**Mi az operációs rendszer (2 féle definíció)?**

1.A számítógépem állandóan futó program

2. Az összes program, ami a szállítótól "op. rendszer"-ként érkezik

**Miért van szükség operációs rendszerre?**

Felhasználók kényelmét szolgálja, hatékony gépkihasználás

**Melyek napjaink fő asztali, mobil és szerver operációs rendszerei?**

Asztali: Windows macos linux

Mobil: Android IOS windows

**Nevezze meg és magyarázza el az operációs rendszerek fő feladatait.**

**Végrehajtási környezet:**

Olyan környezet, ahol a felhasználók és programjaik hasznos munkát végezhetnek.

A számítógép **hardver szolgáltatásainak bővítése**

Elrejti a „piszkos” részleteket, könnyű felhasználhatóságot biztosít

**Erőforrás kiosztás**

Kezeli a rendszer erőforrásait:

CPU,

Központi tár,

Merevlemez stb.

Tulajdonságai:

Hatékony

Biztonságos

Igazságos felhasználás.

* Vezérlő program:

Vezérli a **felhasználói programok** működését,

a felhasználói programok helyett **vezérli a perifériák** működését

**Miben és hogyan segített az offline I/O műveletek bevezetése a korai számítógépes rendszerekben?**

Bemenet szalagról (lényegesen gyorsabb mint a kártya).

Egy másik gép végzi a másolást kártyáról a szalagra.

Kimenet: szalagról

Egy másik gép végzi a nyomtatást a szalagról.

+2 gépet igényel

A kisegítő gépek egyszerűbb/olcsóbb gépek

Az I/O műveletek felgyorsulnak, a processzor kihasználtsága nő.

A három számítógép párhuzamosan működik.

Periféria-független programok, szabványos felületek.

**Hogyan működik a pufferelés? Miben és hogyan segíti ez a megoldás az operációs rendszerek működését?**

CPU és a periféria vezérlő egyidejűleg működnek.

A perifériáról történő beolvasás a pufferbe megy.

A CPU az adatokat a pufferből olvassa ki

Az adatfeldolgozás és az I/O művelet átlapoltan történik.

A kimenet hasonlóan működik

A sebességingadozásokat jól kompenzálja

Csak akkor jelent hosszú távon sebességnövekedést, ha

az I/O műveletek összes ideje és

a CPU műveletek összes ideje

kb. azonos (ami általában nem igaz)

**Mit jelent a multiprogramozás?**

A CPU sebessége megnőtt (IC-k)

érdemes kihasználni az időt, amíg egy munka a lemezről olvasás eredményére vár

erre az időre másik program működhet.

*Az OS egyszerre több munkát futtat.*

**Mit jelent, hogy egy rendszer időosztásos?**

Az időosztásos rendszerek (timesharing, multitask systems) *közvetlen interaktív kommunikációt* biztosít a felhasználó és a programja, ill. az OS között.

Adatok tárolására közvetlen (on-line) állományrendszerben.

A felhasználó interakciója nagyon lassú, közben az OS más tevékenységet tud végrehajtani.

Gyors reakció a parancsokra, válaszidő (response time) kicsi. Sűrűn kell a programok között kapcsolgatni.

Több felhasználó is lehet. Felhasználók függetlenül használják a gépet, mintha mindenki egy saját gépen dolgozna.

**Mit jelent a valós idejű rendszer?**

Kemény valós idejű rendszerek (hard real-time)

Szigorúan definiált és betartott válaszidők

Kritikus rendszerek (pl. atomreaktor, járművek)

Adattárolás: RAM, ROM, másodlagos tárolást (diszk) nem támogatják

Általános célú operációs rendszerek nem támogatják

Puha valós idejű rendszerek (soft real-time)

A válaszidők betartására törekednek, csúszás megengedett

Nem kritikus folyamatirányítási feladatok

Multimédia rendszerek, virtuális valóság

**Mik a szigorú valós idejű rendszerek? Soroljon fel tipikus példákat.**

Kritikus rendszerek (pl. atomreaktor, járművek)

**Mik a puha valós idejű rendszerek? Soroljon fel tipikus példákat.**

Multimédia rendszerek, virtuális valóság

**Mit jelent a beágyazott rendszer? Miben különleges egy beágyazott operációs rendszer?**

Tipikus alkalmazási területek:

Háztartási berendezések

Járművek

Játékok/Robotok

Nem általános célú:

Nem független az alkalmazástól (összefordul)

Korlátos erőforrások

Beágyazott környezetben kevés memória, gyenge CPU

A szolgáltatások is korlátosak.

Az OR-nek csak a szükséges része fordul be.

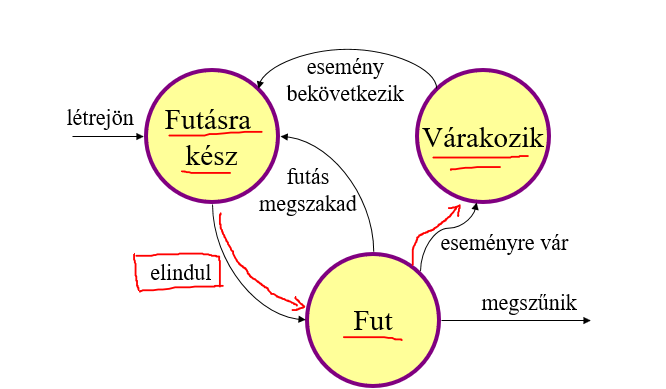
1. **Mi a folyamat?**

* Egy olyan program amelyet éppen végrehajtunk

1. **Mi a különbség és kapcsolat a program és folyamat között?**

* **Folyamat:** A végrehajtás alatt lévő program.

1. **Rajzolja fel a folyamatok működését leíró egyszerű állapotátmeneti gráfot. Mik a gráf csomópontjai és mit jelentenek az élek?**

* 

1. **Magyarázza el, mit jelent a „Fut” állapot.**

* A CPU a folyamathoz tartozó utasításokat hajtja végre, CPU-nként egyetlen ilyen folyamat lehet.

1. **Magyarázza el, mit jelent a „Futásra kész” állapot.**

* Minden feltétel adott, a CPU éppen foglalt. Több ilyen is lehet a rendszerben.

1. **Magyarázza el, mit jelent a „Várakozik” állapot.**

* A folyamat várakozni kényszerül, működését csak valamilyen esemény bekövetkezésekor tudja folytatni. Több ilyen is lehet a rendszerben.

1. **Magyarázza el, mit jelent az „Létrejön” állapotátmenet.**

* A programok a háttértárban vannak eltárolva valamilyen állomány formájában és ezek akkor kelnek életre amikor elindítjuk őket és folyamat lesz belőlük.
* **Folyamat létrehozása:**
* Rendszer indításkor(boot) (pl.: felhasználói interfészek, démon)
* A folyamatot létrehozó rendszerhívás hatására (pl.: program indít programot)
* Felhasználó által (pl.: program indítás)
* Új batch-job indítása 🡪 Kötegelt rendszereknél az OS dönti el mikor kezdhet futni egy új munka (kötegből ilyenkor kiválasztunk egy új folyamatot)
* Az a folyamat amely létrehozhat egy folyamatot az a szülő folyamat
* A létrehozott folyamat a gyermek folyamat
* **Tulajdonságai:** Tetszőleges mélységü, Hierarchikus struktúra
* Egy folyamatnak erőforrásokra van szüksége (pl.: memória, perifériák)
* OP rendszer adhat erőforrást vagy a gyerek osztozhat a szülő erőforrásain (és újabbakat alap esetben nem kap)
* Befolyásolhatjuk a programnak az indulását
* Ha a szülő elindít egy gyerek folyamatot 🡪 párhuzamosan fut VAGY bevárja a gyerekének/gyerekeinek befejeződését

1. **Magyarázza el, mit jelent az „Elindul” állapotátmenet.**

* Ha egy folyamat futásra kész volt akkor ő egy futó állapotba juthat amikor elindul, (az op. rendszer elindítja őt)
* Ha a CPU felszabadul, *egy* folyamat futhat
* **Futó állapot kiválasztása:**
* Ütemezés szerint (CPU ütemezés)

1. **Magyarázza el, mit jelent az „Futás megszakad” állapotátmenet.**

* Futó állapotból futásra kész állapotba kerül
* **Okai:**
* Önként lemond a CPU ról (Kooperatív viselkedés) 🡪 folyamat részéről bekövetkező viselkedési forma
* **Kooperatív viselkedés:**
* Hosszú feladatok esetén újra ütemezést kér

1. **Magyarázza el, mit jelent az „Eseményre vár” állapotátmenet.**

* A folyamat fut és várakozó állapotba kerül, azért mert valamilyen erőforrásra vagy eseményre szüksége van a további futáshoz, de ez most nem áll rendelkezésre
* Az OP. rendszer feljegyzi hogy a folyamat kire, mire vár
* Több folyamat várhat ugyan arra az erőforrásra

1. **Magyarázza el, mit jelent az „Esemény bekövetkezik” állapotátmenet.**

* Az erőforrás (források) rendelkezésre állnak, a várt esemény bekövetkezett, de még nem kerül lefuttatásra (futásra kész állapot)

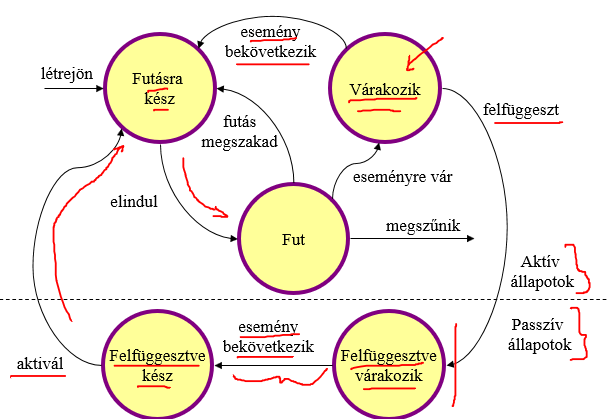
1. **Magyarázza el, mit jelent az „Megszűnik” állapotátmenet.**

* **Úgy szűnik meg:**
* Befejeződik (programnak vége van, futása leáll)
* Hiba miatt leáll
* Felhasználó által leállíthatjuk a futó folyamatot
* Ha egy folyamat befejeződik akkor az általa használt erőforrásainkat az OP rendszer felszabadítja

1. **Mit jelent a „preemptív” ütemezés? (🡪 modern oprendszerek)**

OP Rendszer 🡪 Elveheti a CPU t akkor is ha a folyamat nem kényszerül várakozásra (időosztásos rendszerek)

1. **Rajzolja fel a folyamatok működését leíró kibővített állapotátmeneti gráfot. Mik a gráf csomópontjai és mit jelentenek az élek?**

****

1. **Mit jelent az aktív vagy passzív állapot?**

* Aktív állapot: fut, futásra kész, várakozik
* Passzív állapotok: felfüggesztve várakozik, felfüggesztve kész

1. **Magyarázza el, mit jelent az „Felfüggesztve vár” állapot.**

* A CPU nem kap erőforrásokat, nem fog futni, nem kerülhet futásra kész állapotba.

1. **Magyarázza el, mit jelent az „Felfüggesztve kész” állapot.**
   * Az oprendszer felfügegszti, erőforrásokat elveszi, néhányat megtarthat
2. **Magyarázza el, mit jelent az „Felfüggeszt” állapotátmenet.**

* Az oprendszer felfügegszti, erőforrásokat elveszi, néhányat megtarthat

1. **Magyarázza el, mit jelent az „Aktivál” állapotátmenet.**

* Az OP rendszer az erőforrásokat vissza adja

1. **Magyarázza el, mit jelent az „Esemény bekövetkezik” állapotátmenet a passzív állapotok esetén.**

* Várakozó => futásra kész állapotátmenet történik, ha a folyamat által várt esemény bekövetkezik (például az elindított be/kivitel befejeződik, az erőforrás felszabadul, a szemaforra jelzés érkezik, az üzenet megérkezik stb.).

1. **Mit jelent a környezetváltás?**

* Környezetváltás olyan folyamar, ahol a futó, elhagyja a futó állapotot és egy futásra kész folyamat elindul

1. **Mit kell menteni egy környezetváltás során?**

* Az éppen végrehajtott programot tudnunk kell, hogy hol jár, milyen állapotba van. A CPUnak és az OS tudnia kell hogy hol hagytuk abba a program végrehajtását környezetváltás esetén.

1. **Mi a folyamat állapota? Adjon példákat.**

* Megfelelő program legyen a folyamat.

1. **Mi a végrehajtó gép állapota? Adjon példákat.**

* Azokkal a tulajdonsággal rendelkezzen, mely az adott programhoz szükség.

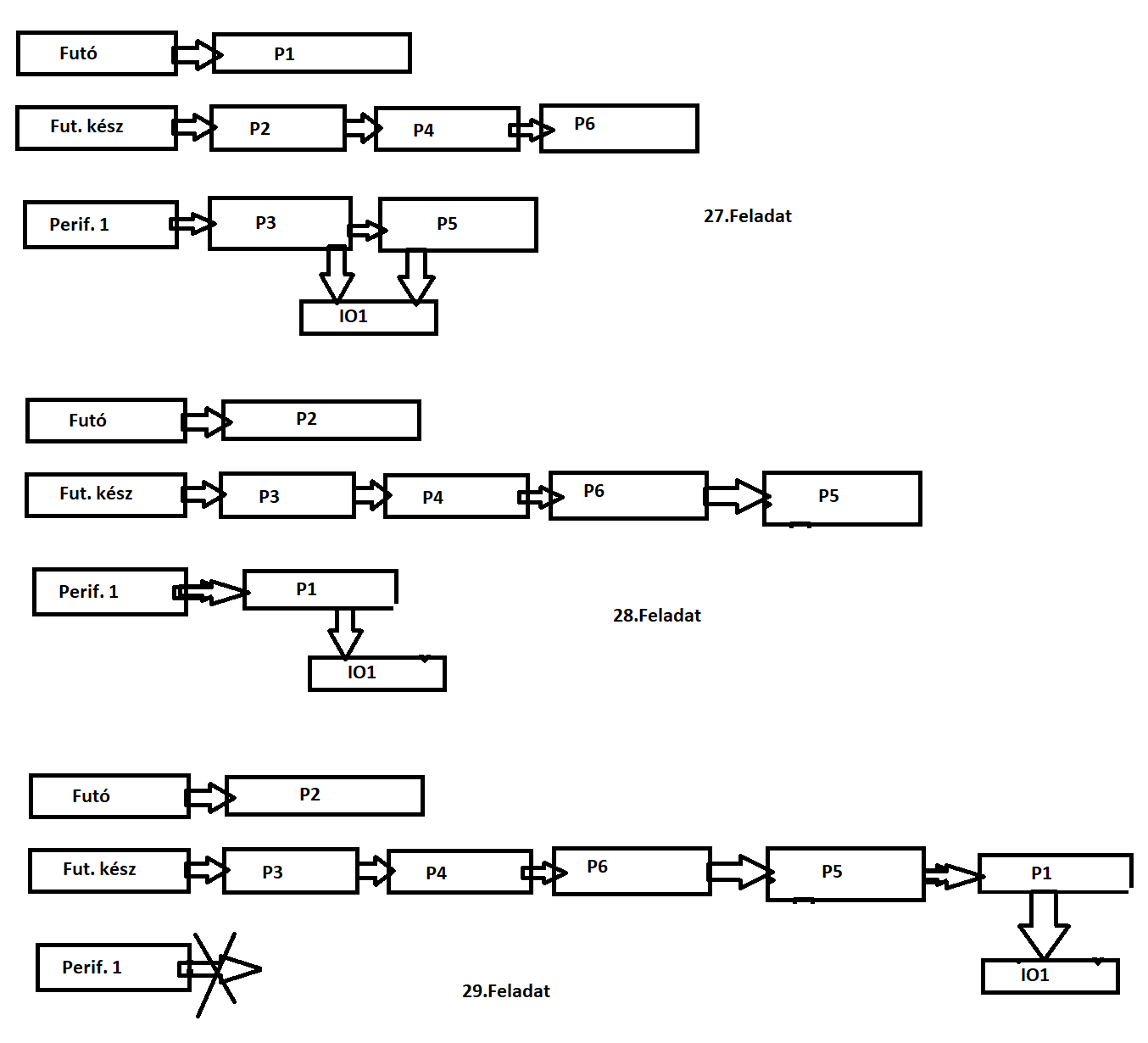
1. **Mi a PCB, mit tartalmaz?**

* Process Control Block – PCB 🡪 Adatszerkezet
* Ezekkel fogjuk leírni a folyamatainkat, tároljuk el a folyamatokhoz kapcsolodó információkat.

1. **Mi az IOCB, mit tartalmaz?**

* Input Output Controll Block – IOCB 🡪 Adatszerkezet
* Azon információkat tároljuk el, mely egy adott folyamat adott perifériával kapcsolatos tevékenysége.

1. **Adott az alábbi hat folyamat: P1, P2, ..., P6: A hozzájuk tartozó folyamatleírók rendre PCB1, PCB2, ..., PCB6. P1 éppen fut, P2, P4, P6 futásra kész állapotban vannak, P3 és P5 pedig az IO1 perifériára várakoznak. Rajzolja fel a folyamatleíró és IO-leíró blokkokból az operációs rendszer által felépített adatstruktúrákat! Nevezze meg az ábrán használt elemeket!**
2. **Módosítsa az előző feladat során rajzolt ábrát: mi történik, ha a P1 folyamat egy perifériára várakozni kényszerül?**
3. **Módosítsa az előző feladat során rajzolt ábrát: mi történik, ha a P1 folyamat perifériaművelete befejeződik?**



1. **Mi a szál? Mi a szál és a folyamat közötti különbség?**
   * Folyamat: A végrehajtás alatt lévő program.
   * Szál: Egy végrehajtás alatt álló program rész, vagy folyamat rész.
   * Szálnak meg van a 3 tulajdonsága: fut, futásra kész, várakozik
   * Különbség:
     1. A szánal nincsen saját memóriája, nincsen saját erőforrása
     2. Egy szálnak csak a regiszterek tartalma és a verem a saját tartalma
     3. Gyorsabban müködik a szálak közötti váltás a folyamatokhoz képest.
     4. Másik előny: Osztott erőforrások gyors kommunikációt tesznek lehetővé.
2. **Mi a megszakítás?**
   * Interrupt: Egy speciális processzor funkcionalitás, ennek hatására egy külső forrásból érkező esemény képes félbe szakítani, az aktuálisan folyó program végrehajtását. A vezérlés átkerül egy speciális programrészbe (megszakítás után) un. interrupt kezelő.
3. **Nevezzen meg tipikus interrupt forrásokat.**
   * Interrupt: Egy speciális processzor funkcionalitás, ennek hatására egy külső forrásból érkező esemény képes félbe szakítani, az aktuálisan folyó program végrehajtását. A vezérlés átkerül egy speciális programrészbe (megszakítás után) un. interrupt kezelő.
4. **Magyarázza el egy interrupt kezelésének menetét az operációs rendszer szempontjából?**

* A megszakítások a számítógép munkájának összehangolásában játszanak fontos szerepet. A megszakítási rendszer a folyamatok közben keletkező események feldolgozására szolgál. Ezen események lehetnek szinkron jellegűek, melyek keletkezése egy program futása közben meghatározható helyen és időpontban várható (például túlcsordulás), aszinkron események, melyek várhatóak, de időpontjuk előre nem ismert (például adatbeolvasás), valamint váratlan aszinkron események, amelyek keletkezése nem várható (ilyen lehet egy hardverhiba). A megszakítási kérelem jelzi a processzornak valamely esemény bekövetkeztét, amely egy kiszolgáló folyamatot indít el egy későbbi időpontban (megszakítás időpontja).
* A megszakítás tulajdonképpen a futó folyamat felfüggesztése annak kiszolgálása céljából.

1. **Milyen módon történhet a környezetváltás az interruptkezelés során? Mikor melyik módszert használjuk?**
   * Két különböző kezelési módszer létezik:
     1. Gyors
        1. Nincs környezet váltás ; Nem lesz másik folyamat, amelyik elindul
        2. Az OS csak egy rövid „rutinja” indul el
        3. A legszükségesebb mentéseket végezzük el ahhoz, hogy az OS rutinja ne ronditsa el a futó folyamat állapotát.
     2. Lassabb
        1. Megszakítás során elindul az OS ütemezője és kiveszi a futófolyamatot a futó listából 🡪 Elvégzi az adminisztrációt 🡪 betesz egy másikat 🡪 Ezután indul el az új ütemezett folyamat.
   * Melyik módszer mikor:
     1. Gyorsnál:
        1. Egyszerű
     2. Lassabbnál:
        1. Komolyabb környezet váltás esetén: ugyanazt a mechanizmust végezzük, mint a folyamatoknál.
2. **Mit jelent az, ha két folyamat független?**
   * egymás működését nem befolyásolják,
   * aszinkron működés:

* egymással párhuzamosan is végrehajtódhatnak,
* Nincs időbeli függőség

1. **Mit jelent az, ha két folyamat függ egymástól? Milyen esetei lehetnek a függésnek?**
   * + logikailag független folyamatok, de megosztott erőforrás használat (pl. több user, azonos gépen dolgozik)
     + logikailag függő folyamatok
     + közösen oldanak meg valamely feladatot
     + együttműködnek, kommunikálnak, közös változók, stb
2. **Milyen tipikus felhasználói esetek indokolhatják együttműködő folyamatok használatát?**

* Erőforrások megosztása
  + - átlapolt működés, jobb kihasználtság
* Számítások felgyorsulása (több processzor)
  + - Számítások párhuzamosítása, végrehajtási sebesség nő.
* Felhasználók kényelme
  + - Egy időben több feladat megoldása.
* Modularitás
  + - Egy adott folyamat kisebb részekre való bontása
    - Jobb áttekinthetőség
    - Bizonyos feladatoknál (párhuzamos részek) kézenfekvő modell.
    - A függőséget valahogy biztosítani kell

1. **Mit jelen a szinkronizáció?**

* A folyamat végrehajtásának olyan időbeli korlátozása, ahol ez egy másik folyamat futásától illetve egy külső esemény bekövetkezésétől függ.

1. **Mi a szinkronizáció 3 tipikus alkalmazási esete?**

* Precedencia (előidejűség)
* Egyidejűség
* Kölcsönös kizárás (versenyhelyzet, kritikus szakasz)

1. **Mit jelent az előidejűség? Adjon rá saját hétköznapi példát (ne az előadás példáját).**

* P1 folyamat *U*1 utasítása előbb hajtódjon végre, mint P2 folyamat *U*2 utasítása

1. **Mit jelent az egyidejűség (vagy randevú)? Adjon rá saját hétköznapi példát (ne az előadás példáját).**

* Két vagy több folyamat bizonyos utasításait (*Sk; Sj*) egyszerre kellelkezdeni.
* Két folyamat találkozása (randevú).
* Két folyamat bevárja egymást mielőtt további működését elkezdené.

1. **Mit jelent a kölcsönös kizárás? Adjon rá saját hétköznapi példát (ne az előadás példáját).**

* A résztvevő folyamatok utasításainak sorrendjére nincs korlátozás, de egy időben csak egyik futhat.

1. **Mi a versenyhelyzet? Adjon rá példát, magyarázza el a problémát.**

* Több párhuzamosan futó folyamat közös erőforrást használ. A futás eredménye függ attól, hogy az egyes folyamatok mikor és hogyan futnak, ezáltal hogyan (milyen sorrendben) férnek az erőforráshoz.
* Elkerülendő, nagyon nehéz debuggolni!
* Példa: *A mért hőmérséklet (T) értékét egy két szó hosszúságú változóban (TK) tároljuk. A hőmérő folyamat a hőmérsékletet szavanként beírja a változóba, a szabályzó folyamat pedig kiolvassa a változót és annak értékét használja.*

1. **Mi a kritikus szakasz? Adjon rá példát!**

* Kritikus szakaszoknak nevezzük a program olyan (általában osztott változókat használó) utasításszekvenciáit, amelyeknek egyidejű (párhuzamos) végrehajtása nem megengedett.
* Példa: Tipikus kritikus szakasz a közös memória használata.

1. **Mi a kapcsolat a kritikus szakasz és a kölcsönös kizárás között?**

* Kapcsolat: Van egy A és egy B folyamat, ha az A belép egy kritikus szakaszba akkor elindul az A folyamat kritikus szakasza és egy T időpillanatban kilép ebből a szakaszból
* Kölcsönös kizárás: B folyamat elindul A val együtt, de B folyamat nem tud belépni a kritikus szakaszba mert az A már benne van, tehát B nek megkell várnia míg A kilép a kritikus szakaszból, amint A kilép B belép a kritikus szakaszba

1. **Sorolja fel és magyarázza el a kritikus szakaszok megvalósításával szemben állított elvárásokat (3 db).**

* Biztosítsa a kölcsönös kizárást
* Egy időben csak egyetlen folyamat hajthatja végre a kritikus szakaszban lévő utasításokat.
* Haladjon
* Ha nincs folyamat a kritikus szakaszban, de van több belépő, akkor az algoritmus ezek közül véges idő alatt kiválaszt egyet és beengedi a kritikus szakaszba.
* Folyamat várakozása korlátozott legyen (ne éheztessen)
* Csak véges számú esetben előzhetik meg.

