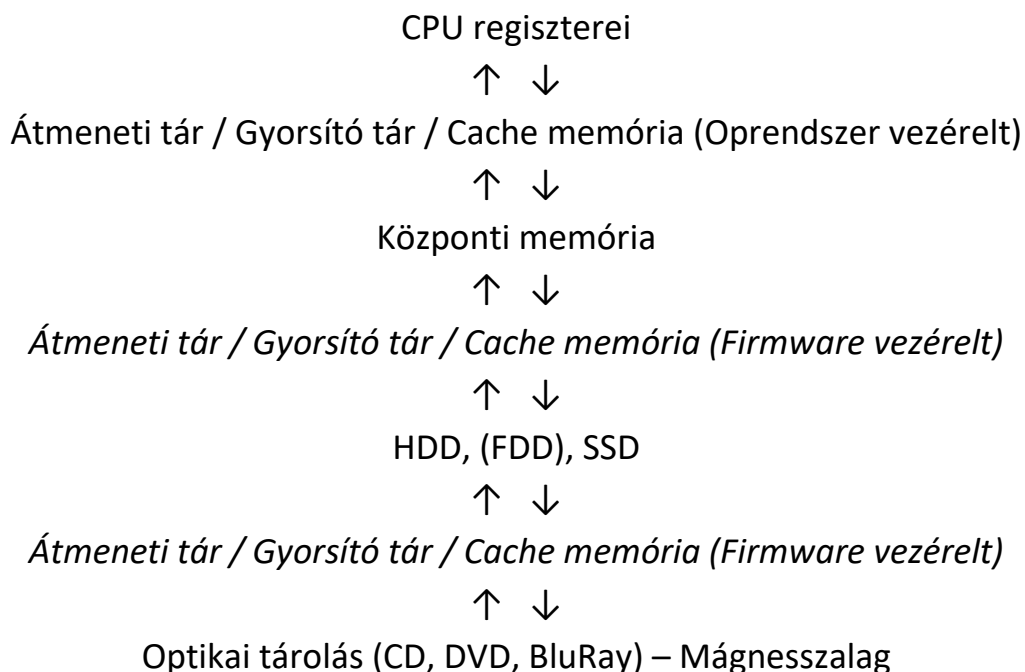


## Előadás\_#07.

### 1. Tárkezelés

A tárolóeszközök több szempont szerint is csoportosíthatóak (sebesség, kapacitás, mozgó alkatrészek aránya, stb.), mégis a legfontosabb csoportosítási szempont az, hogy az adott tároló a számítógép kikapcsolása után is megőrzi-e tartalmát, vagy a számítógép kikapcsolásával elfelejti azt. Így megkülönböztetünk „nem felejtő”, azaz permanens (Persistent, Non-Volatile), illetve „felejtő”, azaz nem permanens (Non-Persistent, Volatile) tárolókat.

A tárolóeszközök hierarchiája a tárolókapacitás illetve a sebesség viszonylatában alkot egységes rendszert. A két előbb említett paraméter jellemzően fordított arányosság szerint viszonyul egymáshoz. Az egységnyi adat tárolásának költsége is egyre csökken a tárterület méretének növekedésével. A következő felsorolás az első sorában a leggyorsabb, de legkisebb kapacitású tárolók, míg az utolsó sorában a leglassabb elérésű, de legnagyobb kapacitású tárolók családjai találhatóak.



Más megközelítésben: A központi tár után következnek a belső permanens tárolók, a külső permanens tárolók, illetve a biztonsági másolatok különböző megvalósításai. Léteznek ezen kívül nem „oprendszer közeli” tárolási megoldások: hordozható eszközök (Pen Drive, mobil HDD), internet, felhő alapú tárolás. (Bár pont a Windows Vista és az azt követő rendszerek képesek bizonyos típusú pendrive-okat rendszergyorsításra használni, de ez inkább lakossági (azaz kommersz) megoldás, mint vállalati (azaz professzionális), tekintettel a megoldás hibátűrésére.

Az egyes szintek között (az adott szintek kiszolgálásához megfelelő) gyorsító tárákat, cache memóriákat is találhatunk. [A „cache” szó angol/francia eredetű, jelentése rejtkehely. A hangzás összecseng a „cash” – készpénz szóval, támogatva azt a (jelen esetben téves) logikát, hogy a készpénz gyorsan rendelkezésre áll...]

