OPERÁCIÓS RENDSZEREK

Fazekas, Gábor

OPERÁCIÓS RENDSZEREK

Fazekas, Gábor

Publication date Debrecen, 2011.

Tartalom

Végszó

A tananyag a TÁMOP-4.1.2-08/1/A-2009-0046 számú Kelet-magyarországi Informatika Tananyag Tárház projekt keretében készült. A tananyagfejlesztés az Európai Unió támogatásával és az Európai Szociális Alap társfinanszírozásával valósult meg.

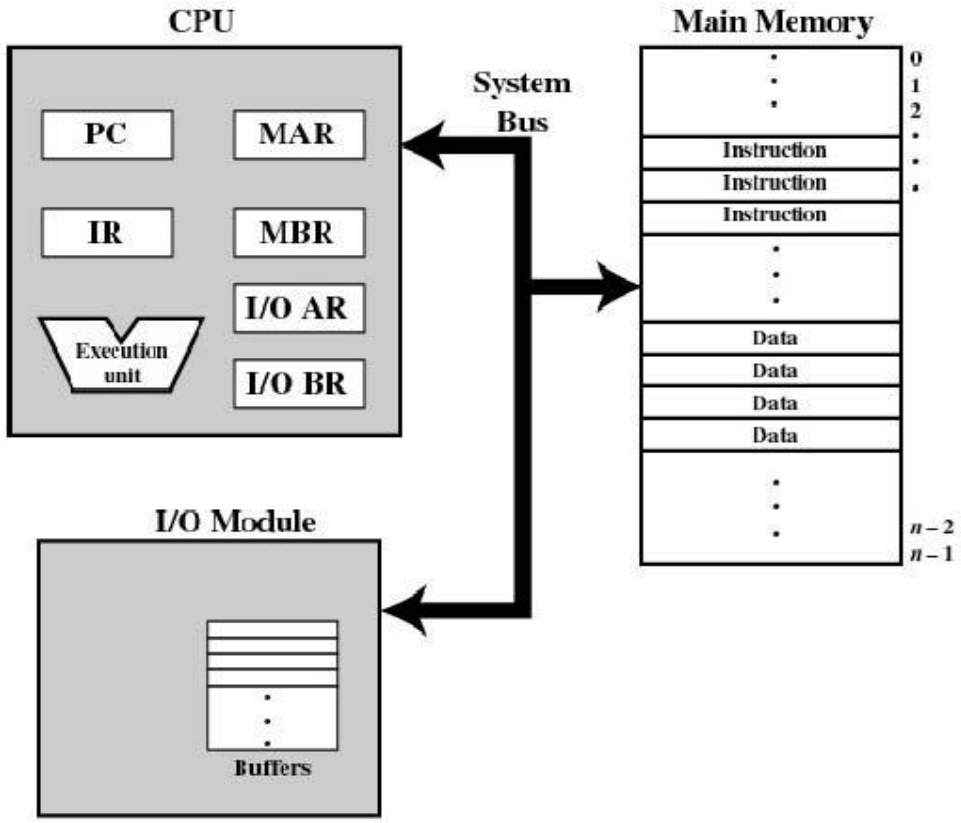
Nemzeti Fejlesztési Ügynökség http://ujszechenyiterv.gov.hu/ *06 40 638-638*

1. fejezet - Számítógépes rendszerek: szerkezeti jellemzők

1. Fő szerkezeti elemek

- **Processzor** (CPU)
- **Fő memória:** memóriának, illetve operatív tárnak, főtárnak is nevezik
- **I/O egységek** (I/O processzorok, I/O adapterek): másodlagos memória eszközök, másodlagos tárak, háttértárak, kommunikációs eszközök, terminálok
- **Rendszerbusz**: vezérlés és adatátvitel a processzor(ok), a memória és az I/O egységek között
- **Az operációs rendszer** megkísérli a hardver erőforrások kihasználását optimalizálni.
  - A felhasználóknak számos szolgáltatást biztosít
  - Például kezeli a másodlagos memóriát és az I/O eszközöket

A fő szerkezeti elemek egy funkcionális sémája:



Jelölések:
• MAR - Memory Address Register
– a következő írás/olvasás memóriacíme
• MBR - Memory Buffer Register

– memóriába küldendő adatok tárolása
– memóriából olvasott adatok tárolása
• I/OAR - I/O Address Register
– kijelöl egy bizonyos I/O eszközt
• I/OBR - I/O Buffer Register
– a processzor és a I/O eszközök közötti kommunikáció adattárolására
• PC/IR Programszámláló- és Utasításregiszter

2. A processzor regiszterei

- Regiszterek: az operatív tárnál gyorsabb és kisebb kapacitású memóriaegységek a CPU-n belül, melyek a feldolgozás alatti ideiglenes adattárolásért felelősek:
  - Felhasználó által látható regiszterek:
    - lehetővé teszik a programozó számára, hogy csökkentse az operatív tárra való hivatkozások számát
  - Vezérlő- és állapotregiszterek:
    - a processzor használja önmaga vezérléshez
    - az operációs rendszer és a programok egyes rutinjai használják a programok futásának vezérléséhez

2.1. A programozó számára látható regiszterek

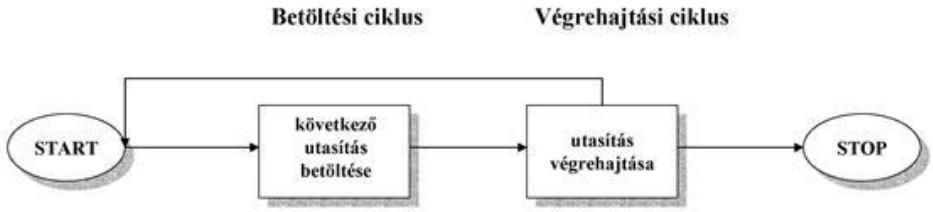
- Gépi kóddal elérhető, általános használatú: felhasználói- és rendszerprogramok is használhatják
- Regisztertípusok: adat, cím, állapotkód
- Adatregiszter: programozó által kioszthatő, módosítható
- Címregiszter: adatok memóriacímeit és utasításokat tartalmaz
  - Index: egy báziscím hozzáadásával kapjuk meg a címet
  - Szegmensmutató: a memória szegmensekre osztása esetén, egy offset és a szegmensmutató együttese határozza meg a címet
  - Veremmutató: a veremmemória legfelső elemét jelöli ki
- Állapotkód regiszterek: műveletek végrehajtásának eredményeként a processzor ír bele, programok által elérhető, de közvetlenül meg nem változtatható (Pl.: egy aritmetikai eredmény pozitív, negatív, nulla, vagy túlcsordult-e)

2.2. Vezérlő- és állapotregiszterek

- Programszámláló (Program Counter – PC / Instruction Pointer - IP)
  - a következő végrehajtandó utasítás címét tartalmazza
- Utasításregiszter (Instruction Register - IR)
  - a végrehajtandó utasítást (ennek bináris kódját) tárolja
  - utasítástípusok:
    - Processzor-memória: adattovábbítás a memória és a processzor között
    - Processzor-I/O: „adattovábbítás” a perifériák (I/O adapter) és a processzor között
    - Adatfeldolgozó: aritmetikai, vagy logikai művelet végzése adatokon

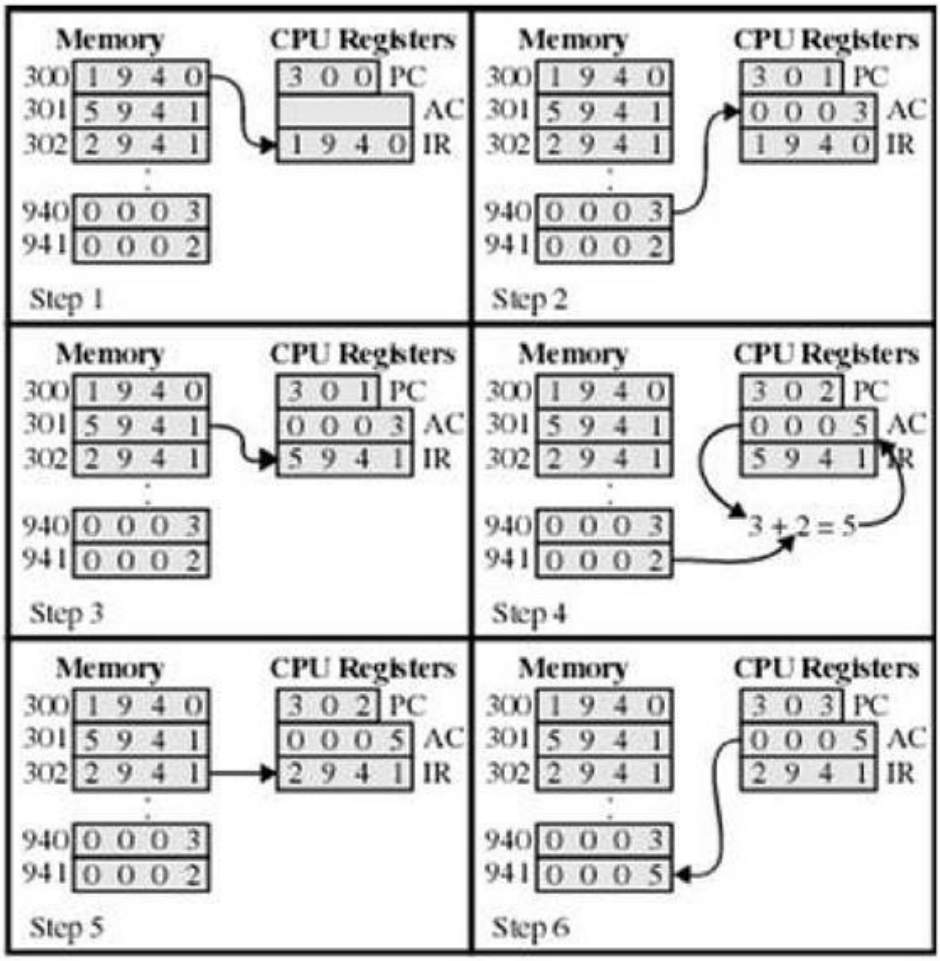
    - Vezérlő: az utasításvégrehajtás sorrendjének megváltoztatását okozza
- Programállapotszó (Program Status Word - PSW)
    - a processzor állapotát írja le
        - állapotkódok
        - megszakítás engedélyezése/letiltása
        - rendszergazdai/felhasználói (kernel/user) mód

3. Utasításvégrehajtás



- a programszámláló tartalmazza a következő betöltendő utasítás címét
- a processzor betölti az utasítást a memóriából
- a programszámláló értéke minden betöltés után „eggyel nő”

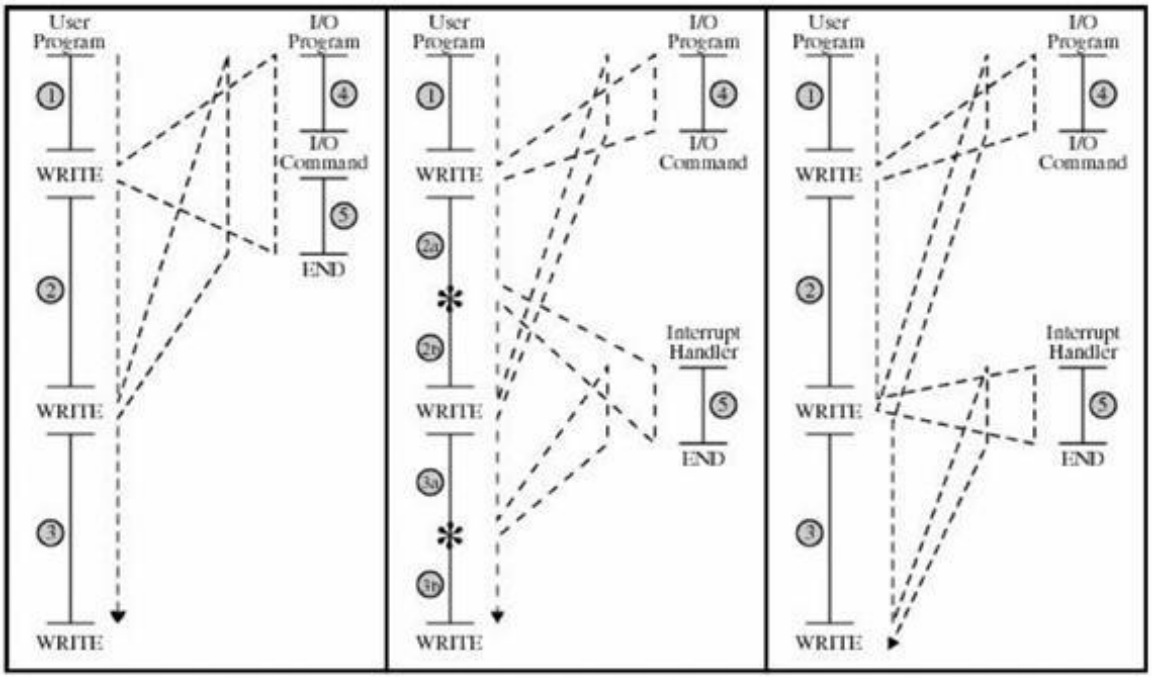
3.1. Egy program végrehajtása



4. Megszakítások

- A normális utasításvégrehajtási sorrend megszakítása:
  - Egy külső esemény hatására létrejövő folyamatfelfüggesztés olyan módon, hogy a felfüggesztett folyamathoz való visszatérés lehetséges
  - A végrehajtás alatt álló utasítássorozat feldolgozása valamelyik utasítás végrehajtása után „megszakad”, új sorozat „kezdődik”
- A feldolgozás hatásfokát növeli:
  - Például a processzor más program utasításait hajthatja végre, amíg egy I/O művelet folyamatban van

Programok végrehajtásának folyamata megszakítással és anélkül



4.1. Megszakítás kezelő

- Program, amely meghatározza a megszakítás okát és végrehajtja azokat az eljárásokat, amelyek ebben az esetben szükségesek (a vezérlést megsza-kításkor kapja meg)
- A megszakítás (interrupt) átadja a vezérlést a megszakítás-feldolgozó rutinnak. Ez általában a megszakítási vektor segítségével történik, amelynek megfelelő elemei tartalmazzák a megszakítási osztályokhoz tartozó feldolgozó rutin első végrehajtandó utasításának címét.
- A megszakítási rendszernek tárolnia kell a megszakított utasítás címét.
- A megszakítási jel forrását tekintve egy megszakítás lehet külső (pl. I/O, Timer, Hardver), vagy belső (szoftveres megszakítás)
- A megszakítás feldolgozó rutin (operációs rendszer része!) közvetlen feladatai:
  - a további megszakítások letiltása, („maszkolás„)
  - a CPU állapotának megőrzése
  - a megszakítás okának, körülményeinek részletesebb elemzése
  - a megszakított programhoz történő visszatérés megszervezése

4.2. A megszakítások osztályai

- Program:
  - aritmetikai túlcsordulás
  - nullával való osztás
  - nem létező művelet végrehajtásának megkísérlése
  - felhasználói memóriaterületen kívülre való hivatkozás (szegmentációs hiba)

  - „rendszerhívás” (szoftver által direkt módon kiváltott megszakítás: INT, SVC, Tr, ...)
- Rendszeróra (időzítő, időadó, timer)
- I/O berendezés/adapter által kiváltott
- Hardverhiba

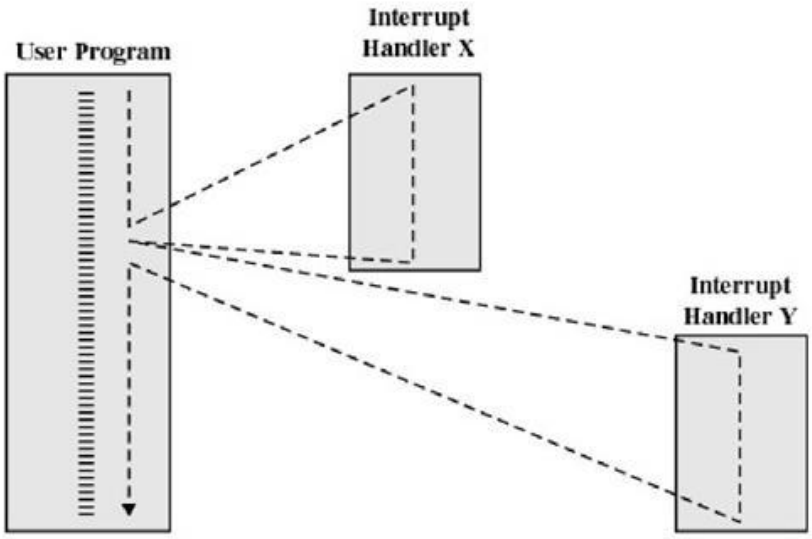
4.3. Megszakítási ciklus

- egy-egy gépi utasítás végrehajtása után a processzor „megvizsgálja / érzékeli, van-e megszakítás”
- ha nincs, betölti a program(címszámláló) szerinti soron következő utasítást a memóriából
- ha egy megszakítás függőben van, felfüggeszti a program végrehajtását, és elindítja a megfelelő megszakításkezelőt.

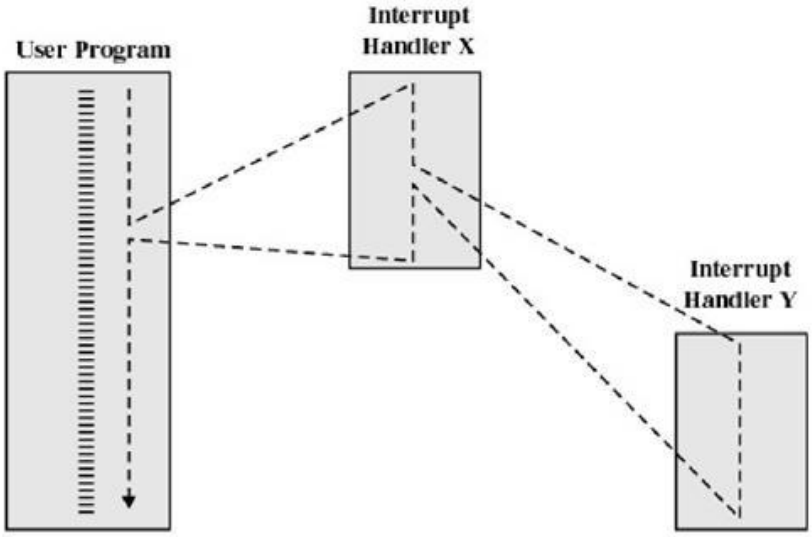
Ezt szemlélteti a következő ábra:

4.4. Többszörös megszakítás

- Újabb megszakítások letilthatók, amíg egy megszakításkérlem feldolgozás alatt áll, hogy el ne vesszenek (lost interrupt), ilyenkor a processzor figyelmen kívül hagy minden újabb megszakításkérést



(a) Sequential interrupt processing



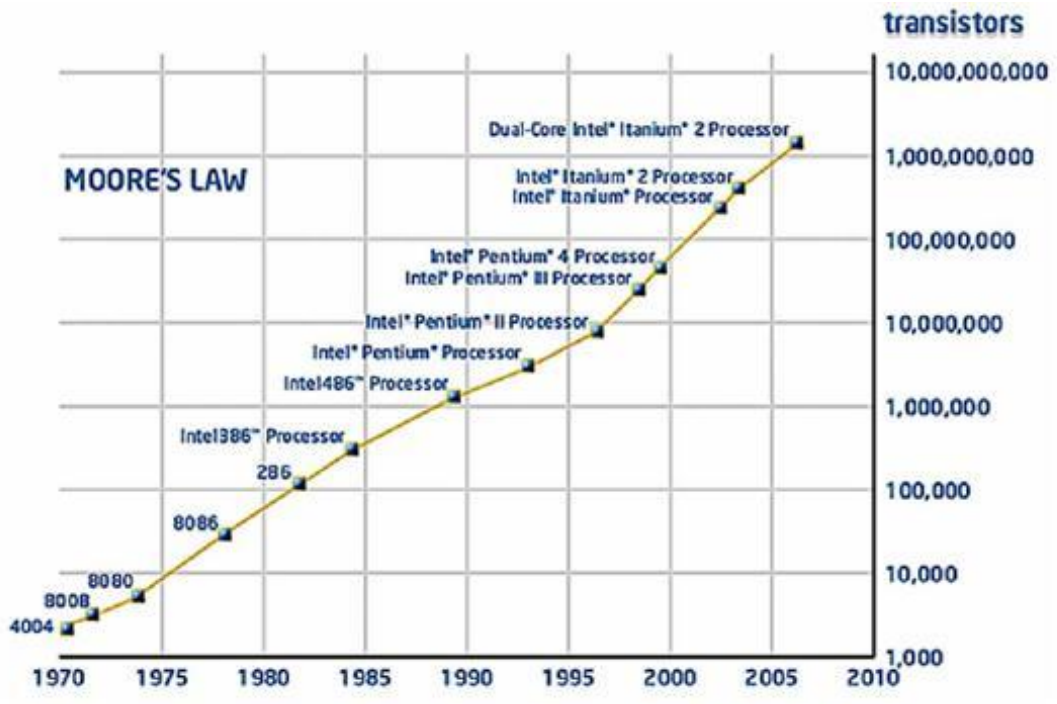
(b) Nested interrupt processing

4.5. Megszakítási sorrend és prioritás

- A megszakítások letilthatók (maszkolás), amíg a processzor befejez egy feladatot, és függőben maradnak addig, amíg a processzor újból engedélyezi a megszakítást
- A megszakításkezelő rutin feladatának elvégzése után a processzor további megszakítások fogadására kész
- A magas prioritású megszakítások várakozásra késztetik az alacsonyabb prioritású megszakításokat
- Alacsonyabb prioritású megszakításkezelő megszakítható
- Példa: egy kommunikációs csatornán való bevitelt gyorsan fogadni kell, hogy hely legyen a következő bevitelnek, (az I/O szűkkeresztmetszet!)