1. **Magyarázza el, hogy hogyan lehet interrupt-tiltással kritikus szakaszt megvalósítani. Mi a megvalósítás előnye és hátránya? Hol alkalmazzák?**

* Vannak olyan utasítások is amelyek nem kritikusak és vannak kritikusak is
* Az Op rendszernek úgy jellezzük hogy vannak a kódban kritikus utasítások, hogy létrehozunk egy entry és exit szekciót
* Sokféle megvalósítási lehetősége lehet:
* hardware közeli 🡪 legegyszerűbb az interrupt tiltás
* busy waiting megoldások
* Szemafor (Magas szintű)
* (Van még magasabb is)
* Interrupt tiltás:
* kritikus szakaszba való belépés jelzése 🡪 disable\_interrupt();
* enable\_interrupt(); 🡪 kilépés
* Előnye:
  + - a megszakítást semmi nem szakíthatja félbe
    - ha a kritikus szakasz elindul akkor garantáltan végrehajtódik mivel kikapcsoljuk az interupptot
  + Hátránya:
    - Veszélyhelyzetet tud létrehozni 🡪 az a folyamat amely letiltja az interruptot az az interrupt végrehajtásának idejére lefagyasztja az egész rendszer tehát az Op rendszer sem fog tudni futni 🡪 Ha nem fut az Op rendszer akkor egy folyamatnak joga van az egész számítógépes rendszerünket megfagyasztani
* Alkalmazása:
* Semmiképpen nem pc s környezetben
* Beágyazott rendszerben, jól kézben tartott kódban elképzelhető
* Linux magban vannak interrupt tiltással megoldott kritikus szakaszok, itt is veszélyes, de Op rendszer magja jól védett

1. **Magyarázza el, miért hibás a naiv programozott megközelítés?**

* Egy időben teszteli a flageket mindkét folyamatnál. Nincsen a folyamatok között sorrend. Ellenőrzés és flag beállítás nem egyszerre történik. Pl. egyik folyamat tesztel és a futási eredmény tesztelő, a következő utasításban megkezdődik a lefoglalás (beállítása a flagnek). Ha a két folyamat közé beékelődik egy folyamat azonos időben, akkor a működése nem megfelelő. Az ellenőrzés és flag beállítását megszakíthatatlan módon lehetne végrehajtani, elkerülhető lenne a probléma.

**15. Ismertesse a szigorú váltás (strict alteration) algoritmus működését.  
16. Mit jelent a spin lock?**

* (A közös változó a **Turn**. Jelzi, hogy melyik folyamatnak van lehetősége belépni a kritikus szakaszba. Előre meghatározott módon és sorrendben lehet belépni kritikus szakaszba.) Két Processes változó esetén a process 0 (nulla) várakozásra kényszerül, amíg **Turn** változó értéke 0 nem lesz. Addig végzi a műveletet amíg Turn változó 1 nem lesz. Ha 0 a változó, akkor belép a kritikus szakaszba és ha végzett átállítja a változót 1-re és tovább megy a saját nem kritikus szakaszára. A kettes process arra vár, hogy a változó 1 legyen.
* Amint ez a változó 1 lesz, belép a kritikus szakaszba és ha végzet beállítja a **Turn** változót 0-ra. Ekkor újraindul a process 0. Elkezdi elölről a végrehajtást, megint beléphet a kritikus szakaszba, utána process 1, innentől felváltva tudnak belépni a kritikus szakaszba. Mindin a kritikus műveletet befejező szakasz írja át a Turn változót. Egymásnak adják át a helyet a kritikus szakaszba. Gond lehet a folyamatok sebesség különbég, így előfordulhat, hogy az egyik process feleslegesen sokat vár.
* A mechanizmus neve **SpinLock,** ebben a végtelen ciklusban addig hajtjuk végre az üres utasításokat a process-en belül, ameddig engedélyt nem kapunk (a Turn változó érték megadásáig) a kilépésre a ciklusból. A SpinLock-ban a processor működik, de nem csinál semmi értelmeset. Végtelen NOPE utasítást hajt végre lefoglalja a CPU-t. megoldás hátránya.

**17.** **Ismertesse a Peterson algoritmus működését.**

* Peterson algoritmus mentes a korábbiak gyermek betegségeitől. (A be- és kilépő fv-ek (void-ok)) **Define**-al ki kell kötni a folyamatok számát. Változók megadása (int, string, double), annyi elem van, ahány folyamat és mindegyik folyamat a saját elemét írja. Nincs versenyhelyzettől adódó eltérés Interested jelentés: a folyamat be akar-e lépni a folyamat kritikus szakaszába (0 nem, 1 ha igen).
* Közös változó: A Turn (PPT 20. dia) sorrendiséget állapít meg, mindegyik folyamat fogja írni. tömb, minden folyamat a saját elemét tudja módosítani. Interest: a program be akar-e lépni a kritikus szakaszába. A fv a sorszámával fog meghívódni, vagyis 0-val vagy 1-el. A fv kiszámítja a másik fv azonosítóját, majd a folyamat beírja a saját interested változójába, majd beállítja megint a saját indexére.
* Ezután következik a **SpinLock**, addig várakozunk a ciklusban, amíg a másik folyamat (interestedOther) igaz és a Turn változó a saját process index. Ha egy mésik folyamat szeretne belépni a kritikus szakaszba, akkor az a folyamat hasonlóan jár el. A saját interested változóját igazra állítja, a process betölti a turn-be és ebben az esetben a SpinLock futni fog. A második process addig várakozik a végtelen ciklusban amíg az első pr ki nem lép, vagyis az első interested process értéke hamis nem lesz.
* A Turn változóért a versenyhelyzet fellép, ha egyszere próbálnak belépni a kritikus szakaszba. Mindkét folyamat beállítja a saját interested változóját. A saját indexű elemét, majd beállítják a Turn változó érétékét. A Turn azt az érétket bírja, aki később írt.

**18. Ismertesse a TestAndSet utasítás működését.  
20. Ismertesse a kritikus szakasz megvalósítását TestAndSet utasítás segítségével.**

* **TestAndSet**. Egy belépő és egy kilépő utasítást készítünk. Addig pörgünk, amíg a Flag 0 nem lesz. Függetlenül a Flag értékétől beállítjuk 1-re. Ha a Flag 1 volt, akkor változatlan marad, ha meg 0 akkor 1-re állítjuk. Ha a Flag 1-es volt akkor addig hajtja végre a végtelen üres utasítást, amíg a fleg értéke 0 nem lesz. Ebben a TestAndSet utasítás visszatér 0-val és azonnal átállítja Flag értékét 1-re.
* Ha két folyamat próbál belépni a saját kritikus szakaszába, az a folyamat, amely először hajtja végre a TestAndSet utasítást az visszatér 0-val, a második szükség szerüen 1-est fog találni, hiszen az előző beállította ezt 1-re. Csak egy folyamat léphet be a kritikus szakaszba, a kilépő utasítás beállítja a Flag-et 0-ra. Flag=0-val kilép. A TestAndSet folyamat megszakíthatatlan. Legfeljebb egy folyamat láthat 0-ás Flag-et. Ez a megoldás nem garantálja az „éhezés” kiküszöbölését, nincs garancia arra, hogy ne kelljen az adott folyamatnak sokáig várakoznia.

**19. Ismertesse a Swap utasítás működését  
21. Ismertesse a kritikus szakasz megvalósítását Swap utasítás segítségével.**

* **Swap**. (segéd változót myFlag=1, Flag=0) SpinLock-ban pörög amíg myFlag=1, tehát egyszer biztosan belépünk a ciklusba A ciklusban Swap utasítással kicseréli a myFlag és Flag értékeit. A következő ciklus 0-t talál a Flag-ben, kilép a SpinLock-ból, belép a kritikus szakaszába és végrehajtja a kritikus műveleteket. Majd kilépéskor a Flag-et 0-ra állítja.
* Mivel 1 a Flag értéke, egy másik folyamat, ami be akar lépni a myFlag=1 értékét csereberéli a Flag=1 értékével. Tehát csupa 1-es értékeket fog olvasni egészen addig amíg ki nem lépünk a kritikus szakaszból, a Flag értékét át nem állítjuk 0-ra, amikor is a másik folyamat swap utasítása bele fogja tenni a myFlag változójába a 0-t és ő fog tudni kilépni a SpinLock-ból és belépni a kritikus szakaszba.
* A swap utasítás megszakíthatatlan, tehát egyetlen utasítás fogja végrehajtani azt a swap-ot, ami a Flag=0-t goja kicserélni a myFlag=1-el és ezután az összes többi folyamat már az 1-eseket csereberéli az 1-esekkel, tehát várakozni kényszerülnek. itt sem garantált az „éhezés” kiküszöbölése.

1. **Ismertesse a szemefor működését, két operációját.**

* Dijkstra alkotta meg. Olyan speciális adattípus, amely egy nemnegatív egész szám, és ehhez az adattípushoz, egy p operáció nevű művelet és egy v operációs nevű művelet tartozik. A p operáció végzi a vizsgálást és a belépést, ő végzi a kritikus szakaszba való beléptetést. A v operáció a kilépési művelet lesz, a kritikus szakasz elhagyása.
* Működési elve: A szemafor nem negatív szám értéke addig „pörög” a spin lockban, amíg változó értéke nem lesz legalább egy. Ha 1-nél kisebb a változó értéke, akkor várakozik, üres utasításokat hajt végre. Ha a változó értéke 1 volt, akkor tovább megy és csökkenti a változó értékét eggyel. EZ a vizsgálás belépés. Kilépés esetén a szemafor értékét eggyel megnöveljük

1. **Mi a mutex? Ismertesse a mutex műveleteit.**

* A mutex olyan speciális szemafor, amelynek csak két értéke lehet: 1-es érték 🡪 ez a nyitott (unlocked) 0-ás érték 🡪 ez a zárt (locked)
* Műveletei: Mutex-lock 🡪 Ez felel meg a P operációnak Mutex-unlock 🡪 Ez felel meg a V operációnak

1. **Hogyan lehet szemafor segítségével megvalósítani az előidejűséget? Ismertesse a működést.**

* Előidejűség egy szemaforral: Feladat: P1 folyamat *U***1** utasítása előbb hajtódjon végre, mint P2 folyamat *U***2** utasítása. Működése: Kezdetlegesen a szemafor értékét állítsuk 0-ra (s=0). Ezt követően áthaladás vizsgálatot kell tenni az *U***2** elé.
* Ha a P2 folyamat előbb ér az *U***2** -höz, mint a P1 az *U***1** -hez, akkor a V utasítás megállítja a P2 utasítás futását, addig amíg a P1 folyamat V utasításával fel nem emeli a szemafort. Amikor az U1 utasítás lefutott, akkor a P1 folyamat felemeli a szemafort és ezt követően a P2 tovább fog futni és végrehajtja az ő utasítását.

1. **Hogyan lehet szemafor segítségével megvalósítani az egyidejűséget (randevút)? Ismertesse a működést.**

* Randevú = Szinkronizáció.
* Feladat: *U***1** és *U***2** utasítás egyszerre hajtódjon végre.
* Működése: Két szemafort használunk (s1 és s2, mindegyik értéke: 0, azaz tilos). A P1 folyamatnál az utasítás elé, beteszünk egy V utasítást az S1 szemaforhoz (ez felemeli a szemafort) és utána egy áthaladás vizsgálatot a 2-es szemaforra. Itt felemeljük a 2-es szemafort és utána megkíséreljük az áthaladást az 1-es szemaforon. Ebben az esetben a szemaforok keresztbe vannak elhelyezve, így a P1 folyamat elvégzett V utasítás azt a szemafort fogja felnyitni, amelyikre a P2 folyamat várakozik. P2 esetében ez fordítva működik.

1. **Hogyan lehet szemafor segítségével megvalósítani a kölcsönös kizárást? Ismertesse a működést.**

* Probléma: *U***1** és *U***2** kódrészletek ne tudjanak egyszerre végrehajtódni.
* Feladat: ne tudjanak átlapolodni, (magyarul: impulzusmodulációt végrajtani *(Még magyarosabban: Az impulzusmoduláicó: Az Impulzussorozat valamelyik jellemzőjének változtatása az átvinni kívánt jel, pl. beszéd, pillanatnyi értékeinek megfelelően*.)) vagy az egyikert vagy a másikat kell végrehajtani, de nem egyszerre! Megoldáshoz egy szemafort használunk és kivételesen felemeljük (s=1) (Lásd: PPT/29.dia). U1 elé beteszünk egy P1 utasítást, mögé egy V utasítást. A kritikus szakaszon úgy tud áthaladni a folyamat, hogy először a P utasításon menne át, majd az áthaladást követően a szemafort felemeli és tovább halad. A másik folyamat ugyanúgy viselkedik.
* Ha egyszerre próbálnak meg belépni a kritikus szakaszba: Ha P1 folyamat gyorsabb volt, belép a kritikus szakaszba és s=0, akkor elkezdi végrehajtani a saját kritikus szakaszát. Ha ekkor megérkezik P2, akkor ő is megkísérli a szemaforon az áthaladást, de az most 0 értékű, ezért P2 várakozni fog, nem tud U2-höz menni. Addig várakozik, amíg P1 ki nem lép a kritikus szakaszból és végre nem hajtja a V operációt. Ezt követően P2 belép és végig hajtja a ciklust, de P1 nem tud belépni, így most ő fog várakozni. Az *s* kezdő értékétől függ, hogy hány folyamat lehet egyszerre a kritikus szakaszban. Legtöbbször 1, ezért általában bináris szemafort (mutex) használunk.

1. **Ismertesse a szemafor megvalósítását az operációs rendszer szintjén (folyamatok kezelésével).**

* Elvi megvalósítással: Busy waiting ciklussal, ahol Spin Lock található, amely addig fog futni, amíg az s értéke eggyé válik, vagy annál nagyobbá, majd az s értékét lecsökkentjük. Hátránya: Ez blokkolt állapotban is állandóan futna, így állandóan használná a CPU-t.   
  Multiprogramozott megoldás (a jobb megoldás) A *P*(*s*) az üres utasítás helyén a folyamattól elveszi a CPU-t és várakozó állapotba teszi (feljegyzi a szemaforhoz tartozó valamilyen adatszerkezetekbe). A *V* (*s*)-nél nem csak a számláló nő, hanem egy folyamatot (ha van) futásra kész állapotba tesz (többféle stratégia lehet).

1. **Ismertesse a Monitor működését.**

* Sokkal magasabb szitű szinkronizációs primitív, mint a szemafor
* Gyűjtemény, amelybe eljárásokat, változókat, adatszerkezeteket tudunk bele helyezni és ezeket hívhatjuk szabadon
* Hasonlóan kell elképzelni, mint az objektum orientált programozásban az objektum metódustait és az objektumon belüli változókat kívülről nem tudjuk kezelni csak az eljárásokon keresztül, de a monitor sokkal több mint egy hagyományos objektum
* Egyszerre csak egy eljárás lehet aktív a monitoron belül! 🡪 nem jöhetnek létre versenyhelyzetek

1. **Ismertesse az akadály (barrier) működését.**

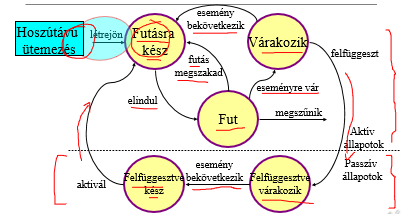
* Randevűk létrehozására szolgál
* Magas szinten, kevés hiba lehetőséggel, jól áttekinthetően lehet vele szinkronizációs mechanizmust leprogramozni
* A, B, C, D elé blokkoló utasítást teszünk és amikor mindegyik oda ér akkor ez a barrier ledől és egyszerre tovább engedi minda 4 folyamatot így ezek a randevú után párhuzamosan tudnak tovább haladni
* Több folyamat (A, B, C, D) szinkronizálására szolgál
* Az akadályt elérő folyamatok blokkolódnak
* Az összes folyamat megérkezésekor az akadály ledől, az összes folyamat egyszerre folytatja futását.

**1. Az ütemezés milyen szintjeit különböztetjük meg?**

* Hosszútávú (long-term):
* Középtávú (medium-term):
* Rövidtávú (short-term): Másodpercenkénti munkavégzés
* Laptop: Rövid, közép
* Szuperszámítógép: mind3 megtalálható.

**2. Ismertesse a hosszútávú ütemezés célját és működését. Mutassa be a folyamatok állapotátmeneti gráfján ennek helyét.**

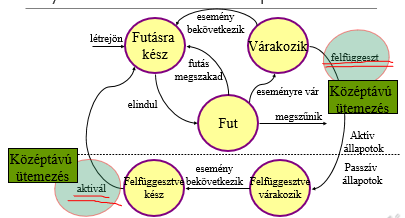
* A háttértáron várakozó, még el nem kezdett munkák közül melyek kezdjenek futni.
* Nem egy kritikus feladat, ritkán kell futnia; egy munka befejeződésekor választunk ki egy új elindítandót.
* Ritkán fut, így nem kell, hogy gyors legyen.
* Követelmények:
  + Olyan munka-halmaz (job-mix) előállítása, amely a rendszer erőforrásait kiegyensúlyozottan használja.
  + CPU-korlátozott és periféria-korlátozott munkák egyenletesen forduljanak elő



* + - Hosszútávú üt 🡪 Létrejön 🡪 Futásra kész 🡪 elindul 🡪 Fut 🡪 eseményre vár 🡪 Várakozik 🡪 esemény beköv. 🡪 Futásra kész 🡪 Fut 🡪 Felfüggeszt 🡪 Felfüggesztve várakozik 🡪 esemény beköv 🡪 felfüg. kész 🡪 aktivál 🡪 futásra kész 🡪 fut 🡪 megszűnik
* Fontos része: Létrejön

**3. Ismertesse a középtávútávú ütemezés célját és működését. Mutassa be a folyamatok állapotátmeneti gráfján ennek helyét.**

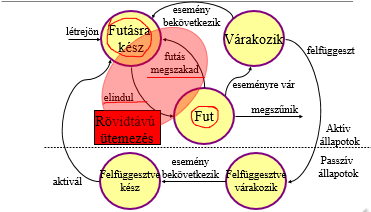
* A rendszer időszakos terhelés-ingadozásait egyes folyamatok felfüggesztésével (passzív állapotba helyezés) illetve újraaktiválásával próbálja kiegyenlíteni. (Időnként túlterhelődik, időként nincs terhelve)
* Felfüggesztés esetén folyamat a háttértáron tárolódik, megfosztják elvehető erőforrásaitól.
* Az ilyen folyamatok (Passzív) nem versengenek tovább az erőforrásokért.



* Fontos része: Várakozik 🡪 Felfüggesztve vár 🡪 Felfüggesztve kész 🡪 Futásra kész +Aktivál és felfüggeszt

**4. Ismertesse a rövidtávú ütemezés célját és működését. Mutassa be a folyamatok állapotátmeneti gráfján ennek helyét.**

* Mely futásra kész folyamat kapja meg a CPU-t (kerül futó állapotba)
* Gyakran fut, ezért gyorsnak (Másodpercek alatt többszáz váltás) kell lennie, különben a rendszer túl sok időt töltene az ütemezéssel, elvéve a CPU-t a folyamatoktól.
* Az ütemező mindig a tárban van, része az OS magjának.



* + Fontos része: Futásra kész 🡪 Elindul 🡪 Fut 🡪 Futás megszakasd

**5. Mit jelent a CPU-löket és az I/O-löket?**

* CPU-löket: A folyamat ezen idő során csak CPU-ra és az operatív tárra van szüksége. A CPU löket átlagos hossza folyamatonként változik.
* I/O-löket/Periféria löket: Ezen időszak alatt a folyamat perifériás átvitelt hajt végre, annak lezajlására várakozik.
  + A folyamatnak periféria löket alatt nincs szüksége a CPU-ra.

**6. Az állapotátmeneti gráf melyik élén történik ütemezés, és ezek közül mely esetekben van biztosan környezetváltás? Miért?**

* 1.típus A futó folyamat várakozni kényszerül. Melyik élén történik ütemezés? : Eseményre vár
* 2.típus A futó folyamat lemond a CPU-ról, vagy elveszik tőle. Melyik élén történik ütemezés? : Futás megszakad
* 3.típus A folyamat felébred, futásra késszé válik. Melyik élén történik ütemezés? : Esemény bekövetkezik
* 4.típus A futó folyamat befejeződik. Melyik élén történik ütemezés? : Megszűnik
* Környezetváltás lesz: 1. és 4. típusnál, mert a folyamat nem folytatja a működését.
* 2. és 3.típusnál nem mindig van környezet váltás

**7. Mit jelent a preemptív és a nem preemptív ütemezés?**

* Nem preemptív: Ha egy folyamattól, miután megkapta a CPU-t, nem lehet azt elvenni.
  + a folyamat csak az általa kiadott utasítások hatására vált állapotot.
  + erőforrásra, eseményre várakozás
  + befejeződés vagy a CPU-ról önként lemondás
* Preemptív: ha az OS elveheti a futás jogát egy folyamattól
  + futásra kész állapotba teszi a futó folyamatot és
  + egy másik (futásra kész) folyamatot indít el.
  + Pl. időosztásos, valósidejű OS-ek

**8. Definiálja a CPU kihasználtság mértékét.**

* A CPU azt mutatja meg hogy az idő hány százalékát használja folyamatok utasításainak végrehajtására.
* Ha jó a kiosztás nagy %-ba a programok futnak
* Ha rossz a kiosztás, akkor a CPU henyél (idle) vagy a rendszer-adminisztrációra fordított idő (rezsi) sok.

**9. Definiálja az átbocsátó képesség mértékét.**

* Megmutatja, hogy az Op.rendszer egy időegység alatt hány munkát tud lefutatni.

**10. Definiálja a körülfordulási idő mértékét.**

* Azt jelenti hogy mennyi idő telik el attól, hogy ezt a program elindítjuk, rendszerbe helyeztük, process folyamatot csináltunk belőle, addig amíg ez a munka lefut.
* (Teljes élettartama a folyamatnak)
* (Egy munka a rendszerbe helyezéstől számítva mennyi idő alatt fejeződik be.)
* Az a jó, amelyik minél előbb elvégzi az utasítás(oka)t

**11. Definiálja a várakozási idő mértékét.**

* Egy munka (vagy folyamat) mennyi időt tölt várakozással (futásra kész állapot, várakozó állapot, felfüggesztett állapotok, (long-term) előzetes várakozás).
* Jó az, ami rövid idő alatt gyorsan lefut.

**12. Definiálja a válaszidő mértékét.**

* Időosztásos (interaktív) rendszereknél fontos. Az Op.Rendszer reakció ideje, a kezelői parancs után a rendszer első látható reakciójáig eltelt idő.

**13. Ismertesse a Legrégebben várakozó (FCFS) ütemező algoritmus működését és tulajdonságait.**

* Amikor egy folyamat futásra kész állapotba kerül, akkor őt betesszük a futásra kész folyamatok közé, a várakozási sor végére.
* Az ütemező pedig mindig a sor elején álló folyamatot kezdi el futtatni.
* Tulajdonság:
* Nem preemptív - Ha elkezdi futtatni a folyamatot, akkor végig futatja, nem szakítja félbe.
* Egyszerűen megvalósítható -
* Konvoj hatás (egy hosszú CPU löketű folyamat feltartja a mögötte várakozókat).

1. **Ismertesse a Körforgó (Round Robin) ütemező algoritmus működését és tulajdonságait.**

* Preemptív algoritmus. A mai korszerű időbeosztásos rendszerek ezen alapulnak.
* Működése: a folyamatok időszeleteket kapnak, előre mehgatározott idő keretek mellyeket arra használhatnak, hogy mennyi ideig futnak. Ha nem sikerül az adott időn bellül elvégezni a műveletett, akkor az Operációs rendszer elveszi a végrehajtás jogát, várakozásra kényszerülnek és elindítja a következő folyamatot. Amint az összes folyamaton végigérünk, az első folyamattól újrakezdődik és ha az adott folyamat van még futtni valója, akkor ismét egy időszelet erejéig futthat.
* Az időszelet nem lehet túlságosan nagy, ha viszont túl kicsi, akkor a sok környezet váltás rontja a hatásfokot. (Ha nagyon sokat környetetett váltunk akkor nagy lesz a rezsi idő). Úgy szokták beállítani az idő szeleteket, hogy a CPU lüketek kb 80%-a legyen rövidebb mint ez az időszelet. Úgy választjuk az időszeleteket, hogy a folyamatok kb 80% essen az időszelettől ballra, 20%-a pedig ettől jobbra. Általában 10-100ms nagyságrendbe esik.
* Példa: 3 folyamat P1=20ms, P2=3ms, P3=6ms, ütemező Tslice (időszelet)=4ms.
* A P1 folyamat 20 ms-ból felhasználja a teljesen 4ms időszeletét (P1-ből még hátra van 16ms), majd az OpRe átadja a futási jogot a P2 folyamatnak. P2 azonban nem tudja kihasználni a teljes 4ms időszeletett, ugyanis 3ms alatt elvégzi a folyamatait (P2 lefutott). P3 megkapja a futási jogot és a teljes 4ms időszeletet kuhasználja (P3-ból maraf 2ms). Az OpRe elveszi a futási jogot, eddig 11ms-nál járunk (4+3+4ms), majd visszatér a P1-re. P1- 4ms, marad még 12ms a P1-ből. P2 teljesen lefutott, ezért átadja P3-nak a jogot. P3 kezdetekor az összes futási idő 15ms, miután P3 elvégzi a maradékját az összes 17ms lesz. Mivel P2 és P3 végzett, ezért P1 addig ismétlődik, amíg P1 is bejezi a sajátját. A végén az összesített futási idő 29ms lesz.
* Időszelet méretének hatása a környezetváltásra.
* Ci folyamat =10ms (/időlökete 10ms). Ha Tslice=12ms, akkor ebben az egyetlen időszeletben képes lesz lefutni, szóval nincs szükség környezet váltásra, 0 db környezet váltás lesz. Ha Tslice=6ms, akkor szükségünk lesz egy 6 és 4ms-os időrészre, tehát 1 db környezetváltozásra lesz szükség a futtatásához. Ha Tslice=1ms akkor 10 db időszeletre lesz szükség, vagyis 9db környezet váltásra volt szükség, hogy be tudja fejezni a működését.

