# Mindaz, amit a memóriáról tudni érdemes

Elérkeztünk következő nagy témakörünkhöz: a memóriakezeléshez.

Vessünk egy pillantást az *1. ábrára*, ami egy jellemző folyamatrendszerű operációs rendszer felépítését mutatja. Eddig a két legalsó réteggel foglalkoztunk, a folyamat- és eszközkezelővel – ezeket az operációs rendszer magjának nevezzük. Ezen a szinten közvetlenül tarthatunk kapcsolatot a számítógép alkatrészeivel, és gyakorlatilag bármit megtehetünk. A továbbiakban az ábrán világoskékkel jelölt elemekről lesz szó, amelyet kiszolgálóknak (server) szokás nevezni, és a felhasználói alkalmazások számára kínálnak bizonyos szolgáltatásokat, olyanokat, amelyeket ők maguk képtelenek elvégezni. A kiszolgálók nem a rendszermag részei! Sokkal kevesebb jogosultsággal bírnak (például közvetlenül nem érhetik el a beviteli–kiviteli kapukat), mint például az eszközkezelők. A három legfontosabb kiszolgáló: a memóriakezelő, a fájlrendszer és a hálózatkiszolgáló. Az utóbbival nem foglalkozunk, mivel részletes ismertetése meghaladná e sorozat kereteit. A fájlrendszerre azonban részletesen vissza fogunk térni, ugyanis ez lesz a következő, egyben utolsó nagy témakörünk.

Mindenekelőtt azonban van még néhány fontosabb összefüggés a kiszolgálók és az operációs rendszer további részei között, amelyekre érdemes rávilágítanunk. Már sokszor kitértünk rá, hogy az operációs rendszerek két legfontosabb feladata az erőforrás-kezelés és a gép szolgáltatásainak kiterjesztése (az úgynevezett rendszerhívások bevezetésével). Láthattuk, hogy az erőforrások kezelésének oroszlánrészét a rendszermag végzi. Vajon mi foglalkozik a rendszerhívásokkal? A választ talán már sejtjük is: a kiszolgálók. Ők azok, akik értelmezik a beérkező rendszerhívásokat és az alsóbb rétegeket utasítják a kért feladat elvégzésére.

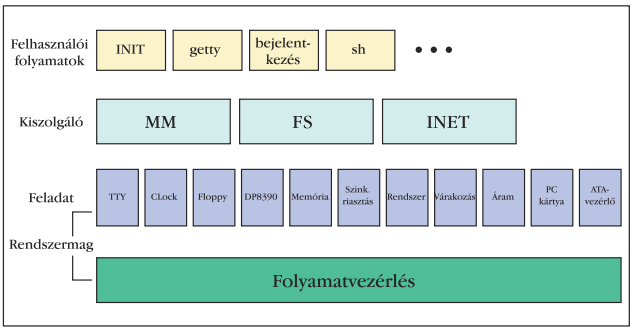
Fontos még megemlítenünk, hogy a kiszolgálók maguk is folyamatok, és alig különböznek valamiben a közönséges felhasználói folyamatoktól. A két lényegbevágó különbség az, hogy egyrészt magasabb szinten futnak, tehát több mindenhez van jogosultságuk (már ha az adott gép processzora támogatja a védelmi szinteket, például ilyen a Pentium). Másrészt a kiszolgálófolyamatok a rendszer indulásától kezdve egészen a leállításig folyamatosan futnak.

Ennek a kiépítésnek az az előnye, hogy a kiszolgálókat akár egy másik, távoli gépen is futtathatjuk. Ennek köszönhetően – apró módosítások árán – a fájlrendszert akár távoli fájlkiszolgálóként is használhatjuk.