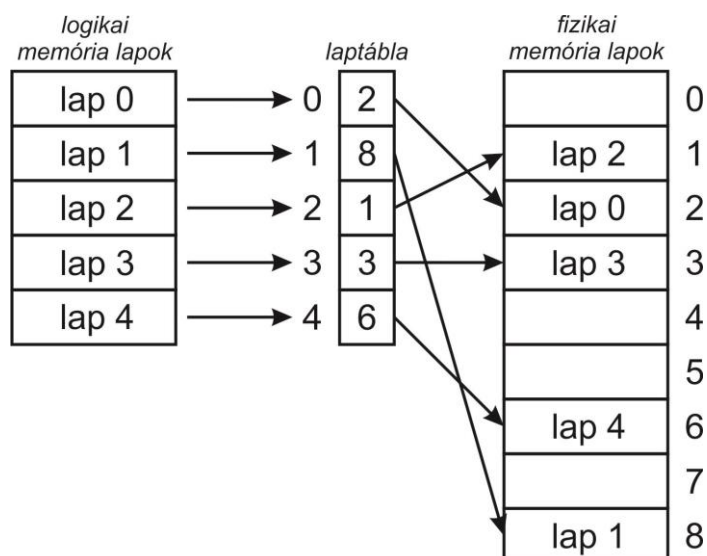
Az operációs rendszer, illetve az egyes intelligens eszközök firmware-e ezen átmeneti táruk segítségével olyan módon képes az adatátvitelt gyorsítani, hogy megkísérli egy megfelelő algoritmus segítségével azokat az adatokat előre áttölteni a gyorsító tárba, melyekre nagy valószínűséggel a közeljövőben szükség lesz. Egy adott címen elhelyezkedő adat esetében például az adott cím környezetében található adatok is betöltésre kerülnek (hátha jók lesznek valamire...). A gyorsítás tehát azon alapul, hogy a gyorsító tár gyorsabb, mint az egyik irányból hozzá kapcsolódó, gyorsítandó tár. Így ha a gyorsítandó (rész)tartalom korábban már bekerült a gyorsító tárba (valamely cache vezérlő algoritmus szerint), az ilyen adatokat már nem a lassú működésű területről, hanem a gyors cache tárolóból lehet előhívni.

Például egy CPU esetében a keresés a leggyorsabb helyen, azaz az L1 cache-ben indul, az L2-ben majd az L3-ban folytatódik. Legrosszabb esetben az adat a RAM-ban lesz megtalálható, illetve legeslegrosszabb esetben háttértárból kell előhívni.

Az operációs rendszer feladata a hatékony memória gazdálkodás, az operatív memória hatékony kezelése illetve kiosztása a folyamatok részére. A multiprogramozott rendszerek esetében, ahol egyszerre több program is a memóriában tartózkodik a hatékony memóriakezelés kiemelt fontosságú feladat.

A memória kezelés legfontosabb elvei:

- A programok fizikai tárigényének csökkentése
  - A fizikai és a logikai címtartományok összerendelése.



- A programok bizonyos részei, eljárásai csak akkor töltődnek be a memóriába, amikor azokra szükség van. (pl. hibakezelő rutin)
- A több program által használt eljárások elég, ha csak egy példányban, egy fizikai címtartományban tárolódnak. Így a programok különböző logikai címtartományai ugyanarra a fizikai címtartományra hivatkoznak. Ezen elv gyakorlati megoldásai például az Overlay (Egymást átlapoló programrészek) és a DLL (Dynamically Linked Library / Dinamikusan kapcsolódó [kölcson]könyvtár).

Az overlay technika biztosítja az olyan programok használhatóságát is, melyek egyszerre nem férnének el a számítógép belső memóriájába. Az overlay átlapolást jelent. A gyakorlatban az overlay területre a programnak mindig csak az éppen szükséges (futás alatt álló) része töltődik be, felülírva az esetleges előzőleg bent lévő adatokat.

A DLL technika megvalósulásának alapját a \*.dll rendszerfájlok képezik, melyek dinamikusan, azaz csak akkor töltődnek be az operatív memóriába, ha azokat a program meghívja, vagyis ilyenkor dinamikusan szerkesztődik össze a hívó program és a DLL-ből meghívott eljárás. Az eredeti angol „library” azaz könyvtár kifejezés arra utal, hogy a fájl adatokat tárol, amelyeket a programok szükség esetén „kikölcsönöznek”. (Kissé zavaró lehet, hogy a könyvtár az informatikában más megközelítésben mást [is] jelent...)