1. **Ismertesse a statikus priorotás alapú ütemező algoritmus működését és tulajdonságait.**

* Adottak a folyamatok, azok ideje (ms-ben megadva) és a prioritásuk. A prioritásokat számmal jelöljük. Mindig az a folyamat lesz az első, amelynek a legmagasabb a prioritásszáma. Minél nagyobb a szám, annál kisebb a prioritás (Ha pl. a második folyamat prioritása 1, akkor az fog kezdődni, ha a negyedik folyamat prioritása 2, akkor az lesz a második folyamat, ami lefut.). Ha két folyamat prioritása megegyezik, akkor teljesen mindegy, hogy melyiket vesszük előbb. A Statikus prioritásnál meg kell határoznunk az átlagos várakozási időt, amit úgy kapunk meg, hogy összeadjuk az egyes folyamatok várakozási idejét (ms-ben van megadva, általában az első lefutott folyamatnak nincs várakozási ideje, mert az fut le először, nem kell várnia), majd el elosztjuk a folyamatok számával (átlag)

1. **Ismertesse a Legrövidebb löketidejű (SJF) ütemező algoritmus működését és tulajdonságait.**

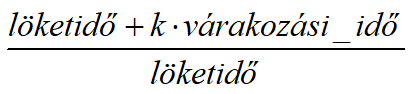
* Nem preemptív algoritmus, a futásra kész folyamatok közül a legrövidebb löketidejűt indítja. Nincs konvoj hatás, optimális körülfordulási idő, optimális várakozási idő. Azt kell futtatni először, annak kell először prioritást adni, amelynek a legrövidebb a löketideje. (pl. Egy bevásárlóközpontban az mehet először a pénztárhoz, aki legkevesebb árucikket, terméket tette bele a kosarába). A példában a nempreemptív azt jelenti, hogy ha egy vásárló a péztárhoz érkezik, azt szolgálják ki elsőként, hiába van mögötte (később érkezett) olyan vásárló, aki kevesebb terméket tett bele a kosarába.
* Alkalmazása:
* Hosszú távú ütemezés
* Rövid távú ütemezés (RT rendszerek)

1. **Ismertesse a Legrövidebb hátralévő idejű (SRTF) ütemező algoritmus működését és tulajdonságait.**

* Az SRTF az SJF preemotív változata. Teljesen hasonló a működés, a cél az, hogy a rövid futási idejű algoritmusok kerüljenek előre. A különbség annyi, hogy amennyiben egy új folyamat futásra kész állapotba kerül, akkor felülvizsgáljuk a folyamatainkat. Az éppen futó folyamatnak mennyi a hátralévő ideje, és mennyi a várakozók közül a legrövidebb futási időnk. Ha a várakozók közül van egy rövidebb futási idővel rendelkező folyamat, akkor az fog előre kerülni, és az éppen futó folyamatot kivesszük. A folyamat megszakítása és egy másik elindítása környezetváltozást igényel, ezt az időt is figyelembe kell vennünk.

1. **Ismertesse a Legjobb válaszarány  ütemező algoritmus működését és tulajdonságait.**

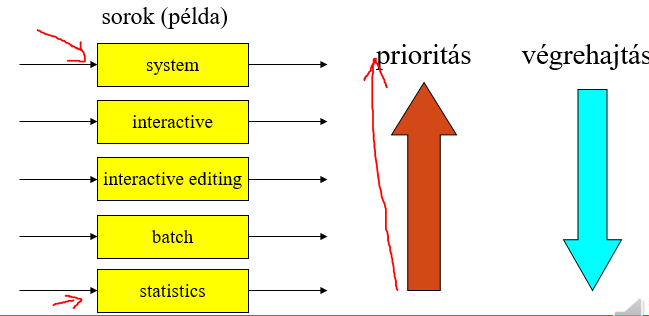
* Ismertesse a Legjobb válaszarány ütemező algoritmus működését és tulajdonságait.
* Az SJF algoritmus változata, ahol a várakozó folyamatok öregednek. A kiválasztás (a löketidő helyett) a



* válaszarány szerint megy végbe (a legnagyobbat választjuk ki), ahol k egy jól megválasztott konstans. Minél nagyobb a válaszarány, annál nagyobb prioritást adunk a folyamatnak. Éretelemszerűen minél nagyobb lesz a várakozási idő, annál nagyobb lesz a válaszarány is, és így magas prioritású lesz. Amíg viszont pici a várakozási idő, addig a rövidebb löketidők jutnak előnyhöz. Hogy a k konstans mekkora, azt attól függően kell, hogy milyen dinamikus tulajdonságokat várunk el az ütemezőtől.

1. **Ismertesse a Statikus többszintű sorok ütemező algoritmus működését és tulajdonságait.**

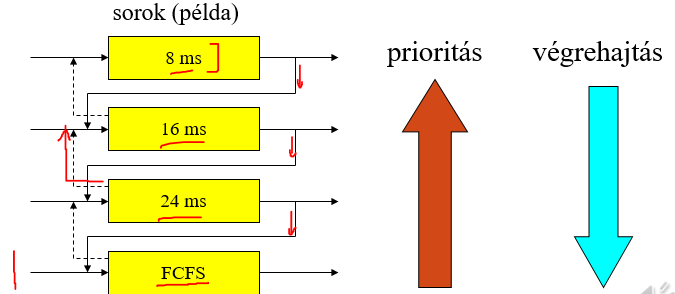
* Egy folyamat mindig a saját sorában marad
* Statikus állandó sorokat tartalmaznak
* a felső sorok fontosabbak mint az alsók és fentről indulunk amint kiürült a legfelső megyünk egyel lejebb majd ha ez is ki ürült megyünk még lejebb és így tovább
* A folyamatot elindulásakor valamilyen kritérium alapján besorolunk egy várakozó sorba.
* A folyamat élete során végig ugyanabban a sorban marad.



* Egy lehetséges példa a prioritások besorolására:
* rendszer folyamatok (magas prioritás, közvetlen hatással vannak a rendszer működésére)
* interaktív folyamatok (biztosítani kell a felhasználó számára az elfogadható válaszidőt)
* interaktív szövegszerkesztők (kevésbé kritikusak)
* kötegelt feldolgozás (általában akkor futnak, ha "van idő")
* rendszerstatisztikákat gyűjtő folyamatok (alacsony prioritás, nincsenek közvetlen hatással a rendszer működésére)

1. **Ismertesse a Visszacsatolt többszintű sorok ütemező algoritmus működését és tulajdonságait**

* Löketidő = az az idő amennyi alatt a folyamat lefut
* FCFS = First Come First Served
* A sorok között lehet vándorolni
* A sorhoz időszeletet rendelünk
* A végrehajtás hasonló a statikushoz, fentről lefele haladunk és ha a felső kiürül akkor jöna következő
* Az Op rendszer monitorozza a folyamatokat és viselkedésük alapján pakolgatja sorról sorra
* Ha pl.: 8ms az ideje egy sornak akkor elvárjuk hogy a feladatát befejezze ennyi idő alatt, ha nem fejezi be a folyamat a működését ennyi idő alatt akkor az Op rendszer leállítja azt és lejebb rakja egy sorral ahol már 16ms áll a rendelkezésére ha itt sem fér bele akkor lejebb kerül a 24ms es sorba ahonnan a legalsóba kerülhet ahol nincs követelmény és a legrégebben várakozó hajtódik végre amint a fölötte lévő sorok kiürültek
* Egy folyamat feljebb is kerülhet egy sorral a rövid futási ideje miatt



1. **Mit jelent a multiprocesszoros ütemezés témakörében a heterogén és homogén rendszer?**

* Heterogén rendszerek esetében egy folyamat csak bizonyos processzorokon futhat.
* Homogén rendszerekben a futásra kész folyamatokat közös sorokban tárolja.

1. **Mit jelen a szimmetrikus és aszimmetrikus homogén multiprocesszoros rendszer?**

* Szimmetrikus multiprocesszoros rendszer:
* Minden CPU saját ütemezőt futtat, amely a közös sorokból választ. A sorok osztott használatához a kölcsönös kizárást biztosítani kell!
* Aszimmetrikus multiprocesszoros rendszer:
* Az ütemező egy dedikált CPU-n fut, ez osztja szét a folyamatokat a szabad CPU-k között.

1. **Az előadás egyszerű példáihoz hasonló ütemezési feladatok és számítási feladatok. Ajánlott az előadás példáinak önálló megoldása.**

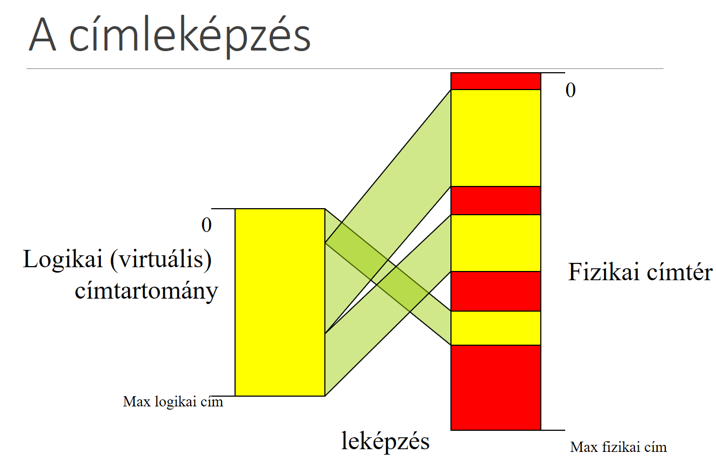
**1. Mit jelent a logikai (virtuális) cím?**

* Azt értjük logikai címtartománynak, ahol a programunk logikailag működik. Ez egy folytonos címtartomány, ami lineárisan nő és 0-tól kezdődik és egy adott maximális értékig tart.

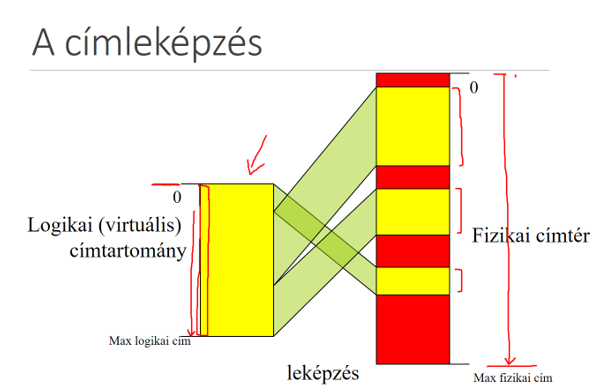
**2. Mit jelent a fizikai cím?**

* A fizikai címtartomány azt mutatja meg, hogy a valóságban hol helyezkedik el a programunk az operatívtárban. Nem a 0 fizikaicímtől helyezkednek el a programok végrehajtása nem is folytonosak, több darabból is állhatnak.

**3. Ismertesse a középtávútávú ütemezés célját és működését. Mutassa be a folyamatok állapotátmeneti gráfján ennek helyét.**



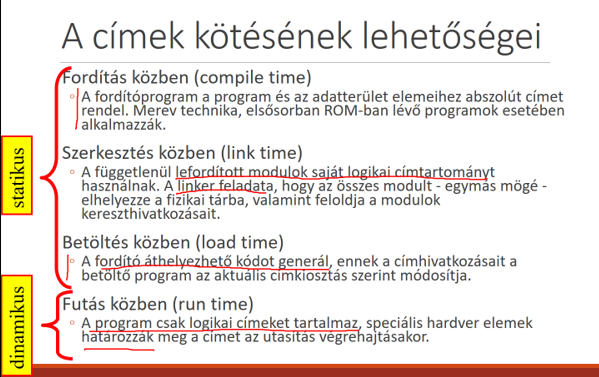
* Láthatjuk, hogy a bal oldali logikai címtartományból töltjük meg a fizikai címtartományba a dolgokat és ezt összevissza teszi a logikaicímtartományból a dolgainkata fizikai címtartományba. (Bal oldali a folytono logikai címtartomány.) (Jobb oldal Pl.: hogyan akarja betölteni a memóriát, ha 8 GB-nk van akkor a max fizikai cím a 8 GB-t)
* Szóval a bal oldalkában van a programunk és ez a jobb oldalban helyezkedik el több darabban, ahogy az ábra is mutatja



**4. Mit jelent a statikus címleképzés/címkötés? Milyen fajtái vannak?**

**5. Mit jelent a dinamikus címleképzés/címkötés?**

* Statikus:
  + Az első három módszert hívjuk statikus kötésnek, mivel ott már előre eldöntödik véglesen az, hogy a kód hol fog helyet foglalni a fizikaimemóriában. Ezeknek a lehetőségei a képen látható: Fordítás közben, Szerkesztés közben, Betöltés közben.
* Dinamikus:
* Ez ugye nem előre eldöntettet, hanem a futás közben fog kiderülni/kialakulni az, hogy egy adott logikai címtartományban mozgó kód az éppen hova fog kerülni a fizikai címtérben. (EZ A KORSZERŰ) Az ütemező mindig a tárban van, része az OS magjának.



**6. Mi a logikai és fizikai címek kapcsolata statikus, illetve dinamikus címkötés esetén?**

* Szóval a logikai címünk az, amit a kódunk tartalmaz, ami futásközben egy változó címre (Pl.: memória) uta vagy egy ugróutasításban szerepel tehát amit a CPU utasítás végrehajtás közben generál. Ellenben a fizikai cím a memória valós címe, amely akár meg is egyezhet a LOGIKAI címmel STATIKUS címkötés esetén (Fordítás közben, Szerkesztés közben, Betöltés közben.), de a FIZIKAI és a LOGIKAI címek különbözni fognak DINAMIKUS címkötésesetén. (Futás közben)

**7. Mi az MMU? Mi célt szolgál?**

* Az a hardver egység, amely a virtuális címeket fizikai címmé alakítja.

**8. Ismertesse a bázis-relatív címzés működését.**

* A program tetszőleges helyre betölthető. A bázisregisztert a betöltési kezdőcímre állítva a program végrehajtható.

**9. Ismertesse az utasításszámláló-relatív címzés működését.**

* Pozíció-független kód. Csak relatív címeket tartalmaz.

**10. Mit jelent a dinamikus betöltés? Ismertesse a működését.**

* A programhoz tartozó egyes eljárások a háttértárban vannak, ha valamelyikre szükség van, azt egy speciális programrészlet betölti.
  + Hívás
  + Ellenőrzés: memóriában van?
  + Speciális programrészt betölti a memóriába, vezérlést átadja

**11. Ismertesse a dinamikusan betöltött könyvtárak működését.**

* A programban használt rendszerkönyvtárak eljárásai helyett csak egy csonk kerül a programba.
* A csonk tartalmaz egy hivatkozást:
  + a könyvtárra
  + és az azon belüli eljárásra
* A csonk első meghívásakor az OS az eljárást betölti a tárba
* A következő hívások már az eljárást hívják meg időveszteség nélkül.

**12. Mit jelent a dll-pokol?**

* A dinamikusan betöltött könyvtárak hátránya. Ha megjelenik egy verzió valamilyen dinamikus könyvtárból, akkor azt használja a program, ami készülés alatt van. Ha a dinamikus könyvtár(dll) nem működik teljes mértékben, akkor ehhez a hibás működéshez előkészítik az adott programot.
* Ha esetleg publikálnak egy javítást a dll-hez, akkor nem biztos, hogy a program helyesen fog működni. Gyakran előfordul, hogy valamilyen programhoz valamilyen verziójú dll-re van szükség, tehát nem ”szereti” az újabbat. Fontos, hogy minden program a megfelelő dll-t fogja használni.

**13. Ismertesse az átfedő programrészek (overlay) működését.**

* A program szerkezetében vannak közös adatok és programrészek, és vannak olyan részei is, amelyekre bizonyosan nem lesz szükség egyszerre (mindig csak legfeljebb 1-re 🡪 ezek az átfedő részek) Ebben az esetben alkalmazható az overlay technika.
* Ezeket az átfedő részeket egyesével töltik be a memóriába, amikor szükség van rá. Ha befejeződik egy rész használata, akkor annak a helyére kerül a következő átfedő programrész. Itt nincs szükség az OS támogatására.
* Az átfedés számára fenntartott tárterület a legnagyobb átfedő programrész hosszával egyenlő.

**14. Hogyan működik a memóriaallokáció egypartíciós rendszerekben?**

* Az egypartíciós rendszerek a korai, legegyszerűbb rendszerek voltak. Nagyon egyszerű volt a memóriaszervezése.
* Müködése:
  + Van egy védett terület, amely az OS területe. Itt találhatóak mindazon rendszerek és folyamatok, amelyek a működéshez nélkülözhetetlenek 🡪 Ez az operációs rendszer memóriája.
  + Ezen felül van egy felhasználói terület. Ezt a területet egy folyamat használja egyszerre úgy, hogy ezt betöltik az első szabad címre, ami az OS területe felett van. Itt futnak a felhasználói programok.
  + Ha az OS-nek szüksége van memóriára, akkor
    - vagy áthelyezi a programot,
    - vagy a nem használt területről allokál.
  + A memória két nagy részre van osztva:
    - felhasználó memóriaterület
    - OS
  + A kettő határát a háttérregiszter mutatja meg (ahol az egyik terület vége van, és a másik kezdete), valamint az OS területének védelmére szolgál.
  + A felhasználó memóriaterületre a háttértárról lehet a programokat betölteni.
* Az OS kétféle módon futhat egypartíciós rendszerben:
  + Felhasználói módon: Futás közben egy hardver figyeli (egy védelem), hogy minden hivatkozás a határregiszter felett legyenen. Az OS-be való címzés tilos!
  + Rendszer módon: Rendszerhíváskor a védelem kikapcsol, az OS az egész címtartományt eléri.

1. **Hogyan működik a memóriaallokáció többpartíciós rendszerekben? Mik a fix- és változó partíció méretű rendszerek?**

* Ha egy multiprogramozott rendszer áll rendelkezésre, akkor lépnek életbe a többpartíciós rendszerek. A multiprogramozás elve megköveteli, hogy több folyamat legyen egy időben a tárban.
* Erre kétféle megoldás lehetséges:
  + Fix partíciós rendszerek: Itt az OS feletti felhasználói területet felbontják fix méretű partíciókra (ezeket futás közben nem változtatják)
  + Változó partíciós rendszerek: Az OS az igényeknek megfelelően kisebb vagy nagyobb partíciókat hoz létre és rendelkezésre bocsátani a folyamatoknak.

1. **Mit jelent a belső tördelődés? Fellép-e a fix méretű partíciók esetén? Fellép-e a változó méretű partíciók esetén?**

* A fix, illetve a változó méretű partíciók egyik hatása. Ez akkor jön létre, ha a folyamatok nem használják ki a rendelkezésére bocsátott partíciót. Ez tipikusan fix partíciók esetén alakul ki.
* A partíciók belsejébe jön létre, mert ha a felhasznált terület nem telítődik meg az adott partíción, akkor a maradék lesz szabad tárterület. Ekkor tördelődik két részre a partíció (egy felhasznált és egyszabad területre.

1. **Mit jelent a külső tördelődés? Fellép-e a fix méretű partíciók esetén? Fellép-e a változó méretű partíciók esetén?**

* A fix, illetve a változó méretű partíciók másik hatása.
* Lefoglalnak valamekkora partíciót, Ha a másik partíció nem közvetlenül az előző partíció feletti területen kezdődik, akkor van közte egy ”rés”, vagy szabad tárterület. Ezt nevezzük külső tördelődésnek, hiszen nem egybefüggő az elfoglalt memória.
* A szabad memóriaterületek elszórva, kisebb darabokban, nem szomszédos részeken találhatók meg. Ezek általában túl kicsi területűek ahhoz, hogy elhelyezzünk rajtuk egy programot, ezért ezek elveszett területeknek számítanak.

1. **Mit jelent a garbage collection? Milyen rendszerek esetén van rá szükség? Hogyan működik?**

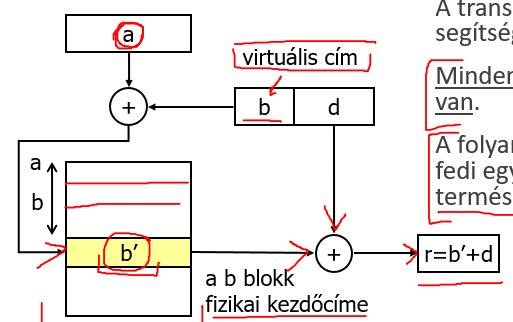
A külső tördelődés bizonyos fok után lehetetlenné teszi újabb  
folyamat elindítását (elég a szabad terület, de a leghosszabb  
összefüggő szabad terület nem elég a folyamatnak).  
Megoldás: a szabad helyek tömörítése.  
Tömörítő algoritmusok:  
◦ időigényes (nem biztos hogy megéri futtatni, esetleg jobban járunk, ha  
megvárjuk, míg néhány folyamat befejeződik).  
◦ HW támogatást igényel.  
◦ váratlanul lehet szükség rá (ezért pl. egy interaktív rendszer válaszideje  
hirtelen megnőhet)  
◦ a tárterületek mozgatását körültekintően kell végrehajtani (az áthelyezési  
információk megőrzése, stb.

1. **Mit jelent a tárcsere (swap)?**

* Tárcsere során az OP rendszer egy folyamat tárterületét, akár egy részét vagy teljes részét, kidobhatja az operatív tárból és a háttértárra másolhatja, ily módon az operatív tárban helyet tud felszabadítani más folyamatok számára. Időigényes művelet, sokkal több idő, mint egy környezetváltás. Ne legyen túl sűrűn tárcsere, mert sok időbe telik.  
  Módszerei:
  + Hardver támogatás
  + mesterséges folytonosság: Virtuális címtartományban folytonos program a valóságban nem az.
* A futó folyamatoknak nem az egész címtartománya van a központi tárban:
* A nem érvényes hivatkozás (nincs a központi tárban) esetén automatikus programrészlet betöltés (OS – hardver együttműködés).

1. **Ismertesse a futás közbeni címleképzés működését.**

* Adott egy folyamat, ami a logikai címtartományban működik, ahol virtuális címeket generál. Ezeket kéne futás közbe leképezni fizikai kezdőcímekre.



* Blokktábla: Itt tároljuk a címleképezéséhez, a cím megkötéséhez szükséges. Minden egyes blokkhoz tartalmaz egy bejegyzést, amely az adott blokk fizikai kezdőcímét fogja tartalmazni.
* Blokktábla kezdőcíme: (a), amit egy regiszter tartalmazza. Ha az (a) kezdőcímhez hozzáadjuk a (b) blokkcímet akkor a blokktábla azon bejegyzésére fogunk mutatni, ahol az ahhoz a blokkhoz tartalmazó fizikai cím (b’) található. A (b’)-höz ha hozzáadjuk a (d) eltolást, akkor megkapjuk a virtuális cím megfelelője (r) .
* **FONTOS!** Minden folyamatnak saját blokktáblája van.
* Virtuális címeket két részre kell osztani, felső rész: blokkcím (b), alsó rész: eltolás (d)
* Blokkcím (b): A blokkoknak a sorszámát fogja tartalmazni. Egy kód sok-sok blokkra osztható
* Egy cím tehát a blokkok sorszámát és az eltolást fogja tartalmazni. 🡪 Így keletkezik a virtuális cím, ami lehet ugróutasítás vagy egy memóriában található adatok címzése.