## Memóriarendszer

Az operációs rendszerek esetében a memória pazarlása megengedhetetlen fényűzés. Főleg, ha figyelembe vesszük azt az örök igazságot (amelyet gyakran Parkinson törvényeként is emlegetnek), amely szerint a programok mindig kitöltik a rendelkezésre álló memóriát. Könnyen elképzelhető, hogy nemsokára 1 GB memória már egy kiló kenyér árával fog vetekedni, ugyanakkor abban is biztosak lehetünk, hogy a jövő alkalmazásai már nem fogják annyival beérni, mint a maiak.

Világunkra az is jellemző, hogy többféle memóriát forgalmaznak. Akad nagyon gyors, ám rendkívül kicsi, felejtő és drága. Ugyanakkor nagyon olcsó, rettentő nagy kapacitással bíró,




*1. ábra* A folyamatstrukturált operációs rendszer elvi felépítése

tartalmát áramszünet esetén is megtartó, ellenben borzalmasan lassú is található közöttük. Az az egy azonban, amelyre mindenki áhítozik, a határtalan kapacitással és sebességgel bíró, mindenki számára megfizethető és tartalmát örök időkre megjegyző memória egyelőre sajnos még csak a vágyálmok birodalmában létezik.

Ha létezne, bizonyára minden gépbe ezt szerelnék, és nem lenne ennyi gond a memóriakezeléssel. Mivel nem ez a helyzet, a legtöbb számítógépben található memóriák a következő módon szerveződnek rendszerbe: legfelül a kicsi, ám rendkívül gyors és drága gyorstár (cache), alatta a már nem annyira gyors, de nagyobb kapacitású és olcsóbb központi memória (ez a RAM), végül a legalsó rétegben az akár több gigabájtos, olcsó, lassú, de nem felejtő lemezes tároló található. (A központi memóriát gyakran operatív tárnak is nevezik. Ez arra utal, hogy maga a futtatott program is itt foglal helyet.)

A hierarchikus memória kezelése nem egyszerű feladat: nemcsak azt kell megszerveznünk, hogy a szabad memóriarészeket nyilvántartsuk, és hatékonyan osszuk ki őket a futó programok között, hanem figyelni kell arra is, hogy mit melyik memóriában tárolunk. Ha egy folyamat például blokkolt állapotban van, akkor a központi tárból nyugodtan kitehetjük a lemezre, ezzel is felszabadítva némi helyet a többi program számára.

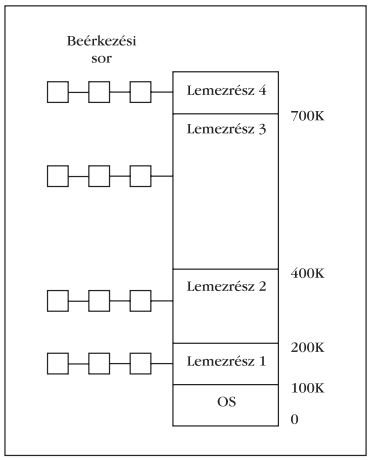
Az operációs rendszernek azt a részét, ami ezeket a feladatokat megvalósítja, memóriakezelőnek nevezzük.

## Memóriakezelés lemez nélkül

A legegyszerűbb memóriakezelők nem használják a lemezt, csak a központi memóriával gazdálkodnak. Ezek közül is a legalapvetőbb az, amikor a memóriában egyszerre csak egy programot tárolunk – ezt a taktikát alkalmazta az MS-DOS is. Az operációs rendszer a lemezről betölti a kívánt alkalmazást a központi tárba (tekintet nélkül arra, hogy mi volt benne előzőleg), majd végrehajtja. Amint a program lefutott, kezdődhet minden elölről. Ennek a módszernek a megvalósítása rendkívül egyszerű, viszont nem teszi lehetővé több program egyidejűleg való futtatását, mivel az egész memóriát egyetlen programnak adományozta.