- A memória hézagmentes kitöltése
  - Egy számítógép működése közben programok indulnak, futnak majd befejeződnek, azaz végül az adott program által használt memóriaterület felszabadul. Különböző méretű programok lefutása után a legkülönbözőbb méretű, felaprózódott szabad helyek keletkeznek a memóriában. Célszerű tehát a darabokból álló szabad helyek összevonása egybefüggő szabad helyé. Elvileg és gyakorlatilag egy egybefüggő logikai címtartomány nem igényel egybefüggő fizikai címtartományt (mesterséges folytonosság).
- A háttértárak dinamikus használata a memória kiterjesztésére
  - Amennyiben egy program futása valamely okból jelentős ideig várakozni kényszerül (pl. egy másik program eredményére vár), akkor célszerű az operatív tárból a háttértárba áthelyezni. Így az operatív tárból felszabadult helyen más programokat futtathatunk.

A társzervezés módjai és megvalósulásai esetében fontos megjegyezni, hogy a társzervezési algoritmusok szükség képpen mindig veszteségesek, de a veszteség mértéke nem mindegy, hogy mekkora. Cél a minél kisebb veszteség elérése. A címtranszformáció a logikai és a fizikai címek kapcsolatának megvalósítását jelenti.

- Egypartíciós rendszer

Ez esetben az operációs rendszer által használt tárterületen túl, egy darab egybefüggő címtartomány létezik. Az operációs rendszer védelmét egy határregiszter biztosítja, ami a program(ok) által használható legkisebb címet tartalmazza. Az előző mondatban a zárójel azért szerepel, mert ez a megoldási mód jellemzően egyetlen program futása esetén volt használatos.

- Többpartíciós rendszer

A multiprogramozás igényeinek kiszolgálására használatos rendszer, mely több program egyidejű operatív tárban történő tárolását, futtatását teszi lehetővé, minden program számára külön memória partíciót biztosítva. A megfelelő védelemhez partíciónként kettő (egy alsó és egy felső) határregiszterre van szükség.

Ez nagyon szépen hangzik, de fix méretű partíciók esetén nagyon rossz a kihasználtság hatékonysága. Ha a programok nem használják ki a rendelkezésükre álló memóriaterületet, akkor az belső tördelődéshez (Internal Fragmentation) vezet. Azaz kihasználatlan memória területek lesznek az adott partíción belül.

Dinamikusan kiosztott, azaz a programok igényeihez méretezett partíciók esetében viszont külső tördelődés (External Fragmentation) állhat elő. Azaz az adott partíciók között kihasználatlan memória területek keletkezhetnek, melyek összes területe jelentős lehet.

A külső tördelődés okozta veszteségek minimalizálására több algoritmus is rendelkezésünkre áll.

- Legjobban illeszkedő (Best Fit)

Ebben az esetben a foglalás a legkisebb, de még elégséges méretű területből fog megtörténni, így a maradék, azaz a fel nem használt, kihasználatlan terület a lehető legkisebb lesz.

- Első megfelelő (First Fit)

Ebben az esetben a hely keresése a tár elejéről indul, és az első megfelelő méretű terület kiválasztása a cél, függetlenül attól, hogy az mennyivel nagyobb a lefoglalandó területnél. A megoldás jellemzően inkább gyors, mint terület hatékony.

- Következő megfelelő (Next Fit)

Ez az előző algoritmus egy módosított változata, melyben a keresés nem a tár elejétől indul, hanem közvetlenül az utoljára lefoglalt tartomány után, így szintén főleg a sebességre nézve hatékony.

- Legkevésbé illeszkedő (Worst Fit)

Ebben az esetben a foglalás a legnagyobb rendelkezésre álló szabad területből fog megtörténni. A választás logikája az, hogy a rendelkezésre álló legnagyobb szabad területből a választás (azaz a foglalás) után még kellően nagy, azaz másik folyamatok számára is választható méretű terület marad.