4.6. Multiprogramozás

Motivációs háttér: a processzorok és az adatátvitel sebessége jelentősen eltér egymástól és ez az eltérés csak növekszik (Moore törvénye) .



Moore törvényének egy megfogalmazása :

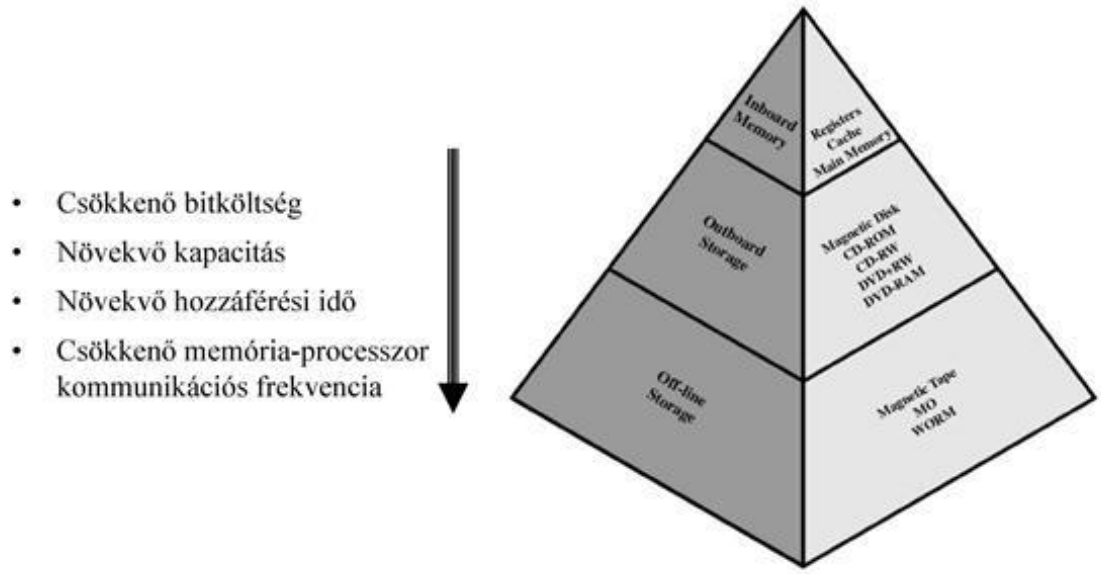
„az integrált áramkörökben lévő tranzisztorok száma minden 18. hónapban megduplázódik”.

A háttértárak sebessége azonban messze nem nő ilyen ütemben!

Multiprogramozás kivitelezése:

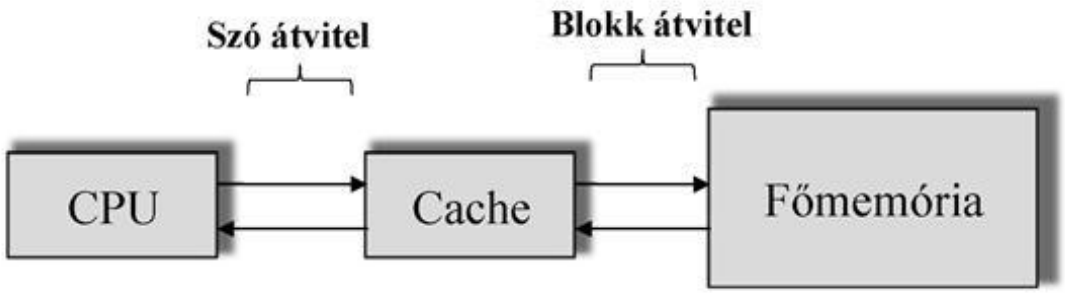
- a processzornak egynél több, az opratív memóriába betöltött programot kell végrehajtania
- a programok végrehajtásának sorrendje függ azok relatív prioritásától illetve attól, hogy várnak-e valamilyen I/O műveletre
- a megszakításkezelő (ütemező) rutin befejeztével a vezérlés nem feltétlenül kerül vissza ahhoz a programhoz, amelyik futása közben a megszakításkérés történt

5. Tárrendszer hierarchia



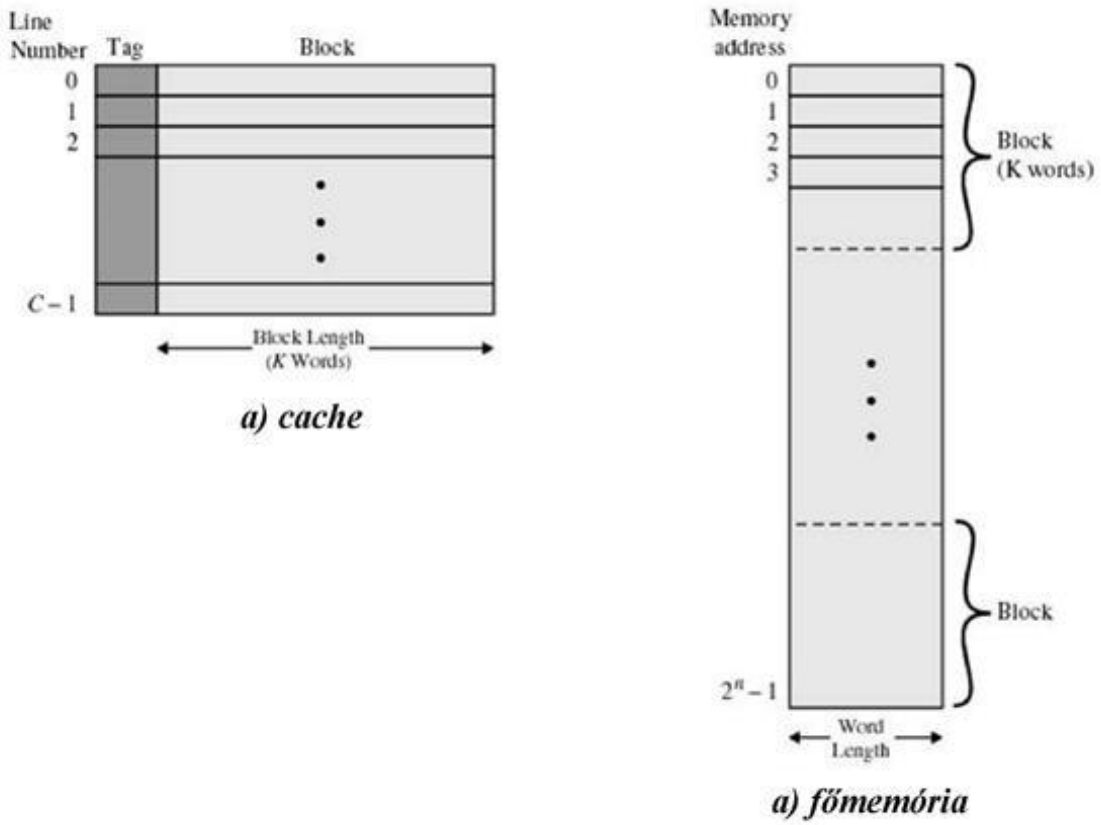
6. Gyorsítótár (cache)

- az operációs rendszer számára "láthatatlan"
- növeli a memóriaelérés sebességét
- a processzor sebessége nagyságrendekkel gyorsabb a memóriánál



- a főmemória és a processzor része is lehet (többszintű cache)
- a processzor először a cache-t ellenőrzi (cache „hit” és „miss”)
- ha a keresett adat nincs a cache-ben, a szükséges információ a főmemóriából a cache-be kerül

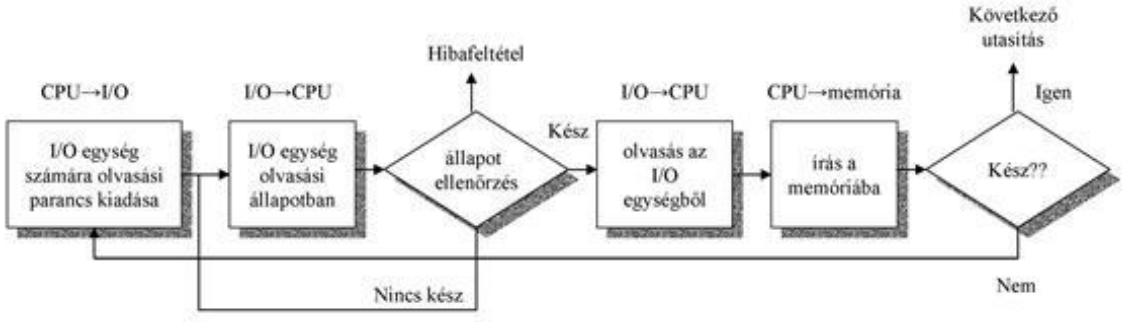
Cache/főmemória szerkezet



7. I/O kommunikációs technikák

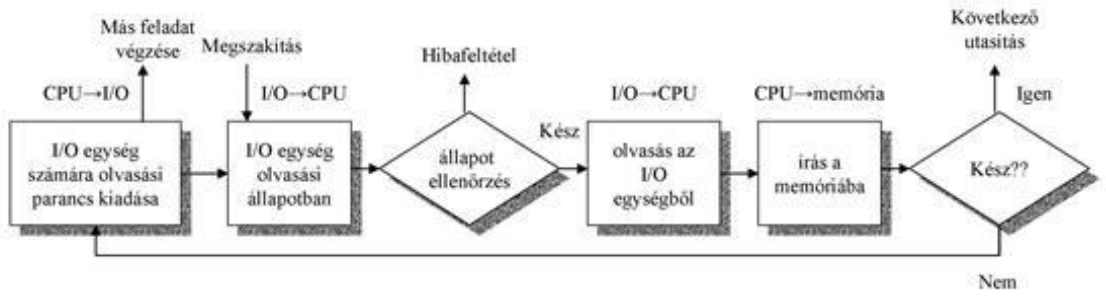
7.1. Programozott I/O

- az I/O modul végzi el a művelet, nem a processzor
- az I/O állapotregiszter bit értékeinek beállítása is megtörténik
- megszakítás nem lehetséges!
- a processzor ellenőrzi a művelet állapotát, amíg az be nem fejeződik



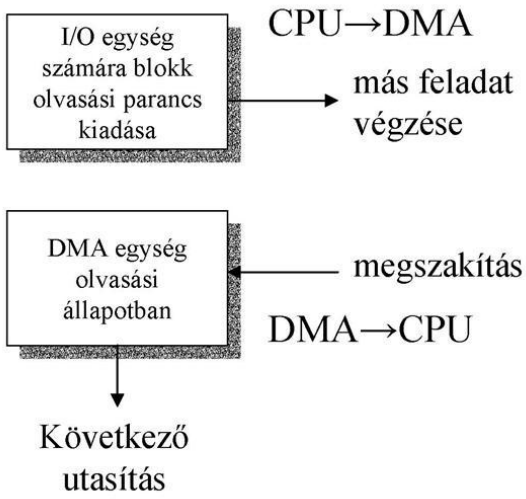
7.2. Megszakítás-vezérelt I/O

- ha egy I/O egység kész adatok cseréjére, a processzornak megszakítást küld
- a processzor más munkákkal foglalkozhat, így nincs haszontalan várakozás (busy waiting)
- még így is sok processzoridőt fogyaszt, mert minden olvasás és írás a processzoron keresztül történik



7.3. Közvetlen memória-hozzáférés (DMA)

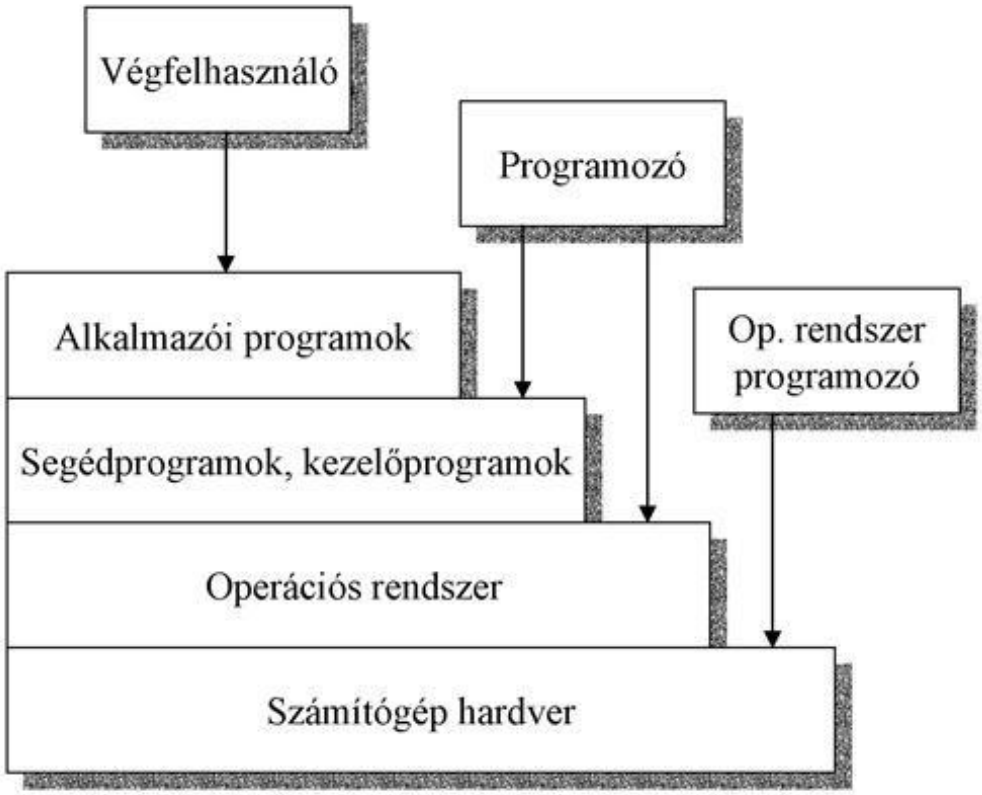
- A processzor engedélyezi az I/O számára a közvetlen memóriahozzáférést
- Adategységek (block) forgalma közvetlenül a memóriába (-ból)
- Megszakítás küldése, amikor a feladat befejeződött (megszakítás blokkonként, nem bájtonként!)
- A processzor csak az adattranszfer elején és végén van bevonva a folyamatba, így mentesíti a processzor az adatcsere felügyelete alól
- a processzor az adatátvitel közben foglalkozhat más feladatok elvégzésével



2. fejezet - Operációs rendszerek: áttekintés

- Operációs rendszer: egy program(rendszer), amely közvetítô szerepet játszik a számítógép felhasználója és a számítógép hardver között.
- Operációs rendszer célok:
  - Felhasználói programok végrehajtása, *a felhasználói feladat-megoldás megkönnyítése.*
  - A számítógép rendszer használatának *kényelmesebbé tétele*.
  - A számítógép hardver kihasználásának *hatékonyabbá tétele.*
- Megjegyzés: az operációs rendszer beszerzése és implementálása a felhasználónak "járulékos költséget (overhead)" jelent.

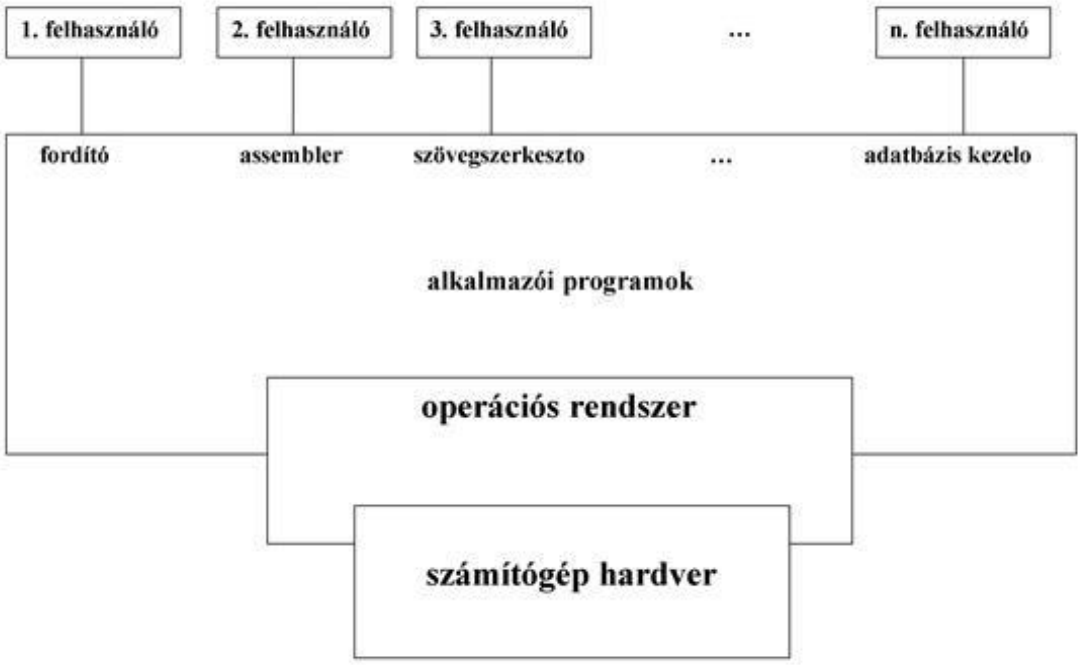
A számítógépes rendszer rétegei



Számítógép rendszerek komponensei (séma)

1. Hardver – az alapvetô számítási erôforrásokat nyújtja (CPU, operatív memória, I/O berendezések).
2. Operációs rendszer – koordinálja és vezérli a hardver erôforrások különbözô felhasználók különbözô alkalmazói programjai által történô használatát.
3. Alkalmazói programok – definiálják azt a módot, ahogyan az egyes rendszer-erôforrásokat a felhasználók számítási problémáinak megoldásához föl kell használni (fordítók, adatbázis kezelôk, videó játékok, ügyviteli programok).

4. Felhasználók (emberek, gépek, más számítógépek).

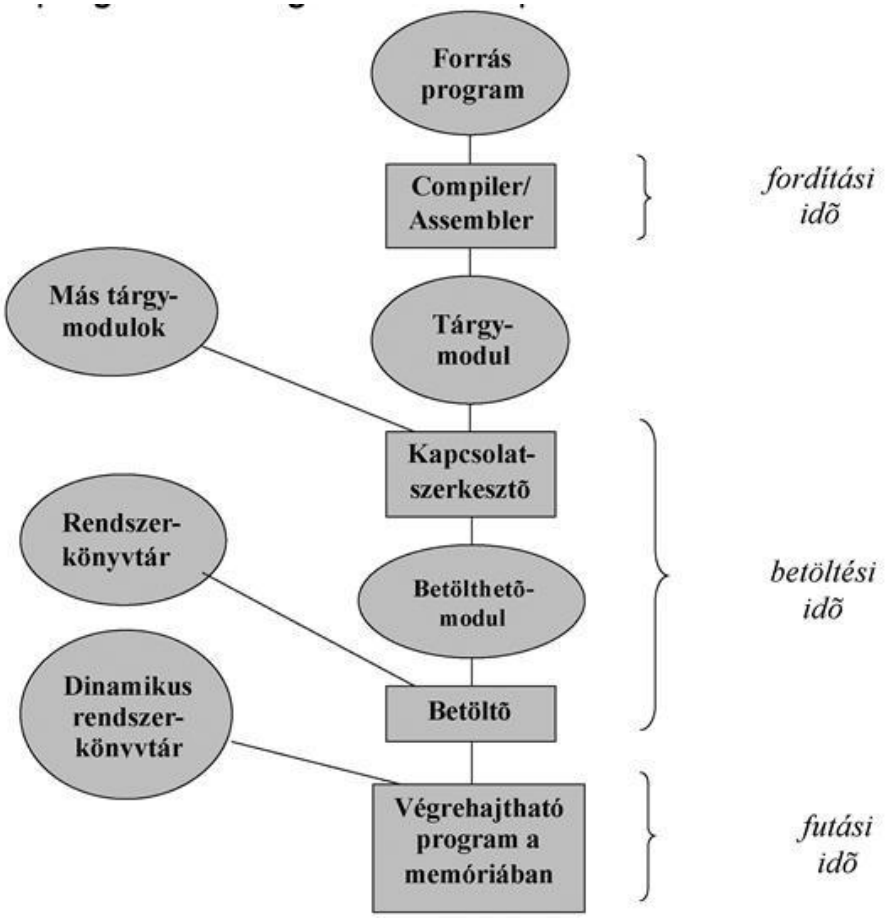


1. Az operációs rendszer szolgáltatásai

- Program végrehajtás (program betöltés és futtatás)
- I/O műveletek (fizikai szint: blokkolás, pufferezés)
- Fájl-rendszer manipuláció (r, w, c, d)
- Kommunikáció – a folyamatok közötti információ csere (ugyanazon, vagy különböző gépeken): Shared memory – Message passing
- Hiba detektálás (CPU, memória, I/O készülékek, felhasználói programok, ...)
- Nem közvetlenül a felhasználó támogatását, hanem a hatékonyabb rendszerműködést segítik:
  - erőforrás kiosztás
  - multiprogramozás, többfelhasználós mûködés
- Accounting – rendszer és felhasználói statisztikák.
- Védelem – minden erőforrás csak az operációs rendszer felügyelete mellett érhető el.

