétválasztva. Ez azért baj, mert ilyenkor összes szabad blokkot nem használhatjuk fel, magyarul a lemez kihasználtsága romlik. Ennek orvoslásaként a fájlrendszerre kell ereszteni egy úgynevezett töredezettségmentesítő programot, ami úgy próbálja „tologatni” a gyakorta használt blokkokat, hogy közöttük a lehető legkevesebb lyuk legyen. Ez a folyamat rendkívül erőforrásigényes. Ha egy fájlrendszer hajlamos a töredezésre, akkor vagy gyakran kell töredezettségmentesíteni (ami a felhasználóktól veszi el a kapacitást), vagy jelentős szabad területről kell lemondanunk.
A láncolt listás helyfoglalás sokkal jobb megoldásnak tűnik, mivel (ahogy a nevéből is sejthető) a fájl által lefoglalt lemezblokkok láncolt listába szerveződnek. (A láncolt lista olyan tárolási forma, amiben a listában szereplő minden elem „tudja”, hogy ki a következő. Így egy tag eléréshez csak a lista első elemének helyét kell ismernünk, utána a láncon elemenként végigmenve megtalálhatjuk a keresett tagot. Ennek különleges formája a kétszeresen láncolt lista, ahol minden elem az előtte állót is ismeri.) Ezt úgy kell elképzelnünk, hogy minden blokk első pár bájtja azt mondja meg, hogy melyik blokk a következő. A fájlok nyilvántartásához itt is elég a kezdőérték ismerete, ami az első blokk címét tartalmazza.
Az ilyen módon megvalósuló fájlrendszereknél nincs szükség töredezettségmentesítésre, mivel minden üres blokkot fel tudunk használni. Ezért viszont komoly árat kell fizetnünk: az egész iszonyatosan lassú. Ha csupán az utolsó blokkhoz szeretnénk hozzáférni, akkor is be kell olvasni az összeset. Ez rendkívül időigényes feladat, mivel itt a fájl által használt blokkok nem

egymás mellett, hanem a lemezen szétszórva találhatók. Így minden blokk beolvasása előtt szükség van a fej pozícionálására. A másik lassító tényező az, hogy nincs lehetőség az egész blokk területét felhasználni, mivel az első pár bájt egy mutató, ami a következő blokkot jelöli meg. Ennek következtében a blokkban tárolható „értékes" adat mérete nem lesz a kettő valamely hatványa. (A lemezblokkok mérete mindig a kettő valamelyik hatványa. Ezért a programok egyszerre általában mindig kettőhatvánnyi méretű adatokat olvasnak be, illetve írnak ki. Mivel a blokk egész mérete nem használható fel, valószínűsíthető, hogy pár bájt „kimarad", így szükség lesz egy újabb blokk beolvasására, illetve kiírására.)

Szerencsére a láncolt listás megoldás hátrányainak búcsút inthetünk, ha a mutatókat (a következő blokk címét tartalmazó bájtokat) nem a blokkokban, hanem egy táblázatba szedjük össze. Ezt a táblázatot indexnek nevezzük, és a memóriában tároljuk, így annak elérési ideje nem lesz számottevő. Ennek köszönhetően hasznos adat tárolására nemcsak a teljes blokkot használhatjuk, hanem a véletlen elérés is sokkal gyorsabb lesz. Ha ugyanis nem az elejétől kezdve akarunk olvasni az állományból, nem kell az összes, a kívánt kezdőpozíció előtti blokkot beolvasni, hanem az indexből kikeresve egyből megkaphatjuk a megfelelő blokk címét.

Ez a módszer jól működött az asztali számítógépeken, ahol pár megabájtos merevlemezekkel kellett dolgozni. Az MS-DOS FAT nevű fájlrendszere is ezt használta. Idővel azonban megjelentek a többszáz, majd a többezer megabájtos merevlemezek is, és a memóriában tárolt táblázat mérete hirtelen megugrott. Például egy 500 MB-os merevlemeznél is már 2 MB-tal kellett számolni. (Ez az MS-DOS esetében megoldhatatlan lett volna, mivel ő közvetlenül csak 640 kilobájt memóriát tudott kezelni. Ezért úgy oldotta meg, hogy nagy lemezeknél nagy blokkméretet – 32 kilobájt – használt, ami rettenetes pazarlásnak számított, különösen sok kis állomány esetében.)

A fájlok megvalósításának másik megközelítése a fájlleíró (inode) használata, amit a Unix-rendszerek is alkalmaznak. Minden állomány rendelkezik egy ilyen fájlleíróval, amelyben egyrészt megtalálhatók a tulajdonságok, és néhány, a fájl által használt blokk. Ha a fájl elég kicsi (tehát ha kevés blokkot használ), akkor az összes blokk címe megtalálható a fájlleíróban. Ha nem, akkor egy másik blokkban folytatódik a felsorolás. Ezt a blokkot egyszeresen közvetett blokknak nevezzük, a címe szintén a fájlleíróban van. Ha az állomány esetleg nagyon nagy, akkor egy kétszeresen közvetett blokk használatára is lehetőség nyílik (ennek helye szintén a fájlleíróból nyerhető ki). Ez azonban már nem a fájl által használt blokkok, hanem további egyszeresen közvetett blokkok címét tartalmazza (a legtöbb Unix fájlrendszere a háromszorosan közvetett blokkok használatát is megengedi). A következő hónapban a Unix könyvtárainak taglalásával innen folytatjuk, továbbá szó lesz a lemezterület-kezelésről is. Olyan kérdéseket érintünk, hogy például miként tudjuk nyilvántartani üres blokkjainkat, illetve milyen módszerekkel tudjuk növelni a fájlrendszer sebességét.

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.