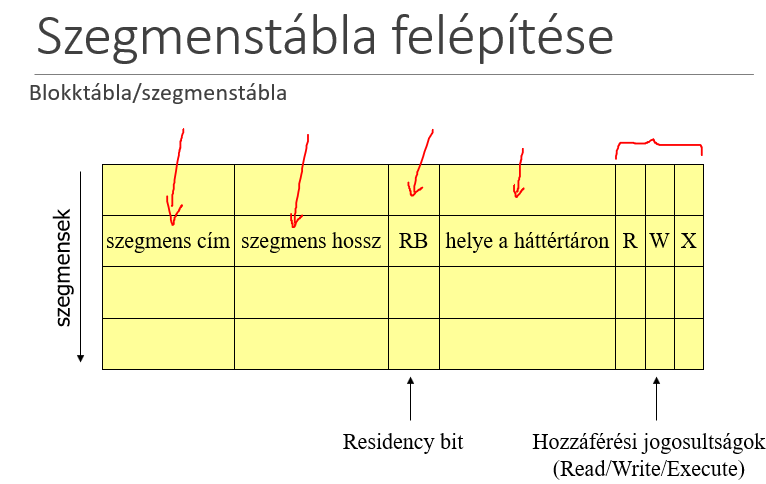
1. **Ismertesse a szegmensszervezés működési elvét, a védelem módszereit.**

* A blokk most egy szegmens lesz, a logikai címtartomány tehát olyan címtartomány, amelyben szegmensek lesznek. Úgy állítjuk össze a szegmenseket, hogy logikai egységeket tartalmazzon pl.: lehet egy szegmensben főprogram, eljárások, függvények, változók, stack, stb.
* Védelme: Egy szegmens nem címezhet ki a saját szegmenséből. Ellenőrzése könnyű, tudnunk kell a szegmens méretét 🡪 Szegmensen belüli cím ≤ szegmens mérete különben megszakítás. A szegmenstábla tárolja a szegmens méretét (limit).
* Hozzáférés ellenőrzés:
  + olvasási jog (szegmens területét olvashatja)
  + írási jog (szegmens területére írhat, az ott lévő értékeket módosíthatja)
  + végrehajtási jog (a szegmensben gépi utasítások vannak, azokat a folyamat végrehajthatja)
  + Jogosultság megsértése, megszakítást generál (segment protection fault)

1. **Mit jelent az osztott szegmenshasználat?**

* (Takarékos memóriahasználat)
* Közös utasítás használata lehet:
  + teljes program, rendszerkönyvtár
  + több folyamat azonos programot futtat
* Közös adatterület
* Folyamatok közötti kommunikáció
* Megvalósítás:
  + A folyamatok szegmenstáblájában valamelyik szegmensnél azonos fizikai cím van.
  + A jogosultságok természetesen különbözőek, biztosítani kell a kölcsönös kizárást.

1. **Ismertesse a szegmenstábla felépítését. Mire szolgálnak az egyes bejegyzések?**



* Szegmens cím: A szegmens neve
* Szegmens hossz: A címzés nem mutat-e ki a szegmensből
* Residency bit (Benttartozási bit): tényleges a memóriába vagy a háttértáron van-e a szegmens
* Hozzáférési jogosultságok: Irható/olvasható/végrehajtható
* Helye a háttértáron: Hol található

1. **Milyen előnyei és hátrányai vannak a szegmensszervezésnek?**

* Előnye: lehetőséget ad arra, hogy több folyamat egy-egy logikai szegmense ugyanarra a fizikai címtartományra legyen leképezve, vagyis, hogy a folyamatok közös eljárásai egyetlen példányban tárolódjanak a memóriában. Könnyen, gyorsan elvégezhető lefutás
* Hátránya: Külső tördelést hozz létre, nagy külső tördelődés, mivel nem azonos méretűek a blokkok.

1. **a, Ismertesse a lapszervezés működési elvét.**

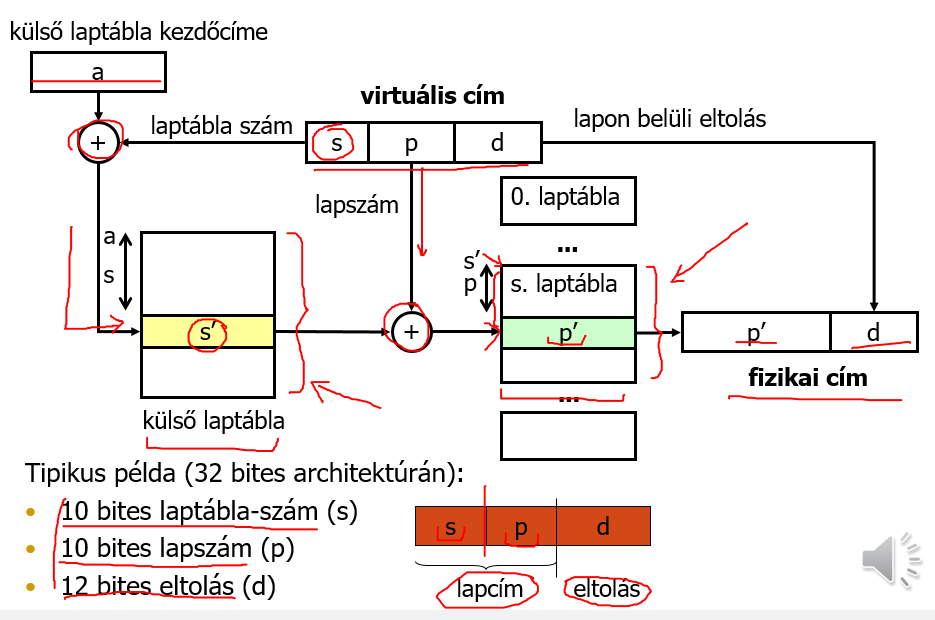
* Azonos méretű blokkok (lap, page) használata
* Megszünteti a külső tördelődést.
* A belső tördelődés elkerülése miatt kis lapokat érdemes választani.
* A lapok mérete mindig 2 hatvány

**b, Hogyan működik az egyszintű címtranszformáció?**

* Címtranszformáció: lényegében egy kétszintű táblakezelés.
* Első szint: laptábla címeket tartalmazó szegmenstábla
* Második szint: fizikai lapcímeket tartalmazó laptábla (szegmensenként egy). A cím három részre tagozódik (szegmenscím, lapcím, lapon belüli eltolás).
* Hozzáférési jogok ellenőrzése a szegmens szervezésének megfelelően történik. Osztott tárhasználat a szegmens szervezésének megfelelően történik.

1. **a, Miért van szükség többszintű címtranszformációra?**

* Laptábla mérete nagy lehet, mivel a lapok száma sok.
* Nehéz gyors elérésű tárban tartani.
* Pl.:
  + - 32 bites címtér (232)
    - 4 Kbites lapok (212)
    - 220 (≈106) db bejegyzés
* Megoldás: laptábla tördelése 🡪 többszintű laptábla.
* Felfogható a laptábla lapozásának is.

**b, Hogyan működik a többszintű laptábla?**

* Jelen esetben 2 szintű laptábla
  + s: laptáblaszáma
  + p: lapnak a száma
  + virtuális cím s p d formátumban jelenik meg
  + a: külső laptábla regiszter
  + a+s = Külső laptáblában megkapjuk azt az indexet amelyet el akarunk olvasni jelen esetben az s’
  + s’: tartalmazza azon laptábla kezdőcímét amelyikkel dolgoznunk kell
  + s’+p = Ebben a laptáblában megkapjuk a megfelelő indexet és ide fogunk bele címezni és itt megtaláljuk a p’ értéket amely mostmár a lapunk fizikai kezdőcímét tartalmazza ehhez hozzáfűzve az eltolást megkapjuk a fizikai címet
  + két lépésben állítottuk elő a fizikai kezdőcímet, először a külső laptáblában aztán pedig a kiválasztott laptáblában keresgéltünk és így állítottuk elő a fizikai címet

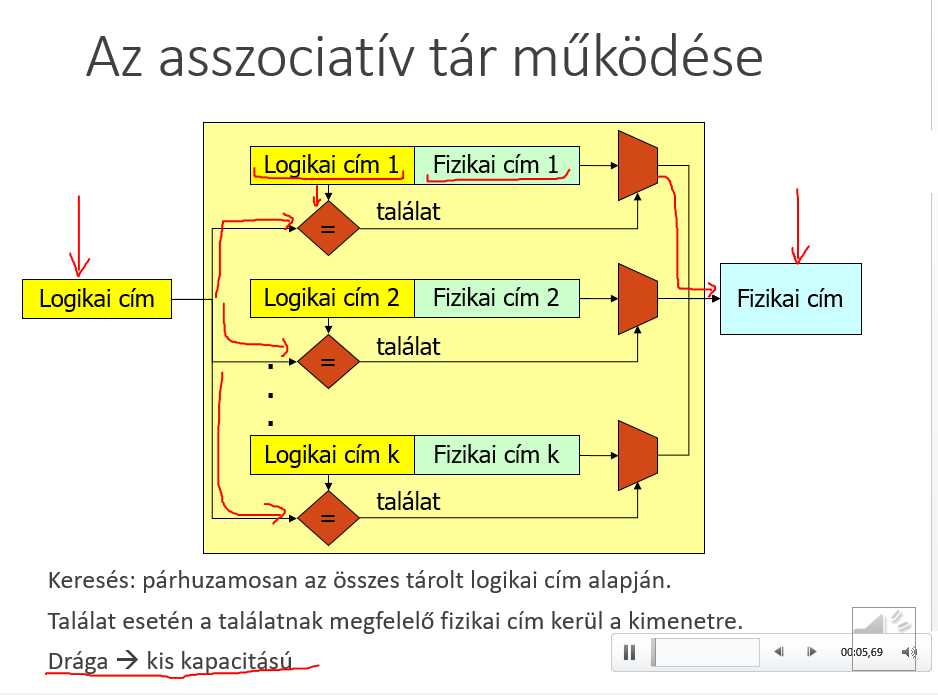
1. **a, Miben segít az asszociatív leképzés?**

* Speciális gyors elérésű tár (asszociatív tár) segíti a címzést :
* translation look-aside buffer (TLB)
* A laptábla gyorsító tárban a várhatóan gyakran használt lapok címét tároljuk.
* A tár mérete itt sem elég nagy.
* A gyakorlatban a kombinált technikák (laptábla + gyorsítótár) használhatók.

**b, Hogyan működik?**

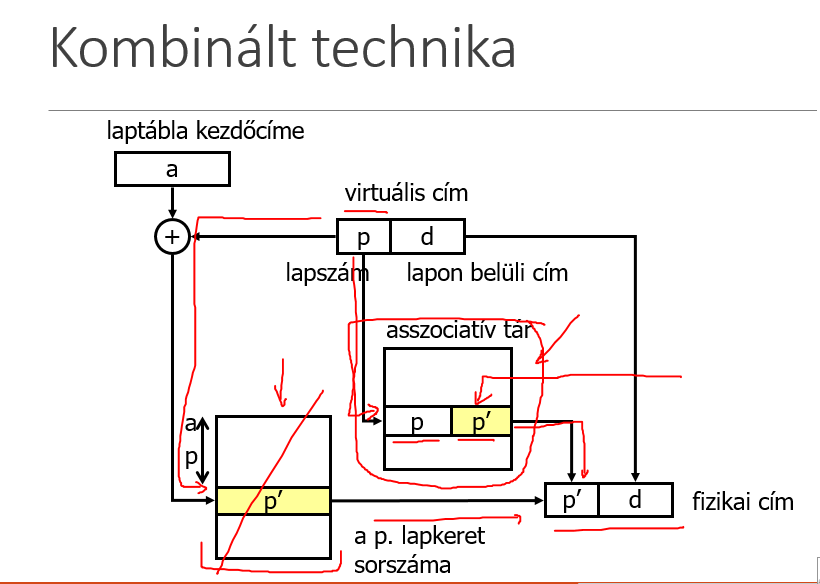
* Asszociatív tárban(TLB ben) eltároljuk a gyakran használt lapok címeit és ezekben keressük meg a logikai címekhez tartozó lapok kezdőcímeit. Gyorsan megtaláljuk mivel nagyon gyorsak a tárak, méreteik kicsik.

1. **Hogyan működik az asszociatív tár?**



* Feladat: logikai címből lapkeret sorszámhoz tartozó fizikai lapkeret sorszámot kell előállítani
* Párhuzamosan sok sok ágon keres
* logikai címet összehasonlítjuk az eltárolt logikai cím 1-gyel, ha ez a kettő megegyezik akkor megtaláltuk az ehhez a logikai címhez tartozó fizikai címet és egy multiplexeren keresztül megtaláltuk az eredmény
* ,ha nem találtunk akkor még egy másikon találhatunk hisz párhuzamosan folyik a keresés
* ,ha egyik ágon sincs találat akkor az asszociatív tárunkban nem találtuk meg ezt, nem tudunk eredményt szolgáltatni

1. **Hogyan működik a kombinált (laptábla+asszociatív tár) címképzés?**



* A laptáblában elindítjuk a keresést a virtuális címnek a lapszám részével
* A laptáblában megtalált lap keret sorszámát összefűzzük az eltolássa és megkapjuk a fizikai címet
* a laptáblában való keresés viszonylag lassú lehet
* de ezzel párhuzamosan elindíthatjuk az asszociatív tárban való keresést
* azaz a p lapszámot beengedjük az asszociatív tár bemenetére és megnézzük hogy van e találat
* ,ha van akkor azonnal tudjuk a p’ értékét és ezt máris szolgáltatjuk és abbahagyhatjuk a keresést
* ,ha nincs találat, nem volt az asszociatív tárban még eltárolva ez a p, p’ páros akkor ki kell keresni a laptáblából és ekkor beírhatjuk az asszociatív tárba
  + - ( a dián lévő szöveggel 🡪
    - A fizikai lapcím keresése egyszerre kezdődik mind az asszociatív tárban, mind a direkt laptáblában.
    - Ha az asszociatív tárban van találat, akkor a direkt keresés leáll.
    - Az asszociatív tárban lévő lapokat frissíteni kell, környezetváltás esetén az asszociatív laptáblát is cserélni kell.
    - Egy adott időszak alatt csak a teljes címtartomány kis része van kihasználva, így a találati arány elég magas lehet (80-99%).
    - itt a vége a dián lévő szövegnek :D)

1. **a, Hogyan működik a kombinált szegmens- és lapszervezés?**

MULTIX operációs rendszer címtranszformációja

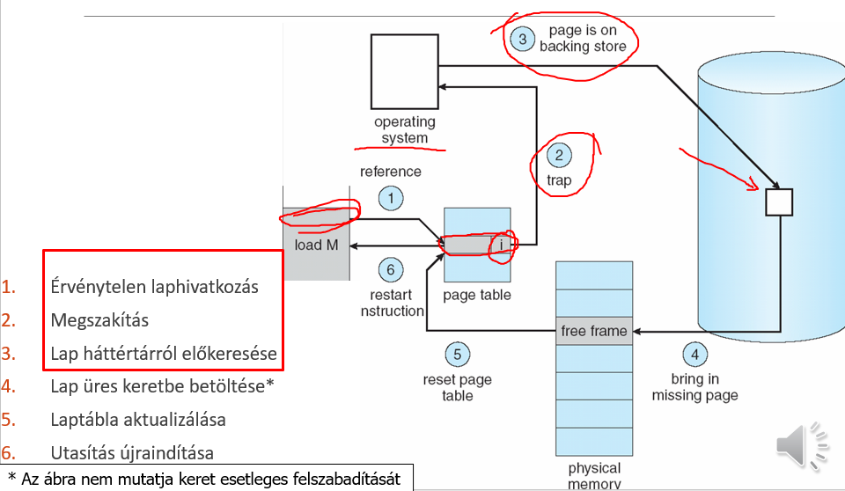


* lapcím és az eltolás a d ben van egyesítve
* szegmens tábla kezdőcíme + szegmens sorszáma = szegmens táblába való index és itt elolvassuk a szegmens hosszát és a szegmens kezdő címét ami most egy laptáblában a kezdő címe
* összehasonlítjuk a szegmens hosszát és a jelenlegi eltolás értéket
* ,ha a legális tartományban vagyunk akkor tovább megyünk 🡪 d
* ,ha nem akkor egy szegmens hibát okozunk
* szegmensen belüli eltolásunk nem más mint egy (p) lapcím és a lapon belüli (d’) eltolás
* laptábla kezdőcíme (a szegmens táblából) + (p) laptábla sorszám 🡪 így beletudunk címezni a laptáblánkba
* lapunk fizikai kezdő címe (f) ehhez hozzáfűzzük a (d’) eltolásunkat és megkapjuk a fizikai címet amivel betudunk címezni a fizikai memóriába

**b, Mik az előnyei az egyszerű szegmensszervezéshez és egyszerű lapszervezéshez képest?**

* Egyesíti a két technika előnyeit.
  + Lap szervezés: nincs külső tördelődés, nem kell a teljes szegmensnek a tárban lennie.
  + Szegmens szervezés: tükrözi a folyamat logikai társzerkezetét, hozzáférési jogosultság megoldható.
* Címtranszformáció: lényegében egy kétszintű táblakezelés.
  + Első szint: laptábla címeket tartalmazó szegmenstábla
  + Második szint: fizikai lapcímeket tartalmazó laptábla (szegmensenként egy).
* A cím három részre tagozódik (szegmenscím, lapcím, lapon belüli eltolás).
* Hozzáférési jogok ellenőrzése a szegmens szervezésének megfelelően történik.
* Osztott tárhasználat a szegmens szervezésének megfelelően történik.
* az első szinten egy olyan szegmens táblánk van ami mostlaptábla címeket tartalmaz
* a második szinten pedig olyan laptáblánk van amely fizikai címeket tartalmaz már
* a külső laptábla helyett egy szegmens táblát használunk
* 3 részre osztjuk a címet, -szegmens cím, -lap cím, -lapon belüli eltolás
* a kívül lévő szegmensben ellenőrizzük a hozzáférési jogosultságokat és ezzel minda két technika előnyeit ötvözni tudjuk

1. **Mi a virtuális tárkezelés?**

Olyan szervezési elvek, algoritmusok összessége, amelyek biztosítják, hogy a rendszer folyamatai logikai címtartományainak csak egy (A folyamat futásához szükséges) része legyen a központi tárban.

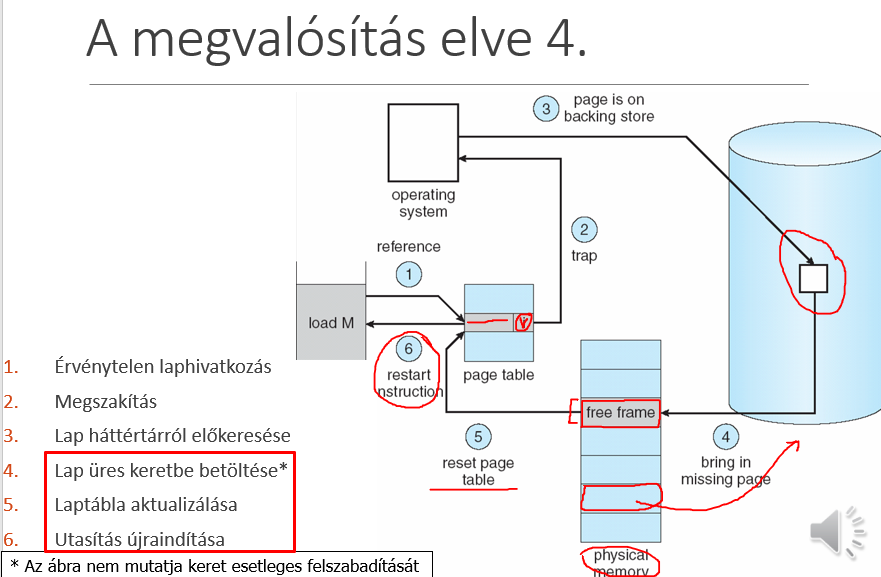
1. **Miért van szükség virtuális tárkezelésre?**

Azért van rá szükség, hogy automatikusan cserélődjön igényekhez megfelelően

1. **Ismertesse a virtuális tárkezelés menetét lapkezelés esetén.**

A programunk a Loadban található. Itt végrehajtunk egy olyan utasítást, amely egy memóriában jelenleg nem létező címre vonatkozik *(ábrában: i).* Olyan lapra mutat, amely nincs a memóriába. *(Érvényesség bit értéke i azaz invalid lap)* Nincs tehát a memóriába az a cím, amire hivatkozunk, így ez az utasítás nem hajtható végre. Az érvénytelen laphivatkozás egy megszakításhoz vezet *(ábrában 2 – trap),* az operációs rendszer ***(továbbiakban OR)*** átveszi a vezérlést. Megszakad tehát a folyamatunk futása, az OR fog futni. Az OR-nek elő kell keresnie a háttértárról a hiányzó lapot *(ábrában 3 – page is on backing store).* Ezt megtudja tenni, hiszen arra alkalmas hivatkozást fog találni a lap táblában. Megtaláltuk a háttértáron a hiányzó lapot. Ezt most az operatív tárba kell „átszállítani” *(ami még mindig jobban hangzik, mint az átfuvaroz).*

Alapvetően két eset lehetséges. Vagy van szabad laphely a központi tárban, vagy nincs. Amennyiben nincsen szabad laphelyünk, akkor az OR csinálni fog neki egyet. Ha van szabad laphely, akkor pedig beolvassuk a kívánt blokkot a memóriába, a szabad helyre. Ez rendkívül költséges művelet, hosszú időt vesz igénybe, ezért ezeknek a műveleteknek a számát elsősorban optimalizálni kell, másodsorban pedig szükséges, hogy a folyamatot az OR várakozó állapotba helyezze, más folyamatok kezdhetnek el futni és amikor a kívánt blokk a tárba bekerül, akkor a folyamat futásra kész lesz.



*(A folyamat az ábrában a bekarikázott négyzettől fog folytatódni)* Meg van tehát a háttérlapon a hiányzó lapot *(bekarikázott négyzet)* és ezt be kell hozni az operatív tárba *(ábrában physical memory)*. Ha van szabad laphely *(ábrában free frame)*, akkor egyszerű a dolgunk, erre a szabad laphelyre, behozzuk a háttéron lévő lapot és készen is vagyunk. Ha nincsen szabad laphely, akkor készíteni kell egyet. Kiválasztunk egy foglalt laphelyet *(ábrában a bekeretezett üres téglalap)*, ezt el kell távolítani és ki kell írni a háttértárra és hogyha ez mér felszabadult, akkor a felszabadult laphelyre be lehet hozni a kívánt lapunkat. Behoztuk a kívánt lapot, adminisztrálnunk kell a következő lépésben *(ábrában 5 – reset page table)*. A laptáblába be kell jegyeznünk annak a tényét, hogy ez a lap bent van és azt is be kell jegyeznünk, hogy hol található. *(ábrában: page table (i))* Az itt található invalid bitet *(i)* átállítjuk validra *(v)* és a valid mellé be fogjuk írni annak a lapkeretnek a sorszámát, ahol az aktuális keret helyet kapott. A laptábla elkészült, bekerült a memóriába a kívánt, hivatkozott utasítás/adat, az utasítást végre lehet hajtani.

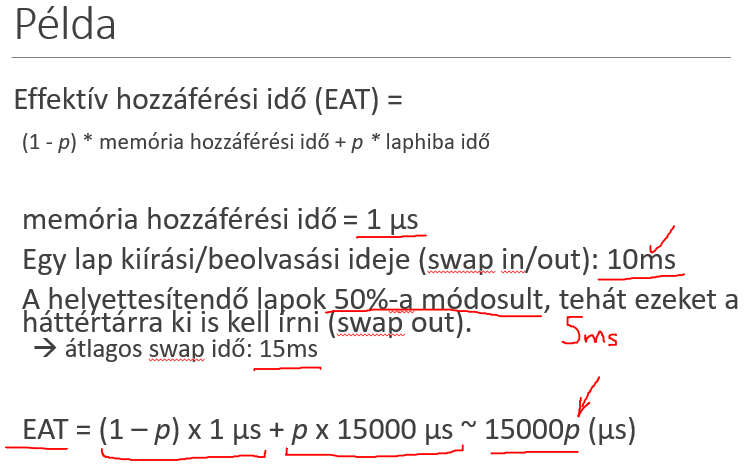
1. **Ismertesse az effektív hozzáférési idő jelentését és számítását.**

Effektív hozzáférési idő: Az az idő, amelybe egy átlagos lap elérése kerül.

Számítása: (1-p) \* memória hozzáférési idő + p \* laphiba idő

ahol p=A laphiba gyakorisága

* Mivel a laphiba kiszolgálása 5 nagyságrenddel nagyobb lehet, így *p*-nek kicsinek kell lennie.



**a, Mit jelent az igény szerinti lapozás?**

Mindig csak azt a lapot hozzuk be amelyre igény van

**b, Mik az előnyei és hátrányai?**

Előnye: Egyszerű a lapot kiválasztani, a tárba csak a biztosan szükséges lapok kerülnek be

Hátránya: Új lapokra való hivatkozás mindig laphibát okoz.

**a, Mit jelent az előretekintő lapozás (anticipatory paging)?**

Az OS megpróbálja kitalálni, hogy a folyamatnak a jövőben melyik lapokra lesz szüksége és azokat "szabad idejében" betölti.

Az előretekintő lapozás egyre népszerűbb.

**b, Mik az előnyei és hátrányai?**

Ha a jóslás gyors és pontos, akkor a futási sebesség jelentősen felgyorsul

Ha a döntés hibás, a felesleges lapok foglalják a tárat.

A memória ára jelentősen csökken, így a mérete nő, a hibás döntés ára (a felesleges tárfoglalás) egyre kisebb.

1. **Mi lenne az optimális lapcsere stratégia?**

Akkor lenne **optimális**, ha azt a lapot választaná ki, amelyre legtovább nem lesz szükség.

A lapcserét nagyban gyorsítja, ha a mentést csak akkor végezzük el, ha az a lap a betöltés óta módosult. A hardver minden lap mellett nyilvántartja, hogy a lapra írtak-e a betöltés óta (*modified* vagy *dirty* bit).