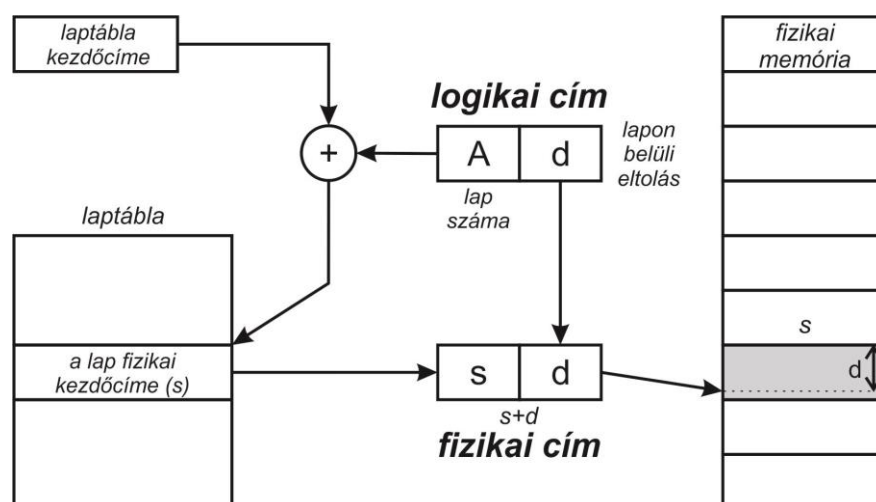
Amennyiben a dinamikus kiosztás folyamatos átméretezéssel, dinamikus áthelyezhetőséggel is társul, akkor a tördelődések szinte teljesen elkerülhetők, de ne felejtjük el az ehhez szükséges erőforrásigényt, és a ráfordítandó időt. A gyakorlatban hatékonyabb működést eredményez az, ha beletörődünk némi memória veszteségbe (és kellően nagyméretű memóriánk van ennek elviselésére).

- Tárcsere (Swapping)

A tárcsere esetében az operatív tárat időben osztjuk meg a folyamatok között. Azaz egy program vagy az operatív tárból vagy valamilyen háttértárból tartózkodik. Erre jellemzően akkor van (klasszikus esetben volt...) szükség, amikor egyszerre csak egy futó program fér el az operatív tárból.

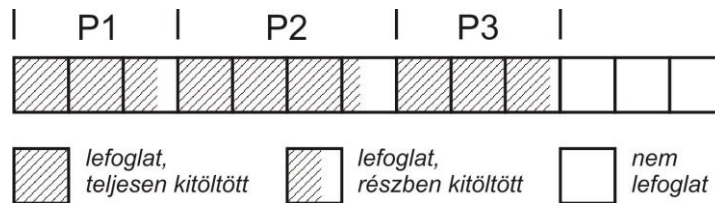
- Lapszervezés

A logikai és a fizikai címtartomány egyaránt azonos, rögzített méretű lapokra van osztva. Jellemző a kettő egész hatványai (azaz az informatikai szempontból kerek számok) szerinti kiosztás. Fix méretek esetében (pl. kész szoftverek) célszerű a használata.



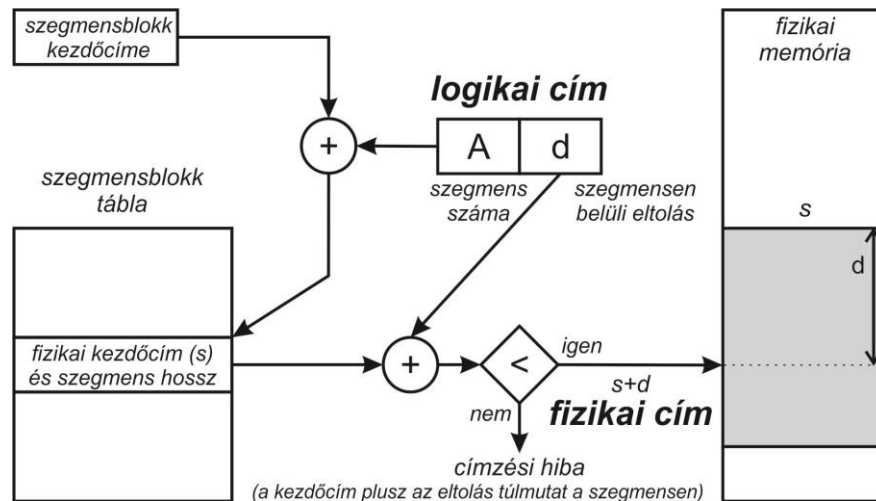
A logikai címtartományban egy címet, azaz egy tároló rekeszt a logikai lapcím és egy eltolás érték jelenít meg. Szükség van ezen kívül egy laptáblára is, mely a logikai lapcímek sorrendjében a hozzájuk tartozó fizikai lapok kezdőcímeit tartalmazza. Mivel a lapméret adott, így eleve nem definiálhatunk azon túlmutató eltolási értéket. Ez esetben a címtranszformációhoz csupán egy összeadás szükséges.