Egy többfeladatos operációs rendszerben azonban a memóriát egynél több programmal is meg kell osztani. A már legendásnak

számító OS/360-as rendszerben a következő módszert alkalmazták: a memóriát több különböző méretű részekre, úgynevezett lemezrészekre osztották fel, amelyek méretét az operátornak a rendszer indulásakor előre meg kellett határoznia. Amikor a program elindult, a rendszer megkereste magának azt a legkisebb lemezrészt, amelyikbe a program még belefért. Miután megtalálta, berakta a lemezrész várakozási sorába, és várta, hogy felszabaduljon. Miután felszabadult, a program betöltődhetett a kiválasztott memóriarészbe, és elkezdődhetett a végrehajtása. Ennek a módszernek MFT (Multiprogramming with Fix Tasks, lásd *2. ábrán-kon*) volt a neve.




*2. ábra*
Memóriakezelés az OS/360-asban

Az MFT-vel rengeteg gond akadt. Pazarló módszernek bizonyult, hiszen előre megadott méretű lemezrészek voltak, így hiába használtuk ki csak a felét a számunkra kiosztott memóriaszeletnek, akkor is elveszett az egész. Az operátornak ezért mindig mesterkednie kellett, hogy alkalmas méretű lemezrészeket hozzon létre, figyelembe véve olyan szempontokat is, hogy például az adott munkanapon inkább nagyobb vagy kisebb memóriaigényű programokat fognak-e futtatni.
A másik baj az volt, hogy könnyen előfordulhatott olyan eset, amikor egy-egy kisebb lemezrészért több program is sorban állt, míg a nagyobbak senkinek sem kellettek. Ekkor egy olyan módosítást találtak ki, hogy csak egy várakozási sor legyen, és ha egy lemezrész kiürül, akkor a rendszer azt a programot tölti be a megüresedett helyre, amelyik a legjobban ki tudja használni a lemezrészt és amelyik ezek közül legelöl áll a sorban. Ezzel a megoldással az volt a gond, hogy kirekesztő módon viselkedett a nagyon kicsi alkalmazásokkal szemben, mert ha kizárólag a méretüknél sokkal nagyobb lemezrészek ürültek ki, csak akkor töltődhettek be az üres helyre, ha nem várakozott másik, náluk nagyobb alkalmazás egy üres memóriaszeletre.
Az MFT a legkönnyebben megvalósítható olyan memóriakezelő módszer, amelyik azt támogatja, hogy egyszerre több program is a memóriában legyen. Sok évig sikerrel alkalmazták a nagygépes rendszereken – fent említett hátrányai ellenére. Ma már azonban sehol sem alkalmazzák, és egyetlen rendszer sem támogatja.