2. Az operációs rendszerek evolúciója

A programok feldolgozásának lépései: fordítás, szerkesztés futtatás



A programok feldolgozásának módjai (soros, kötegelt, időosztásos)

- Soros feldolgozás
  - nincs operációs rendszer; a felhasználó közvetlenül a hardvert éri el, a programozó egyben operátor is, egyfelhasználós rendszer
  - problémák: drága erőforrások alacsony hatásfokú kihasználása (hosszú beállítási idő, gyenge CPU kihasználtság)
- Kötegelt feldolgozás (Simple Batch)
  - Monitorok
    - a futó programok vezérlésére használtos szoftver
    - feladatok egymáshoz kapcsolása
    - a program visszaadja a vezérlést a monitornak amikor befejeződik
    - a felügyelőprogram mindig a memóriában van és futásra kész
- Időosztásos rendszerek (Time Sharing)
  - több felhasználó / program használhatja a CPU-t egymás után, azonos ideig

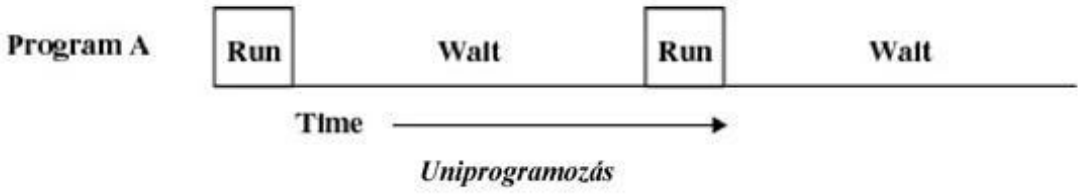
2.1. A kötegelt feldolgozás

- Rezidens monitor
  - a munkák (job) vezérlésére használatos program, mely állandóan a memóriában van és futásra kész

  - a felhasználó nem operátor, a beállítási idő csökkentésének érdekében a munkákat egymáshoz kapcsolva, egymás után végezzük el
  - minden program végrehajtása végeztével visszaadja a vezérlést a monitornak, mely ezután automatikusan betölti (loader) a következő munkát
- Job Control Language (JCL)
  - speciális programozási nyelv, mely a monitor számára biztosít utasításokat (Pl.: mely forrásnyelvi adatokon milyen fordítót használjon)
- Szükséges hardverjellemzők:
  - memóriavédelem: a monitort tartalmazó memóriaszegmens megváltoztatásának letiltása
  - időzítés: jobok meggátolása abban, hogy kisajátítsák a rendszert
  - lefoglalt (privilégizált) utasítások: csak a monitor által használható utasítások
  - megszakítások: rugalmasságot biztosítanak a felhasználói programok vezérléséhez

Egyfeladatos feldolgozás:

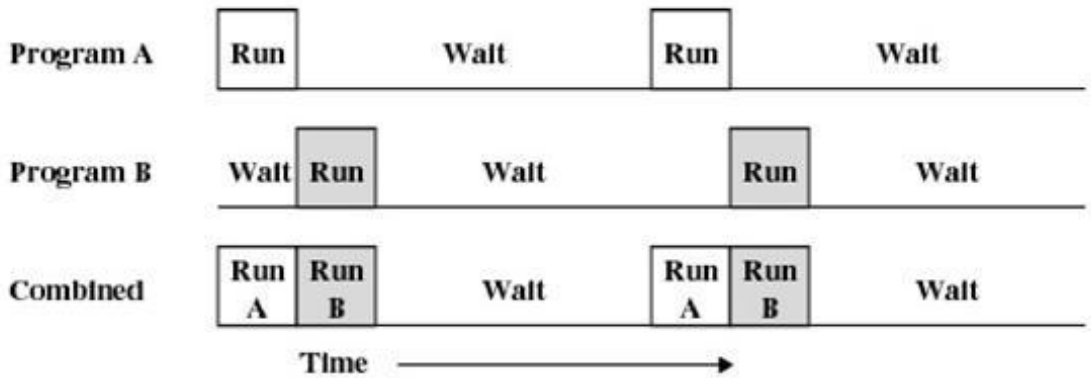
- a processzornak várnia kell egy I/O utasítás befejeződésére mielőtt továbblép, az I/O és a CPU műveletek nem fedhetik át egymást
- probléma: I/O lassú a processzorhoz képest (pl. kártyaolvasó lassú), CPU nem megfelelően kihasznált



Uniprogramozás

Többfeladatos feldolgozás (multitaszking):

- Egy időben több program is található a főmemóriában: ha egy programnak I/O műveletre kell várnia, a processzor átvált egy másik program végrehajtására (nem párhuzamos futás!)
- A CPU idő kiosztása valamilyen stratégia szerint történik
- bizonyos hardverelemek szükségesek (I/O megszakítás támogatása)



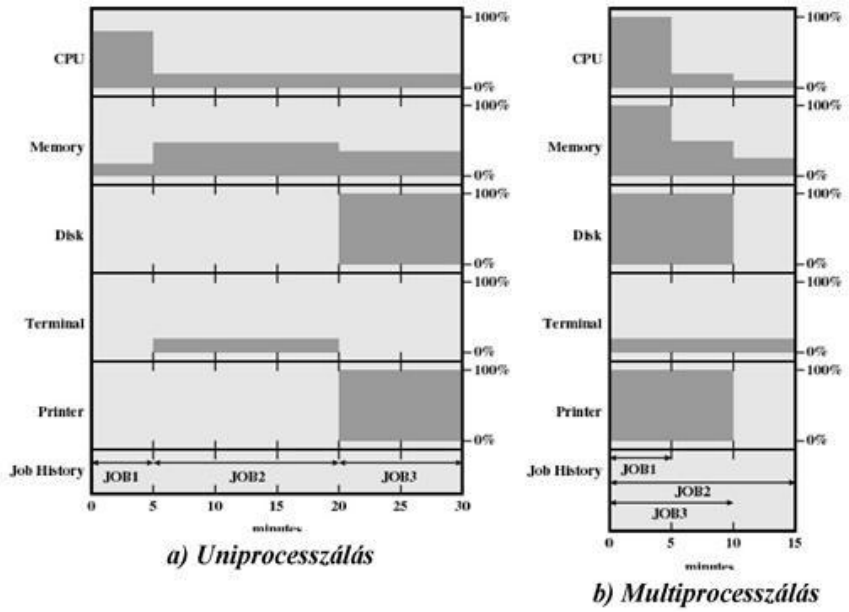
Multiprogramozás

A multiprogramozás által az operációs rendszerekkel szemben támasztott követelmények

- Az I/O-nak az operációs rendszer részéről történő teljes körű felügyelete. (adatvédelem!)
  - Az I/O-t az operációs rendszer nem egyszerűen támogatja, hanem végrehajtásához elkerülhetetlen.
  - Hardver feltételek: kernel/supervisor mode, privileged operations
- Memória gazdálkodás
  - a rendszernek fel kell osztania a memóriát a futó jobok között.
  - Hardver feltételek: kernel/supervisor mode, privileged operations, segmentation
- CPU ütemezés
  - a rendszernek választani kell tudni a futásra kész jobok között.
- Készülékhozzárendelés
  - Nem “jut” minden jobnak, printer, lemez, stb.

Uniprocesszálás és multiprocesszálás összehasonlítása:

	MUNKA1	MUNKA2	MUNKA3
Munka típusa	Számítás	I/O	I/O
Időtartam	5 perc	15 perc	10 perc
Szükséges mem.	50K	100 K	80 K
Kell lemez?	Nem	Nem	Igen
Kell terminál?	Nem	Igen	Nem
Kell nyomtató?	Nem	Nem	Igen



a) Uniprocesszálás

b) Multiprocesszálás

Kihasználtság uniprocesszálás és multiprocesszálás esetén

	Uniprocesszálás	Multiprocesszálás
Processzorhasználat	22%	43%
Memóriahasználat	30%	67%
Lemezhasználat	33%	67%
Nyomtatóhasznákat	33%	67%
Eltelt idő	30 min.	15 min.
Frekvencia	6 jobs/hr	12 jobs/hr
Átlagos válaszidő	18 min.	10 min.

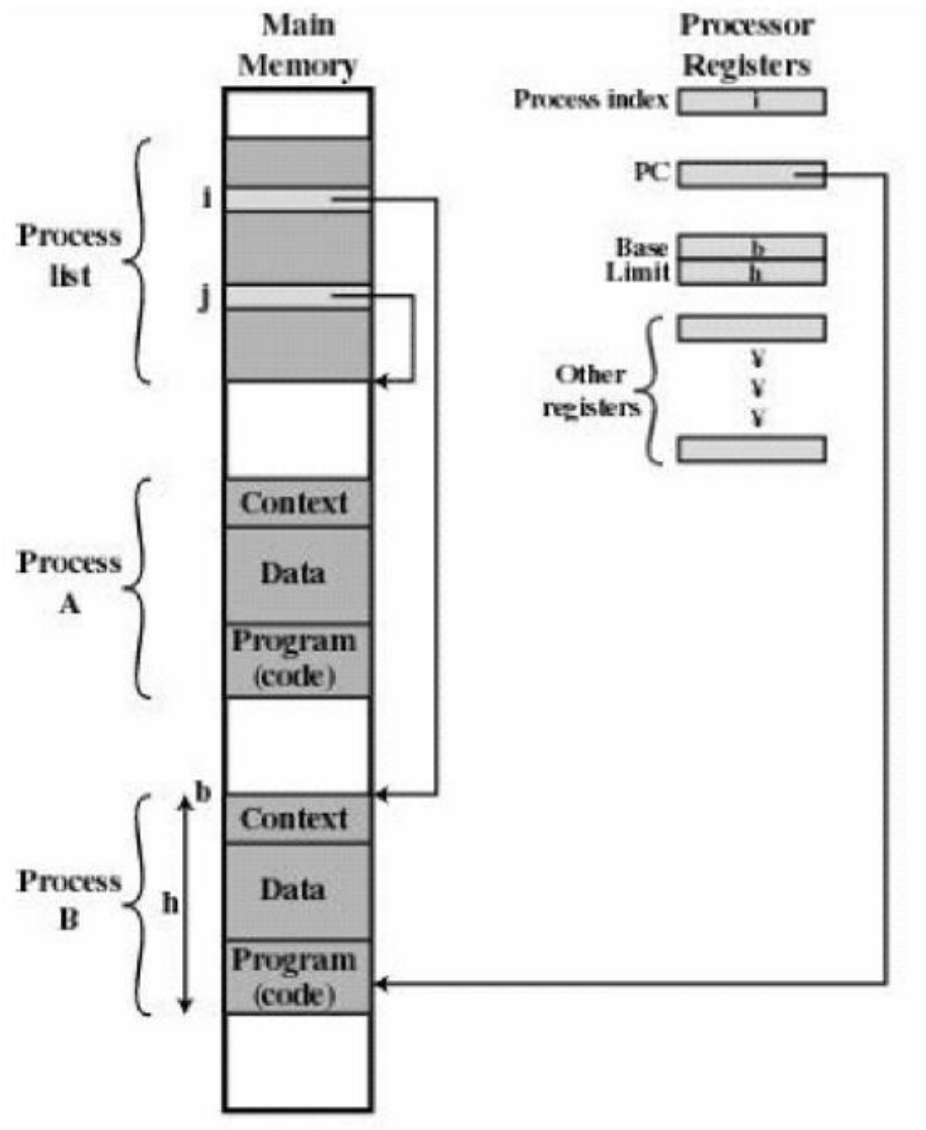
2.2. Időosztásos rendszerek

- A kötegelt rendszerek hátránya: nincs interaktivitás!
- a CPU váltakozva áll olyan joboknak a rendelkezésére, melyek a memóriában, vagy a lemezen találhatók. (a CPU-t csak olyan job kaphatja meg, amely éppen a memóriában van.)
- Egy job a lemezről a memóriába, ill. a memóriából a lemezre betölthető/ kimenthető az ütemezési stratégiának (időosztás!) megfelelően. (Process!)
- A rendszer és a felhasználó között online kommunikációt tételezünk fel; ha az operációs rendszer befejezi egy parancs végrehajtását, a következő „vezérlő utasítás”-t nem a kártyaolvasóról, hanem a felhasználó klaviatúrájáról várja.
- A processzoridő több felhasználó között van megosztva
- Több felhasználó együttesen éri el a rendszert terminálok használatával (interaktivitás)
- Egy – adatokat és utasításkódokat tároló – online fájlrendszer áll a felhasználók rendelkezésére.

3. Operációs rendszer komponensek

3.1. Folyamatkezelés, processzusok, folyamatok

- Processzus: végrehajtás alatt álló program. A processzusnak bizonyos erőforrásokra (pl. CPU idő, memória, állományok, I/O berendezések) van szüksége, hogy a feladatát megoldhassa.
- Egy végrehajtható programból, a hozzákapcsolódó adatokból és a végrehajtási környezetből tevődik össze (az összes információ, ami ahhoz szükséges, hogy az operációs rendszer kezelni tudja a processzust)
- Az operációs rendszer az alábbi tevékenységekért felel a processzusok felügyeletével kapcsolatban:
  - processzus létrehozása és törlése
  - processzus felfüggesztése és újraindítása
  - eszközök biztosítása a processzusok szinkronizációjához és kommunikációjához



Egy processzus tipikus végrehajtása

3.2. Memóriakezelés

- Az operációs rendszerek szempontjából az operatív memóriát bájtokból (szavakból) álló (absztrakt) tömbnek tekintjük, amelyet a CPU és az I/O vezérlő megosztva (közösen) használ.
- Processzusok elszigetelése
  - egymástól független processzusok ne legyenek egymásra hatással
- Automatikus kiosztás és kezelés
  - a memória kiosztása a programozó számára átlátható legyen
- Moduláris programozás támogatása
- Védelem és hozzáférésvezérlés
  - a memória felosztása lehetővé teszi, hogy egy program megcímezzen egy másik programhoz tartózó memóriateret (veszélyeztetheti egyes programok integritását)

- Az operációs rendszer a következőkért felelős a memóriakezelést illetően:
  - nyilvántartja, hogy az operatív memória melyik részét ki (mi) használja
  - eldönti, melyik processzust kell betölteni, ha a memória felszabadul
  - szükség szerint memóriaterületeket foglal le és szabadít fel a szükségleteknek megfelelően

3.3. Másodlagos tár kezelés

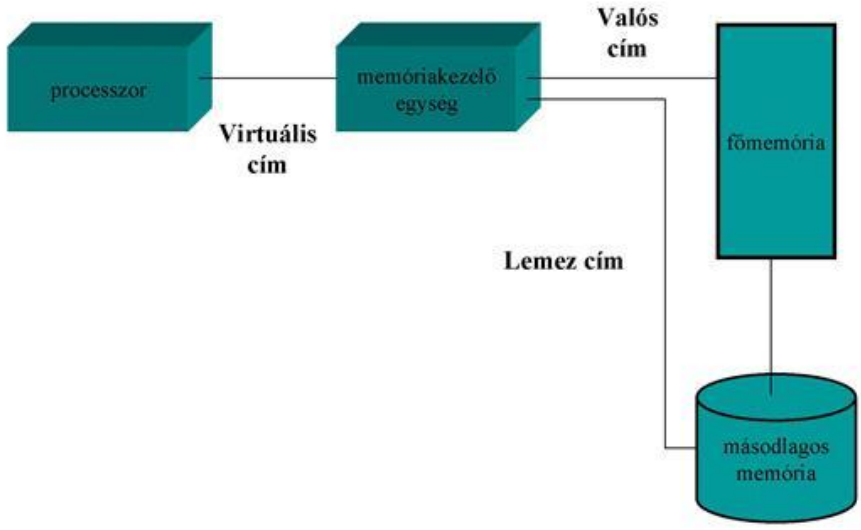
- Mivel az operatív tár (elsődleges tár) törlődik (és egyébként sem alkalmas arra, hogy minden programot/adatot tároljon), másodlagos tárra van szükség.
- A merevlemezes tár a másodlagos tár legelterjedtebb megjelenése
- Az operációs rendszer a következőkért felelős a másodlagos tár kezelését illetően:
  - Szabadhely kezelés
  - Tárhozzárendelés (allokálás)
  - Lemezelosztás, ütemezés (scheduling)

Fájlrendszer kezelés:

- az információ (adat) névvel rendelkező objektumokban, a fájlokban tárolódik
- egy fájl kapcsolódó információ (adatok) együttese, amelyet a létrehozója definiál
- az operációs rendszer a következőkért felelős a fájlkezelést illetően:
  - fájl, könyvtár létrehozása és törlése
  - fájlokkal és könyvtárakkal történő alap-manipulációhoz nyújtott támogatás
  - fájlok „leképezése” a másodlagos tárba
  - fájlok mentése stabil adathordozóra.

3.4. Virtuális memória

- Logikai szempontok szerinti memóriacímzést biztosít a programok számára
  - nem kell tekintettel lenni arra, hogy mennyi fizikailag elérhető főmemória áll rendelkezésre
- Egy program úgyis „futhat”, hogy a program és a hozzákapcsolódó adatok egy része a lemezen tárolódik
  - a program mérete akár nagyobb lehet, mint az egész főmemória mérete
- Lapozó rendszer (paging system)
  - a programok (logikai címtartománya) fix méretű blokkokra vannak osztva (szeletelve!), ezek a lapok (page)
  - a virtuális cím egy lap sorszámból és a lapon belüli eltolásból (offset) áll
  - az egyes lapok bárhol elhelyezhetők a főmemóriában (keret, frame)
    - a lapozó rendszer dinamikus hozzárendelést szolgáltat a virtuális és a fizikai cím között



A virtuális memória címzése

3.5. Az operációs rendszer egyéb feladatai

- Információvédelem és biztonság
  - hozzáférés vezérlése (access control): a felhasználó rendszerhez való hozzáférésének szabályozása
  - információáramlás vezérlése: a rendszeren belüli adatáramlás vezérlése és az adatok felhasználóhoz történő szállításának végzése
  - igazolása annak, hogy a hozzáférés és az adatáramlás vezérlése az előírásoknak megfelelően működik
- Ütemezés és erőforráskezelés elvei
  - méltányosság: az összes processzus számára egyenlő és korrekt hozzáférést biztosítani különböző érzékenység: a
  - különböző típusú munkák között különbséget lehet és kell tenni
  - hatásosság: cél a teljesítmény maximalizálása, a válaszidő minimalizálása, és a lehető legtöbb felhasználó kiszolgálása

4. Modern rendszerek jellemzői

- Mikrokernel architektúra
  - a kernel csak néhány alapvető szolgáltatást nyújt
    - alapvető ütemezési feladatok
    - processzusok közötti kommunikáció (interprocess communication - IPC)
- Multithreading
  - a processzusok szálakra osztása, mely szálak szimultán képesek futni
- Objektum-orientált kivitelezés
  - a kis kernelhez való moduláris kiterjesztések hozzáadásának lehetősége

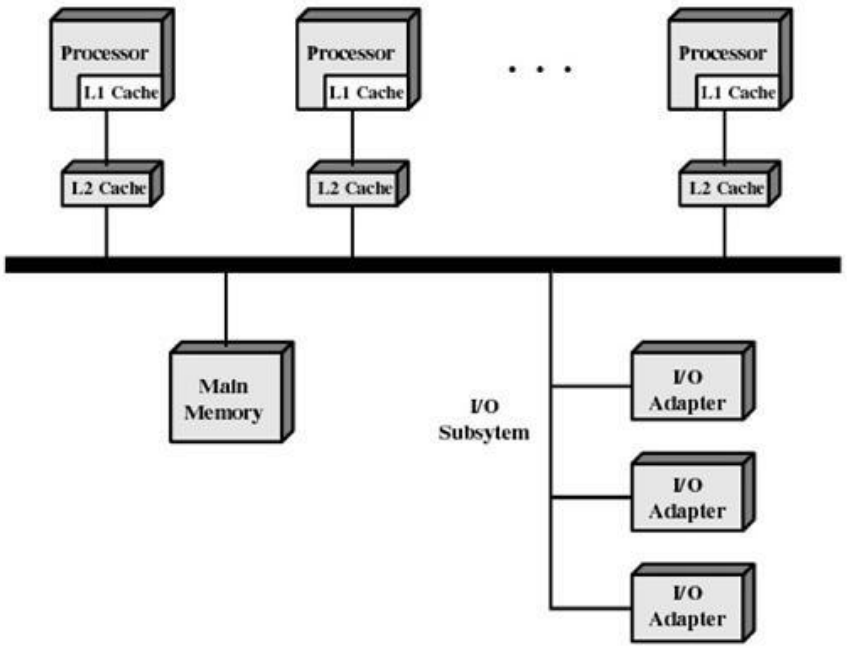
- a programozó testre szabhatja az operációs rendszert anélkül, hogy a rendszerintegritást veszélyeztetné

Párhuzamos rendszerek

- Szimmetrikus multiprocesszálás
  - több processzor, melyek ugyanazon főmemórián és I/O rendszeren osztoznak
  - minden processzor az operációs rendszer azonos változatát (másolatát) futtatja, melyek egymással szükség szerint kommunikálnak
  - több processzus futhat egyszerre teljesítménycsökkenés nélkül
  - I/O és ütemezési problémák léphetnek fel
- Asszimetrikus multiprocesszálás
  - minden processzor a hozzárendelt specifikus feladatot (task) oldja meg
  - a taskok egymással kommunikálhatnak.