A lapszervezés kísérőjelensége a belső tördelődés, hiszen az egy folyamathoz tartozó fix méretű lapok közül a legutolsó nagyon ritkán van pont tele. A lapokon belül a kihasználatlan részek más folyamat részére nem oszthatók ki.



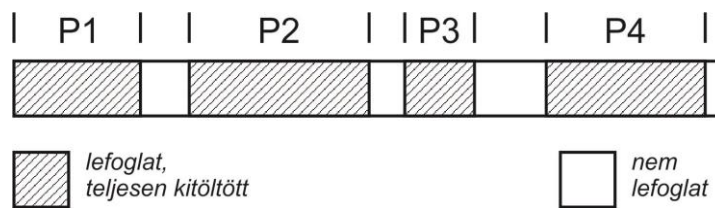
- Szegmensszervezés

A logikai és a fizikai címtartomány különböző méretű lapokra van osztva. Előnye, hogy a lapok az adott programok igényeinek, logikai szerkezetének megfelelően alakíthatóak ki. Változó méretek esetében (pl. adatállományok) célszerű a használata.



A logikai cím ez esetben is két komponensből áll. A tábla kezdőcímét a folyamat környezetét leíró adatszerkezetben tárolja a rendszer. A változó szegmensméret miatt a szegmensen belüli eltolás maximális hossza is szegmensenként eltérő. Ez esetben a címtranszformáció kicsit bonyolultabb, elvégzéséhez két összeadás, és egy méretellenőrzés szükséges.

A szegmensszervezés kísérőjelensége a külső tördelődés, hiszen a változó méretű szegmensek nem feltétlenül képesek a memóriát teljesen kitölteni.



- Kombinált szervezés

A szegmensszervezés és a lapszervezés együttes használata, mely a ma használatos rendszerekre jellemző.

Kombinált szervezés – illetve szegmensszervezés és lapszervezés esetén is – minden egyes folyamat logikai címtartománya 0-tól (a saját nullájától!) indul, de az azonos logikai címekhez nyilván más-más fizikai címek tartoznak, a saját táblák szerint.

## 2. Virtuális memóriakezelés (VM)

A virtuális memória egy, az operációs rendszer és/vagy a számítógép hardvere által nyújtott szolgáltatás, amit jellemzően egy külső tárolóterület igénybevételevel, a futó program(ok) számára transzpárens módon biztosítja, hogy a program végrehajtáskor a központi vagy operatív memória fizikai korlátai észrevétlenek legyenek. Ez gyakorlatilag azt jelenti, hogy a folyamatok által felhasználható logikai címtartományoknak csak a háttértárak kapacitása szab határt.

Az operációs rendszer úgy szabadít fel operatív memóriát az éppen futó program számára, hogy a memóriában tárolt, de éppen nem használt lapokat kiírja a külső tárolóra, amikor pedig ismét szükség van rájuk, visszaolvassa őket. Mivel a merevlemez sebessége töredéke a memória sebességének, nagyon sok múlik azon, hogy a virtuálismemória-kezelő milyen stratégiát alkalmaz az operatív memóriából kimozgatandó lapok kiválasztásakor.

Az alapvető cél, hogy „minden” az operatív tárban legyen objektíve elérhető, főleg több folyamat esetén. Valójában elég, ha csak az éppen futó folyamat futásához éppen szükséges adatok vannak az operatív tárban, de emellett minden más folyamat úgy hivatkozhat logikai memória címekre, mintha azok fizikai címek volnának. A tárolt információk (folyamatos) mozgatása az operatív tár és a háttértár között valósíthatja meg ezt a modellt.

A fizikai címekhez kötött logikai címek használata alapvető a modern operációs rendszerek működése szempontjából.