## Relokáció és védelem

Még mielőtt belemerülnénk a ma használatos memóriakezelési eljárásokba, nem árt, ha előtte megemlítünk két olyan nem elhanyagolható nehézséget, amelyet minden memóriakezelési eljárásnak meg kell tudnia oldani.
Amikor egy programot lefordítunk, az utolsó lépésben, amit összeépítésének vagy szerkesztésnek nevezünk, a főprogramot, az eljárásokat és az osztott könyvtárakat (erről bővebben a következő részben szólunk) egy közös címtérben helyezzük el. A szerkesztést végző alkalmazásnak (linkernek) tudnia kell, hogy a főprogram hol fog kezdődni, és merre helyezkednek el majd azok az eljárások, amelyek a végrehajtás során meghívásra kerülnek.
Nehézség abból adódik, hogy a betöltéskor minden program más címtartományba kerül, és sosem tudhatjuk előre, hogy a mi programunkat melyik memóriarész fogja megkapni. Amikor a program végrehajtása elkezdődik és egy olyan utasítást kap, hogy például hívjuk meg az 50-es címen lévő eljárást, gond van. Az 50-es cím az operációs rendszer területéhez tartozik – a program nyilvánvalóan nem arra a címre kíván ugrani, hanem a saját lemezrészének kezdetéhez viszonyított 50-es címre. A megoldás az lenne, hogy a lemezrész kezdeti címéhez hozzáadjuk a program által megadott címet, így megkapjuk a meghívandó eljárás valódi helyét. Ezt a műveletet nevezzük eltolásnak (relocation).
Az operációs rendszerek különböző módon oldják meg a fenti feladatot, azaz a áthelyezést. Az előbb már említett OS/360-as például a program betöltésekor önműködően módosította az utasításokat, úgy, hogy minden címhez hozzáadta a kiválasztott lemezrész kezdőcímét. Ehhez azonban az kellett, hogy a szerkesztőprogram egy térképet helyezzen el, ami a rendszernek megadja, hogy mely értékeket kell úgymond „relokálni". Ez a módszer még ma is használatos, főleg a mikroszámítógépek világában.
Az áthelyezéssel szorosan összefügg a védelem kérdése. Az imént bemutatott módszerben a programok abszolút címekkel dolgoznak, ezáltal némi ügyeskedés segítségével könnyen írhatunk olyan programot, amelyik képes belepiszkálni más folyamatok memóriájába, vagy akár az operációs rendszer számára fenntartott területre is.
A védelem megvalósításának módja szorosan összefügg az áthelyezéseknél alkalmazott módszerrel, sőt az is fontos, hogy az adott rendszer milyen gépen fut. A legtöbb eszköz ugyanis segít e két dolog megvalósításában az operációs rendszernek. Maradjunk a szokásos példánál az OS/360-assal. Itt a memóriát 2 KB-os blokkokra osztották, és minden blokkhoz hozzárendeltek egy négybites kulcsot, amelyeket csak az operációs rendszernek áll módjában megváltoztatni. Amikor egy program hozzá akart férni egy memóriablokkhoz, a programállapotszó-regiszterbe (PSW) be kellett tölteni az adott blokkhoz tartozó kulcsot. Az ellenőrzés eszközszinten történt, és ha a kód nem egyezett, a művelet nem hajtódott végre.
Sokkal elterjedtebb volt egy másik módszer, amelyik egyszerre kínált megoldást az áthelyezésés a védelem kérdésére. Az eszköz két egyedi regiszterrel rendelkezett: a bázis- és a háttérregiszterrel. Amikor egy folyamat megkapta a futási lehetőséget, a bázisregiszterbe betöltődött a program memóriarészének kezdőcíme, a háttérregiszterbe pedig annak mérete. Innentől kezdve az eszköz minden memóriacímet a bázisregiszterhez viszonyított, így az operációs rendszernek többé nem kellett a kód módosításával foglalkoznia. A háttérregiszter jelenléte a védelem kérdését is megoldotta, mivel ha a program egy olyan címre akart hivatkozni, amelyik saját memóriarészén kívül volt (azaz a háttérregiszternél egy nagyobb értékre hivatkozott), a rendszer azonnal gyilkolt.
Ezt a módszert sok helyen alkalmazták, még az első IBM PC-kben található Intel 8088-as processzorok is ezen alapultak (habár itt mellőzték a háttérregisztert, így a védelmi részét gyakorlatilag kiiktatták). A 286-os processzoroktól kezdve azonban egy teljesen más módszer jelent meg, amit védett módnak (protected mode) nevezünk. De minderről majd a következő részben szólunk.

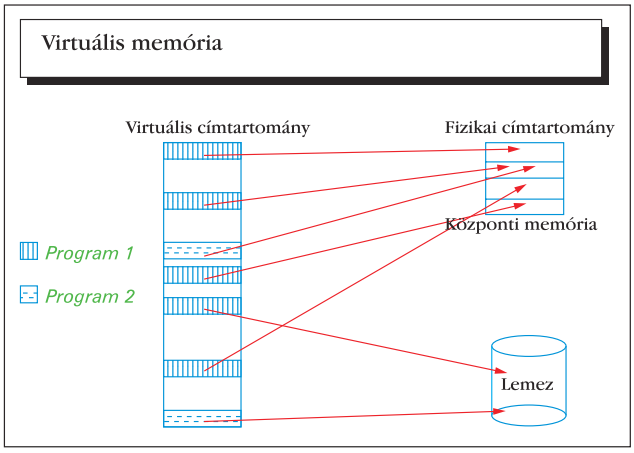
## Csereterület és virtuális memória

Az emberiségnek hamar szembesülnie kellett azzal a ténnyel, hogy a memória sosem elég, és az összes futó program egy-