Elosztott rendszerek

- A számításokat több processzor között osztják meg
  - lazán kapcsolt/csatolt rendszerek – a processzorok saját lokális memóriát és rendszer órát használnak. A kommunikáció nagy kapacitású adatvonalak, vagy telefonvonalak segítségével történik
  - elosztott rendszerek előnyei: erőforrás megosztás, számítási teljesítmény növelés, túlterhelés védelem, növekvő megbízhatóság, kommunikáció



Szimmetrikus multiprocesszálás

Valós idejű rendszerek (real-time)

- gyakori megjelenési formája valamilyen dedikált alkalmazás (pl. tudományos kísérlet támogatása, orvosi képfeldolgozás, ipari kontroll, kijelző rendszerek) irányító-felügyelő rendszere
- a „kiszolgálás” azonnal megkezdődik! Jól definiált, rögzített idejű korlátozások vannak

- „hard” („merev" valós idejű) rendszerek
  - a másodlagos tár korlátozott, vagy teljesen hiányzik; az adatokat az operatív memóriában (RAM), vagy akár ROM-ban tárolják
  - fogalmi konfliktus az időosztásos rendszerekkel
- szoft” („puha" valós idejű) rendszerek.
  - korlátozott szolgáltató programok az ipari kontroll, a robotika területén
  - a fejlett operációs rendszer szolgáltatásokat igénylő alkalmazásoknál (Multimédia, VR, AR) igen hasznosak.

5. A Windows 2000 és a Unix

5.1. A Windows 2000

- a 32 bites mikroprocesszorok teljesítményének kiaknázására fejlesztették ki
- teljes többfeladatos feldolgozást biztosít egyfelhasználós környezetben
- kliens/szerver modell megvalósíthatóság

Windows 2000 architektúra:

- moduláris szerkezet a rugalmasság érdekében
- sokféle hardverplatformon képes futni
- más operációs rendszerekre írt alkalmazások bő választékát támogatja
- módosított mikrokernel architektúra
  - nem teljesen szabályos mikrokernel arcutektúra
  - módosítás: több, mikrokernelen kívüli rendszerfüggvény is kernel módban fut
- bármelyik modul kivehető, frissíthető, vagy helyettesíthető a rendszer újraírása nélkül

Réteges szerkezet:

- Hardver absztrakciós réteg (Hardware abstraction layer - HAL)
  - elkülöníti az operációs rendszert a platformfüggő hardverkülönbségektől
- Mikrokernel
  - az operációs rendszer legtöbbet használt illetve legalapvetőbb komponenseit tartalmazza
- Eszközkezelők (device driver)
  - a felhasználói I/O függvényhívásokat fordítja le specifikus I/O hardvereszközök felé irányuló kérelmekké

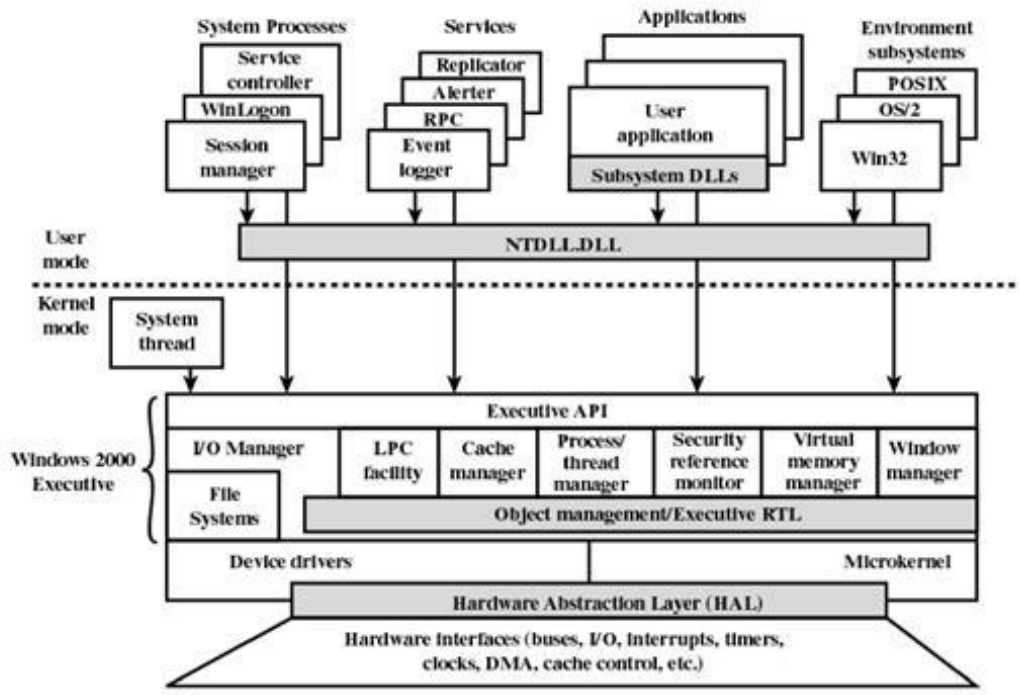
Adminisztratív modulok:

- I/O kezelő, objektumkezelő, biztonsági monitor, processzus/szál menedzser, helyi eljáráshívó (local procedure call - LPC) szolgáltatás, virtuális memóriakezelő, gyorsítótár kezelő, grafikai modulok

Felhasználói processzusok típusai:

- rendszert támogató processzusok (bejelentkezés, session manager)

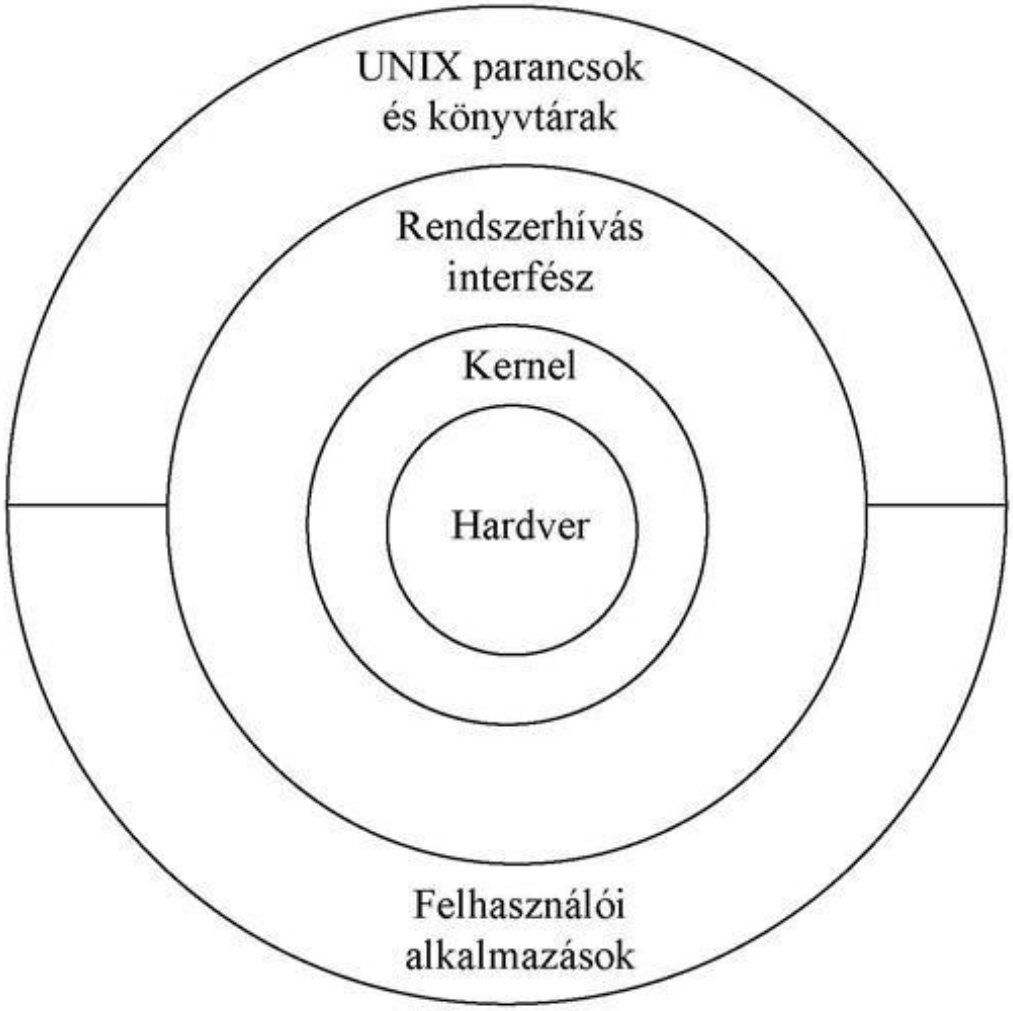
- szerver processzusok,
- környezeti alrendszerek processzusai,
- felhasználó alkalmazások



Windows 2000 architektúra

5.2. A Unix

- az operációs rendszer lefedi a teljes hardvert
- az operációs rendszert gyakran csak kernelnek (mag) hívják
- sok felhasználói szolgáltatás és interfész
  - héj (shell)
  - C fordító



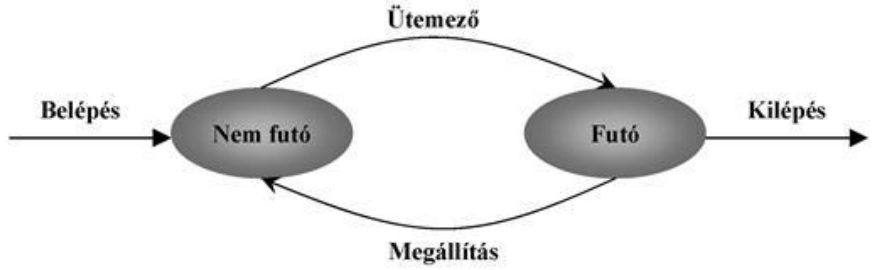
Unix architektúra

3. fejezet - Processzus leírás és vezérlés

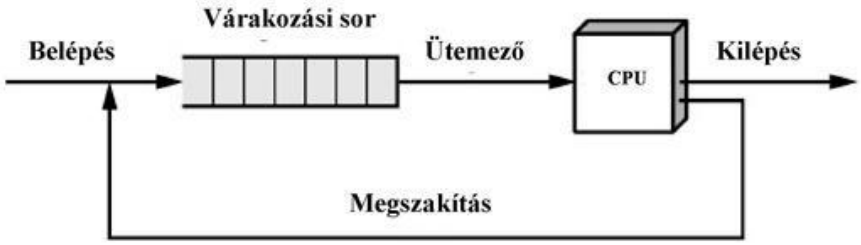
1. Processzus állapotok

Folyamat (processzus): végrehajtás alatt álló program. Alapvetően két állapotban lehet: futó, nem futó.

1.1. Két állapotú processzus modell



A kétállapotú processzus modell



A várakozási sor (queue)

1.2. Processzusütemezés és létrehozás

- Ütemező (dispatcher):
  - – program, mely a processzor processzusokkal való ellátását végzi
  - – megszakítás vagy processzusfelfüggesztés esetén a várakozási sorból választ ki végrehajtásra egy másik processzust
  - – megóvja a rendszert attól, hogy egy processzus kisajátítsa a processzoridőt
- Processzus létrehozása:
  - – Az operációs rendszer létrehozza a processzus kezeléséhez szükséges adatszerkezetet és a főmemóriából címteret foglal le a processzus számára.
  - – Okai:
    - új kötegelt munka (batch job) benyújtása
    - új felhasználó terminálról való bejelentkezése
    - az operációs rendszer által létrehozott processzusok valamilyen szogáltatásnyújtás érdekében (pl. nyomtatásvezérlés)

- egy már létező processzus is létrehozhat processzust (egymással kapcsolatban álló processzusok kommunikációját meg kell oldani!)

1.3. Processzusmegállítás (befejezés)

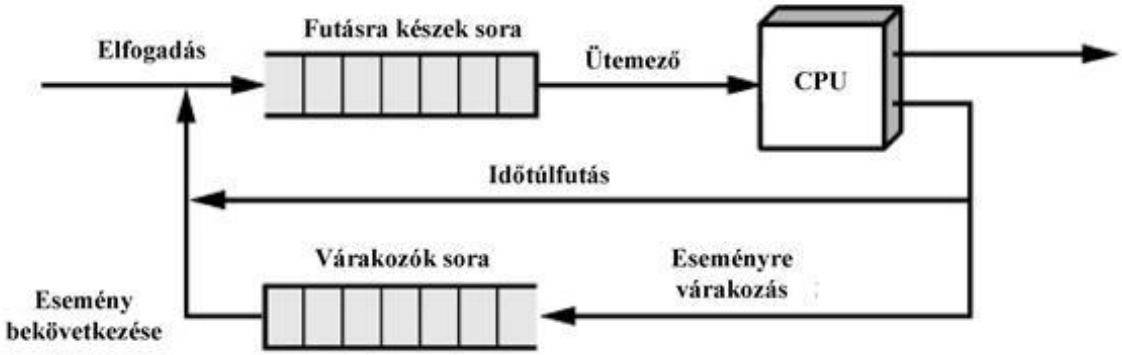
- Kötegelt munka kiadja a "Halt/Stop" utasítást
- Egy felhasználó kijelentkezik
- Alkalmazásból való kilépés
- Bizonyos hibafeltételek teljesülése
- A megállítás/befejezés okai lehetnek:
  - Normális processzusbefejezés
  - Időhatár túllépése
  - Memória nem áll rendelkezésre
  - Memóriahatárok megsértése (nemlétező cím, bounds violation, segmentation fault)
  - Védelmi hiba: például írás csak olvasható fájlba
  - Számolási hiba
  - Időtúllépés
  - I/O hiba
  - Érvénytelen utasítás: adat „végrehajtása”
  - Privilegizált utasításvégrehajtásának megkisérlése: az utasítás csak kernel (operációs rendszer) módban hajtható végre
  - Használhatatlan adatsor
  - Operációs rendszer beavatkozása (preempció)
  - Szülő processzus és így az utód processzus is megszakad (kaszkád termináció)
  - Szülő processzus által történő megszakítás

1.4. Öt állapotú processzus modell

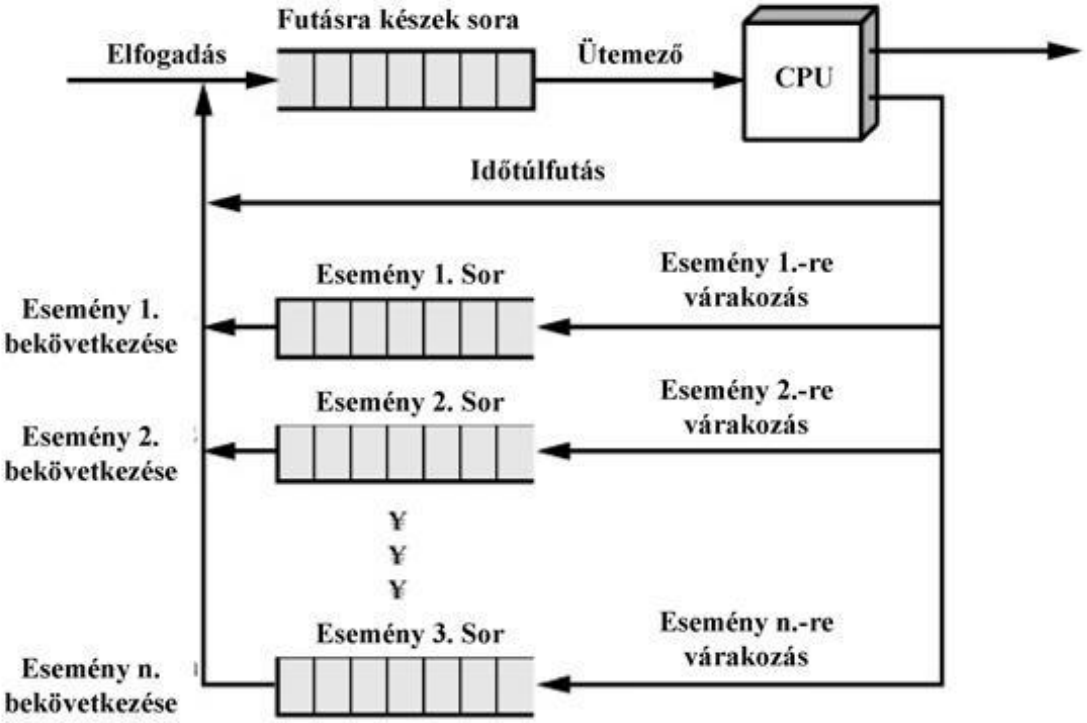
- A két állapotú modell elégtelensége:
  - – néhány nem-futó állapotban levő processzus készen áll a végrehajtásra, míg mások blokkolva vannak (I/O várakozás)
  - – az ütemező nem választhat csak úgy processzust a lista legvégéről
  - – az ütemezőnek végig kellene vizgálnia a listát a legrégebbi nem blokkolt processzus után keresve
  - – nem futó processzusok kettéválasztásának szükségessége:
    - futásra kész (ready) állapot és blokkolt (blocked) állapot
- A processzusok öt állapota:
  - futó (running)

- futásra kész (ready)
- blokkolt, vagy eseményre (I/O) várakozó (blocked)
- új (new): újonnan létrehozott processzus, mely nincs még a főmemóriában
- befejezett (terminated): processzus, melyet az operációs rendszer kivon a végrehajtandó processzusok közül

1.5. Várakozási sor használata



Egyszeres várakozási sor



Többszörös várakozási sor

1.6. Processzusfelfüggesztés

- A processzor sokkal gyorsabb, mint az I/O rendszer, így előfordulhat, hogy az összes processzus I/O-ra vár (a processzor üresjáratban van....)
- Ezen processzusok memóriából lemezre történő mozgatásával memória szabadítható fel új processzusok számára (swap in, swap out) - SWAPPING
- A processzus lemezre történő áthelyezésével a processzus blokkolt állapotból felfüggesztett állapotba kerül
- Felfüggesztett lista (suspended queue): felfüggesztett processzusok listája

1.7. Két felfüggesztett állapot

- Probléma: egy felfüggesztett processzus időközben futásra késszé válhat
- Két új állapot szükséges:
  - – blokkolt, felfüggesztett
  - – futásra kész, felfüggesztett

1.8. A processzusfelfüggesztés okai

- Swapping:
  - – az operációs rendszernek főmemóriát kell felszabadítani, hogy egy készen álló processzust be tudjon tölteni
- Egyéb operációs rendszerhez köthető okokból:
  - – például az operációs rendszer felfüggeszthet olyan processzust, amely egy hiba okozásával gyanúsítható

- Interaktív felhasználói kérelem:
  - – egy felhasználó a program végrehajtásának felfüggesztését kérheti (pl. erőforráshasználati okok miatt)
- Időzítés:
  - – olyan processzus ideiglenes felfüggesztése, mely periodikusan hajtódik végre (naplózó illetve rendszermonitorozó processzusok)
- Szülő processzus általi kérelem:
  - – egy szülő processzus felfüggesztheti az utód processzust annak vizsgálata illetve megváltoztatása céljából

2. Processzus vezérlés

2.1. Processzusleírás

- Az operációs rendszernek információra van szüksége a processzusok és erőforrások pillanantnyi állapotáról
- Az operációs rendszer az általa felügyelt egységekhez táblázatokat rendel
- Négy ilyen táblázat (operációs rendszer függő):