Elvárások:

- a programok gyorsabban töltődjenek be, mert induláskor csak a szükséges kódrészletek töltődnek az operatív tárba
- több folyamatot futtathatunk egy időben, ha csak az van a fizikai memóriában, ami tényleg szükséges
- tudjunk olyan folyamatokat is futtatni (egyszerre akár többet is), melyek több memóriát igényelnek, mint a rendelkezésre álló fizikai memória
- a folyamatok legyenek képesek osztozni közös erőforrásokon, adatokon és adatszerkezeteken, illetve kódokon
- a folyamatok megoszthatnak memóriaterületeket egyedi hozzáférésekkel (írás, olvasás), ezekről nyilvántartást kell készíteni

A laphiba, mint fogalom azt jelenti, hogy olyan memórialapra hivatkozunk, amely nem az operatív tárban, hanem valamelyik háttértáron van a hivatkozás pillanatában. A laphiba mentes időszakok hossza a memória méretétől, illetve a futtatni kívánt programok számától és méretétől függ.

Probléma akkor keletkezik, ha a fenti feltétel nem teljesül. Az operatív memóriában nem található lapra történő hivatkozáskor következik be a laphiba. A laptábla bejegyzés azt jelzi, hogy az elérni kívánt lap nincs a fizikai memóriában, illetve a logikai cím nagyobb egy előre megadott korlátnál, vagy a folyamat olyan memóriacímet próbált elérni, amihez nem tartozott memória, vagy tiltott módon akar hozzáférni a memóriához (pl. csak olvasható részt írni akar), akkor a program a futását megszakítja.

A laphibát megszakításként kell értelmeznie az operációs rendszernek, és kezelnie kell. Például: Ha az adott lap nincs a memóriában, de a HDD-n megtalálható, akkor azt onnan be kell olvasni, majd az operációs rendszernek vissza kell adnia a vezérlést a megszakított folyamatnak.

Amennyiben a fizikai memória az előbbi esemény bekövetkeztekor éppen teljesen tele van, akkor legelőször is helyet kell felszabadítani. [lásd: Lapcsere algoritmusok] Eközben a folyamat természetesen várakozik.

Lapozási stratégiák:

- Igény szerinti lapozás:
  - Csak laphiba esetén, és csak a laphibát megszüntető lapot hozza be a fizikai memóriába.
  - Csak a szükséges lapok vannak/legyenek a fizikai memóriában.
  - Az új lapra vonatkozó hivatkozás jellemzően hosszú várakozást eredményez.



- Előretekintő lapozás:
  - Előre tekintve (becslés) az operációs rendszer megpróbálja „kitalálni”, hogy mely lapokra lesz szükség, és azokat is behozza. (Azt, hogy a fizikai tár mekkora része használható fel erre a célra, korlátozni kell.)
  - Feltétel a megfelelő mennyiségű szabad erőforrás (CPU, HDD, fizikai memória).
  - Ha a behozott lapokra tényleg szükség lesz (sikeres a becslés), akkor csökken a laphibák száma.
  - Ha nem, akkor feleslegesen használunk erőforrásokat, és lassítottuk a rendszert.

#### Lapcsere algoritmusok:

Az alaphelyzet az, hogy tele van a fizikai memória, és laphiba történik. Melyik lapot írjuk ki a háttértárba? Legalább egyet választanunk kell, hiszen ennek a helyére kerül majd be a kívánt adat a háttértárból. (áldozat kiválasztás / Victim Selection)

- Optimális algoritmus  
(gyakorlatilag nem realizálható az abszolút optimális választás, de viszonyítási alapnak mindenképpen jó)
- Legrégebbi lap (FIFO)  
(a múlt alapján próbálja meghatározni a jövőt, a tényleges használatot nem veszi figyelembe, csak a sorrendet)
- Újabb esély algoritmus (Second Chance)  
(mivel tárol egy úgynevezett „Reference bit”-et a korábbi használatokról, szintén a múlt alapján dönt, de a korábbi használatok alapján)
- Legrégebben nem használt (Least Recently Used, LRU)  
(egy úgynevezett „last used” időbélyeget tárol)
- Legkevésbé használt (Least Frequently Used, LFU)  
(szintén egy számlálót használ a döntéshez)
- Utóbbi időben nem használt (Not Recently Used, NRU)  
(kétszintű prioritást használ a döntéshez, két bittel: „hivatkozott” illetve „módosított”)

Minél több tárolt adat segíti a döntést, annál inkább HW támogatást igényel a módszer, különben a futási ideje megnőhet.