szerre általában nem fér el a memóriában. Ezért a központi tárat ki kell egészíteni a lemezzel. A lemezes tárolók a memóriához képest ugyan lassúak, viszont sokkal nagyobb kapacitással rendelkeznek, tehát nyugodt szívvel odapakolhatunk minden olyan dolgot, amire az adott pillanatban nincsen szükség, például a blokkolt folyamatok memóriarészeit.
A lemez és az operatív tár közötti ide-odapakolás megvalósítására kétféle megközelítés is létezik. Az első módszer a csere. Ilyenkor a programokat csak teljes egészükben mozgathatjuk a lemez és a központi memória között.
A csere azonban nem jelent megoldást arra a helyzetre, ha a futtatni kívánt program meghaladja a rendelkezésre álló szabad memória méretét. Ilyenkor hajdanán az volt a bevett szokás, hogy a programokat rétegekre darabolták, és ezeket a rétegeket megszámozták. Először a 0. réteg töltődött be a memóriába és kezdett el futni. Amikor a végrehajtás befejeződött, betöltötték a következő részt, és folytatódott tovább a program végrehajtása. Ezt overlay módszernek nevezik. Csak egy gond volt vele: programjának részekre való bontását a programozónak saját magának kellett megoldania. (Szerencsére a rétegek cseréjét az operációs rendszer maga is el tudta végezni). Ez nem volt igazán szórakoztató elfoglaltság, ezért egyre inkább felmerült az igény egy olyasfajta megoldás megalkotására, amely a programozót megkíméli az effajta lélekölő tevékenységektől.
Ekkor jött a nagy ötlet: a virtuális memória. Lényege az, hogy a programnak mindig csak azok a részei tartózkodnak a memóriában, amelyekre éppen szükség van – a többit a lemezen tároljuk. A virtuális memória másik vonzó tulajdonsága, hogy több program együttes futtatásánál is alkalmazhatjuk. Ebben az esetben több program darabjai tartózkodnak a memóriában.
Háromféle megvalósítását ismerjük a virtuális memóriának: a lapozást, a szegmentálást és a két módszer egyszerre való alkalmazását.