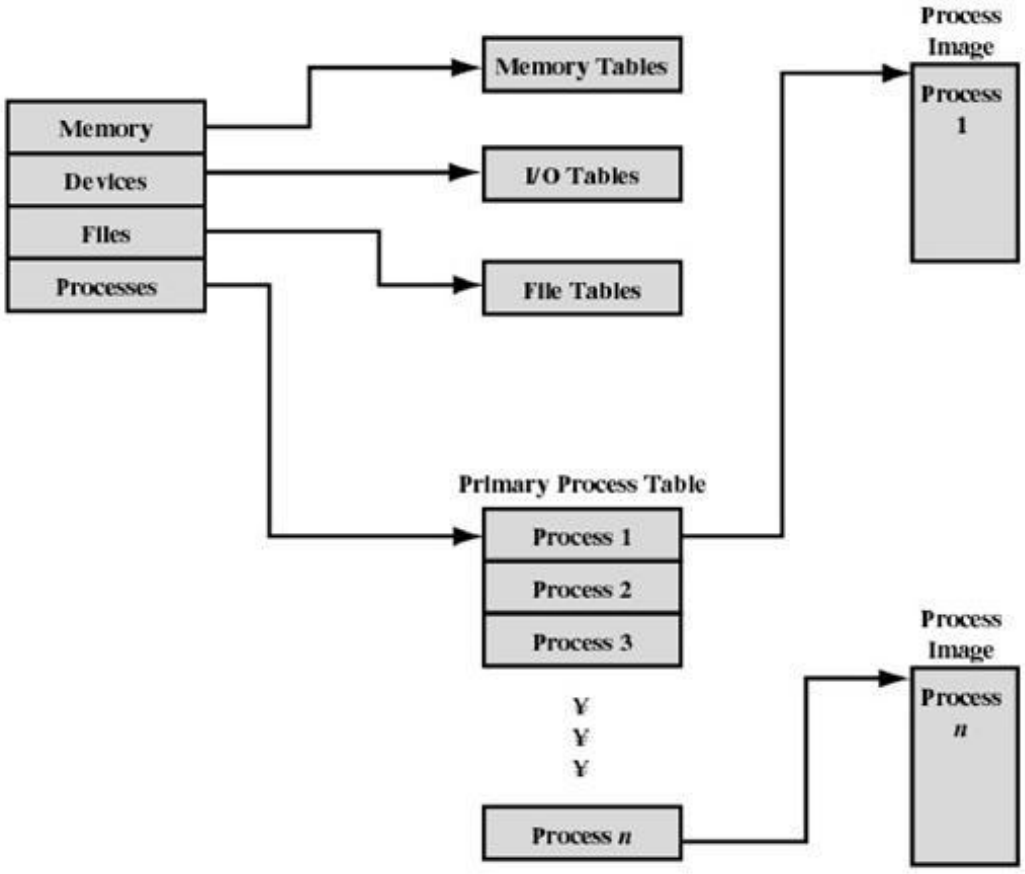
2.2. A processzustábla

- Hol található meg a processzus?
- Jellemzők, melyek szükségesek a processzus kezeléséhez:
  - – processzus azonosító (ID)
  - – processzus állapot
  - – elfoglalt memóriaterület

Processzuskép (Process Image):

- Felhasználói adat
  - – lokális és globális változók illetve definiált konstansok számára fenntartott adat területek
- Felhasználói program
  - – a processzus során végrehajtandó program(ok)

- Rendszer verem (System stack)
  - – rendszerhívások paramétereinek tárolása
- Processzusvezérlő blokk (Process Control Block - PCB)
  - – az operációs rendszer számára a processzus vezérléséhez szükséges adatok

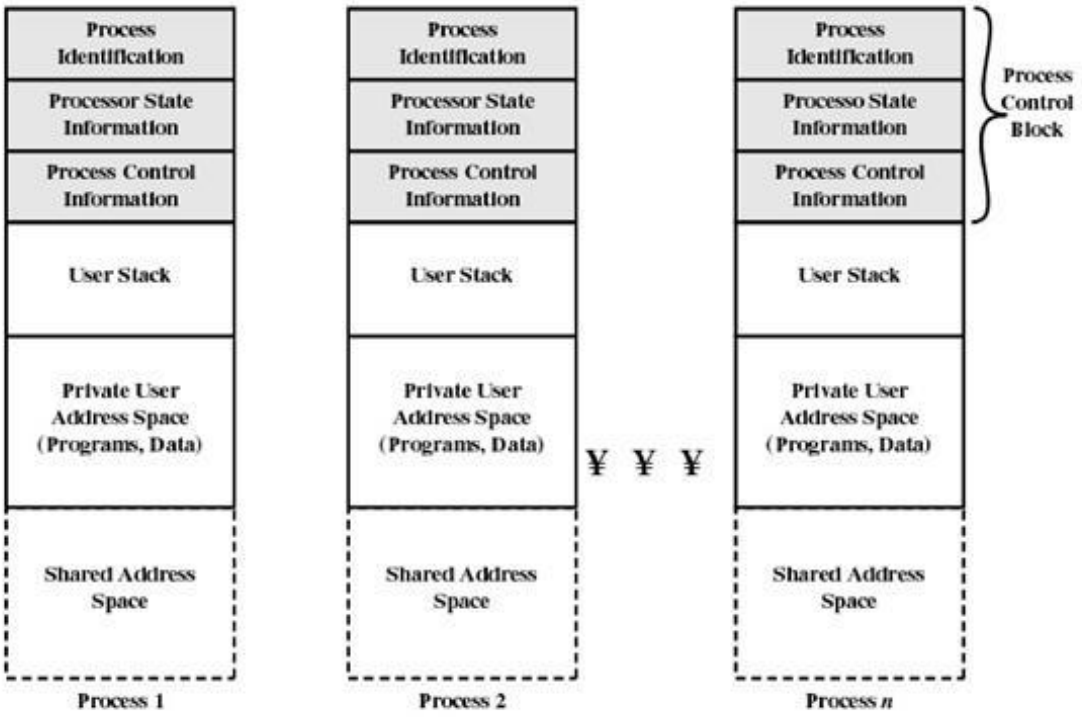


Az operációs rendszert vezérlő táblák

2.3. A processzusvezérlő blokk elemei

- Processzusazonosítás
  - – processzusazonosító: egyedi numerikus azonosító
    - az elsődleges processzustábla egy indexe is lehet
  - – szülőprocesszus azonosítója
  - – felhasználóazonosító
- Processzorállapot információ (*Processor State Information*)
  - – felhasználó által látható regiszterek állapota
  - – vezérlő- és státuszregiszterek állapota: processzorregiszterek, melyek a processzor működését vezérlik
    - programszámláló: a következő végrehajtandó utasítás címét tartalmazza

    - állapotkód: a legutolsó aritmetikus vagy logikai művelet eredményét tartalmazza (előjel, nulla, átvitel, egyenlő, túlcsordulás)
    - státuszinformáció: megszakítás bekapcsolva/kikapcsolva, végrehajtó mód
  - – veremmutatók (Stack Pointer) állapota
    - minden processzushoz társítva van egy vagy több "last-in-first-out" (LIFO) rendszerverem
    - ez a verem a rendszerhívások és eljárások számára paraméterek és címek tárolására szolgál
    - a veremmutató ezen verem tetejére mutat
- Processzusvezérlő információ (Process Control Information)
  - – ütemezési és állapot információ: ez az információ szükséges az operációs rendszernek, hogy az ütemezési feladatát elvégezze
    - processzusállapot: a végrehajtásra kijelölt processzus készenléti fokát határozza meg (futó, futásra kész, várakozó, leállított).
    - prioritás: egy vagy több mező írja le a processzus ütemezésének prioritását. (alapértelmezett, azonnali, megengedhető legmagasabb)
    - ütemezéssel kapcsolatos információ: a használt ütemezési algoritmustól függ. Például a processzus várakozással telt idejének mértéke, ill. a legutolsó végrehajtás során eltelt idő
    - esemény: milyen eseményre várakozik a processzus, hogy az végrehajtható legyen?
  - –adatrendszerezés
    - egy processzus más processzushoz csatolódhat valamilyen rendszer szerint. Például szülő-gyerek viszonyban lehet más processzus(okk)al. A PCB ilyen szerkezetek, viszonyok kialakítását támogatja, más processzusra mutató pointerek alkalmazásával
  - – processzusok közötti kommunikáció
    - több jelző illetve üzenet is rendelhető két független processzus kommunikációjához
    - ezen információk egy része vagy egésze a processzusvezérlő blokkban tárolható és tartható fenn
  - – processzus privilégiumok
    - a processzusoknak privilégiumok adhatók, amelyek a számukra elérhető memóriát és a végrehajtható utasítások típusait határozzák meg
  - – memóriakezelés
    - ez a rész laptábla mutatókat tartalmazhat, mely a processzushoz rendelt virtuális memóriát írja le
  - – erőforrás felhasználás
    - a processzus által használt erőforrásokat (pl. megnyitott fájlok) jelezheti
    - a processzor illetve más erőforrás felhasználásának történetét is tartalmazhatja
    - ez az információ az ütemezőrendszer számára lehet fontos



PCB: Processzusvezérlő táblák

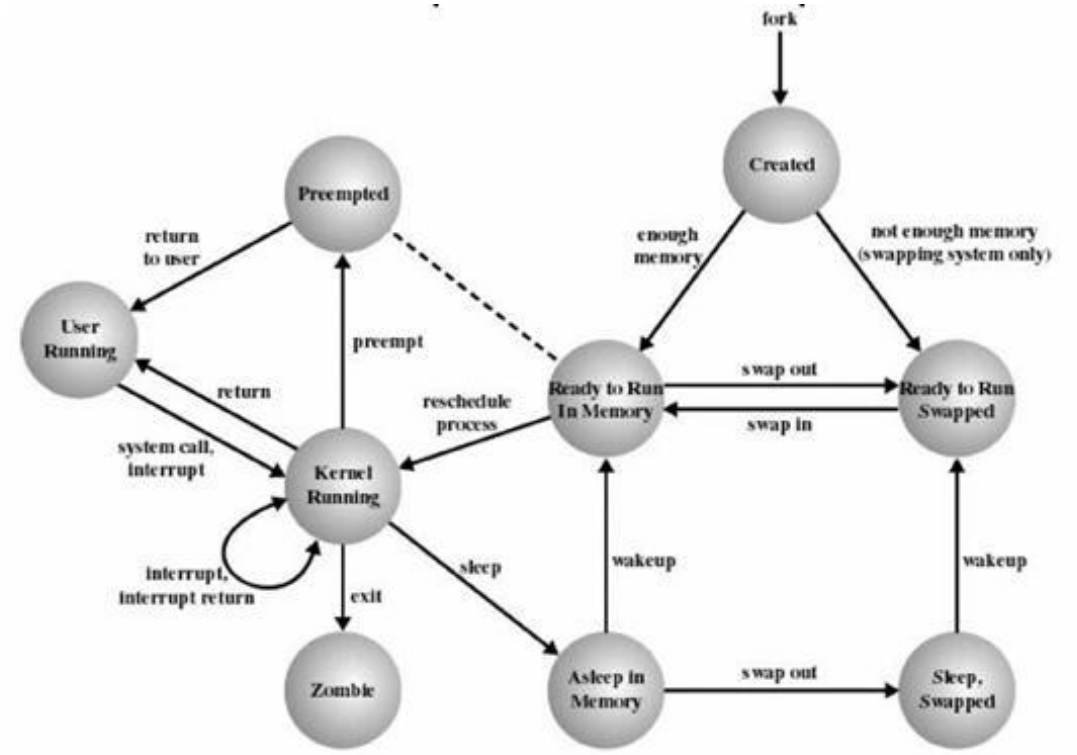
2.4. A processzusvezérlés folyamata

- Végrehajtás módjai:
  - – felhasználói mód
    - csökkentett privilégiumokkal járó mód
    - felhasználói programok tipikusan ebben a módban kerülnek végrehajtásra
  - – kernel mód
    - több privilégiummal rendelkező mód
    - teljes felügyelet a processzor (és összes utasítása), a regiszterek és a memória felett
- Processzuslétrehozás lépései:
  - – egyedi processzusazonosító hozzárendelése
  - – tárfoglalás a processzus számára – processzusvezérlő blokk inicializálása
  - – megfelelő kapcsolatok beállítása
    - ütemezési sorhoz szükséges listához történő kapcsolódás
  - – egyéb adatrendszerek létrehozása
    - könyvelési fájl fenttartása
- Processzusváltás okai:
  - megszakítás

- – óramegszakítás
  - a processzus a maximális időszeleten túlfut
- – I/O megszakítás
- – laphiba
  - a memóriacím a virtuális memóriában lévő adatra hivatkozik, amit először a főmemóriába kell áthozni, csak ezután futhat tovább a processzus

- csapda (trap)
  - – hibaesemény
  - – a processzus „Kilépés” állapotba történő mozgatását jelentheti
- rendszerhívás (INT) – operációs rendszer valamely szolgáltatásának (funkció) hívása

3. A Unix processzus kezelése

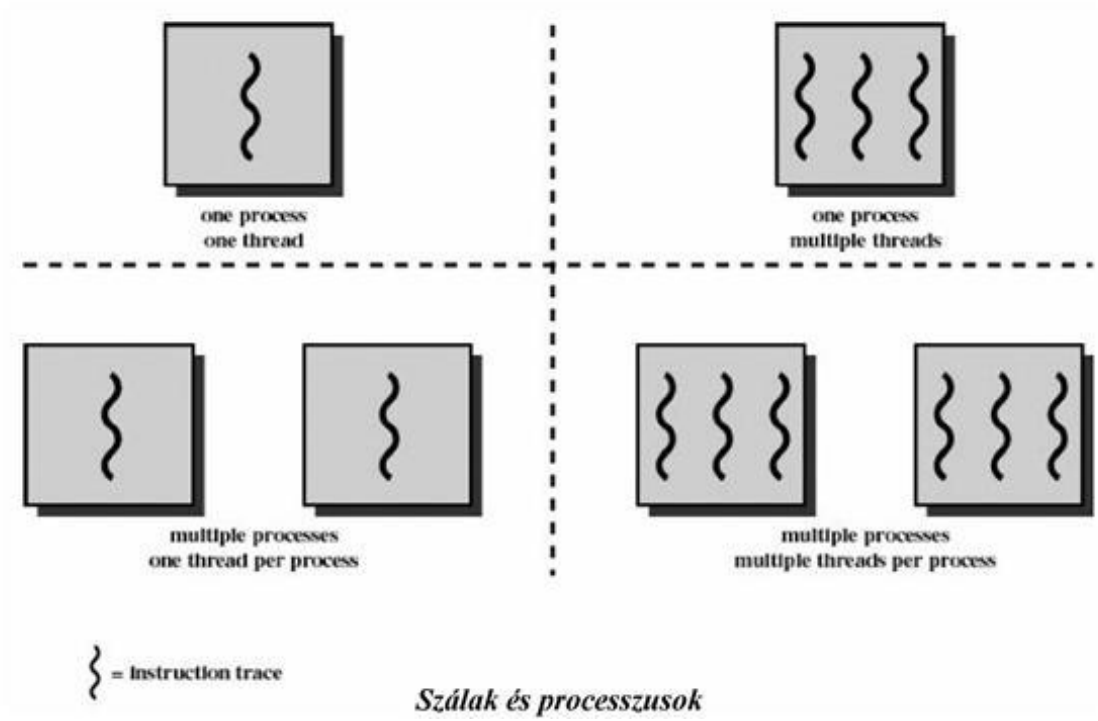


A UNIX processzusállapotai és átmenetei

4. fejezet - Szálak, mikrokernelek

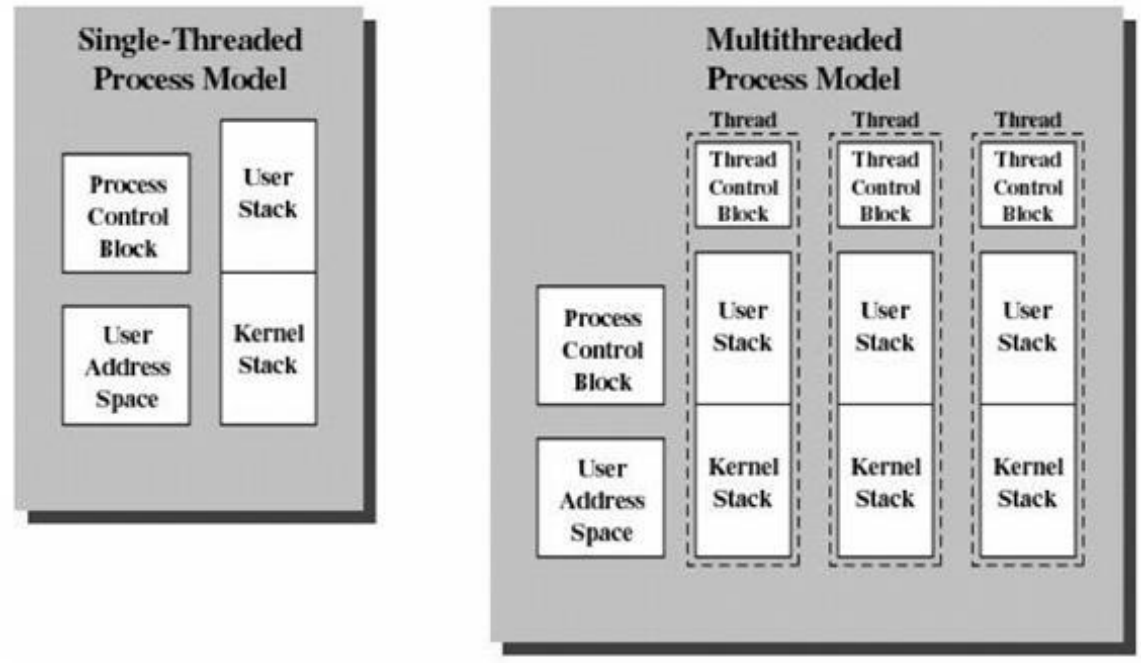
1. Folyamatok és szálak

- A processzusokkal kapcsolatban két jellemzőt lehet megemlíteni:
  - – erőforráskiosztás: a processzus számára virtuális címtartomány van lefoglalva a processzus kép (process image) tárolásához
  - – ütemezés/végrehajtás: a processzus végrehajtása egy programvégrehajtási útvonalat követ, mely kereszteződhet más processzusok végrehajtásával
- Ezen jellemzők egymástól függetlenek, az operációs rendszer egymástól függetlenül kezelheti őket:
  - – Processzus
    - erőforráskiosztás alapegysége
    - virtuális címtartomány, főmemória
    - I/O eszközök és fájlok
  - – Szál (vagy könnyűsúlyú processzus, újraindított programkód)
    - processzor kiszolgálás, ütemezés alapegysége
    - ütemezés és kiszolgálás operációs rendszer vezérlése szerint
    - a szálak olyan mechanizmust szolgáltatnak, amely lehetővé teszi a szekvenciális processzusoknak a rendszerhívások blokkolását, s közben a „párhuzamosság elérését”
- Többszörös szálak (Multithreading)
  - Az operációs rendszer támogathatja egy processzuson belül több vezérlési szál végrehajtását
    - – MS-DOS csak egyszeres szálakat támogat
    - – UNIX támogat párhuzamos felhasználói processzusokat, de egy processzuson belül csak egy szálat
    - – Windows 2000, Solaris, Linux, Mach, és OS/2 támogatja a többszörös szálakat



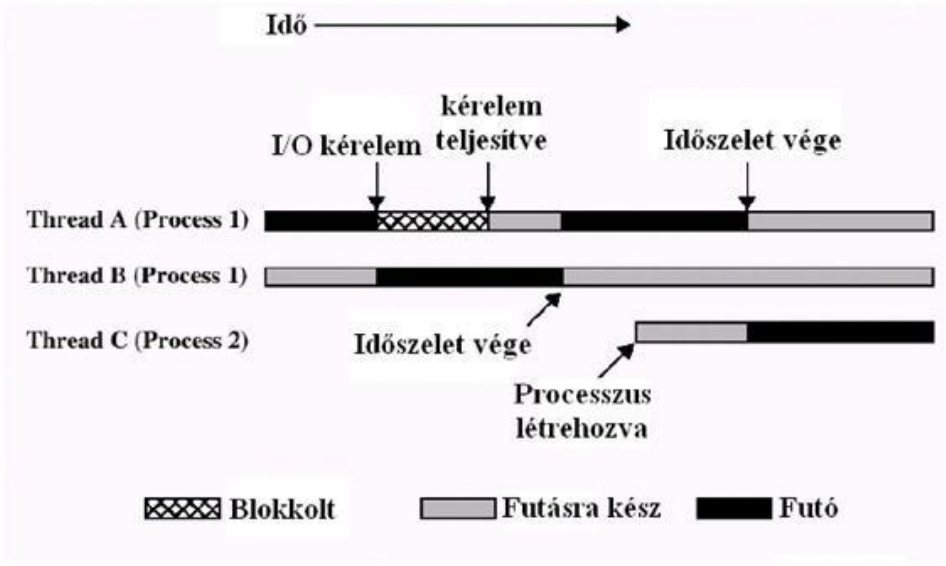
Szálak és processzusok

- Egy processzuson belül egy vagy több szál lehetséges a következő jellemzőkkel
  - – végrehajtás állapota (futó, készen álló, stb.)
  - – tárolt „szálkörnyezet”
    - program címszámláló, verem tartalma, regiszterkészlet, gyerekszálak, lokális változók számára memória
  - – a processzushoz lefoglalt memóriához és erőforrásokhoz való hozzáférés
    - ugyanazon processzushoz tartozó szálak (task) közösen használják
- Események, melyek egy processzus összes száljára hatással vannak
  - – egy processzus megszakítása az összes szál megszakításával jár
- Szálak használatának előnyei:
  - egy szál létrehozásához kevesebb idő kell, mint egy processzus létrehozásához
  - kevesebb idő egy szál megszakítása, mint egy processzusé
  - ugyanazon processzuson belüli szálak közötti átváltás kevesebb idővel jár, mint processzusok között
  - mivel az egy processzuson belüli szálak a memórián és a fájlokon osztoznak, a kernel segítségül hívása nélkül tudnak kommunikálni