Fontos szempont az eddig felsoroltak mellett a terheléelosztás, azaz amikor az operációs rendszer és egy háttértár terheltsége is alacsonyabb, olyankor lehetőség nyílik a virtuális memória lapjainak rendezésére, azaz olyan lapok mozgatására is, melyek cseréje csak hosszabb távon elengedhetetlen.

Az operatív tár, azaz a fizikai memória és a virtuális tárkezelés stratégiai algoritmusai nem csak egy folyamat szempontjából vizsgálható meg, hanem a rendszer egésze szempontjából is. A lapcsere algoritmusok végrehajtása közben merül fel az a kérdés, hogy egy adott folyamat laphibája esetén a végrehajtás milyen hatással lehet a többi folyamatra. A döntési helyzet pedig az, hogy a lapcsere algoritmus csak a laphibát produkáló folyamat saját lapjai közül (lokális memória-gazdálkodás) válasszon, vagy a rendelkezésre álló teljes lapkészletből (globális memória-gazdálkodás).

A globális memória-gazdálkodás egyenletesebb terhelés eloszlással jár, de szükségképpen előtérbe helyezi a nagyobb, több hivatkozással bíró folyamatokat, tulajdonképpen folyamatosan kiszorítva a kicsi folyamatokat. A lokális memória-gazdálkodás közel egyenlő esélyt biztosít a kicsi és a nagy folyamatok részére, viszont így túl statikus ahhoz, hogy a terhelés ingadozásokat hatékonyan legyen képes kezelni.

A két együttesen megvalósítandó cél – a multiprogramozás fokának javítása (és ezzel hatékonyabb erőforrás kihasználás biztosítása), valamint a virtuális memória lapok számának csökkentése (amivel egyre csökken az egy folyamathoz rendelt lapok száma, és így a laphibák gyakorisága növekszik, a CPU feleslegesen terhelődik a laphiba megszakításokkal) – egyszerre és együttesen nem valósítható meg, valamilyen kompromisszumra van szükség.

Az optimális szint eléréshez először is el kell dönteni, hogy melyik paraméter optimumát próbáljuk megvalósítani. A multiprogramozás fokának van elvi optimuma, de gyakorlatilag azt mondjuk, hogy minél nagyobb, annál jobb. A másik paraméter, az egyes folyamatokhoz tartozó lapok száma viszont egyértelműbben optimalizálható. Minden folyamatnak maximum annyi lapra lehet szüksége, hogy a legösszetettebb, legtöbb lapra hivatkozó utasítása is le tudjon futni, illetve a lapok száma nem kell hogy több legyen, mint a folyamat logikai címtartománya.

Az optimumot, azaz az egyensúlyi állapotot úgy fogalmazhatjuk meg, hogy az egyes folyamatok esetében a két laphiba közötti átlagos futási idő haladja meg a laphiba lekezelésének idejét. Másképp fogalmazva, egy folyamatnak legalább annyi lapot kell biztosítani az egyensúlyi állapothoz, amennyi lapra a laphiba lekezelésének ideje alatt hivatkozik. Ez az érték a működő lapkészlet (Working Set). Az elmélet természetesen ez esetben is szebb, mint a gyakorlat, hiszen egy programnak a futása közben, különböző futási szakaszaiban a működő lapkészlete más és más méretű lehet.

Az operációs rendszer a lokális és a globális memória-gazdálkodás között az egyensúlyt egy úgynevezett dinamikus lokális memória-gazdálkodásban találhatja meg, melyben a laphibákat lokálisan kezeli, de a folyamatok igénye szerint képes a lapok globális átcsoportosítására is. Ha számottevő az eltérés egy folyamat működő lapkészlete és a folyamat által a valóságban használt lapok számával, akkor célszerű a lapokat átcsoportosítani. Tekintettel arra, hogy a laphiba gyakoriságának mérése lényegesen egyszerűbb, mint működő készlet meghatározása, a kiindulási alap a laphiba gyakorisága. Túl sok laphiba esetén a folyamathoz kiosztott lapok száma nem elegendő, a lapok számát növelni kell. A másik esetben, amikor szinte elenyésző a laphibák száma, felmerül a kérdés, hogy a folyamat nem birtokol-e esetleg túl sok lapot – másik futó folyamatok kárára.

### 3. Hasznos linkek

[Mi a virtuális memória? \(Microsoft\)](#)

[A virtuális memória méretének módosítása \(Microsoft\)](#)