1. **Soroljon fel megvalósítható lapcsere stratégiákat.**

Algoritmusok:

* Véletlen kiválasztás
* Legrégebbi lap (FIFO)
* Újabb esély
* Óra algoritmus
* Legrégebben nem használt lap
* Legkevésbé használt lap
* Mostanában nem használt lap

1. **Ismertesse a Legrégebbi lap (FIFO) lapcsere stratégia működését, megvalósítását.**

A tárban lévő legrégebbi lapot cseréli le.

Megvalósítása egyszerű FIFO listával történik.

Hibája:

Olyan lapot is kitesz, amelyet gyakran használnak

Felléphet egy érdekes jelenség. Ha növeljük a folyamatokhoz tartozó lapok számát, a laphibák száma esetenként nem csökken, hanem nő! (Bélády-anomália)



**1. eset: 3 lapkeret van**



* behozzuk a 3 lapot és betöltjük őket az üres lapkeretbe
* tele van a 3 lapkeret
* 4. lapkeretre való hivatkozás: a legrégebben használt lapkeretet kidobjuk (1-est) behozzuk a 4 est
* 1. lapkeretre való hivatkozás: kidobjuk a 2 est, behozzuk az 1 est
* 2. lapkeretre való hivatkozás: kidobjuk a 3 mast , behozzuka a 2 est
* 5. lapkeretre való hivatkozás: kidobjuk a 4 est beírjuk az 5 öst
* 1.lapkeretre való hivatkozás: bent van a memóriában
* 2. lapkeretre való hivatkozás: bent van a memóriában
* 3. lapkeretre való hivatkozás: kidobjuk az 1 est behozzuk a 3 mast
* 4. lapkeretre való hivatkozás: kidobjuk a 2 est behozzuk a 4 est
* 5. lapkeretre való hivatkozás: memóriában van
* **összesen 9db laphiba**

**2, eset, 4 lapkeret van**

**Laphivatkozások: 1,2,3,4,5,1,2,3,4,5**

* nagyobb a memória 🡪 4 lapkerettel gazdálkodunk 🡪 ügyesebb futás
* betöltjük a 4 lapkeretet
* 5. lapkeretre való hivatkozás: kidobjuk az 1 est, behozzuk az 5 öst
* 1. lapkeretre való hivatkozás: kidobjuk a 2 est, behozzuk az 1 est
* 2. lapkeretre való hivatkozás: kidobjuk a 3 mast, behozzuk a 2 est
* 3. lapkeretre való hivatkozás: kidobjuk a 4 est, behozzuk a 3 mast
* 4. lapkeretre való hivatkozás: kidobjuk az 5 öst, behozzuk a 4 est
* 5. lapkeretre való hivatkozás: kidobjuk az 1 est behozzuk az 5 öst
* **összesen 10db laphiba 🡪 4 alap(1,2,3,4) + 6 laphiba**
* több memórája volt és mégis lassabban futott 🡪 **Béládi anomália**

1. **Mi a Bélády-anomália?**

Nagyobb lapkereteknél több hiba tapasztalható.

1. **Ismertesse az Újabb esély (Second Chance) lapcsere stratégia működését, megvalósítását.**

FIFO-t javítja az algoritmus. Minden laphoz „R” hivatkozás bit. Kezdetben a hivatkozás biteket töröljük, amikor egy lapra hivatkozunk akkor 1-re állítjuk. Amikor lapcsere történik, a következő módon járunk el. Elkezdünk keresni a sor elején, ha a sor elei hivatkozás bit=0, akkor kicseréljük. Ha pedig 1-es, akkor (kap még egy esélyt) a bitet nullázzuk, berakjuk a lapot a FIFO végére és tovább megyünk. Ugyan így végig megyünk az összes lapon. Ezzel a közel móltban hivatkozott lapok kapnak még egy esélyt.

1. **Ismertesse az Óra algoritmus működését. Melyik stratégiát valósítja meg?**

Az újabb esély gyakori implementációja az Óra algoritmus. A lapokat egy körkörös láncba fűzzük össze a betöltés sorrendjébe és a korábbi stratégia szerint vizsgáljuk az R bitet.

„Feláldozás- a 0 R bittel rendelkező lapokat elhagyjuk”

1. **Ismertesse a Legrégebben nem használt lap (Least Recently Used) stratégia működését, megvalósítását.**

Az adatszerkezetet időbélyeggel látjuk el, ami megmutatja mikor hivatkoztunk rá utoljára. Ezalapján az idők szerint rendezzük a lapokat.

Közelítés az optimális algorimusra.

Láncolt listás implementációnál nem kell időbélyeg. A frissen hivatkozott/ behozott lapokat a lista elejére tesszük.

1. **Ismertesse a Legkevésbé használt lap (Least Frequently Used) stratégia működését, megvalósítását.**

Egy számlálót hozzárakunk. Tehát van egy Referencia bitet illetve egy CT nevű számláló minden egyes referencia laphoz. Laphivatkozásnál az R-t billentjük be, ettől függetlenül (ritkábban mint ahogy lapokat hivatkozunk) egy számláló mechanizmus elindul mely körbe megy a lapokon. Amikor a lapnál a R bit =1 akkor megnöveli a lap számlálóját és kitörli az R bitet (gondolom nullázásra gondol). Ha R=0 akkor nem növeli a számlálót. Amikor szükség van egy lapcserére *(„beáldozás”)* akkor azt vélasztjuk, ahol legkisebb a számláló értéke.

Hátránya:

- Ha van egy régebben sokat használt lap a memóriában amelynek magas a számlálója, sokáig bent marad a memóriában. Öregítés mechanizmussal időről-időre csökkentjük.

- A frissen betöltött lapnak nincs esélye a számlálót megnövelni, ezért PageLocking mechanizmussal egy ideig be szokták *fagyasztani*.

1. **Ismertesse a Mostanában nem használt lapok (Not Recently Used) stratégia működését, megvalósítását.**

Leggyakrabban használt lapok a memóriában.

Implementálás: Lap hivatkozásakor *R* bebillentése

Periodikusan minden lapnál egy számlálót növel, ha *R*=1,majd törli az *R*-t.

Lapcsere esetén a legkisebb számláló-értékű lapot dobja ki.

Hátrány:

A valamikor sokat használt lapok a memóriában maradnak (öregítéssel lehet segíteni ezen)

A frissen betöltött lapok könnyen kiesnek (frissen behozott lapok befagyasztása, *page locking*)

Hivatkozott (*R*) és módosított (*M*) bitek használata.

OS időközönként törli az *R* bitet, *M* bitet viszont őrizni kell (lap törlése információ veszteséghez vezetne).

Négy prioritás

nem hivatkozott, nem módosult

nem hivatkozott, módosított

hivatkozott, nem módosult

hivatkozott, módosult

A módszer a legkisebb prioritásból választ véletlenszerűen.

1. **Mit jelent a vergődés (thrashing)?**

Ha egy folyamat vagy rendszer több időt tölt lapozással, mint amennyit hasznosan dolgozik.

Oka:

Folyamat: kevés lap van a tárban, gyakran hivatkozik a háttértáron lévőkre

Rendszer: Túl sok a folyamat, ezek egymás elől lopkodják el a lapokat, mindegyik folyamat vergődni kezd.

Elkerülése: a tárban lévő folyamatoknak biztosítani kell a futáshoz szükséges optimális számú lapot.

1. **Mit jelent, hogy a programok végrehajtása térben és időben lokális?**

Statisztikai tulajdonság: a folyamatok egy időintervallumban csak címtartományuk szűk részét használják.

időbeli:

a hivatkozott címre a közeljövőben nagy valószínűséggel újra hivatkozni fog (ciklusok, eljárások, verem változók, stb.)

térbeli:

hivatkozott címek egymás melletti címre történnek (soros kód, tömbkezelés).

1. **Mit jelent a munkahalmaz?**

A munkahalmaz (*working set*, WS) az elmúlt Δ időben (munkahalmaz ablak) hivatkozott lapok halmaza.

A lokalitás miatt a folyamatnak nagy valószínűséggel ezekre a lapokra lesz szüksége a közeljövőben.

Munkahalmaz mérete folyamatonként és időben is változik (lásd a következő példát).

OS célja minden aktív folyamat számára biztosítani a munkahalmazt.

Pontos méréséhez bonyolult hardver kellene, ezért fix időintervallumonkénti mintavételezéssel oldják meg.

1. **Munkahalmaz modellel megfogalmazva mikor lép fel egy rendszerben vergődés?**

WSSi: az i-ik folyamat munkahalmazának mérete

A példában: WSS(*t1*)=5, WSS(*t2*)=2

A rendszer teljes lapigénye: D=Σ WSSi

A rendszerbeli lapkeretek száma: m

Vergődés akkor lép fel, ha D>m

(vagyis a lokalitások összes mérete nagyobb, mint a rendelkezésre álló memória mérete)

1. **Ismertesse a lapok allokálásának módját a laphiba gyakoriság figyelésével.**

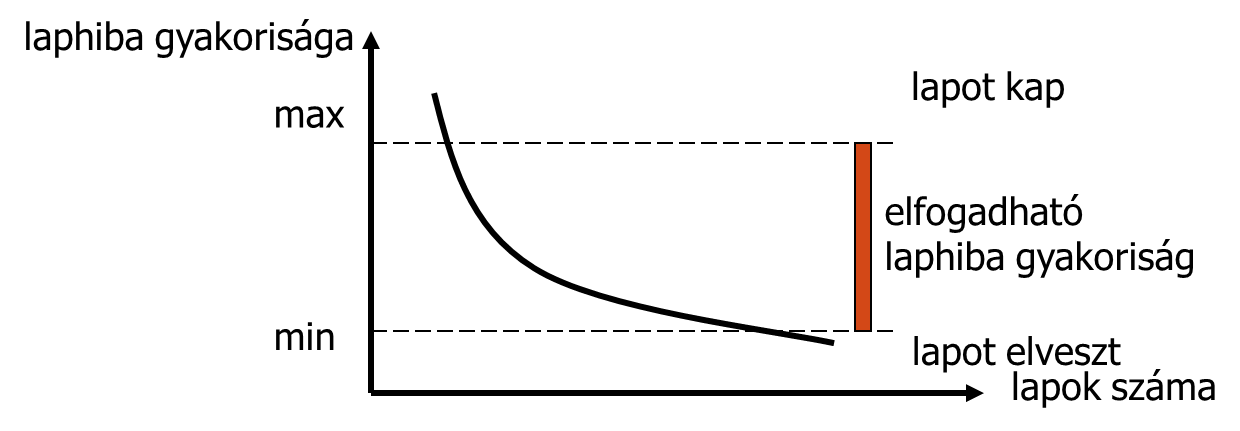
Két egyszerű tényt kell megvizsgálni:

1.Ha egy folyamatnak sok a laphibája, akkor az azért lehet, mert a folyamat számára kevés lapkeret áll rendelkezésére. Ebben az esetben érdemes újabb lapot adni ennek a folyamatnak.

2.Ha a folyamat működése során kevés laphibát figyelünk meg, akkor valószínűleg az történt, hogy a folyamat túl sok lapkeretet bitorol (törvénytelenül birtokol, kevesebb is elég lenne neki), és ezt mások rovására teszi, más folyamatnak kevesebb jut.

Az OS nem méreget, becsülget, hanem a laphibák gyakoriságát vizsgálja.

* Ha ezek száma elért egy maximum szintet, akkor az OS lapot ad a folyamatnak.
* Fordított esetben, ha elér egy minimum szintet, akkor az elvesz lapot a folyamattól.
* Csak akkor indítunk el újabb folyamatot, ha van elég szabad lap.



1. **Milyen előnyei és hátrányai vannak a kis, illetve nagy lapméretnek? A gyakorlatban mekkora lapméretet alkalmazunk?**

Az általában használt lapméret tartomány tipikusan 4kByte és 4Mbyte közötti.

1. Mi történi akkor, ha a lapméret túl nagy?
   * Ha túl nagy, akkor kisebb laptávra elég.
   * Ha nő a lapméret, akkor a perifériás átviteli idő csökkeni fog 🡪 Egy nagyobb blokkot gyorsabb átvinni, mint több kisebbet.
2. Mi történi akkor, ha a lapméret túl kicsi?
   * Ha túl kicsi, akkor sokkal jobban érvényre jut a lokalitás 🡪 Kisebb munkahalmazokat lehet kezelni.
   * Kisebb a belső tördelődés

Valahol a kettő között kell kompromisszumot kötni.

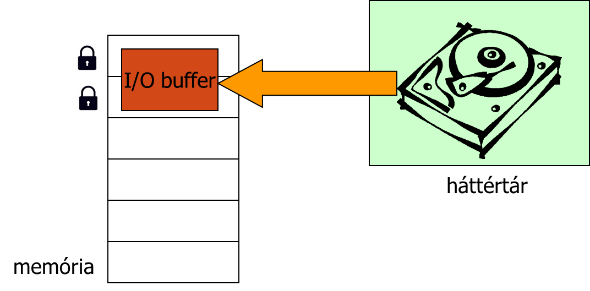
1. **Mit jelent az asszociációs memória fedése? Mondjon példát arra, amikor a lapok tárba fagyasztása szükséges.**

Az asszociációs memória (TLB) fedése azt mutatja meg, hogy mekkora az a memóriatartomány, amit ezen keresztül el tudunk érni.

Az asszociációs memória fedése tehát nem más, mint a lapméret és az asszociációs memóriában létező bejegyzések számának szorzata.

Ha van egy perifériás műveletünk, ami egy céltartományt érint, akkor érdemes befagyasztani azt a címtartományt a művelet idejére, ekkor beszélünk a lapok tárba fagyasztásakor.

Példa (I/O lapok befagyasztása): Ha egy lapot behoznak, akkor azt az első hivatkozásig érdemes befagyasztani (nehogy áldozatul essék egy memóriafelszabadításnak). Ezt úgy valósítjuk meg, hogy minden egyes laphoz rendelünk egy lock bitet.

Egy perifériális művelet, amit a háttértárról akarunk beolvasni a pufferbe. A puffer egy bizonyos lapon/lapokon helyezkedik el, és nyilvánvalóan a művelet idejére ezt a lapot (vagy ezeket a lapokat) be kell fagyasztani, és ekkor nem fog történni probléma a háttértárról való beolvasás közben.



A lakattal ellátott terület van befagyasztva.

1. **Mit jelent a Copy on Write?**

Amikor elindítanak egy folyamatot, akkor a gyermekfolyamat megörökli a szülő folyamatnak a memóriaterületét. Logikailag megkapja azt a területet, azaz ő is használhatja.

Logikailag átmásolják a gyermek számára a szülő memóriaterületét. A valóságban viszont ez nem így működik. A valóságban nem másolnak semmit kezdetben, hanem csak megjegyzik, hogy ez a közös memóriaterület.

Ha bármelyik lapot akár a szülő, akár a gyermek módosítja, akkor ebben a pillanatban történik meg a lap duplikálása. Ekkor készítenek egy másolatot a lapról, és az egyik folyamat a módosított lapot használja, míg a másik a módosítás előttit.

Tehát csak akkor van másolás, hogyha egy írási művelet történik az adott lapon 🡪 ezzel megspórolunk egy csomó memóriamásolást.

(A linux rendszereken elég hatékonyan működik.)

1. **Miért van szükség háttértárra egy számítógépes rendszerben?**

Két megközelítés:

* a, központi tár drága és kicsi a tárolókapacitása
* b, a kikapcsolással az információk elvesznek a központi tárban

1. **Soroljon fel legalább 5 lehetséges háttértár-típust.**

Mágneses, Mágnesszalag, Magneto-optikai lemez, Optikai lemez, Flash (SSD)  
*[Holografikus tárolás és MEMS tárolók – jövő?]*

Legelterjedtebbek a mágneslemezek és az SSD.

1. **Ismertesse a mágneslemezes háttértár fizikai felépítését és működését.**

Működése: Egy mágneslemezben egy mágneses-réteg található, mely vékony, műanyag/fém alapra van felvéve. Két fajta kialakítása létezik: merevlemez (Winchester) és hajlékony lemez (Floppy). Működési elve azonos, a mágneses réteggel bevont lemezek „pörögnek” és ezek felett egy fej található, mely a forgólemez felett kialakuló levegőrétegen „fut”, mely fenntartja a fejet, így nem ér hozzá a lemezhez, viszont amennyiben a fej hozzáér a lemezhez az lemezkárosodást okoz, illetve adatvesztést. Kapacitás: Hajlékonylemez ~1MB; Merevlemez: ~500MB - 1TB vagy nagyobb.

1. **Mit jelen a sáv, cilinder, szektor egy mágneslemez esetén?**

Sáv: A (track) a lemezterület (gyűrű) azon része, amelyet a fej elmozdulás nélkül egy fordulat alatt elér.  
Cilinder: (cylinder) az összes fej alatti sáv  
Szektor: (sector), a sáv azonos méretű blokkokra osztva. Az információátvitel legkisebb egysége: a lemezvezérlő egy teljes szektort olvas vagy ír.

1. **Sorolja fel az adatátvitel idejét meghatározó kiszolgálási időket mágneslemezes háttértáraknál.**

Fejmozgási idő  
Elfordulási idő  
Az információ átviteli ideje

1. **Soroljon fel 3 ütemező algoritmust lemezegységek fejmozgási idejének optimalizálására.**

Legrövidebb fejmozgási idő  
SCAN  
N-SCAN

1. **Ismertesse a legrövidebb fejmozgási idő-alapú ütemezés működését.**

A következő kiszolgált kérés az lesz, amelyiknek a legközelebb van a cilindere.

1. **Ismertesse a SCAN ütemezés működését.**

A fej egy irányban mozog, ezen sorban szolgálja ki a kéréseket, majd irányváltás

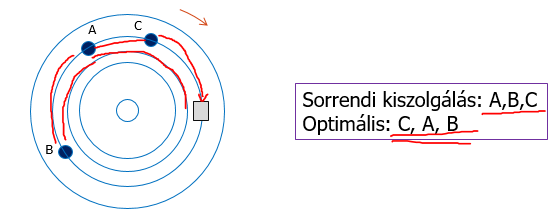
1. **Ismertesse az N-SCAN ütemezés működését.**

SCAN-hez hasonló müködés, de egyszerre csak N kérést szolgál ki.

1. **Hogyan lehet az elfordulási időt optimalizálni?**

Cél: a seek time csökkentése

Megoldás: Az egy cilinderen belüli kérések a lemez aktuális pozíciójának, valamint a szektorok sorrendjének ismeretében a kiszolgálás előtt sorba rendezhetők.



1. **Soroljon fel legalább 5 módszert arra, hogy a lemezegység teljesítményét növeljük. Magyarázza el ezek működését.**

* Lemezterület tömörítése (Disc Compaction)
  + lokalitáson alapul
  + Az egymáshoz tartozó blokkokat a lemezen is egymás mellé tesszük. Időnként egy rendezőprogrammal tömöríteni kell a háttértárat.
* A gyakran szükséges adatok a lemez közepén
* Gyakran használt adatok több példányban
  + több cilinderen is tároljuk
  + így minden fejálláshoz elég közel van
  + Csak nagyon ritkán változó adatokkal érdemes csinálni:
    - adatok konzisztenciája
    - kölcsönös kizárás
* Több blokk átvitele egyszerre
  + Az idő nagy része a fejmozgással telik 🡪 ha már ott vagyunk, vigyünk át minél több blokkot
* Blokkok átmeneti tárolása:

Disc Cache (periférián lévő, központi tár)

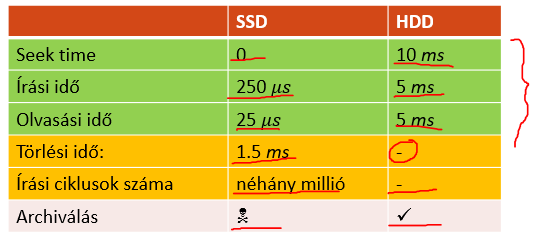
* + write through, egyidejűleg kiírjuk a lemezre
  + copy back, csak akkor írjuk ki ha az átmeneti tárra szükségünk van (teljesítmény növelő, de meghibásodás esetén a módosítások nem kerülnek a lemezre)
* Adattömörítési eljárások használata (Data Compression)
  + Az lemezen az információ tömörített formában van. A ki-be tömörítést a perifériakezelő vagy célhardver végzi.
  + Átvitel sebessége nő, adatvesztés valószínűsége nagyobb

1. **Ismertesse az SSD háttértár fizikai felépítését és működését.**

Nincs benne mozgó alkatrész

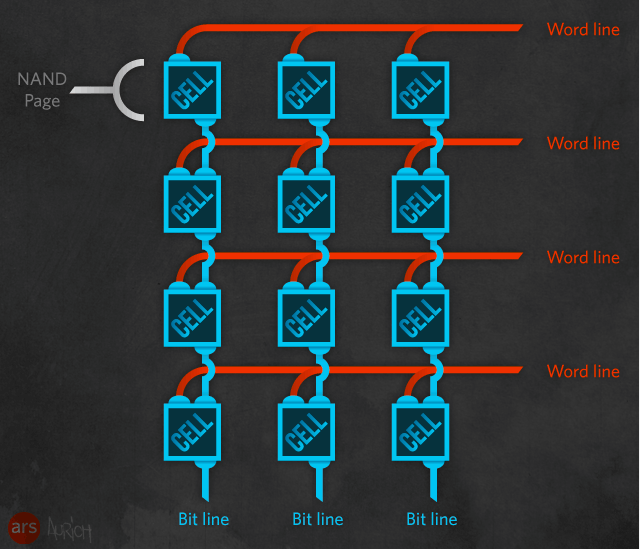
Sokkal gyosabb, mint a HDD

Összehasonlítás

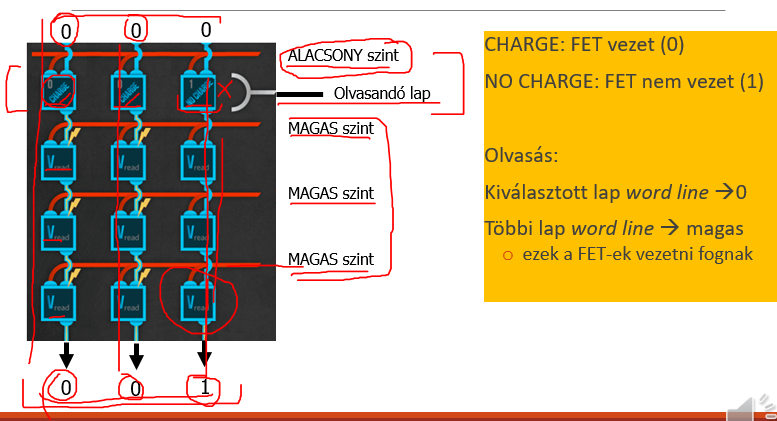


A memóriacellákat rácsos szerkezetbe rendezik

* + - Egy sor: lap
    - A teljes rács: blokk (pl. 256 lap)
  + Törlés: a teljes blokk
  + olvasás: 1 lap (4 lap az ábrán)
  + olvasni bármikor tudunk lapokat
  + írás: 1 lap (csak törölt lap írható!)
  + 🡪 SSD kontroller
  + 1 memóriacella tárolhat:
    - 1 bitet (SLC - *Single Level Cell*) van e benne töltés vagy nincs
    - 2 bitet (MLC - *Multi Level Cell*)
    - 3 bitet (TLC – *Triple Level Cell*)
    - 4 bitet (QLC – *Quad Level Cell*)



1. **Hogyan történik az SSD-n az adatok olvasása?**



SSD-nél a tárolt töltés jelölése a 0 bit

függőlegesen össze vannak fűzve a FET ek és egy ilyen vonal akkor tud vezetni, ha valamennyi FET vezető állapotban van, ha bármelyik zárva van akkor nem vezetnek

bal oldali és a középsőben van míg a jobb oldaliban nincsen töltés, a jobb oldali nem vezet míg a többi igen

az összes többi lap vezetékre magas szintet kapcsolunk ez azt jelenti, hogy a vezérlő elektródára töltéseket viszünk fel így a FET-ek vezető állapotba fognak kerülni

a felső sorban az olvasandó lapon, amelynek a lapvezetékét alacsony szintre állítottuk a FET-ek akkor fognak vezetni, ha töltés van rajtuk nem kényszerítettünk rájuk magas szintet ezért ezek vezető képessége a tárolt töltéstől fog függeni

az összes többi FET-nek magas szintet kényszerítünk a vezérlő elektródájára tehát minden egyé FET vezetni fog a rendszerben

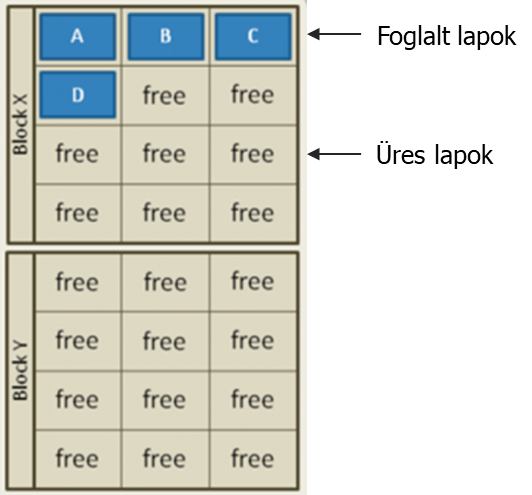
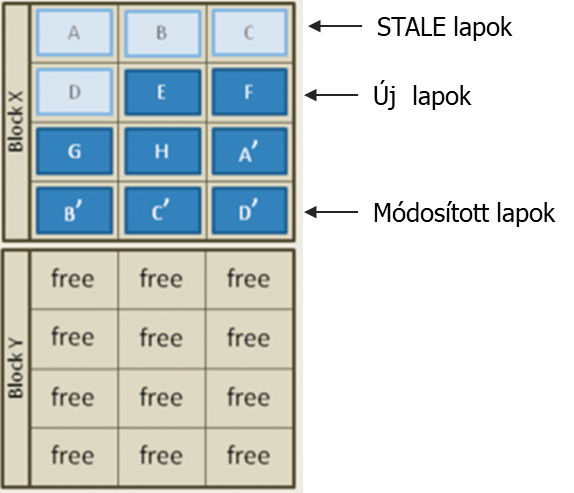
bal oldali soron mindegyik vezet mivel 0 át tárolunk és minden más sor magas szintű, vagyis a felső 0 szint az alsó kiolvasó vezetékszakaszra átmásolódik

ugyan ez történik a középső résznél

a jobb szélső esetben a tárolt töltés 1 es, itt nem tárolunk töltést ezért ez nem vezet, itt egy szakadást találunk, itt 1 es lesz a kimenet

1. **Hogyan történik az SSD-n az adatok írása?**

Adatot írni csak is kizárólag teljes lapon lehet, üres helyre. Első lépésként be kell olvasni a lapot a memóriába (SSD vezérlő memóriája). Ez után a memóriában módosítjuk a lapot, és ezt a lapot vissza kell írni egy üres helyre. Ha már nincs üres hely, akkor az adott (a legrégebbi) lapot töröljük, pontosabban STALE állapotba tesszük.