Egyszeres és többszörös szálak

- Műveletek melyek egy szál állapotát megváltoztatják
  - – származtatás: másik, új szálat származtatni
  - – blokkolás, deblokkolás
  - – befejezés: erőforrások felszabadítása (regiszterek, vermek)

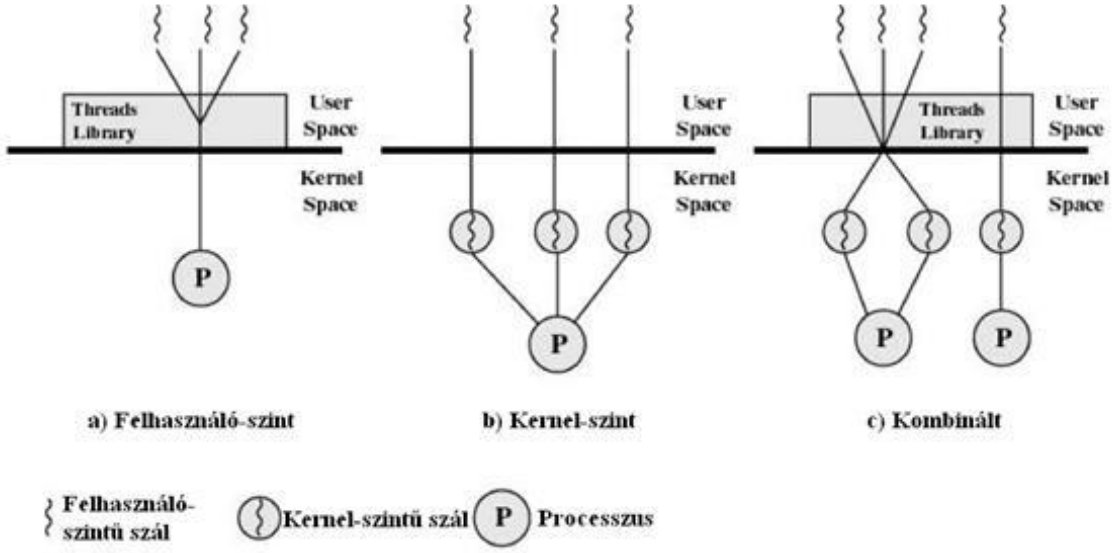


Multithreading egy processzoron

1.1. Szálak megvalósítása

- felhasználói-szintű szálak (User Level Thread - ULT)
  - – a szálak kezelését az alkalmazások (futtató rendszer!) végzik
  - – a kernel nem tud a szálak létezéréről

- kernel-szintű szálak (Kernel Level Thread - KLT)
  - – a kernel tartja fent a processzusok és szálak környezetét
  - – szál alapú ütemezés
  - – Pl: Windows XP, Linux, OS/2
- vegyes megközelítés
  - – szál létrehozása a felhasználói térben
    - az ütemezés és szinkronizáció nagy része is
  - – egy alkalmazáshoz tartozó több ULT leképzése ugyanannyi vagy kevesebb KLT-re
  - – példa: Solaris

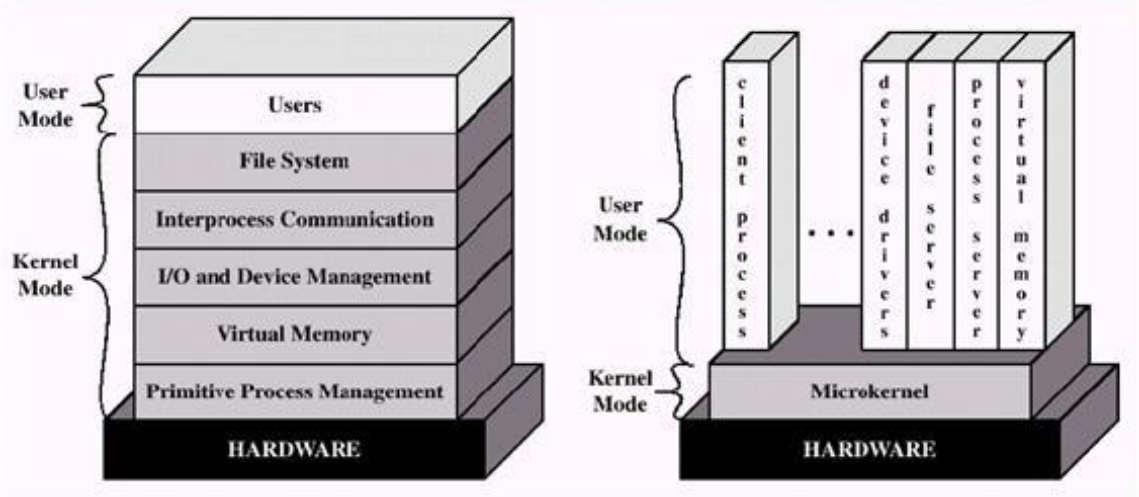


Felhasználó-szintű és kernel-szintű szálak

2. Mikrokernelek

- Kis operációs rendszermag
- Csak az alapvető operációs rendszerfüggvényeket, szolgáltatásokat tartalmazza:
  - – alacsony szintű memóriakezelés
    - hozzárendelni minden virtuális lapot (page) egy fizikai kerethez (frame)
  - – processzusok közötti kommunikáció
    - üzenet (message) az alapvető forma (message passing, MPI)
    - processzusok közötti üzenetváltás memória-memória másolást von maga után
  - – I/O és megszakításkezelés
- Hagyományosan operációs rendszer részeként működő szolgáltatások külső alrendszerekké válnak
  - – eszközmeghajtók

- – fájlrendszerek
- – virtuális memória kezelő
- – ablakkezelő rendszer
- – biztonsági rendszerek

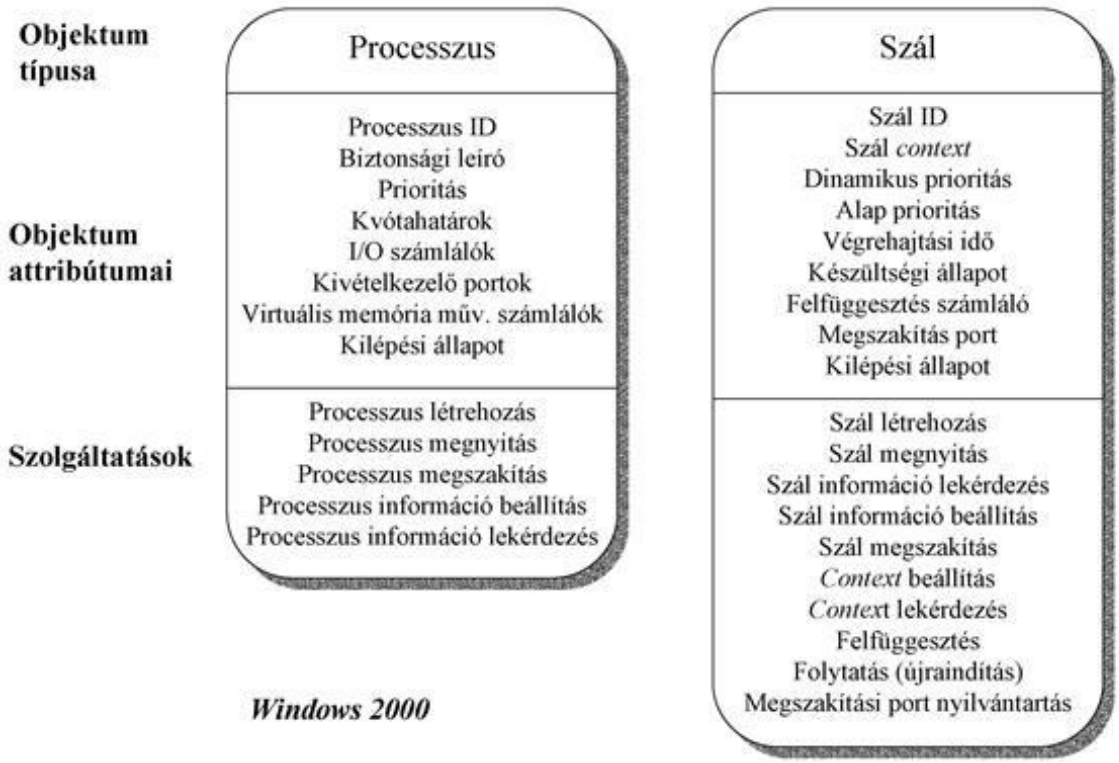


a) Hagyományos kernel *b) Mikrokernel*

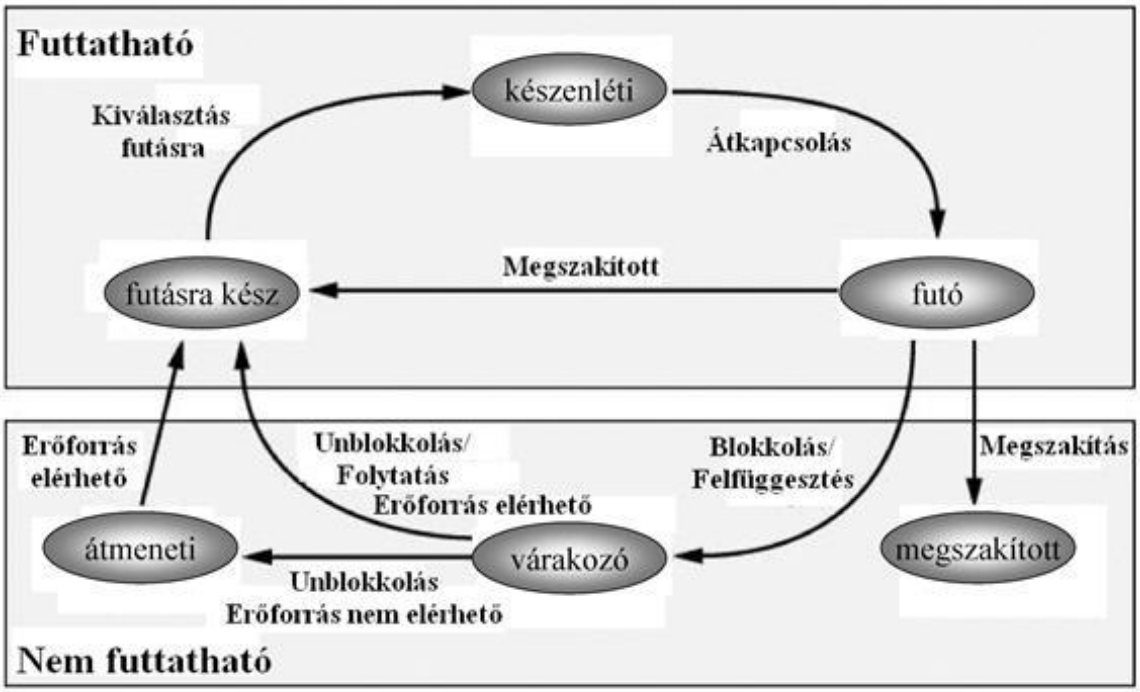
A mikrokernel előnyei

- Egységes felületet biztosít a processzusok számára
  - – a processzusoknak nem kell különbséget tenniük kernel-szintű és felhasználószintű szolgáltatások között
- Kiterjeszthető
  - – új szolgáltatások könnyen hozzáadhatók
- Rugalmas
  - – új szolgáltatások hozzáadhatók, létező szolgáltatások kivehetők, testreszabható
- Hordozható
  - – a rendszer új processzorra való átvitele esetén csak a mikrokernelben szükséges változtatni, az egyéb szolgáltatásokon nem
- Megbízható
  - – moduláris felépítés, egy kis mikrokernel könnyebben és szigorúbban tesztelhető
- Támogatja az osztott rendszerket
  - – az üzenetek küldése anélkül történhet, hogy információnk lenne a célgépről
- Objektum orientáltság

3. A Windows 2000 objektumai



Windows 2000



A Windows 2000 szálállapotai és azok átmenetei

4. Unix-Linux folyamatkezelés, szálak

- Állapotok:
  - futó
  - megszakítható
    - – blokkolt állapot

- nem megszakítható
  - – blokkolt állapot, de nem fogad semmilyen jelet
- leállított
  - – felfüggesztett processzus, csak egy másik processzus pozitív eredményű eseményére indulhat újra
- zombi

5. fejezet - Folyamat szinkronizáció

1. Konkurencia: versenyhelyzetek

- Versenyhelyzetek és az ezzel kapcsolatos problémák:
  - Globális erőforrások (változók) megosztása processzusok között: – ha két processzus megosztott változót használ, a végeredmény a hozzáférés sorrendjétől függővé válik
  - Erőforráslefoglalás (I/O csatornák lefoglalása) processzusok által: – holtponthoz, éhezéshez vezethet
  - A konkurenciahelyzetből származó programozási hibákat nehéz lokalizálni!

Egy processzor:

```
void echo()
{
    chin = getchar();
    chout = chin;
    putchar(chout);
}
```

Több processzor:

```
Process P1
.
in = getchar();
.
chout = chin;
putchar(chout);
.
```

```
Process P2
.
.
in = getchar();
chout = chin;
.
putchar(chout);
```

Tanulság: a megosztott globális változókat védeni kell!

Az operációs rendszer feladatai:

- Aktív processzusok nyomonkövetése
- erőforrások lefoglalása és felszabadítása
  - – processzoridő
  - – memória
  - – fájlok
  - – I/O eszközök
- adatok és erőforrások védelme
- a processzus eredménye független kell legyen más, konkurens processzusok végrehajtásának sebességétől
- Megoldás: kölcsönös kizárás szükséges
  - – kritikus szakasz bevezetése (a program azon része, amelyik nem megosztható erőforrást illetve globális változót használ)
    - egyszerre csak egy processzus léphet be a kritikus szakaszába
    - példa: egy adott időben csak egy processzus számára engedélyezett, hogy a nyomtatónak utasításokat küldjön •
- Kölcsönös kizárás miatt előfordulható problémák:
  - – holtpont (deadlock): processzusok egymásra befejeződésére várnak, hogy a várt erőforrás felszabaduljon
  - – éhezés (starvation): egy processzusnak határozatlan ideig várnia kell egy erőforrás használatára

2. Kölcsönös kizárás: megvalósítás és hardver támogatás

Kölcsönös kizárás megvalósítása:

- Dekker algoritmusa: kölcsönös kizárás megvalósítása két processzusra
  - – aktív várakozás (busy waiting) problémájának megoldása:
    - a processzus folyamatosan ellenőrzi, hogy beléphet-e a kritikus szekciójába (aktív)
    - ugyanakkor ezen kívül semmi produktívat nem csinál (várakozás)

Kölcsönös kizárás megvalósítása A Dekker-algoritmus

```
boolean flag [2];
int turn;
void P0( )
{
   while (true)
   {
      flag [0] = true;
      while (flag [1])
        if (turn == 1)
        {
          flag [0] = false;
          while (turn == 1)
          /* do nothing */;
          flag [0] = true;
        }
      /* critical section */;
      turn = 1;
      flag [0] = false;
      /* remainder */;
   }
}
```

```
void P1( )
{
   while (true)
   {
      flag [1] = true;
      while (flag [0])
        if (turn == 0)
        {
          flag [1] = false;
          while (turn == 0)
           /* do nothing */;
          flag [1] = true;
        }
      /* critical section */;
      turn = 0;
      flag [1] = false;
      /* remainder */;
   }
}
```

```
void main ( )
{
   flag [0] = false;
   flag [1] = false;
   turn = 1;
   parbegin (P0, P1);
}
```

A Dekker-algoritmus

```
boolean flag [2];
int turn;
void P0( )
{
   while (true)
   {
      flag [0] = true;
      turn = 1;
      while (flag [1] && turn == 1)
        /* do nothing */;
      /* critical section */;
      flag [0] = false;
      /* remainder */;
   }
}
```

```
void P1( )
{
   while (true)
   {
      flag [1] = true;
      turn = 0;
      while (flag [0] && turn == 0)
        /* do nothing */;
      /* critical section */;
      flag [1] = false;
      /* remainder */
   }
}
```

```
void main( )
{
   flag [0] = false;
   flag [1] = false;
   parbegin (P0, P1);
}
```

A Peterson-algoritmus

Kölcsönös kizárás hardver támogatásssal

- Megszakítás kikapcsolása
  - – a processzus addig fut, míg egy operációs rendszer szolgáltatást meghív, vagy megszakítása történik
  - – a megszakítás kikapcsolásával szavatolni lehet a kölcsönös kizárást
  - – több processzor esetében (multiprocesszing)
    - a megszakítás kikapcsolása nem garantálja a kölcsönös kizárást!
- Speciális gépi utasítások (szinkronizációs hardver)
  - – a test-and-set (TS) gépi utasítás (bizonyos) architektúrákban egy atomi műveletként képes egy memória szó tartalmát lekérdezni és a szóba egy új értéket beírni (a kettő között megszakítás nem lehetséges)
  - – felhasználása: a közös adat elérésének ténye más processzusok számára érzékelhetővé tehető!

```
boolean testset (int i) {
     if (i == 0) {
          i = 1;
          return true;
     }
     else {
          return false;
     }
}
```

A TS gépi utasítás

Kölcsönös kizárás: gépi utasítások

- Előnyök

- – akármennyi processzusra alkalmazható, egy processzoros és több processzoros esetre is
- – egyszerű, ezért könnyű az ellenőrzés
- – több kritikus szakasz használatát is támogatja

- Hátrányok
  - – az „aktív várakozás” jelentősen fogyasztja a processzoridőt
  - – éhezés (starvation) lehetséges, mikor egy processzus elhagyja a kritikus szakaszt és több, mint egy processzus várakozik
  - – holtpont (deadlock)
    - ha egy kis prioritású processzus a kritikus szakaszban van és egy nagyobb prioritású processzus szeretne belépni a kritikus szakaszba, a nagyobb prioritású processzus megkapja a processzort a kritikus szakaszra való várakozáshoz

3. Szemaforok és alkalmazásaik

- a **szemaforok** (S): speciális, egész típusú (integer) változók, melyeket processzusok végrehajtásának vezérlésére (megállítás/továbbindítás) használhatunk (analógia a vasúti forgalom irányításával!)
  - – (általában) nemnegatív kezdőértéket kaphat
  - – „Wait” művelet csökkenti a szemaforok értékét
    - WAIT(S): S:=S-1; if S<0 then BLOCK(S)
    - BLOCK(S): a hívó processzus "elalszik" az S szemaforon!
  - – „Signal” művelet növeli a szemafor értékét
    - SIGNAL(S): S:=S+1; if S≥0 then WAKEUP(S)
    - WAKEUP(S): "felébreszt" (továbbindít) egyet az S szemaforn alvó processzusok közül
- egy processzus felfüggesztésre kerül, amíg meg nem kapja a továbbíndítási jelet (signal)
- a „wait” és „signal” műveletek nem megszakíthatók!
- a block(S) eljárás felfüggeszti a hívó processzus végrehajtását, és az S szemaforon várakozó processzusok sorához adja

3.1. Termelők-fogyasztók problémája

- egy vagy több termelő adatot generál (termel), melyeket egy pufferbe tesz
- egy egyszerű fogyasztó ezeket az adatokat egyenként veszi ki a pufferből és dolgozza fel
- szinkronizációs problémák miatt egyszerre csak egy termelő vagy fogyasztó érheti el a puffert

```
producer:
while (true) {
    b[in] = v;
    in++;
}
consumer:
while (true) {
    while (in <= out)
    w = b[out];
    out++;
}
```

```
producer:
while (true) {
    while ((in + 1) % n == out)
    b[in] = v;
    in = (in + 1) % n;
}
consumer:
while (true) {
    while (in == out)
    w = b[out];
    out = (out + 1) % n;
}
```

```
/* program producerconsumer */
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while (true)
    {
        produce();
        waitB(s);
        append();
        n++;
        if (n==1) signalB(delay);
        signalB(s);
    }
}
```

```
void consumer()
{
    int m; /* a local variable */
    waitB(delay);
    while (true)
    {
        waitB(s);
        take();
        n--;
        m = n;
        signalB(s);
        consume();
        if (m==0) waitB(delay);
    }
}
void main()
{
    n = 0;
    parbegin (producer, consumer);
}
```

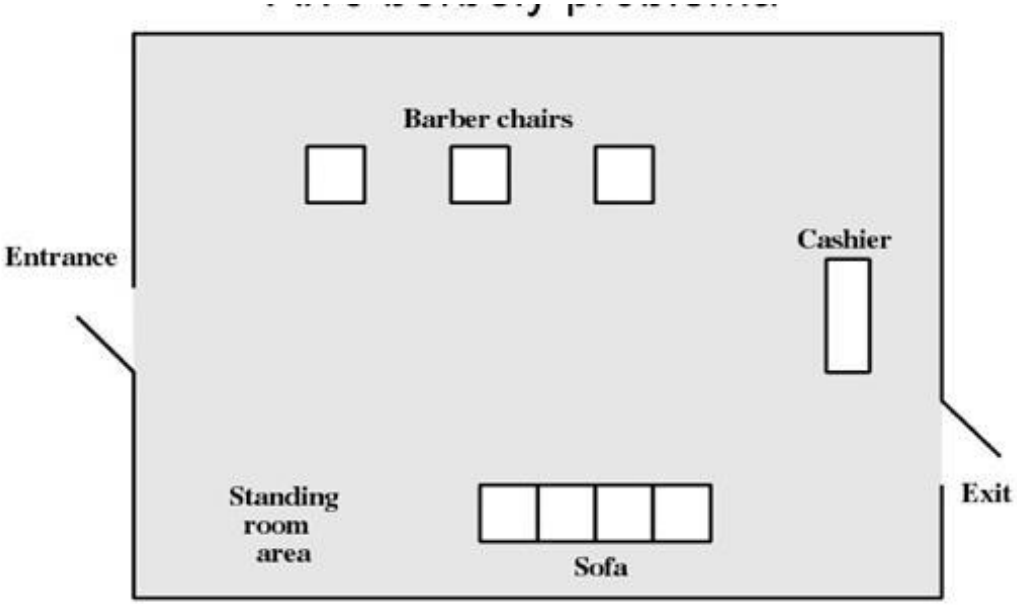
A termelők-fogyasztók probléma egy megoldása szemaforokkal

3.2. Az "alvó borbély" probléma

- **A probléma**:
  - – 3 szék, 3 borbély, és egy várakozó rész
  - – a tűzjelző beállítása maximum 20 vendéget engedélyez az üzletben
  - – a borbélyüzlet esetenként 50 vendéget tud kiszolgálni
  - – vendég nem léphet be az üzletbe, ha az elérte a max. kapacitását
  - – ha bejutott, a vendég leülhet a kanapéra vagy ha az teli van, akkor áll

- – mikor egy borbély szabaddá válik, a kanapén legrégebb óta ülő vendég kerül kiszolgálásra és egyúttal ha van álló vendég, a legrégebben álló vendég foglalhat helyet a kanapén
- – amikor egy vendég hajvágása befejeződött, a díjat bármelyik borbélynak kifizetheti, de mivel csak egy pénztárgép van, egyszerre csak egy vásárló tud fizetni

- **Feladat**: a borbélyok és vendégek beprogramozása versenyhelyzetek kialakítása nélkül!