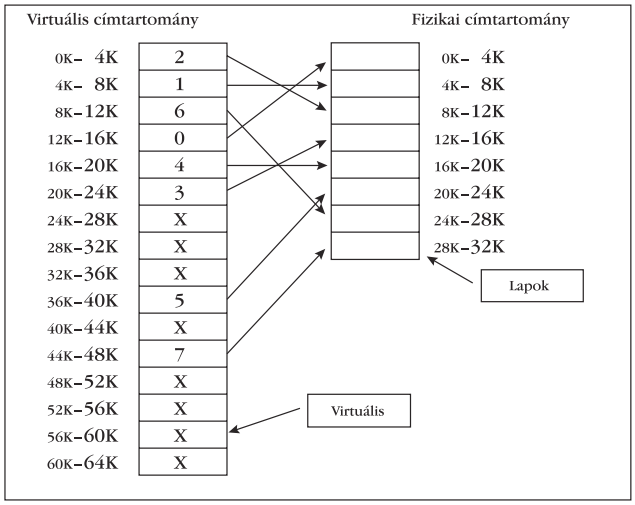
## A lapozás

A virtuális memóriának az a kiinduló gondolata, hogy kettéválasztjuk a címtartomány és a memóriarekesz fogalmát. Hogy ez mit is jelent? Vegyünk példának egy olyan számítógépet, amely mondjuk 4 KB (4096 bájt) fizikai memóriával rendelkezik. A gép regiszterei 16 bitesek, így pontosan 65 536 bájt megcímzésére nyílik lehetőségünk. Az, hogy egy számítógép mekkora memóriát tud kezelni, kizárólag címregiszterei hosszától függ. Azoknak az értékeknek a halmazát, amelyet a címregiszterek felvehetnek, a számítógép címtartományának nevezzük. Jelen esetben ez a 0, 1, ... , 65 535. Ez azt jelenti, hogy legfeljebb 64 KB memóriát tudunk címezni (tehát hiába van a gépben mondjuk 128 KB memória, akkor is csak 64-et lehet belőle használni).
Amikor nem létezett még a virtuális memória, a memóriacímeket két részre osztották: a hasznosakra és a haszontalanokra. Az utóbbi csoportba nyilván azok a címek sorolódtak, amelyekhez már nem tartozott valódi memóriarekesz, azaz jelen példánknál maradva nagyobbak voltak, mint 4095.
Itt az ideje kettéválasztani a címtartomány és a memóriarekesz fogalmát. Számítógépünk egyszerre csak 4096 bájtot tud a memóriájában tartani, de mondhatjuk azt, hogy ezeknek a címei ne feltétlenül a 0 és 4095 közé essenek. Mondhatjuk azt is, hogy a fizikai memória első rekeszéhez rendelt cím legyen inkább a 4096, a második rekeszéhez a 4097, és így tovább. Ha úgy tetszik, egy leképezést (függvényt) határoztunk a címtartományból (0 ... 65 535) a valódi memóriacímekre (0 ... 4095).
Mi ebben a tudomány? Ha a gépben nincs virtuális memória, akkor úgymond állandó leképezésünk van a 0 és 4095 közötti címekre. Ha valamelyik program e tartományon kívül címezne, bizony csúnya véget ér a történet (például egy hibaüzenet kíséretében megszakad a futása). Ha viszont a virtuális memória működik, és tegyük fel, a 8192 és 12 287 közötti címet szeretnénk elérni, akkor a következő történik: először is mindaz kiíródik a lemezre, ami eddig a memóriában volt. Ezután meg kell keresni a 8192 és 12 287 közötti tartományt a lemezen, majd be kell tölteni a memóriába. Legvégül egy új leképezést kell létrehozni a virtuális és a fizikai memóriacímek között (*3. ábra*).