Ha egy blokkon belül maradt pl. 2 üres lapnyi hely és mi 3 lapnyi adatot szeretnénk tenni, akkor a blokkon egy lap érvénytelenné (STALE) válik. Ha fel van töltve egy teljes blokknyi adat, akkor ezt a memóriából az SSD területére írjuk át.

1. **Mit jelent a wear leveling, miért van rá szükség?**

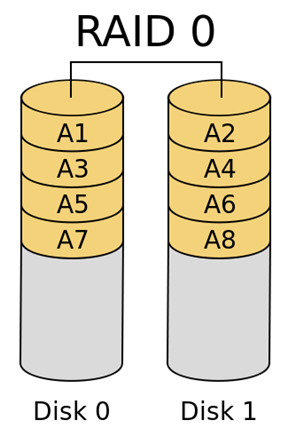
Az SSD korlátozott élettartamú. Minden törlés és minden írás művelet elhasználja a háttértárolót. Ügyelni kell arra, hogy egyenletesen oszoljon el a terhelés a teljes eszközön (az eszköz minden blokkját használjuk), így megnövelhető az SSD élettartama.

Ezt a terheléselosztást nevezik wear leveling-nek. Ez az SSD kontroller feladata, ezt automatikusan elvégzi.

1. **Milyen két alapelve mentén működnek a RAID tárolók?**

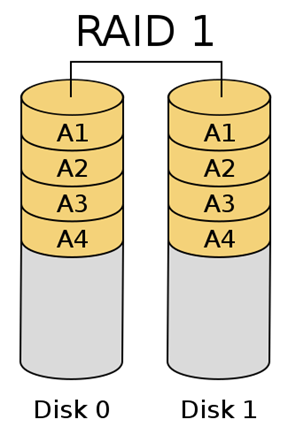
A RAID célja, hogy egy alkalmas elrendezéssel lehetséges az adatátvitel sebességét növelni, másrészt az adatbiztonság növelni.

Alapelvek:

1. Rendelkezésre áll két azonos lemezegység, amit duplikálnak. A cél az, hogy mindkettőn ugyanaz az adat szerepeljen, akkor is, ha írni akarnak egy lemezegységen. Ezt a folyamatot disc shadowing-nak, vagy magyarosan tükrözésnek nevezik.   
   Ha az egyik lemezegység meghibásodik, akkor a másik még rendelkezésre áll a továbbiakban (ebből ugyanúgy lehet másolatot, tükrözést elvégezni). Az elérési sebesség ugyan nem változik, de megbízható és hibavédett.
2. Vegyünk alapul több háttértárat. Az adatbájtok bitjei mind külön-külön bájtokra kerüljenek (pl. 8 bithez 8 darab háttértárat rendelünk). Írás, olvasás esetén ezeket a háttértárakat párhuzamosan tudjuk használni.   
   Előnye, hogy a sebesség a nyolcszorosára növekszik (a példa alapján), 8-szor annyi adatot lehet egy bizonyos időegység alatt mozgatni, viszont ez nem annyira megbízható az előző alapelvhez képest.
3. **Magyarázza el a RAID0-RAID6 tárolók működését.**

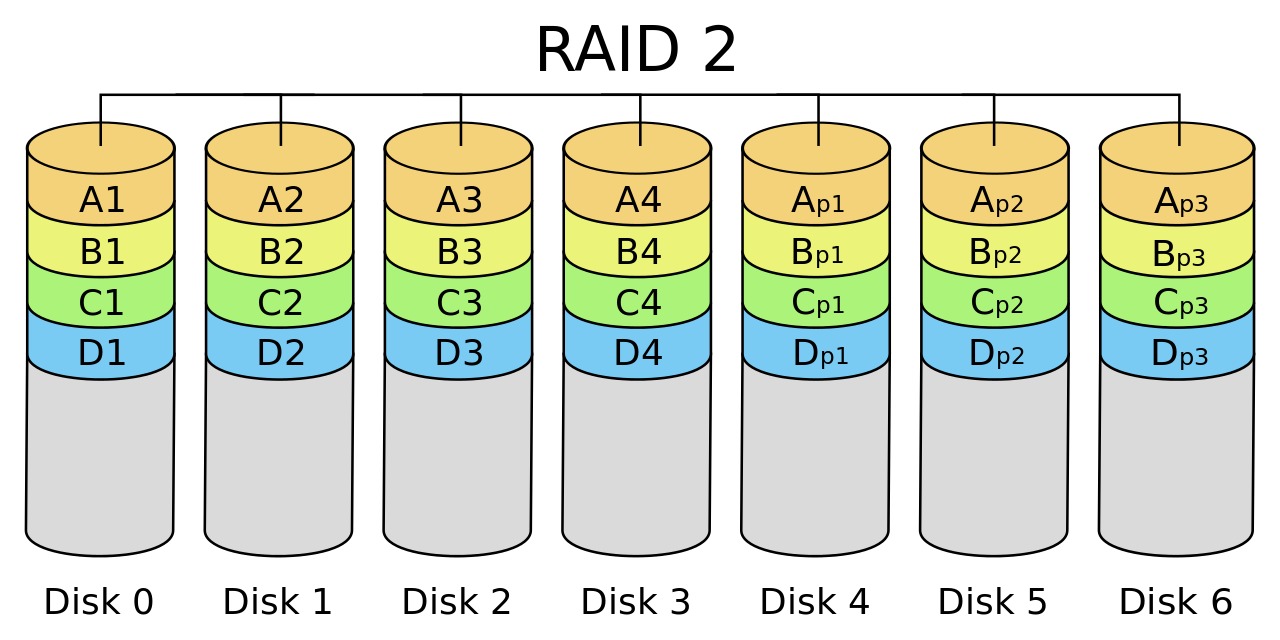
**RAID0 - Non-redundant stripping**

Egymás utáni blokkokat külön háttértárakon helyeznek el. Itt nincs tükrözés. Az adatátviteli sebesség megnő ebben az esetben, viszont, ha valamelyik háttértár megsérül, akkor azt már nem lehet visszahozni.

**RAID1 - Disk mirroring**Minden egyes háttértárat megtükröznek (mirroring). Hiba esetén a tükrön lévő adat használható, a hibás egység erről helyreállítható. Sokkal nagyobb biztonság, de a sebesség nem növekszik.

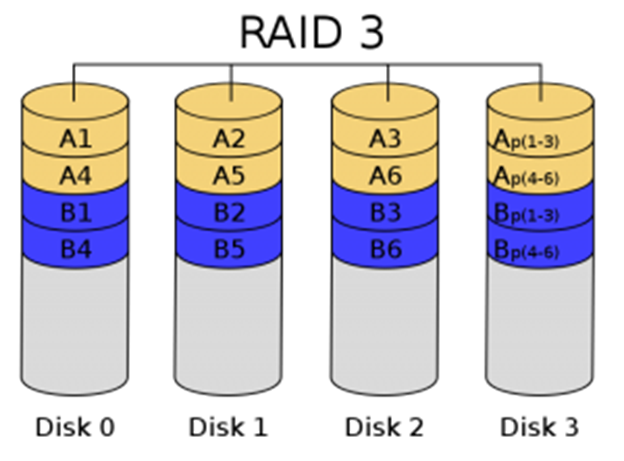
**RAID2 - Memory-style error connection**

Az alapötletet a memória paritásvédelemből veszik. Az adatokat bitenként szervezik, minden bájthoz egy paritásrészt rendelnek. A képen 4 darab adatbithez (A1, A2, B3) 3 darab paritásbitet (Bp3, Dp2) rendelnek hozzá. Ha az egyik háttértár megsérül, akkor a paritásbitek (Ap1, Ap2 stb.) segítségével visszaállíthatóak az adott háttértár adatai.

Előnye a nagy sávszélesség és nagy biztonság kisebb redundanciával, mint RAID 1 esetén.

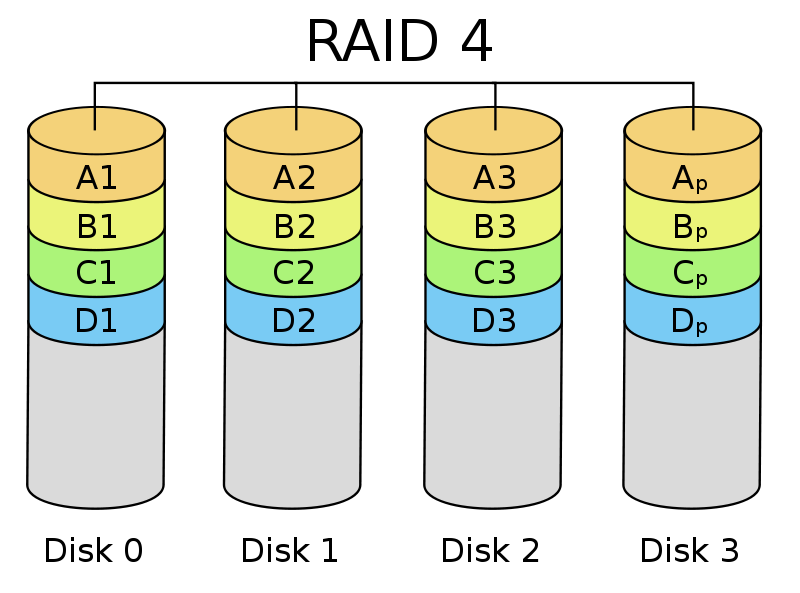
**RAID3 - Bit-interleaved parity**

A háttértáron tudjuk a hibát detektálni, látni fogják, ha az egyik meghibásodott. Ebben az esetben elegendő egy paritásbiteket tartalmazó háttértár. Ha például megsérül az A5-ös adat, akkor tudják, hogy melyik adat hibásodott meg, ezáltal választják ki a paritásbiteket tartalmazó háttértárból a megfelelő adatot (Ap(4-6)).

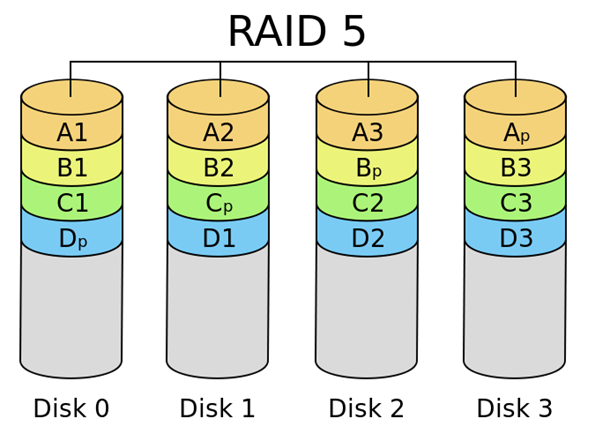
Előnye a nagy sávszélesség és nagy biztonság nagyon kis redundanciával (egyetlen extra háttértárral).

**RAID4 - Block-interleaved parity**

Az adatok tárolását nem bitenként, hanem blokkonként szervezik, ami azt jelenti, hogy az egyes adategységek részei a különböző tárolókon helyezkednek el. Pl. az A1 és az A2 a RAID4 esetén már különböző adattárolón található meg. A paritás tarolónál ugyan ez a helyzet.

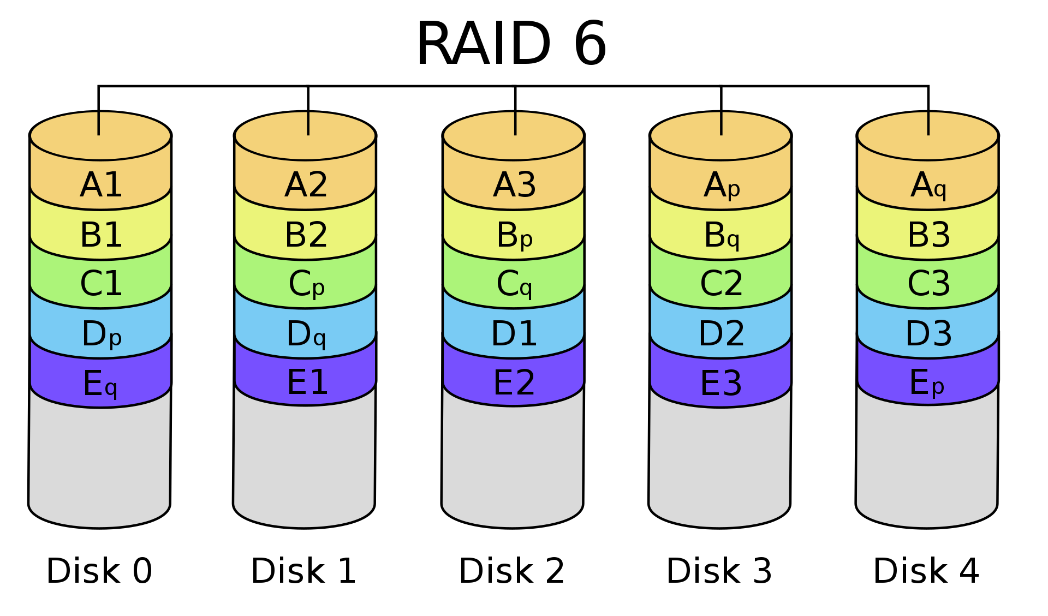
Adatvesztés esetén a partíciós meghajtóról tudjuk visszaállítani az adatokat. Biztonságos, nagy adathalmaz kezelésénél nő az átviteli sávszélesség is.

**RAID5 - Block-interleaved distributed parity**

Az elv az, hogy ne egy külön háttértáron tároljuk a paritásbiteket, hanem helyette elosztják a paritásblokkokat a háttértárolókon. A háttértárak ugyanolyan szerepűek, mindegyik tartalmaz adatblokkokat és paritásblokkokat. A RAID4-hez képest a terhelés kiegyenlítődik.

**RAID6 - P+Q redundancy scheme**

Hasonló elve van, mint RAID5-nek, de több (a képen 2) paritás blokkot tartalmaz. Nagy előnye, hogy véd a többszörös meghibásodás ellen is.



**1. Mi az állomány?**

* Az állomány (file) logikailag összetartozó információk gyűjteménye. A fájl létrehozója dönti el, hogy mi van logikailag összefüggve. Mindegyik file-nak egyedi neve van. Név alatt azt értjük, hogy a teljes elérési út micsoda az adott állományhoz.
* Magas szintű absztrakció, kényelmes kezelni. Csak azokat a tulajdonságokat kell kezelni, amelyeket ténylegesen a felhasználókat érinti. Mindenfajta alacsony szintű részlet el van rejtve az OS által.
* Egy állományt (pl. a neve alapján) meg lehet találni, lehet olvasni, vagy írni.

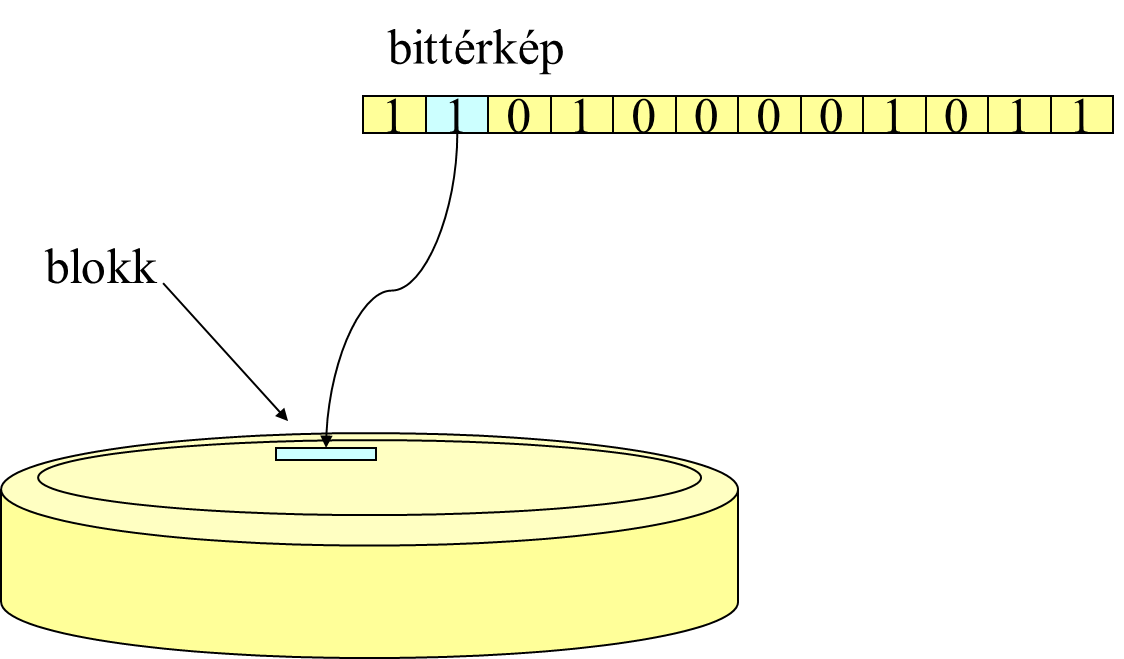
**2. Mi a könyvtár?**

* A könyvtár (pl. mappa) állományok csoportosítására szolgál. A könyvtár tartalmát egyedi katalógus írja le.

**3. Soroljon fel 4 módszert a lemezterületen lévő szabad blokkok nyilvántartására.**

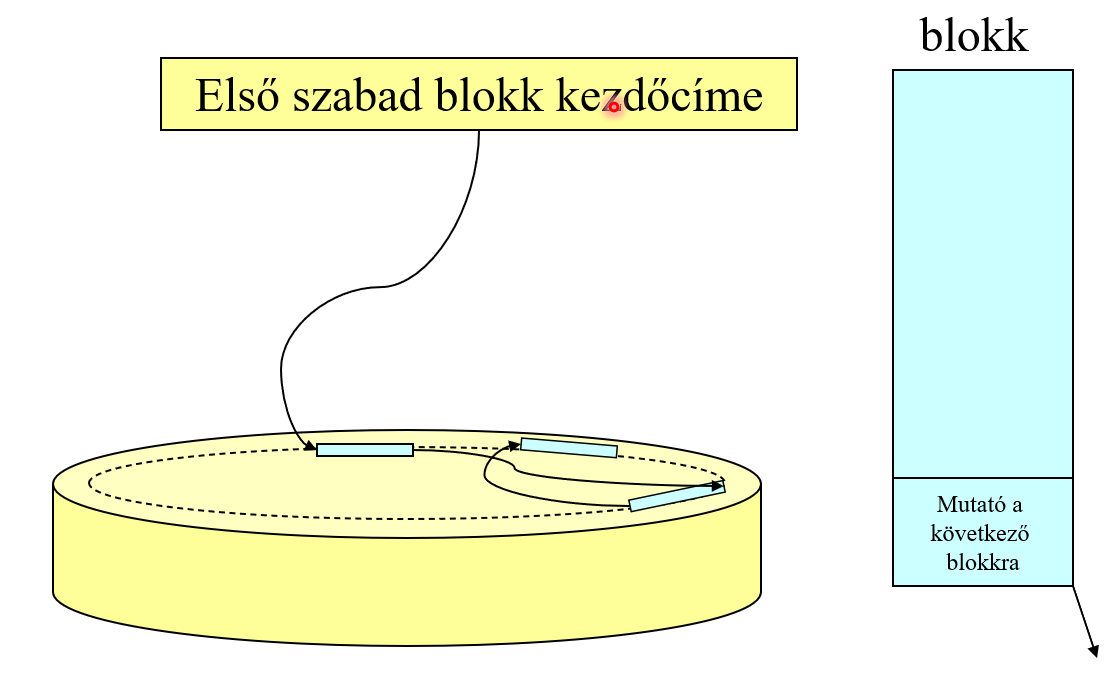
* Bittérkép
* Láncolt lista
* Szabad helyek csoportjainak listája
* Egybefüggő szabad terület tárolása

**4. Ismertesse a szabad blokkok nyilvántartását bittérkép segítségével.**

* A háttértáron rengeteg blokk helyezkedik el, ezekhez rendelnek hozzá egyenként egy-egy bitet. Ezek a bitek fogják jelezni, hogy az aktuális blokk szabad, vagy már foglalt. Ezt nevezzük bittérképnek. Ezt el kell tárolni valahol (pl. háttértáron). Amikor használják ezt az ezközt, akkor célszerű a memóriában eltárolni a gyorsaság végett.
* A képen látható lemezen sok-sok blokk található. Ezek közül egy blokkot látni kiemelve. Minden blokkhoz tartozik a bittérképben egy-egy darab bit, jelen esetben egy 1-es, amely mutatja, hogy ez a blokk már foglalt.

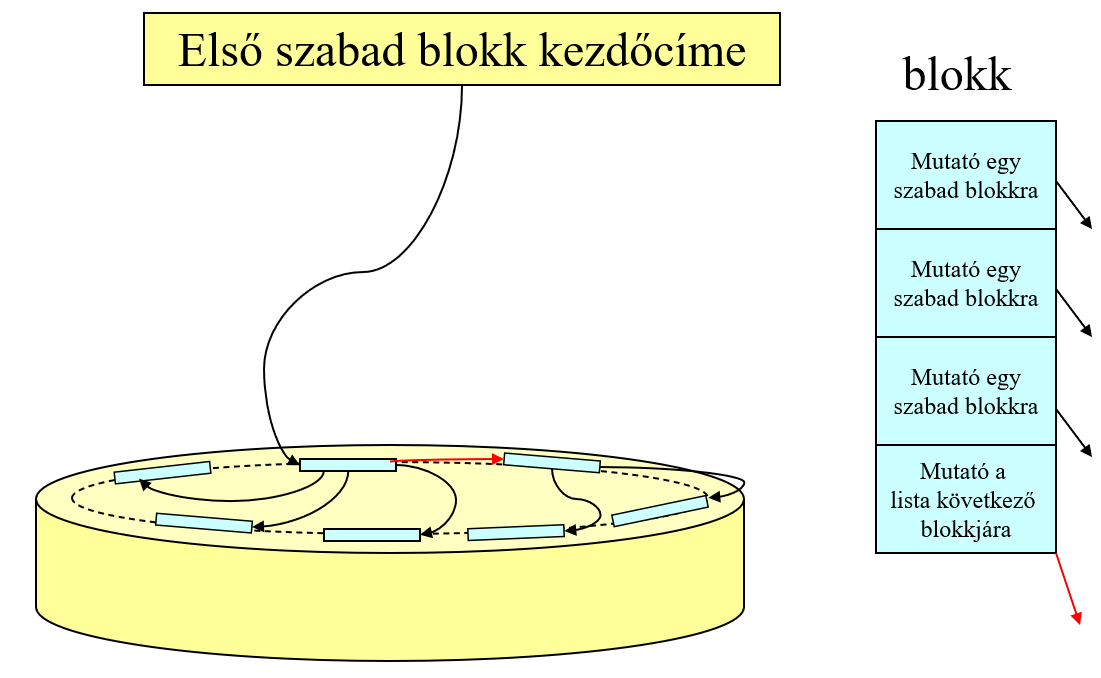
**5. Ismertesse a szabad blokkok nyilvántartását láncolt lista segítségével.**

* Az első szabad blokk címének tárolása, arra felfűzve a többi.
* A blokk területéből vesszük le a cím területét. Nem hatékony, lassú (sok lemezterület).