Az alvó borbély probléma

```
/* program barbershop2 */
semaphore max_capacity = 20;
semaphore sofa = 4;
semaphore barber_chair = 3, coord = 3;
semaphore mutex1 = 1, mutex2 = 1;
semaphore cust_ready = 0, leave_b_chair = 0, payment = 0, receipt = 0;
semaphore finished [50] = {0};
int count;
void customer()
{
   int custnr;
   wait(max_capacity);
   enter_shop();
   wait(mutex1);
   count++;
   custnr = count;
   signal(mutex1);
   wait(sofa);
   sit_on_sofa();
   wait(barber_chair);
   get_up_from_sofa();
   signal(sofa);
   sit_in_barber_chair();
   wait(mutex2);
   enqueue1(custnr);
   signal(cust_ready);
   signal(mutex2);
   wait(finished[custnr]);
   leave_barber_chair();
   signal(leave_b_chair);
   pay();
   signal(payment);
   wait(receipt);
   exit_shop();
   signal(max_capacity)
}
```

```
void barber()
{
   int b_cust;
   while (true)
   {
      wait(cust_ready);
      wait(mutex2);
      dequeue1(b_cust);
      signal(mutex2);
      wait(coord);
      cut_hair();
      signal(coord);
      signal(finished[b_cust]);
      wait(leave_b_chair);
      signal(barber_chair);
   }
}
```

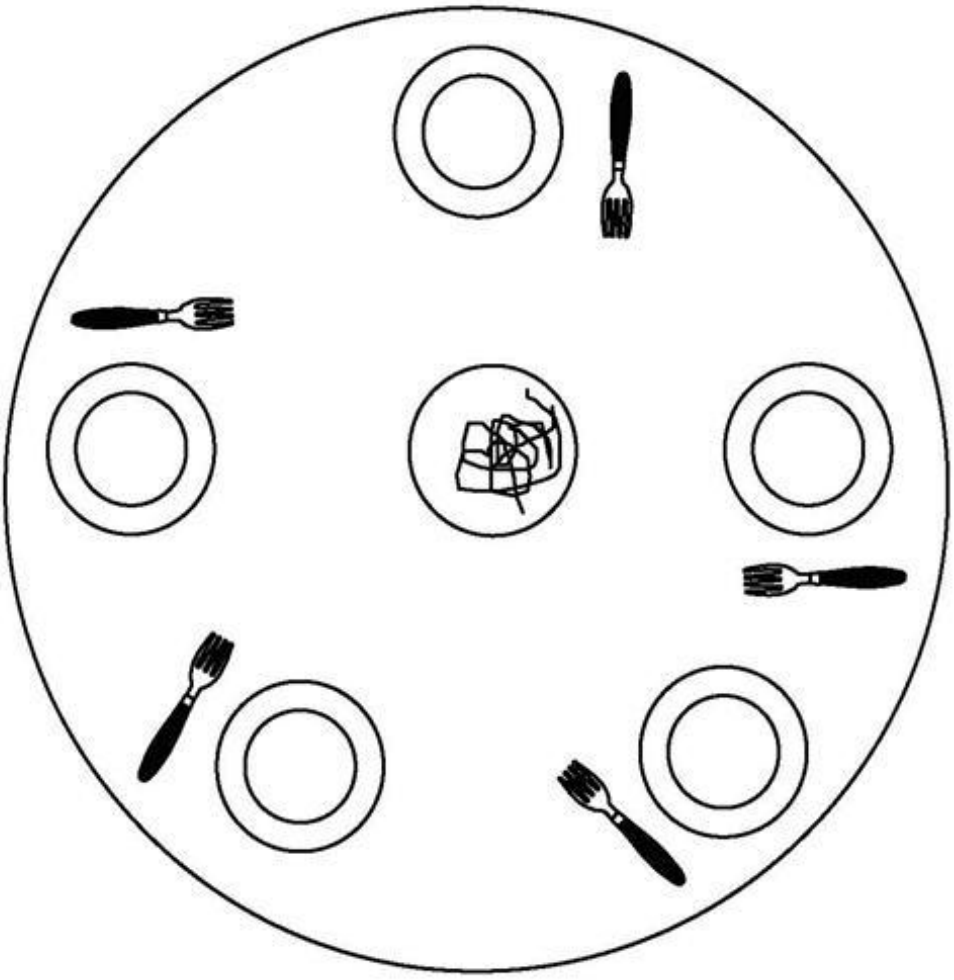
```
void cashier()
{
   while (true)
   {
      wait(payment);
      wait(coord);
      accept_pay();
      signal(coord);
      signal(receipt);
   }
}
```

```
void main()
{
   count := 0;
   parbegin (customer, . . . 50 times, . . . customer, barber, barber, barber,
             cashier);
}
```

Az alvó borbély probléma egy megoldása

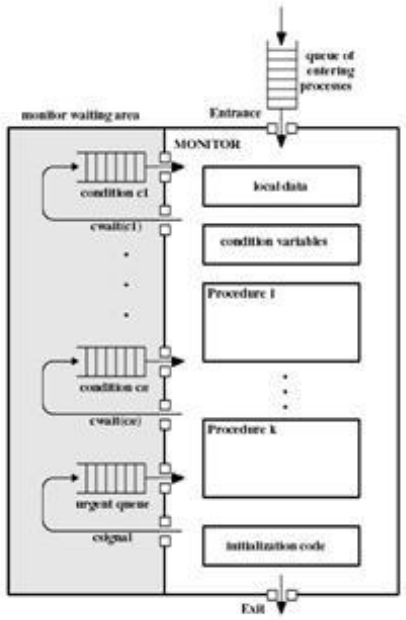
3.3. A vacsorázó filozófusok probléma

- Egy köralakú asztal mellett öt filozófus ül, mindegyik előtt van egy tányér rizs és a szomszédos tányérok között egy-egy evőpálcika.
- evéshez a filozófus a saját tányérja melletti két evőeszközt használhatja úgy, hogy ezeket egymás után kézbe veszi.
- ha befejezte az étkezést, visszateszi az eszközöket, és gondolkodni kezd.
- majd újra megéhezik, stb.



3.4. Monitorok

- a monitorok olyan magas szintű szinkronizációs eszközök, melyek lehetővé teszik egy absztrakt adattípus biztonságos megosztását konkurens processzusok között (a monitor eljárások, változók és adatszerkezetek együttese) …(objektum! Hoare, 1971)
- főbb jellemzők:
  - – a processzusok hívhatják a monitorban levő eljárásokat, de annak belső adatszerkezetét nem érhetik el
  - – minden időpillanatban csak egy processzus lehet aktív a monitorban
  - – megvalósítása például szemaforokkal lehetséges
  - – a kölcsönös kizárás megvalósítását a fordítóprogram/operációs rendszer végzi, így a hibázás miatti holtpontok elkerülhetők!
  - – a blokkoláshoz és ébresztéshez állapotváltozókat (condition típus) használ két rajtuk elvégezhető művelettel (WAIT, SIGNAL). Ezek az állapotváltozók nem számlálók, mint a szemaforok!



Egy monitor szerkezet

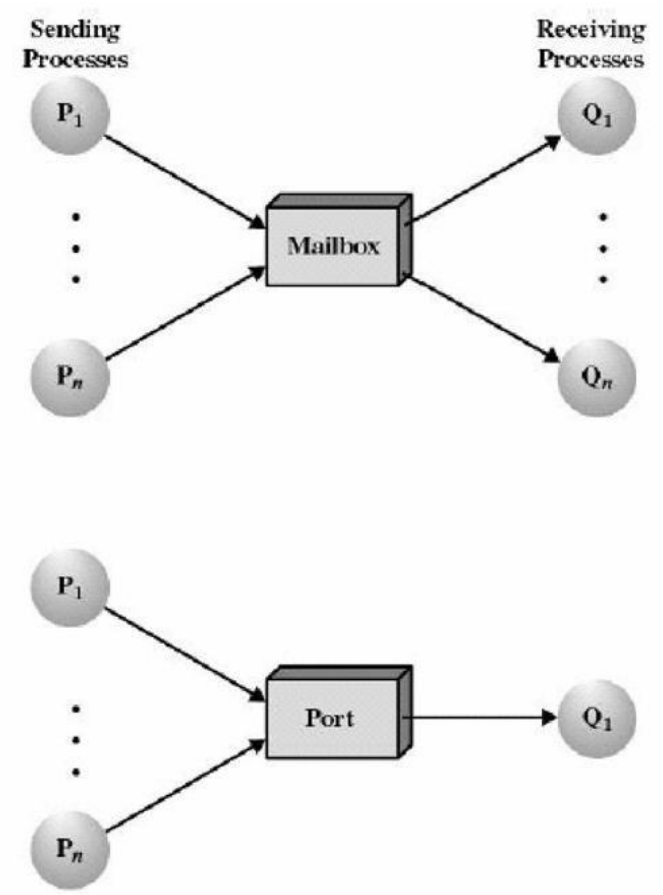
```
type dining-philosophers = monitor
   var state : array [0..4] of (thinking, hungry, eating);
   var self : array [0..4] of condition;
   procedure entry pickup (i: 0..4);
      begin
         state[i] := hungry;
         test (i);
         if state[i] ≠ eating then self[i].wait;
      end;
   procedure entry putdown (i: 0..4);
      begin
         state[i] := thinking;
         test (i+4 mod 5);
         test (i+1 mod 5);
      end;
   procedure test (k: 0..4);
      begin
         if state[k+4 mod 5] ≠ eating
           and state[k] = hungry
           and state[k+1 mod 5] ≠ eating
           then begin
                   state[k] := eating;
                   self[k].signal;
                end;
      end;
   begin
      for i := 0 to 4
         do state[i] := thinking;
   end.
```

A vacsorázó filozófusok probléma megoldása monitorral

4. Folyamatok kommunikációja (IPC)

Az IPC olyan mechanizmust jelent, amely lehetővé teszi, hogy processzusok egymással kommunikáljanak, akcióikat összehangolják ill. szinkronizálják.

- Az IPC kétművelete:
  - – send(message) és
  - – receive(message)
  - – ha a P és Q processzusok kommunikálni szeretnének, akkor szükségük van egy kommunikációs vonalra (communication link)
- Direkt kommunikáció:
  - – send(P, message): küldj egy üzenetet P-nek (utasítás Q-ban)
  - – receive(Q, message): fogadj egy üzenetet Q-tól (utasítás P-ben)
  - – a kommunikációs vonal ebben az esetben automatikusan épül fel a két processzus között (PID azonosító ismerete szükséges!)
  - – a vonal pontosan két processzus között létezik
- Indirekt kommunikáció:
  - – send(A, message): küldj egy üzenetet az „A” Mail-boxba (Mail-box: egy közösen használt, megosztott adatszerkezet)(utasítás Q-ban)
  - – receive(A, message): olvass ki egy üzenetet az A Mail-boxból (utasítás P-ben)
  - – a kommunikációs vonal abben az esetben épül fel a két processzus között, ha közösen használhatják az A Mail-boxot (PID ismerete nem szükséges!)



Processzusok közvetett kommunikációja

Az „olvasók-írók” probléma

- Egy adatot, állományt több processzus megosztva, párhuzamosan használ, egyesek csak olvassák, mások csak írják. Hogyan biztosítható az adatok konzisztenciája?
- Egy stratégia (olvasók prioritása):
  - – párhuzamosan akárhány olvasó olvashatja a fájlt
  - – egyszerre egy író írhat a fájlba
  - – ha egy író éppen fájlba ír, olvasó nem férhet hozzá a fájlhoz
- Egy másik stratégia (írók prioritása):
  - – olvasó nem férhet hozzá a fájlhoz, amint egy író írási szándékot jelez
- Mindkettő éhezéshez (starvation) vezethet!

```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true)
    {
        wait (x);
        readcount++;
        if (readcount == 1)
            wait (wsem);
        signal (x);
        READUNIT();
        wait (x);
        readcount--;
        if (readcount == 0)
            signal (wsem);
        signal (x);
    }
}
```

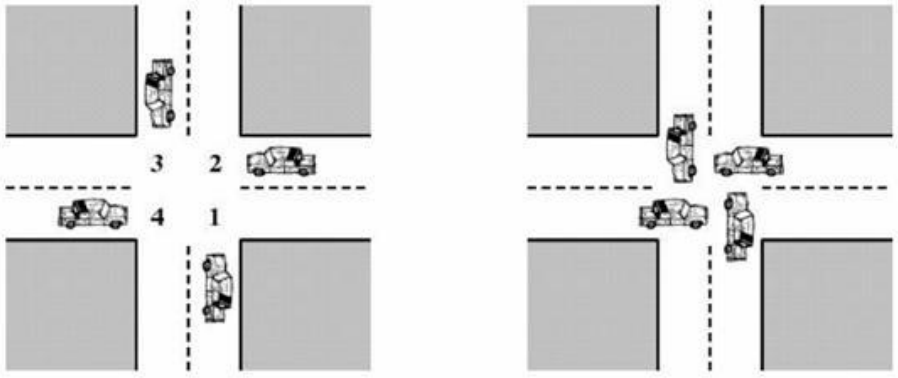
```
void writer()
{
    while (true)
    {
        wait (wsem);
        WRITEUNIT();
        signal (wsem);
    }
}
void main()
{
    readcount = 0;
    parbegin (reader, writer);
}
```

Az írók-olvasók probléma egy megoldása (prioritás az olvasóknál)

6. fejezet - Holtpont és éhezés

1. A holtpont fogalma

- Holtpont fogalma: a rendszererőforrásokért versengő vagy egymással kommunikáló processzusok állandósult blokkoltsága.
- Nincs általános megoldás!!
- Két vagy több processzus erőforrásszükségletek miatt állnak egymással konfliktusban.

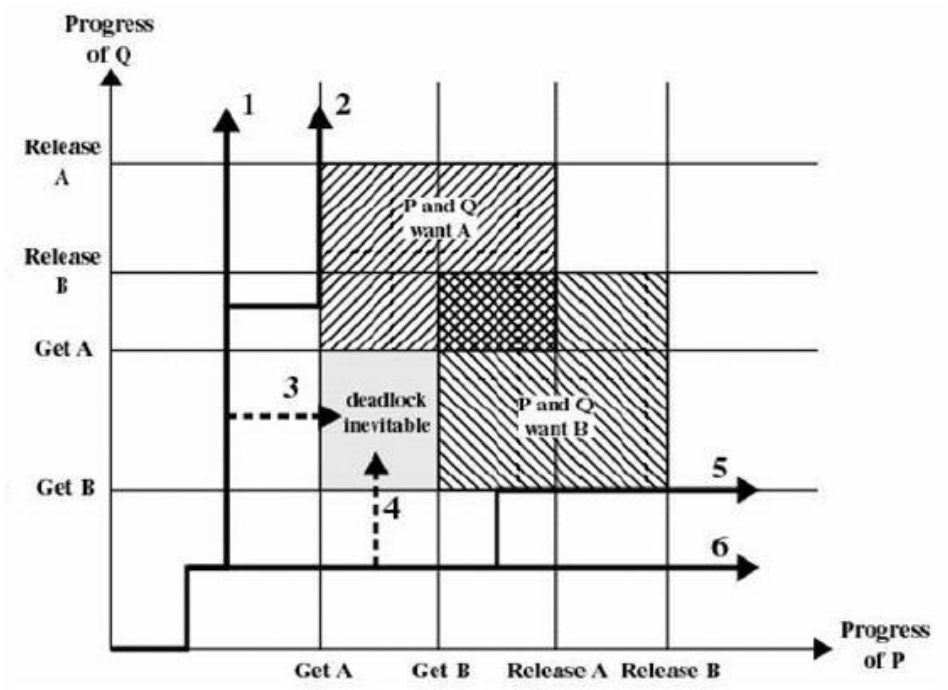


a) Holtpont lehetséges *a) Holtpont*

Holtpont (illusztráció)

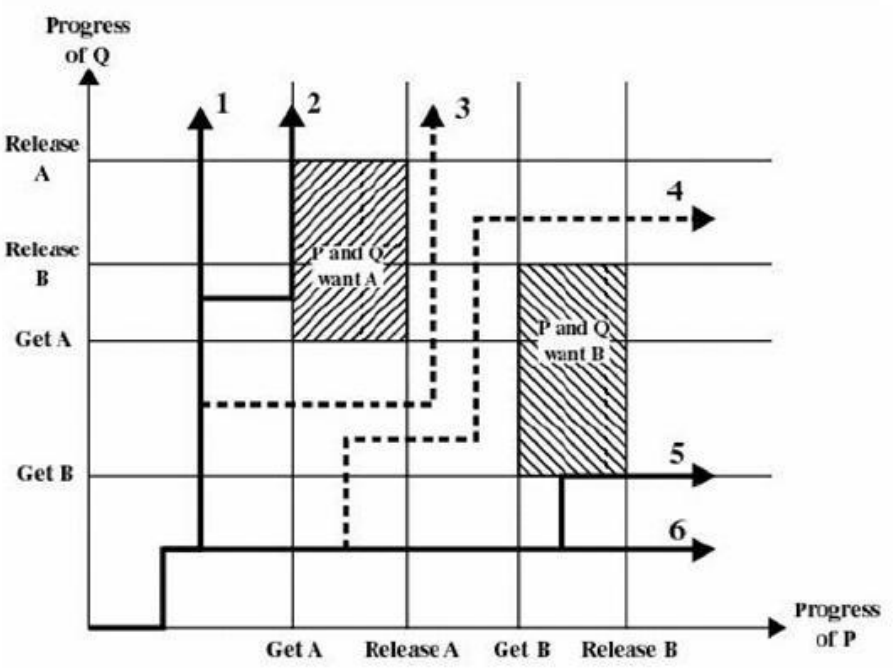
Példa: két processzus (P, Q), két erőforrás (A, B), mindkét processzus igényt tart mindkét erőforrásra. Az alábbi ábra a hat lehetséges végrehajtási útvonalat mutatja (egyprocesszoros rendszerben egyszerre egy processzus végrehajtása lehetséges!)

A 3. és 4. útvonalnál a holtpont elkerülhetetlen!



Példa holtpontra

Példa: két processzus (P, Q), két erőforrás (A, B), csak az egyik processzus (Q) tart igényt egyszerre mindkét erőforrásra. A P processzus az erőforrásokat egymás után használja.



Példa holtpont elkerülésére

Újrahasználható erőforrások:

- egyszerre egy processzus használja de a használat során nem „merül” ki
- processzusok elnyerik az erőforrást, melyet később felszabadítanak, hogy egy másik processzus használni tudja

- például: processzorok, I/O csatornák, fő és másodlagos memóriák, fájlok, adatbázisok és szemaforok
- holtpont következik be, ha mindkét processzus fenntart egy-egy erőforrást és a másikért folyamodik
- a következő ábrán – a végrehajtási sorrend: p0p1q0q1p2q2..... holtpont!