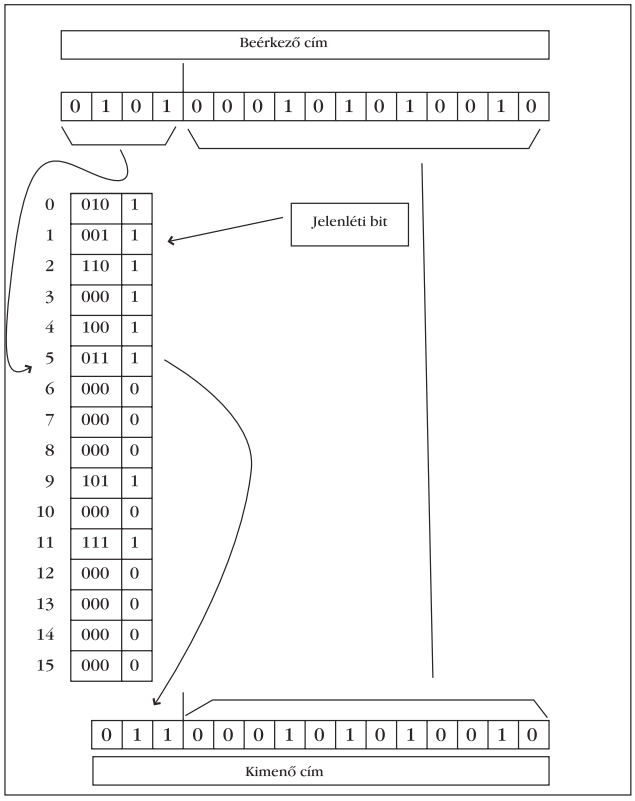



*3. ábra* A virtuális és a fizikai címtartomány kapcsolata

Ezt az eljárást lapozásnak nevezzük, a lemezről beolvasott memóriadarabkákat pedig lapoknak. A fent említett leképezés a virtuális és a fizikai címek között tulajdonképpen egy táblázat, amelyet laptáblának vagy memóriatérképnek hívunk.
(A fenti példában feltételeztük, hogy van elegendő hely a lemezen ahhoz, hogy az egész címtartományt lefedhessük. A való életben ez nincs mindig így, de az elv akkor is ugyanaz marad).
A virtuális és a fizikai címtartományt mindig azonos méretű blokkokra bontjuk (ezek lesznek a lapok), a méret mindig a kettő valamelyik hatványa. Manapság az általános lapméret 512 bájt és 64 KB közé esik, de ez mindig az adott feladattól függ. Így nem ritka a több megabájtos lapméret sem. A fizikai címtartomány egy-egy blokkjába fogjuk betölteni a lapokat, és ezeket lapkereteknek nevezzük.
Fel szeretnénk hívni a figyelmet arra, hogy sem a programozó, de még maga a futó program sem tud semmit minderről.
A programozónak úgy kell megírnia a kódját, mintha nem is lenne lapozás. Ő a memóriát irtózatosan nagynak, folytonosnak és lineárisnak látja. Például nem kell olyan dolgokkal foglalkoznia, hogy az adott memóriaszelet most a központi tárban vagy a lemezen van-e. A virtuális memória másik megvalósítása esetén a következő részben tárgyalt szegmentálásnál már egy kicsit más a helyzet, ott már nem árt, ha a programozó is tud a szegmensek létezéséről.
Mielőtt egy példán részletesen bemutatnánk a lapozás működését, a könnyebb megértés végett sokat segítene, ha a lemezt úgy képzelnénk el, mint azt a dolgot, ami az adott program memóriájának az eredeti változatát tartalmazza, és ami a fizikai memóriában helyezkedik el, az annak a másolata. Természetesen, ha a memóriában megváltozik a tartalom, a változásnak a lemezen is végbe kell mennie.
Lássuk, miként fest ez a gyakorlatban! Tegyük fel, hogy van 32 kilobájtos fizikai memóriánk és ehhez 64 kilobájt virtuális memória tartozik. A lapméret legyen 4 KB, ami azt jelenti, hogy a fizikai memóriában egyszerre nyolc lapot tárolhatunk (más-




*4. ábra* A laptábla




*5. ábra* Az MMU címátalakításának folyamata. Jelen gép processzora 15 bites címekkel dolgozik, de a virtuális címtartomány mérete 16 bit. Az MMU feladata az, hogy a 16 bites virtuális címből 15 bites valódi címet készítsen

ként fogalmazva: nyolc lapkeretet tartalmaz). Mivel a virtuális memória 64 kilobájtos, ezért ott kétszer annyi, azaz 16 lap raktározása lehetséges. Ne feledjük, hogy lemezeinken mindig az összes lap tárolódik, még azok is, amelyek éppen a memóriába is be vannak töltve. Tehát a példánkként szolgáló számítógép összes memóriája együttesen 64 KB és nem 32 + 64 KB! A 64 kilobájtos memóriánkat tehát felosztottuk egyenlő részekre, jelen esetben 4 kilobájtos lapokra, így összesen 16 lappal gazdálkodhatunk. A fizikai memóriába ezek közül be van töltve valamelyik nyolc. Hogy melyik nyolc, azt a laptáblából olvashatjuk ki. Ez tulajdonképpen egy táblázat, ami pontosan annyi bejegyzést tartalmaz, ahány lapunk van. Minden bejegyzéshez egy úgynevezett jelenléti bit tartozik, ami azt mondja meg, hogy az adott lap éppen be van-e töltve a fizikai memóriába vagy sem. Ha be van töltve, akkor azt is kiolvashatjuk a táblázatból, hogy melyik lapkeretében találhatjuk (*4. ábra*).