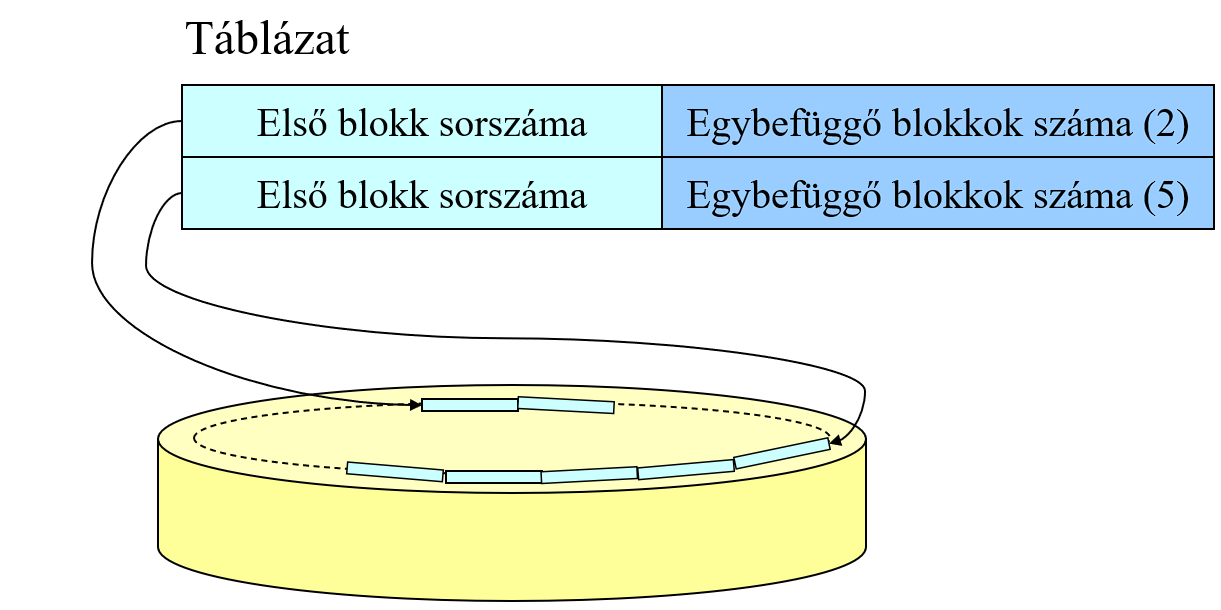
**6. Ismertesse a szabad blokkok nyilvántartását szabad helyek csoportjainak listája segítségével.**

* Láncolt lista javítása. Minden blokk n db (ennyi cím fér el a blokkban) szabad blokkra hivatkozik. n-1 ténylegesen szabad, az n. a lista új elemére mutat.



**7. Ismertesse a szabad blokkok nyilvántartását egybefüggő szabad területek tárolása segítségével.**

* Egy táblázatban tároljuk az összefüggő szabad blokkokat (első blokk sorszám, blokkok száma)



**8. Soroljon fel 4 módszert a lemez blokkjainak allokációjára.**

* Folytonos terület allokációja
* Láncolt tárolás
* Indexelt tárolás
* Kombinált módszerek

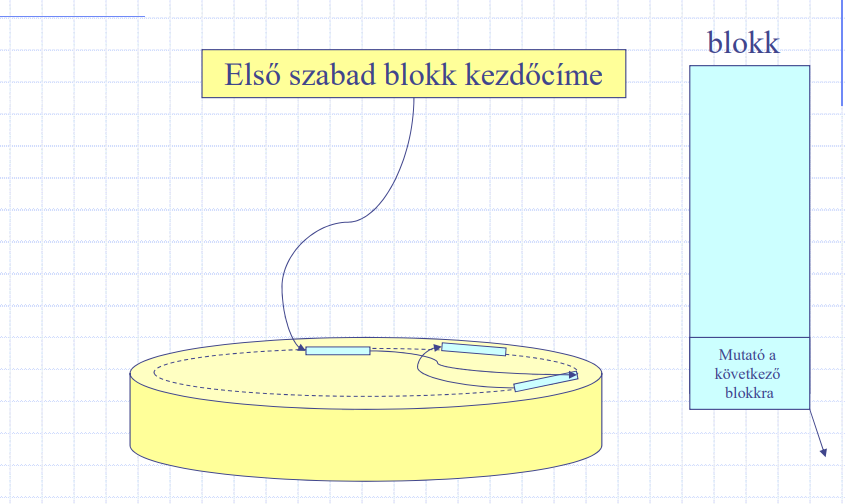
**9. Ismertesse a lemez blokkjainak allokációját a folytonos területek allokációja módszerrel.**

* Az összetartozó információkat egymás melletti blokkokban tároljuk. Első blokk sorszámát és a blokkok számát kell tárolni.
* Hátrányai:
  + Külső tördelés itt is fennáll
  + A megoldáshoz itt is használhatók a már megismert algoritmusok (első illeszkedő, legjobban illeszkedő, legrosszabbul illeszkedő).
  + Nagyméretű tördelés esetén tömöríteni kell.
  + Sokszor nem tudjuk előre, hogy hány blokkra lesz szükségünk. lefoglaláskor becsülni kell, a rossz becslés gondot okozhat (hiba és leállás, átmásolni az eddig lefoglalt területet egy másik helyre)
* Előnyei:
  + A tárolt információ soros és közvetlen elérése is lehetséges.
  + Jól használható tárcsere által kirakott szegmensekre (tudjuk előre a méretet)



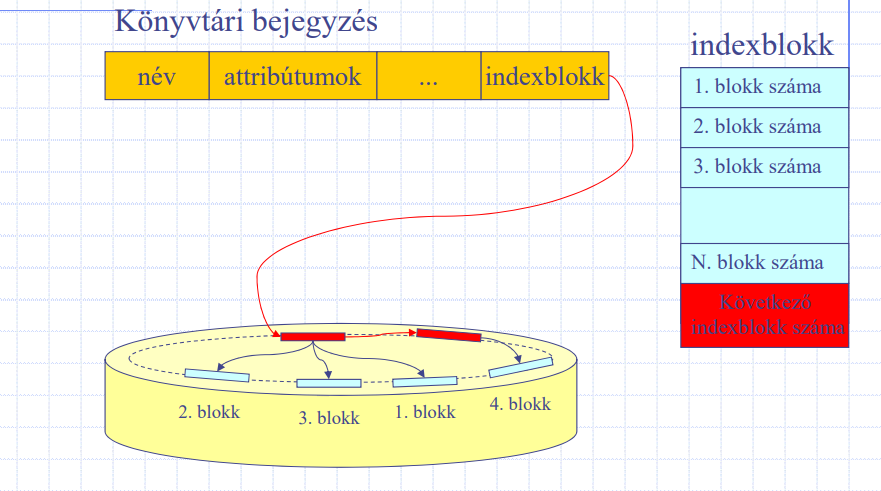
**10. Ismertesse a lemez blokkjainak allokációját láncolt tárolás módszerrel.**

* Az első szabad blokk címét tárolja
* Minden blokk eltárolja a következő blokk címét
* Nem hatékony, nagy lemezterületet igényel



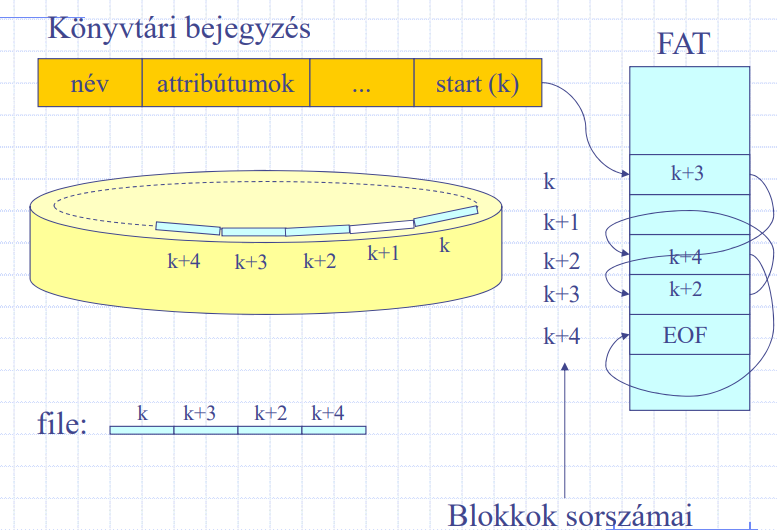
**11. Ismertesse a lemez blokkjainak allokációját indexelt tárolás módszerrel.**

* + Az állományhoz tartozó blokkok címei egy indextáblában vannak.
  + Előnye, hogy közvetlen hozzáférést biztosít és alkalmas a „Lyukas” tárolására (nem minden blokk tartalmaz valós információt)
  + Hátránya, hogy az indextábla tárolása legalább egy blokkot elfoglal (kis állományok esetében pazarló) és az indextábla mérete nem ismert, ezért lehetővé kell tenni, hogy az növekedhessen.
  + A növekedő indextáblára módszerek:
    - láncolt indexblokkok,
    - többszintű indextábla,
    - kombinált mód,
    - kis állományok esetében egy, majd többszintű indextábla



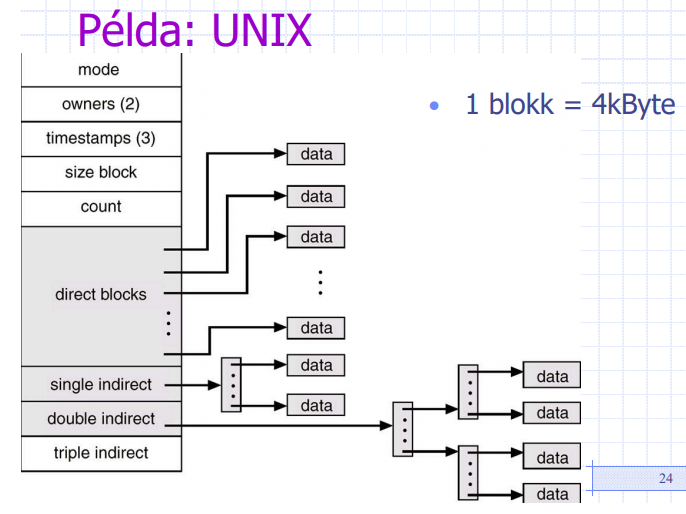
**12. Ismertesse a FAT tábla működését.**

* Láncolt tárolás egy módosított verziója.
* A láncelemeket az állományoktól elkülönítve tároljuk.
* A szabad helyek tárolására is alkalmas a FAT.



**13. Ismertesse a Unix által alkalmazott módszert lemez blokkjainak allokációjára**

* Kombinált módszer, soros hozzáférés esetén láncolt, közvetlen hozzáférés esetén indexelt.
* Kis állomány esetén folytonos, nagy állomány esetén indexelt



**14. Mit tartalmaz egy könyvtári bejegyzés (directory entry)?**

* -Név
* Attribútumok
* Indexblokk

**15. Soroljon fel 5 műveletet állományokon.**

* átvitel, írás, olvasás
  + 1. Közvetlen átvitel esetén, információ címe szükséges
    2. Soros hozzáférésnél az aktuális pozíciót a rendszer növeli és tárolja.
    3. Szimultán írás-olvasás esetén soros hozzáférésnél közös vagy több pozíció használata.
* hozzáadás (append)
* Az állomány végéhez új információt írunk. Az állomány mérete növekszik, esetlegesen új blokk lefoglalása.
* pozicionálás
* Soros hozzáférés esetén megadhatjuk az aktuális pozíciót.
* állomány megnyitása
  1. az állomány megkeresése
  2. jogosultságok ellenőrzése
  3. az elvégezendő műveletek megadása
  4. osztott állománykezelés szabályozása
  5. soros hozzáférés pozíciójának beállítása
* állomány lezárása
* puffer esetén a ki nem írt információ kiírása, osztott állománykezelésnél az állomány felszabadítása.
* állomány végrehajtása
* Az OS létrehoz egy új folyamatot, a programállományt betölti a folyamat tárterületére és elindítja.
* A könyvtárra is hatással levő műveletek:
  1. állomány létrehozása (új bejegyzés, blokkok lefoglalása)
  2. állomány törlése (bejegyzés megszüntetése, blokkok felszabadítása)

**16. Miért van szükség állományok megnyitására/lezárására, amikor azokon művelet végzünk?**

* Megnyitás: így nem kell minden alkalommal a jogosultságokat ellenőriznie az operációs rendszernek, megnyitás során egy adatszerkezetet épít fel amivel a további műveleteket gyorsabban végzi el
* Lezárás: puffer esetén a ki nem írt információ kiírása, osztott állománykezelésnél az állomány felszabadítása, a felépített belső adatstruktúrákat is felszabadítja

**17. Soroljon fel 5 műveletet könyvtárakon.**

* Új könyvtár létrehozása
* Könyvtár törlése
  + 1. csak akkor törölhető, ha üres
    2. törli a benne lévő állományokat
    3. rekurzívan törli a benne lévő könyvtárakat is
* Keresés
  + Egy névhez meg kell találni a hozzátartozó állományt. A keresés típusa függ a nyilvántartás szerkezetétől: rendezetlen keresés, rendezett felező keresés, hash keresés.
* Új bejegyzés létrehozása
  + Nyilvántartás méretének dinamikusan növekednie kell. Rendezett nyilvántartás esetén a rendezettséget meg kell tartani.
* Bejegyzés törlése

**18. Mit jelent az osztott állománykezelés? Hogyan lehet írás esetén az osztott hozzáférést megoldani?**

* Ez is egy erőforrás, amelyet egyidejűleg több folyamat is használni akar.
* Csak olvasás esetén osztottan gond nélkül használható.
* Egész állományra vonatkozó szabályozás (file lock)
  + Az állományt először megnyitó folyamat definiálja, hogy a későbbi megnyitási kérelmekből mit engedélyezhetünk.

1. kizárólagos használat
2. többi folyamat csak olvashatja
   * Írás esetén az olvasó folyamatok mikor veszik észre a változást:
3. azonnal,
4. ha a folyamat lezárta az állományt.
   * Állomány részeire vonatkozó kizárás

* Rekordonként lehet kizárásokat definiálni

**19. Mi a célja az állományok hozzáférés-szabályozásának?**

Cél: jogosulatlanok ne férjenek hozzá állományokhoz, ne végezhessenek rajtuk műveleteket. A jogokat a létrehozója, vagy speciális jogokkal rendelkező felhasználó definiálja.

**1. Hogyan definiáljuk a védelmi problémát?**

A védelmi probléma: minden elemet csak az arra jogosult eljárások érhessenek el és azt helyesen használják.

**2. Milyen fenyegetési források vannak?**

* Belső:
  + rendszerben lévő folyamatok felől, *(Lehetnek tévedésből vagy szándékosan rosszak)*
* Külső:
  + rendszertől független tényezőktől. *(Rendszeren kívül álló rosszindulatú támadók elleni védelem)*

**3. Mi ellen nyújt védelmet a belső biztonsági rendszer?**

Belső biztonsági rendszer (internal security):

* + programhibák elleni védelem,
  + erőforrások jogosulatlan felhasználásának védelme.

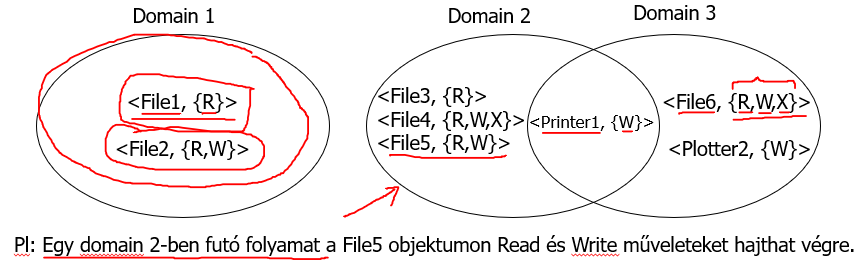
**4. Mit jelent a hozzáférési jogosultság és a védelmi tartomány?**

Védelmi tartomány (protection domain): megadja, hogy a folyamat az adott állapotában mely erőforrásokhoz férhet hozzá és ezeken milyen műveleteket végezhet.

* + Minél szűkebbre kell definiálni, csak a valóban szükséges jogokat megadni (*need-to-know* elv).
  + Védelmi tartomány lehet statikus vagy dinamikus (állapottal változó).

Hozzáférési jogok:

* <Objektum, {jogosultságok listája}>,
* ahol a jogosultságok listája az adott objektumon végrehajtható műveletek egy részhalmaza
* Védelmi tartomány (domain): hozzáférési jogok halmaza

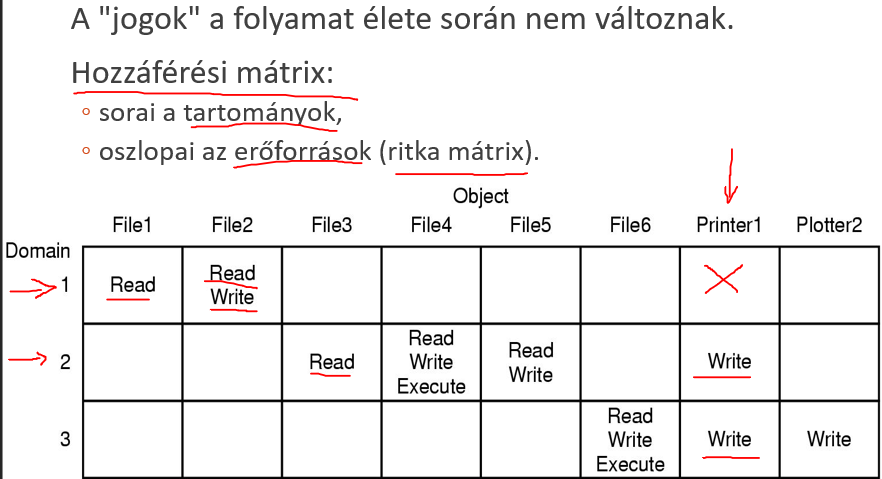


**5. Rajzoljon fel egy példát statikus védelmi tartományokra egy hozzáférési mátrix segítségével. Magyarázza el az oszlopok, sorok és a bejegyzések jelentését.**

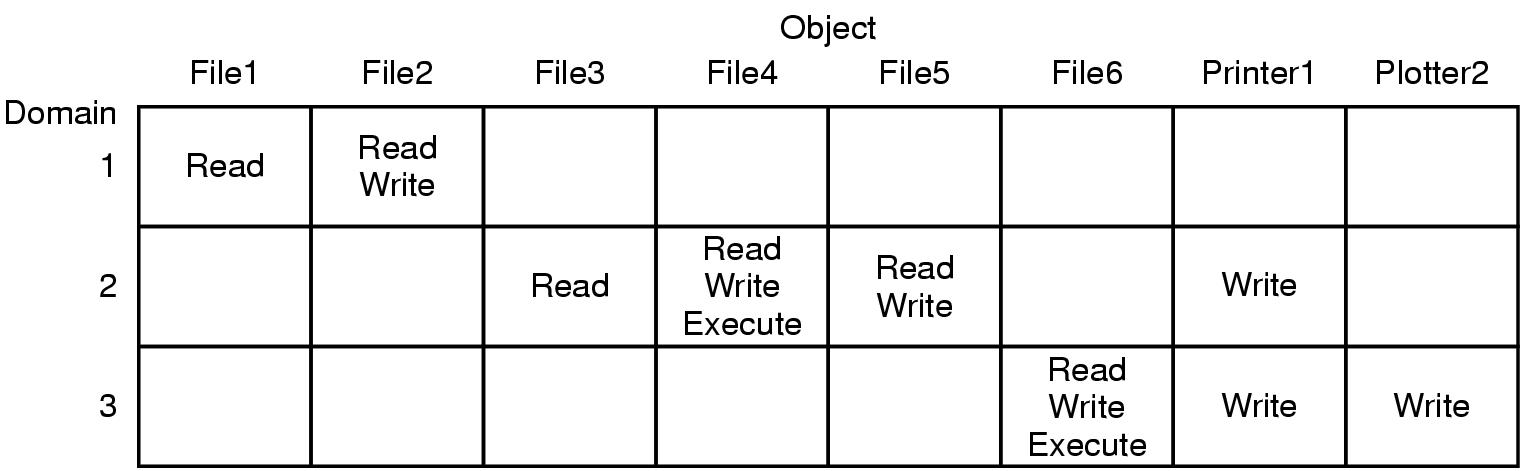
Sor: Tartományokat, domaineket ábrázoljuk

Oszlop: Erőforrások lesznek

Bejegyzések: „Hozzáférési mátrix kevés bejegyzést tartalmazz, így ez egy ritka mátrix” *– PPTben ennyit volt róla – NOTE BY: Yololai*



**6. Magyarázza el a „Globális tábla” adatszerkezetének felépítését.**

Ez egy ritka táblázat, ennek következtében a táblázatban megjelenő bejegyzéseket ábrázolni tudják úgy, hogy a nem üres cellákat sorolják fel.

Ebben az esetben az alábbi módon (<tartomány, erőforrás, művelet>)írhatjuk fel a nem üres tartalmakat:

<Domain1, File1, R>,

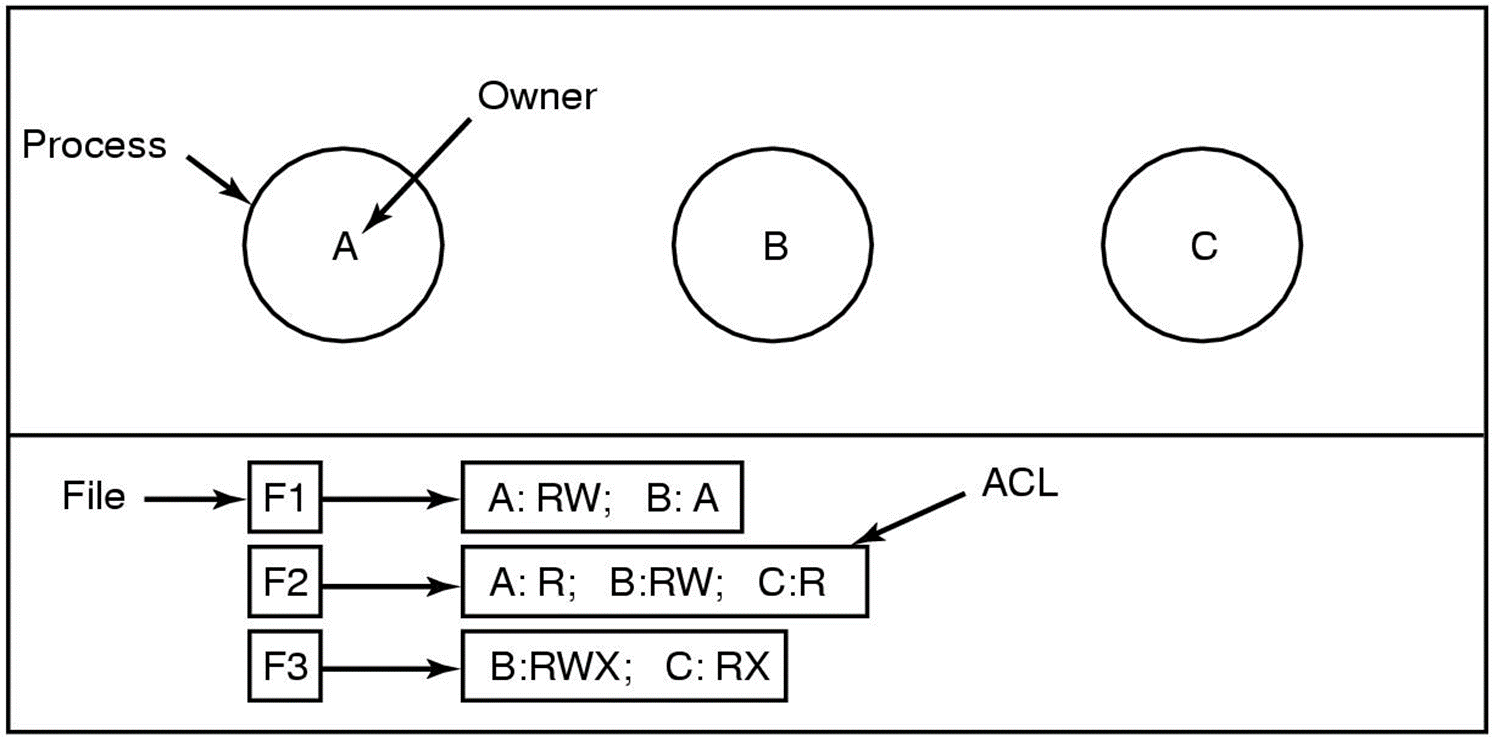
<Domain1, File2, RW>,

<Domain2, File3, R>, … stb.

Nem lehet csoportokat képezni, mindenki számára elérhető erőforrásokat minden tartományhoz fel kell venni. Például a nyomtató esetén, amely több domain-ben is szerepel, érthető módon többször fog szerepelni a bejegyzések között. Mivel ez egy jó nagy adatszerkezet, ezért lassabb lesz a keresési idő, így ez az ábrázolási mód nem sportszerű.

**7. Magyarázza el a „Hozzáférési lista” felépítését és működését. Illusztrálja példa segítségével.**

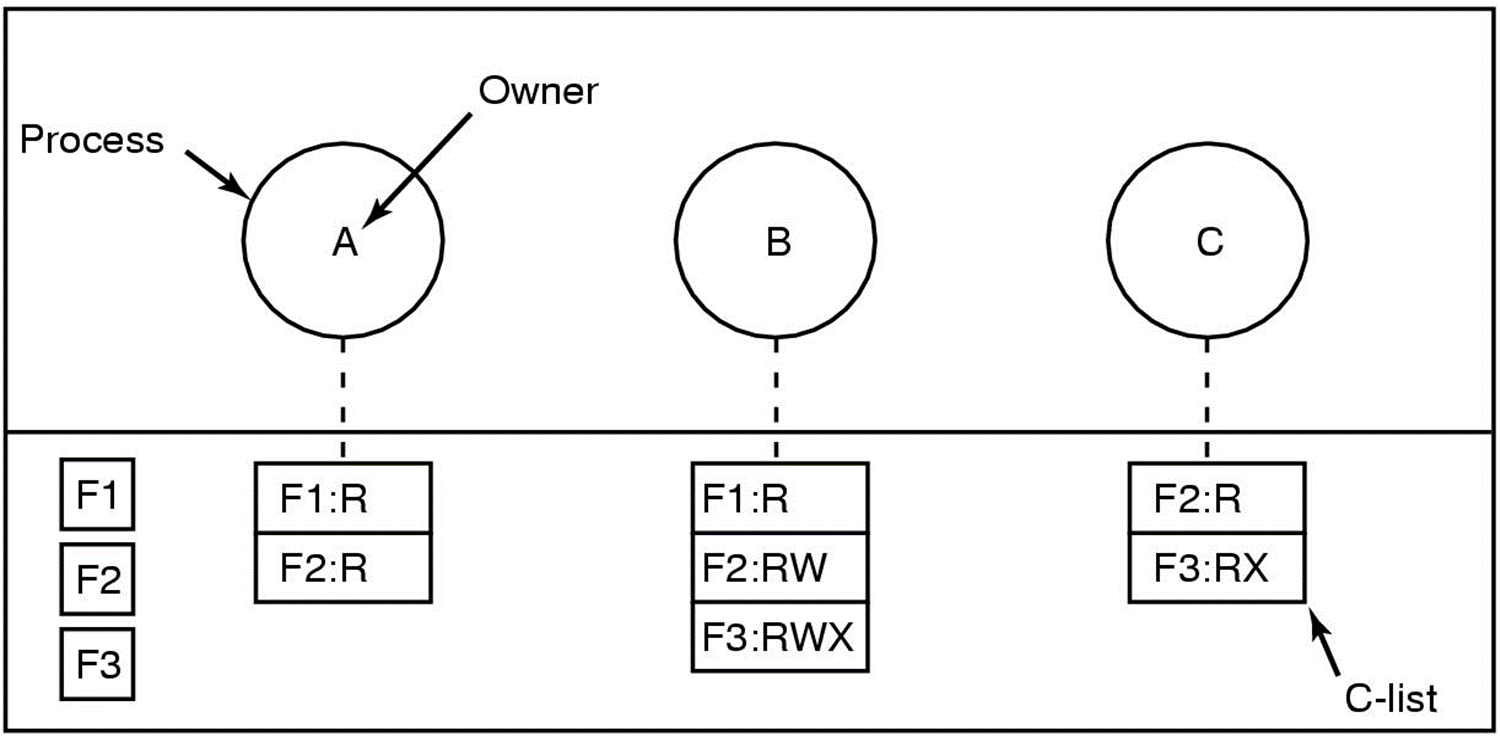
Ez egy alternatív ábrázolási mód, amelynél oszloponként ábrázolják a bejegyzéseket.

Felhasználói igényekhez igazodnak, új erőforrás létrehozásakor lehet definiálni a védelmi tartományokban kiadható műveleteket is.

A fenti képen látható 3 fájl (F1, F2, F3), valamint ezekhez rendelődik egy lista. Ezen listák mutatják, hogy a fájlokon az egyes felhasználók miféle műveletekkel tudnak hozzáférni. (pl. az F2 fájl esetén az A olvashatja, a B írhatja, és olvashatja, míg a C az A-val megegyezően csak olvashatja.

Ezt a listát nevezzük ACL listának (Access Control List).

**8. Magyarázza el a „Jogosítvány-lista” felépítését és működését. Illusztrálja példa segítségével.**

Ez is egy alternatív ábrázolási mód az előzőhöz hasonló logikával, viszont itt soronként ábrázolják a bejegyzéseket. Érdemes megjegyezni, hogy ez a fajta lista nem nyilvános a folyamatok számára (ezt az OS kezeli).

A fenti képen látható a három fájl, vagy adatforrás (F1, F2, F3) és a három felhasználó (A, B, C). Itt a felhasználókhoz vannak kötve az adott fájlok (például, ha az A felhasználót nézzük, akkor neki két írási joga van az F1-es és az F2-es fájlhoz egyaránt).

Ezt a listát pedig C listának nevezzük (Capability List).

A képen asztal látható

Automatikusan generált leírás

**9. Adjon példát hozzáférési listák és jogosítvány listák együttes használatára.**

UNIX rendszerekben egy állomány megnyitásánál a folyamatnak meg kell adnia a műveleteket. Ezt a hozzáférési lista alapján ellenőrzi az OS, majd a folyamat kap egy jogosítványt, a későbbi műveleteknél ezt használja. Az együttes használat következtében valósulnak meg a statikus védelmi tartományok

**10. Mit jelent a statikus védelmi tartomány? Mit jelent a dinamikus védelmi tartomány?**

*Statikus védelmi tartomány:*

Be kell vezetni egy új műveletet mielőtt bármit tennének az adott fájllal. Ezt a műveletet megnyitásnak nevezzük (ez itt az open művelet).

int filedesc = open("alma", O\_APPEND);

write(filedesc, "Egy uj sor**\**n", 11);

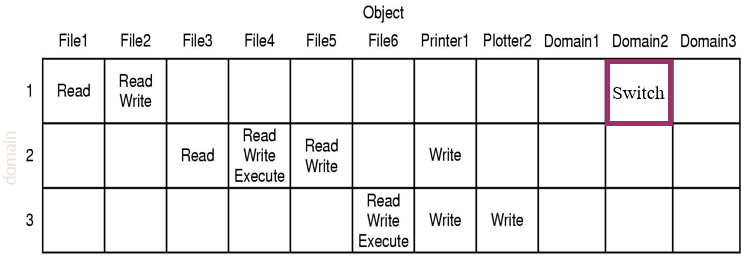
A fenti példában az „alma” nevű fájl szeretnénk APPEND (hozzáfűzés) műveletre megnyitni, ezen belül is írásos hozzáfűzésre. Az OS megtekinti a fájlhoz tartozó hozzáférési jogosultságokat. Megkeresi ennek a fájlnak a megfelelő bejegyzéseit.

Ebben az esetben a tulajdonos az adott fájlt írhatja, olvashatja, végrehajthatja. Az OS látja, hogy a tulajdonos rendelkezi írási jogosultsággal, ezért ő végrehajthatja az APPEND műveletet („kap rá egy jogosítványt”, amit maga az OS állít elő).

A tényleges íráshoz már csak a filedescriptort (filedesc - jogosítvány) kell használni. Ezzel hivatkoznak a már megnyitott fájlra, és az alábbi szöveget szeretnék beírni ("Egy uj sor\n"). Az OS ekkor megtekinti a jogosítványt, és látja, hogy ez írásra alkalmas az alábbi fájlra.

*Dinamikus védelmi tartomány:*

A statikus védelmi tartományokban túl sok erőforrás van (mindet fel kell venni), a dinamikus védelmi tartomány csak a szükségeseket tartalmazza. Hozzáférési táblázat itt is használható.

A trükk csak annyi, hogy a védelmi tartomány is egy erőforrás, ezt is felveszik a mátrix (tábla) oszlopai közé.

Nagyon hasonlít a statikus hozzáférési tábla, amelynek sorai a tartományok, oszlopai az erőforrások. Ehhez a mátrixhoz felvesznek új oszlopokat, amelyeken a tartományok szerepelnek.

Például: Az 1-es számú tartományhoz és egy adott erőforráshoz (file1) milyen jogosultságokkal rendelkezik az adott tartományban futó folyamatnak. Itt definiálnak új műveleteket domainekre is.

Látható, hogy az új művelet a Switch. Ez azt jelenti, hogy az 1-es domainben futó folyamat a Domain2 tartományon, mint erőforráson rendelkezik egy kapcsolási művelet végrehajtási joggal. Amíg ez a Domain1-ben futott, most átkapcsolhat a Switch művelettel a 2-es tartományra, ahol a jogosultságok a domain2-ben megfelelőek lesznek.

**11. Hogyan lehet dinamikus védelmi tartományt megvalósítani? Rajzoljon fel egy példát hozzáférési mátrix segítségével.**

Ezt a feladatot megcsináltam az előzőben (ott van benne a tábla is) – *Note by: Bau*

**12. Adjon példát dinamikus védelmi tartományokon végezhető műveletekre. Magyarázza el ezek jelentését.**

*kapcsolás (switch)*

Az adott tartományból egy folyamat átkapcsolhat az oszlopban kijelölt tartományra.

*másolás (copy)*

Az adott tartományban tartózkodó folyamat hozzáférési jogait átmásolhatja egy másik tartományba. A másoláson kívül lehet: átadás (transfer), amikor a jogok az eredeti tartományból törlődnek; korlátozott másolás, az új tartományból a jogot nem lehet továbbadni.

*tulajdonos (owner)*

Az erőforrás tulajdonosa más tartományokban törölhet vagy adhat az erőforráshoz tartozó műveleteket (oszlopok mentén történő módosítások).

*vezérlő (control)*

Sorok mentén történő módosításokra ad lehetőséget. Csak a tartományokhoz tartozó oszlopokban szerepelhet. Ha egy tartomány vezérli a másikat, akkor annak sorában elhelyezhet, illetve törölhet jogosultságokat.

**13. Milyen módon lehet felhasználókat azonosítani? Adjon ezekre példákat.**

Jogtalan hozzáférések megakadályozására a felhasználókat egyértelműen és megbízhatóan azonosítani kell. Erre létezik néhány módszer:

* Az első, hogy valamilyen attribútuma (tulajdonsága) alapján próbálják a felhasználót azonosítani. Ez lehet biológiai attribútumok is (tipikus példa az ujjlenyomat olvasó, vagy a retina szkenner). Ezek manapság igazán elterjedt módszerek.
* A másik ellenőrzési módszer, amikor valamiféle tárgy birtoklását ellenőrzik. Ez a legrégebibb módszer (pl. akinek kulcsa van a pincéhez, az ki tudja nyitni a pinceajtót). Későbbiekben megjelentek a kártyás beléptetőrendszerek, ahol a kártyák a kulcsok szerepét vették át.
* A harmadik módszer pedig az ismeretekkel operáló azonosítási módszerek (pl. általában egy számtógépes rendszer egy névvel és jelszóval kezdi az egyén azonosítását.

A jelszó biztonsága értelemszerűen a felhasználótól függ. Egyrész jó jelszót kell választani, majd ezt kellő gyakorisággal kell változtatni.

**14. Hogyan tároljuk a jelszóra vonatkozó információkat?**

Ahelyett, hogy a p jelszót tárolnák, ennek egy függvénnyel összekavart változatát ( f(p) ) karaktersorozatot tárolnak. Fontos tulajdonsága, hogy egyirányú. A rendszer a jelszót biztonsági okokból nem tárolja, mert akkor az könnyen ellopható lenne.

**15. Mit jelent, hogy a jelszó kódolására szolgáló függvény egyirányú?**

Hiába ismeri a felhasználó, és hiába lopja el valaki ezt az eltárolt függvényt, nincs esély arra, hogy visszaalakítsa és megkeresse, hogy mi volt az eredeti p jelszó, amiből a függvény keletkezett. Az invertálás közel esélytelen (hatalmas számítási igényt követelnek) az egyirányúság miatt

**16. a, Magyarázza el a jelszó sózásának módszerét.**

A jelszófájl nem *f(p)*-t tárolja (hiszen *p* esetleg gyakori szó!)

* Tárolunk
  + egy véletlen N egész számot, ez a „só”, és
  + *f(pN)*-t.
* Pl. (jelszó: alma)

User2, 67832, f(alma67832),

ahol

* + User2 a felhasználó azonosítója,
  + 67832 a só,
  + f(alma67832) az eltárolt kódolt sózott jelszó.

**b,Mennyiben nehezíti ez meg a támadó dolgát?**

* Teljes lista elleni támadás lehetetlen mert mindenkinek más a sója
* Csak az egyéni támadás lehetsége.

**17. Mire jó a naplózás?**

* A rendszer feljegyzi
  + - az egyes erőforrásokhoz a hozzáférések időpontját,
    - a műveletet és
    - a felhasználót.
* Ez nem véd, de az elkövető utólagos azonosítását megkönnyíti.

**18. Milyen fő alkalmazási területei vannak a kétkulcsos rejtjelzésnek?**

*(Csapóajtó kódok)*

*Alkalmazás:*

* + Üzenet titkosítása
  + Üzenethitelesítés (az üzenetet senki nem módosította)
  + Partner hitelesítés (a küldő hitelesítése), elektronikus aláírás.

**19. Magyarázza el a kétkulcsos titkosítás alapelvét.**

* **Aszimmetrikus** (kétkulcsos, v. nyilvános kulcsú) titkosítás
* Üzenet:
  + - M
* Két kulcsot (függvényt) használ, amelyekre igaz:
  + - S: *titkos*, csak a tulajdonos ismeri
    - P: *nyilvános*, mindenki ismeri
    - S=P-1, tehát S(P(M))=P(S(M))=M
    - P-ből nagyon nehéz (gyakorlatilag lehetetlen) S-et kitalálni

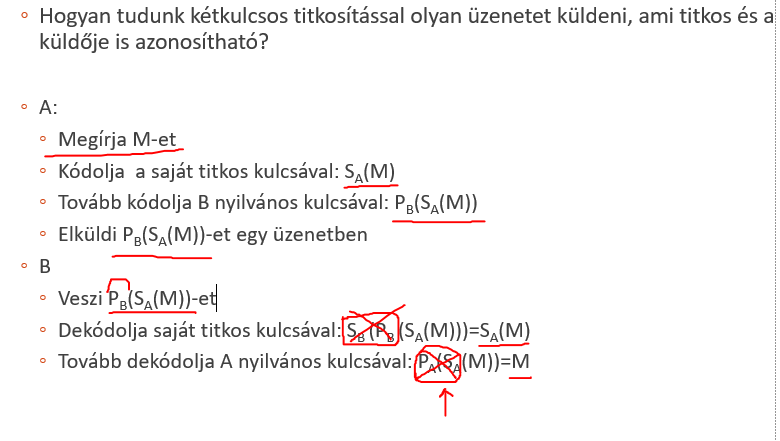
**20. Hogyan tudunk kétkulcsos titkosítással titkos üzenetet küldeni?**

**A:**

* + - Megírja M-et
    - Kódolja a saját titkos kulcsával: SA(M)
    - Tovább kódolja B nyilvános kulcsával: PB(SA(M))
    - Elküldi PB(SA(M))-et egy üzenetben

**B**

* + - Veszi PB(SA(M))-et
    - Dekódolja saját titkos kulcsával: SB (PB (SA(M)))=SA(M)
    - Tovább dekódolja A nyilvános kulcsával: PA(SA(M))=M



**21. Hogyan tudunk kétkulcsos titkosítással üzenetet aláírni?**

* Üzenet titkosítás
* Üzenethitelesítés
* Partner hitelesítése, elektronikus aláírás

**22. Magyarázza el a következő fogalmakat: Trójai faló Rejtekajtó Féreg Vírus Stack és Buffer Overflow** **támadás Túlterheléses támadás**

Trójai faló

* Terjedés: megtévesztés által, E-mailben, reklámmal
* Hatás: Adatlopás, számítógép feletti kontroll átvétele
* Példák: A keresési utakat felhasználva a kívánt rendszerprogram előtt (helyett) indul el, gyakran elgépelt parancsok használata

Rejtekajtó

* Egy programban az írója olyan részeket tesz, amelyek nem dokumentált, jogosulatlan tevékenységeket végezhetnek.
* Léteznek fordítóprogramokban, vagy programkönyvtárakba épített rejtekajtók, amelyek a lefordított kódba is belekerülnek.
* Létezik legális felhasználás is: gyártók által tesztelésre vagy hibaelhárításra

Féreg

* Szaporodok, terjed
* Fertőzött gép újabb áldozatokat keres, ezeket tovább fertőzi
* A fertőzésen kívül vihet magával „csomagot”, ami kárt okoz: fájlok törlése, bizalmas adatok lopása, adatok „túszul ejtése” váltságdíjért, botneteket férgekkel állítják fel

Vírus

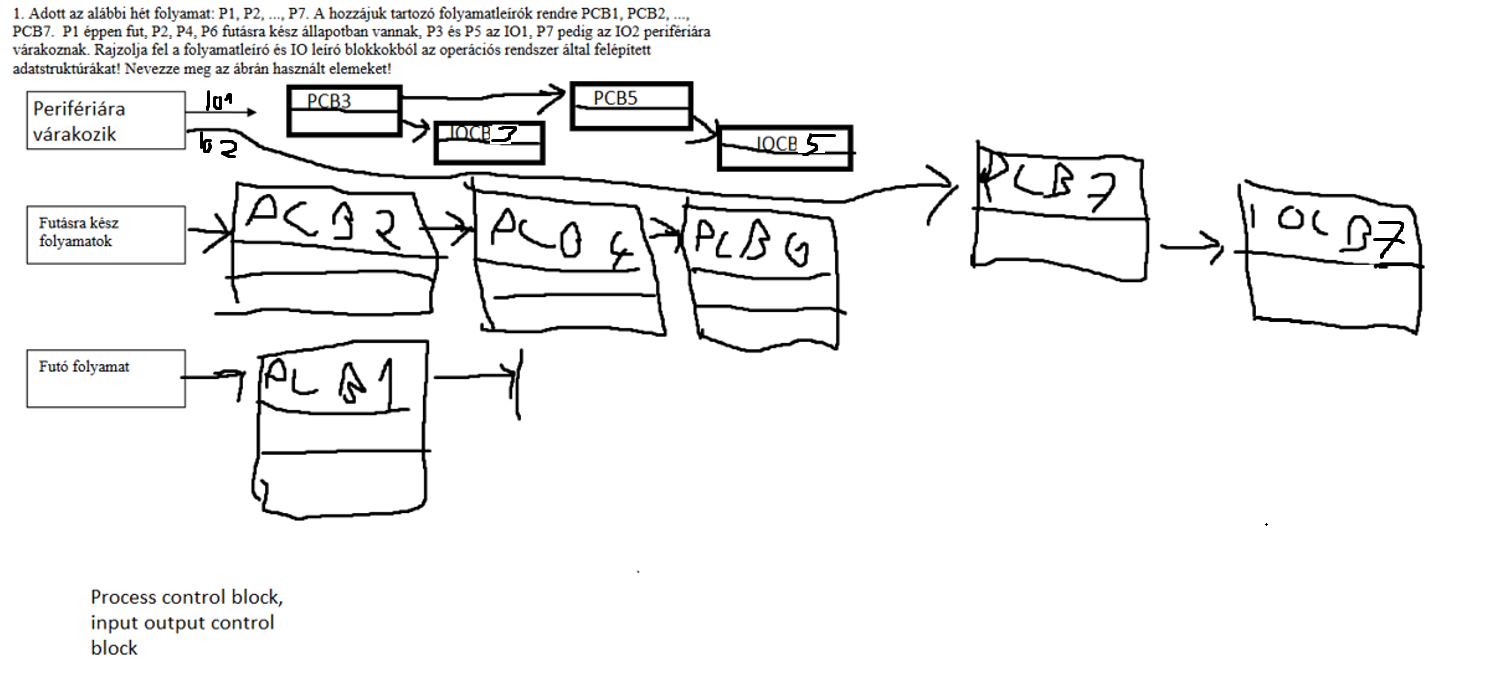
* Más programokba saját utasításait csempészi
* Fertőzhet: végrehajtható állományokat, dokumentumokat, boot szektort, háttértárat
* Jól rejtőzik, nincs saját állománya: jobb rejtőzködés érdekében képes lehet saját magát változtatni
* A fertőzött program indításakor a vírus aktiválódik: tovább terjed, kárt tesz

Stack és Buffer Overflow támadás

* A programban jelen lévő, nem ellenőrzött korlátok kihasználásával a vezérlés saját kódnak átadása

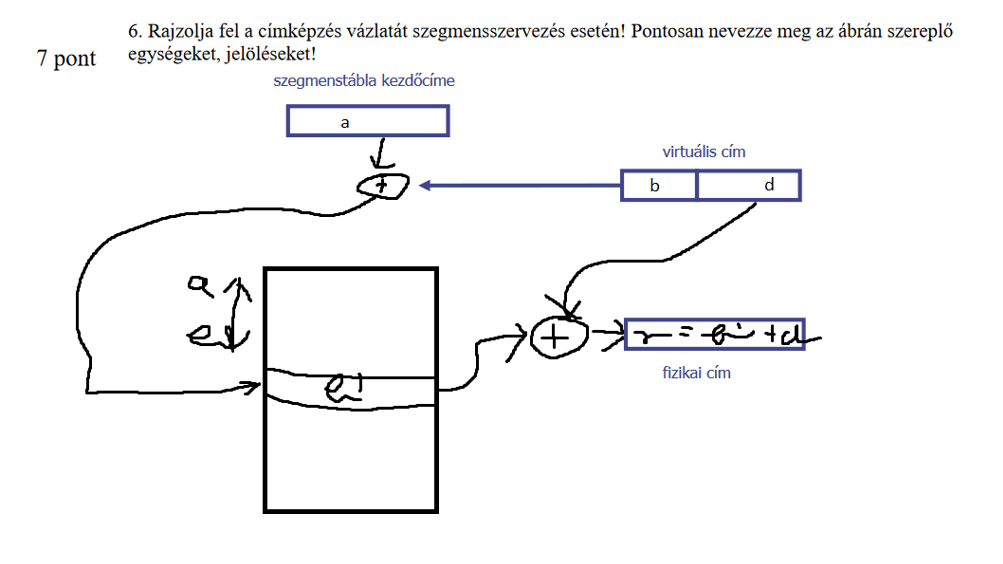
Túlterheléses támadás

* A célzott gép túlterhelése
* hasznos tevékenység megakadályozása
* Fejlettebb formája (DDoS)
* Több gépről indul a támadás
* Védekezés nehezebb
* Általában botnet



A képen szöveg látható

Automatikusan generált leírásA képen szöveg látható

Automatikusan generált leírás

**Egyéb:**

**Laphiba (page fault):** Az előbb írt eset, amikor a laptábla bejegyzés jelzi, hogy az elérni kívánt lap nincs a fizikai memóriában, illetve a logikai cím nagyobb egy előre megadott korlátnál. Ha a memória-hozzáférési igényt az operációs rendszer jogosnak ítéli, akkor intézkedik, hogy a kérdéses lap a fizikai memóriába betöltődjön (ehhez esetleg ugyanezen, vagy valamelyik másik processz egy lapját fel kell áldozni), és a laptáblabejegyzést aktualizálja. (Említsük meg, hogy egyetlen gépi utasítás akár több laphibát is okozhat (egy kétcímes utasítás esetén akár négyet is); minden hiba aktiválja az OS hibakezelő rutinját, aminek a lefutása utána a program újra a megszakadt utasítással folytatódik.)

**Laptábla:** Az a táblázat, amely a logikai címeket fizikai címekre fordítja. Egy-egy bejegyzés egy memórialapra vonatkozik, tartalma tipikusan 4 GiB logikai címtartomány, 4 KiB lapméret: (zárójelben a bitek száma egy tipikus rendszerben)

* Present-bit (1): a lap a fizikai memóriában van-e
* Address (22): lap fizikai címe, ha az a fizikai memóriában van (az alsó 10 bit mindig nulla, azokat nem tároljuk), egyébként tetszőleges. (Az OS használhatja, hogy megtalálja a lap tartalmát a lemezen.)
* Védelem (1-3): a lap írható/olvasható/végrehajtható állapota
* Operációs rendszer számára fenntartott (maradék): az OS például fenntarthat egy bitet annak jelölésére, hogy a lap rezidens, azaz nem szabad kilapozni a memóriából.

**Érvénytelen laphiba (invalid page fault):** Ha az operációs rendszer úgy ítéli meg, hogy a folyamat olyan memóriacímet próbált elérni, amihez nem tartozott memória, vagy tiltott módon akar hozzáférni a memóriához (lásd a memóriavédelemnél), akkor a program futását megszakítja. Szegmentálás esetén az is érvénytelen laphibát okoz, ha a logikai cím *offset* része nagyobb a szegmens méreténél. Megjegyzés: a jelen gyakorlatban a 'segmentation fault' (röviden SEGFAULT) kifejezést 'érvénytelen laphiba' értelemben használjuk.

**Lapozófájl (swapfile, pagefile):** Az a lemezterület, amely a fizikai memóriában el nem férő memórialapokat tartalmazza. Ez rendszertől függően lehet fizikai lemez, lemez-partíció, vagy fájl. Utóbbi esetben előre maximális mérettel létrehozott, egybefüggő (nem darabolódott) fájlt érdemes használni (illetve egyes rendszerekben csak ilyet lehet használni). A fizikai memória és a lapozófájl közötti adatmozgatás az operációs rendszer feladata; érdemes megemlíteni, hogy számos rendszerben nem csak a dedikált lapozófájl használható így, hanem más lemezfájlok is, amelyek tartalmát változatlanul (átalakítás nélkül) akarjuk a memóriában viszontlátni, például végrehajtó programok (executable), programkönyvtárak (shared library, DLL), illetve a felhasználó által memóriába lapozott fájlok (memory mapped file).