Nézzük meg, mi történik, ha a program végre szeretne hajtani egy `MOV REG,20818` gépszintű utasítást (firmware). Ez az utasítás jelentse azt, hogy valamelyik regiszterbe be szeretnénk tenni az 5000-es címen található értéket (és tegyük fel, hogy a címregiszter 16 bit hosszúságú). Természetesen ez egy virtuális cím, amivel a processzor nem igazán tud mit kezdeni, ezért majd át kell alakítani fizikai címmé.

Erről az úgynevezett MMU (Memory Management Unit, azaz a memóriakezelő egység) gondoskodik. Ez egy olyan eszköz, ami általában a processzorlapkán foglal helyet, de van olyan változata is, amelyben külön van választva a processzortól. Akárhogy is, a feladata ugyanaz.

Az MMU tehát megkapja az 20 818-os virtuális címet, és egyből egy 4 bites, úgynevezett virtuális lapszámra és egy 12 bites offsetre bontja. Az, hogy a címből hány bit a lapszám, és mennyi az offset, csakis attól függ, mekkora méretű lapokkal dolgozunk. (Mivel mi 4 KB-osakkal munkálkodtuk, ezért 12 bit az offset, mert $2^{12}$ az pontosan 4096). Ezzel meg is állapítottuk, hogy az 5. virtuális lapon van a keresett adat (a számozást a nullával kezdjük).

A következő feladat, hogy megkeressük a laptáblában az 5. laphoz tartozó bejegyzést, és megnézzük, hogy a memóriában található-e. Ha a jelenléti bit egyre van állítva, akkor igen, és kiolvassuk, hogy melyik lapkeretben lelhető fel – mondjuk az ötödikben. Ekkor már gyerekjáték megadni a fizikai címet, csupán ki kell számolnunk, hogy hol kezdődik az ötödik lapkeret, és hozzá kell adni az offsetet. Az így kapott értéket a processzor már fel tudja dolgozni.

Igen ám, de mi történik, ha az adott lap mégsincs a fizikai memóriában (tehát a jelenléti bit nulla). Ebben az esetben laphibáról beszélünk. Ilyenkor az operációs rendszernek meg kell keresnie a merevlemezen a kért lapot, majd be kell töltenie valamelyik lapkeretbe. Ha ez megtörtént, az egész művelet kezdődhet elölről.

Nem mindegy, hogy melyik lapkeretbe töltjük be az új lapot. Azt az elvet, ami alapján az operációs rendszer eldönti, hogy melyik lap helyére tegye be az újat, lapcserélési algoritmusnak nevezzük.

Vegyük észre, hogy a feladat oroszlánrészét az eszközök végzik el, az operációs rendszernek csak akkor kell intézkednie, ha a kívánt lap nem tartózkodik a fizikai memóriában. A mai korszerű processzorok szinte kivétel nélkül támogatnak valamely virtuális memóriát kezelő módszert. Pentium processzorokban például ez a képesség nagyon fejlett, a lapozás mellett a szegmentálást és a kettő együttes kombinálást is támogatják. Az UltraSparc processzorok esetében is a virtuális memória lapszervezésű. Mindezek ellenére egyre jobban kezd elterjedni a lapkezelés programon keresztüli megoldása. A mai RISC-processzorokat támogató gépek, például az Alpha esetében ez már így működik. Hogy miért ezt a módszert használják, arról a következő részben fogunk szólni.

***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.