Process P

Step	Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Process Q

Step	Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Két processzus újrahasználható erőforrásokért versenyez

Fel/el-használható erőforrások

- processzus által létrehozott és megsemmisített erőforrások
- például: megszakítások, szignálok, üzenetek és I/O pufferekben lévő információk
- két processzus (P1, P2) egymástól vár üzenetet, majd annak megkapása után üzenetet küld a másiknak. Így holtpont állhat elő, hiszen a *Receive* blokkolttá válik (lásd a következő ábrát)

P1

```
. . .
Receive(P2);
. . .
Send(P2, M1);
```

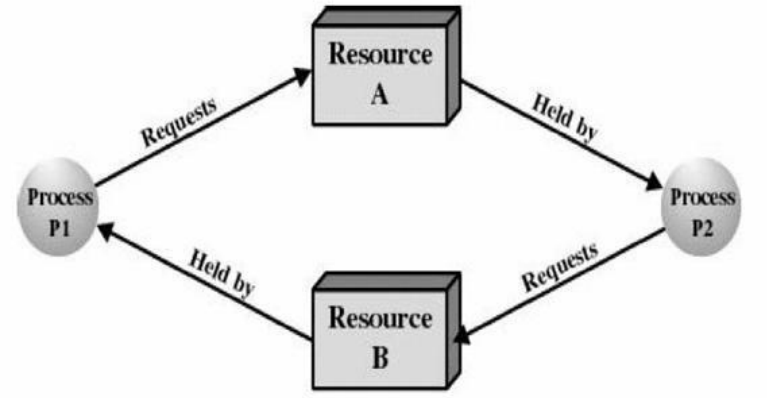
P2

```
. . .
Receive(P1);
. . .
Send(P1, M2);
```

Két processzus felhasználható erőforrások által okozott holtponthelyzete

Holtpont kialakulásához vezető (de egyébként szükséges) stratégiák:

- *kölcsönös kizárás*: egyszerre csak egy processzus használhat egy erőforrást
- *tartani és várni* (Hold-and-wait)
  - – egy processzus lefoglalva tart erőforrásokat, míg más erőforrások megszerzésére vár
- *nincs beavatkozás:*
  - – erőforrást nem lehet erőszakosan elvenni egy processzustól, mely éppen használja
- *körkörös várakozás*
  - – processzusok zárt lánca keletkezik, ahol minden processzus lefoglalva tart egy erőforrást, melyre a következő processzusnak szüksége van



Körkörös várakozás

2. A holtpont megelőzése

Stratégiák szerinti prevenció:

- Kölcsönös kizárás: nincs lehetőség megelőzésre
- Hold and wait:
  - – blokkolni a processzust, amíg az összes számára szükséges erőforrás fel nem szabadul
  - – egy processzushoz rendelt erőforrás sokáig üresjáratban lehet; ezalatt kiosztható más processzus számára
- Nincs beavatkozás:
  - – ha egy processzus számára nem lehetséges további igényelt erőforrás elnyerése, akkor a korábban lefoglalt erőforrásokat fel kell szabadítania
  - – az operációs rendszer beavatkozhat és felszabadíthat egy erőforrást
- Körkörös várakozás:
  - – erőforrások lineáris elrendezése
  - – amíg egy erőforrás elfoglalt, addig csak a listán magasabban levő erőforrás elérhető

3. A holtpont elkerülése

Holtpont elkerülésének két megközelítése:

- ne indítsunk el egy processzust, ha igényei holtponthoz vezetnek!
- ne elégítsünk ki erőforráskérelmet, ha az allokáció holtponthoz vezethet!

Processzus indításának megtagadása:

- n processzus, m erőforrás esetén bevezetésre kerül:
  - erőforrás (Resource) vektor (R1,...,Rm),
  - rendelkezésre álló erőforrások (Available) vektora (V1,...,Vm),
  - allokációs (Allocation) mátrix (A11,....Anm),

- illetve az összes processzus összes erőforrásra vonatkozó igényeinek (Claim) mátrixa (C11,...,Cnm)

- így: egy új processzus akkor indítható el, ha Ri≥C(n+1)i+Σn k=1Cki az összes i-re
- ez nem optimális stratégia, ugyanis a legrosszabbat tételezi fel: az összes processzus egyszerre akarja megszerezni az összes, számára szükséges erőforrást

Erőforrás lefoglalásának megtagadása:

- úgy is nevezik, hogy *bankár algoritmus*
- a rendszer állapota: az erőforrások aktuális kiosztása processzusokhoz
- biztonságos állapot az, amiből legalább egy végrehajtási sorrend lehetséges, mely nem holtponttal végződik (nem biztonságos állapot az, amire ez nem igaz)
- nincs visszaszorítás és beavatkozás!
- bankár algoritmusra vonatkozó korlátok:
  - – a maximum erőforrás-szükségletet előre meg kell állapítani
  - – fix számúerőforrás foglalható csak le
  - – processzus nem léphet ki, amíg erőforrást foglal éppen le

4. A holtpont detektálása

Holtpont detektálási algoritmus:

- allokációs mátrix (A), erőforrás vektor, elérhetőségi vektor
- kérelem mátrix Q bevezetése, ahol qIj jelenti az I processzus által igényelt j típusú erőforrások mennyiségét
- kezdetben minden processzus jelöletlen
- *Az algoritmus*:
  1. jelöljünk meg minden processzust, melynek allokációs mátrixbeli sora csupa 0
  2. legyen W egy vektor, mely megegyezik az elérhetőségi vektorral
  3. keressünk olyan processzust (i), mely jelöletlen, és Qik≤Wk, ahol 1≤k≤m. Ha ilyen nincs, szakítsuk meg az algoritmust!
  4. ha van, jelöljük meg a processzust és állítsuk be az új W-t: Wk=Wk+Aik, ahol 1≤k≤m, majd lépjünk vissza a 3. lépésre
- holtpont létezik, ha az algoritmus végén jelöletlen processzusok maradnak

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request matrix

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation matrix

R1	R2	R3	R4	R5
2	1	1	2	1

Resource vektor

R1	R2	R3	R4	R5
0	0	0	0	1

Available vektor

Helyreállítási stratégia:

- az összes holtpontot okozó processzus felfüggesztése (ez a leggyakoribb)
- az összes holtpontban levő processzus visszaállítása egy előzetesen definiált ellenőrzési pontra és az összes processzus újraindítása
  - – az eredeti holtpont újból bekövetkezhet....
- a processzusok egymás után való leállítása, amíg a holtpont megszűnik, minden egyes processzus leállítása után a holtpontdetektáló algoritmus újraindítása szükséges

- az erőforrások egymás után való felszabadítása, amíg a holtpont szituáció meg nem szűnik (detektáló algoritmus újraindítása minden erőforrás felszabadítás után)
- a processzusok kiválasztása bizonyos megfontolások alapján történik (leghosszabb hátralevő futási idő, legkevesebb lefoglalt erőforrással rendelkező, kisebb prioritású processzusok, stb.) Holtpont észlelése

5. A Unix konkurencia kezelése

A konkurenciakezeléshez használatos objektumok:

- Csatornák (Csövek, *Pipes*)
  - – körkörös puffer, mely két processzus termelő-fogyasztó modellen alapuló kommunikációját teszi lehetővé (first-in-first-out). Kölcsönös kizárás szükséges!
- Üzenetek (*Messages*)
- Osztott memória (*Shared memory*)
  - – leggyorsabb formája a processzusok közötti kommunikációnak
- Szemaforok
  - – a szemafor a következő elemekből áll:
    1. a szemafor aktuális értéke,
    2. a legutóbb a szemaforon működő processzus azonosítója,
    3. azon processzusok száma, melyek arra várnak, hogy a szemafor értéke nagyobb legyen, mint jelenlegi értéke,
    4. azon processzusok száma, melyek arra várnak, hogy a szemafor értéke zérus legyen
- Szignálok
  - – hasonlatosak a hardver megszakításhoz, de prioritás nélküliek
  - – a szingnál az operációs rendszernek "szól", rendszerhívás!

7. fejezet - Memóriagazdálkodás

1. Memóriakezelés

- A számítógép kapacitásának jobb kihasználása megköveteli, hogy egyszerre több processzus osztozzon a memórián (shared memory)
- Egy programot általában bináris formában tárolunk a háttértáron, végrehajtásához be kell tölteni a memóriába, ennek megszervezése a memóriamenedzsment feladata
- Bemeneti sor (Input queue): a végrehajtásra kijelölt programok együttese

A memóriakezelésnek öt követelményt kell teljesítenie:

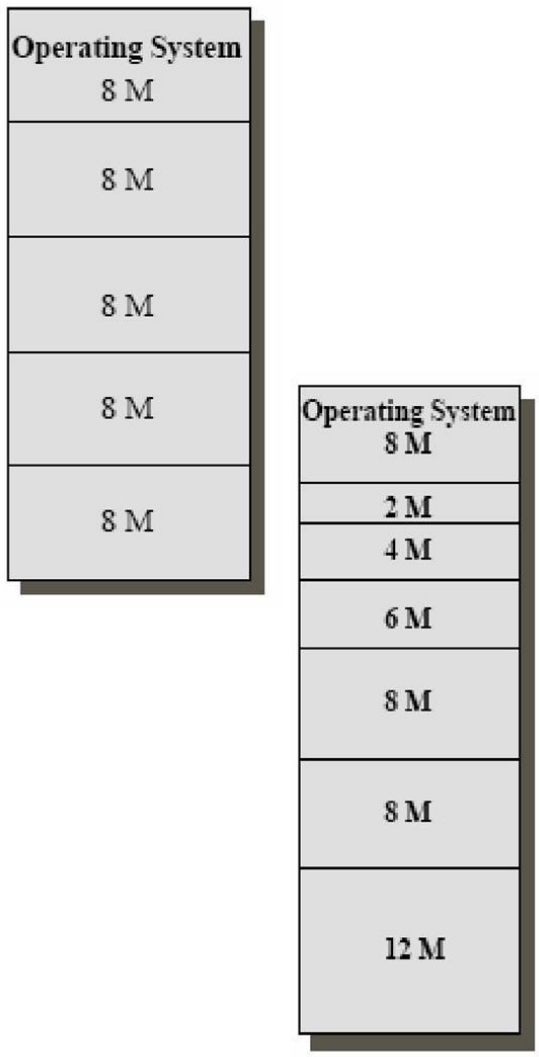
- Relokáció (relocation)
  - – a programozó nem tudja, hogy egy program végrehajtásakor a program a memórián belül hova kerül
  - – a végrehajtás alatt álló programot többször át lehet/kell mozgatni a háttértárba (swap) és vissza, de a kódnak a memóriába való visszamozgatása általában eltérő helyre történik (relocation)
  - – a memóriahivatkozásokat a kódba kell illeszteni az aktuális fizikai memóriacímeknek megfelelően
- Védelem (protection)
  - – a processzusok engedély nélkül nem használhatnak más processzusokhoz tartozó címtartományokat
  - – az abszolút memóriacímeket lehetetlen ellenőrizni a fordítás során, hiszen a program relokációt szenvedhet, így ezt a végrehajtás alatt kell ellenőrizni
- Megosztás (sharing)
  - – több processzus számára engedélyezett ugyanazon memóriaszegmens elérése
  - – jobb lehet, ha minden processzus (személy) egy program ugyanazon másolatát használja, mintha mindenkinek saját másolata lenne
- Logikai szervezés (Logical Organization)
  - – a programokat modulokba érdemes szervezni
  - – a modulok egymástól függetlenül írhatók és fordíthatók
  - – különböző mértékű a modulok védelme (read-only, execute-only)
  - – megosztott modulok
- Fizikai szervezés (Physical Organization)
  - – a program és a hozzá kapcsolódó adatok számára az elérhető memóra kevés lehet
    - ovarlaying: a teljes programnak csak az a része legyen bent az operatív tárban, amelyre ténylegesen szükség van, ezáltal lehetővé válik, hogy különböző modulok a memória azonos régióihoz legyenek rendelve

2. Memória felosztás

Fix partícionálás:

- A memória felosztás fix határokkal rendelkező régiókra

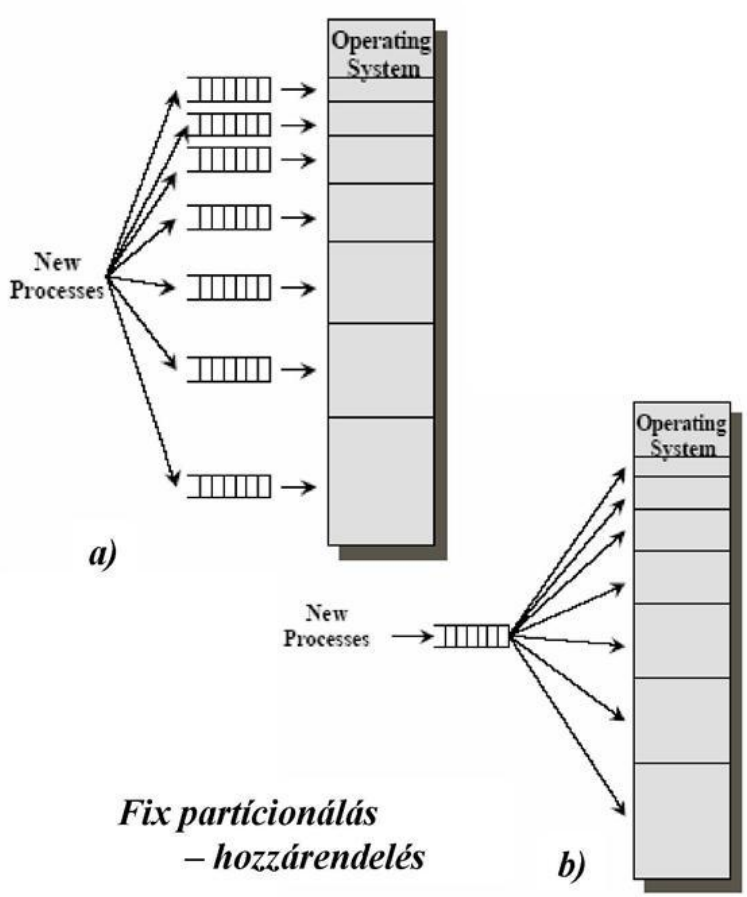
- Egyenlő méretű partíciók kialakítása
  - – bármelyik olyan processzus melynek mérete kisebb vagy egyenlő a partíció méretével, betölthető egy elérhető partícióba
  - – ha az össze partíció tele van, az operációs rendszer kicserélheti egy partícióban levő másik processzussal
  - – Problémák:
    - egy program nagyobb is lehet, mint a partíció, ekkor a programozónak az overlay technikát kell alkalmaznia
    - a főmemória kihasználása nem jó: minden program, méretétől függetlenül egy egész partíciót elfoglal (belső töredezettség - *internal fragmentation*)
  - – Megoldás: nem egyenlő méretű partíciók



Elhelyezési algoritmus:

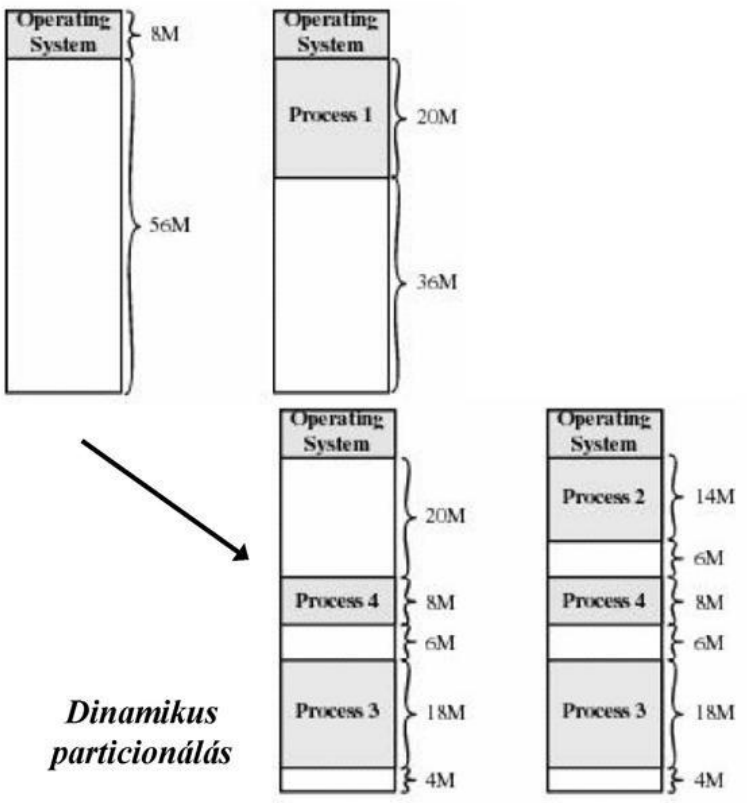
- Azonos méretű partíciók
  - – azonos méret miatt bárhova mehet
- Nem azonos méretű partíciók

- – minden processzushoz a lehető legkisebb alkalmas partíciót választani (belső töredezettség minimalizálása)
- – minden partícióhoz külön bemeneti sor
- – az összes partícióhoz csak egy bemeneti sor



Dinamikus particionálás

- Változtatható számú és nagyságú partíciók használata
- A processzusok pontosan annyi memóriát foglalnak le, amennyire szükségük van
- Végeredményben apró lyukak keletkeznek a memóriában (külső töredezettség - external fragmentation)
- Időnként tömörítés (processzusok egymás mellé tolása) szükséges, hogy az összes szabad memória egy blokkot alkosson. Ez igen sok processzoridőt emészt fel....

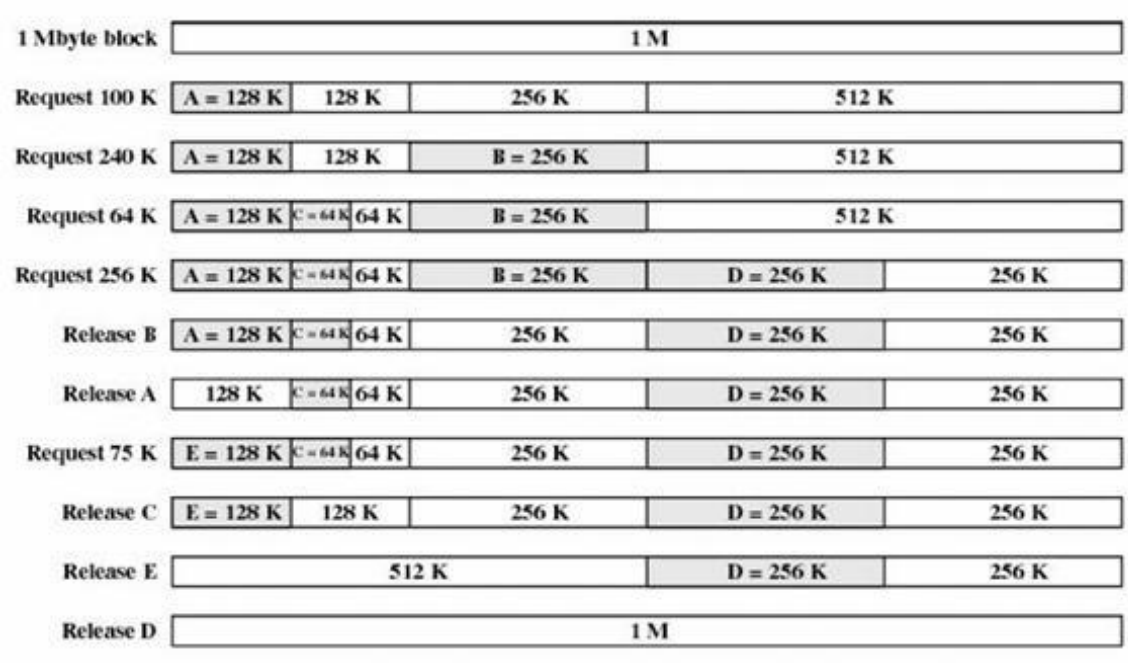


- Az operációs rendszernek kell eldönteni melyik szabad blokkba kerüljön a processzus, ehhez a következő algoritmusokat használhatja:
- Legjobban illeszkedő (Best-fit)
  - – a kérthez méretben legközelebb eső blokk választása
  - – a legrosszabbul teljesítő algoritmus: nagy külső töredezettség, gyakran kell tömörítést végrehajtani
- Első illeszkedő (First-fit)
  - – a memória elejéről számítva az első jól illeszkedő blokk választása
  - – sok processzus felgyűlhet a memória elején, amit minden alkalommal végig kell keresni, mikor egy üres blokkot keresünk
- Következő illeszkedő (Next-fit)
  - – a legutolsó lefoglalt blokktól kezdi a keresést a legjobban illeszkedő blokk után
  - – gyakran foglal le blokkot a memória végéről, ahol a legnagyobb blokk van, így az nagyon hamar kisebb blokkokra darabolódik
  - – tömörítésre van szükség, hogy a memória végén ismét nagy blokkunk legyen

„Buddy” rendszer

- Problémák a fix és a dinamikus partíciókkal:
  - – fix: erősen korlátozza az aktív processzusok számát, a rendelkezésre álló teret nem használja ki hatékonyan
  - – dinamikus: komplikált fenntartani, nagy a tömörítés költsége (processzoridő)

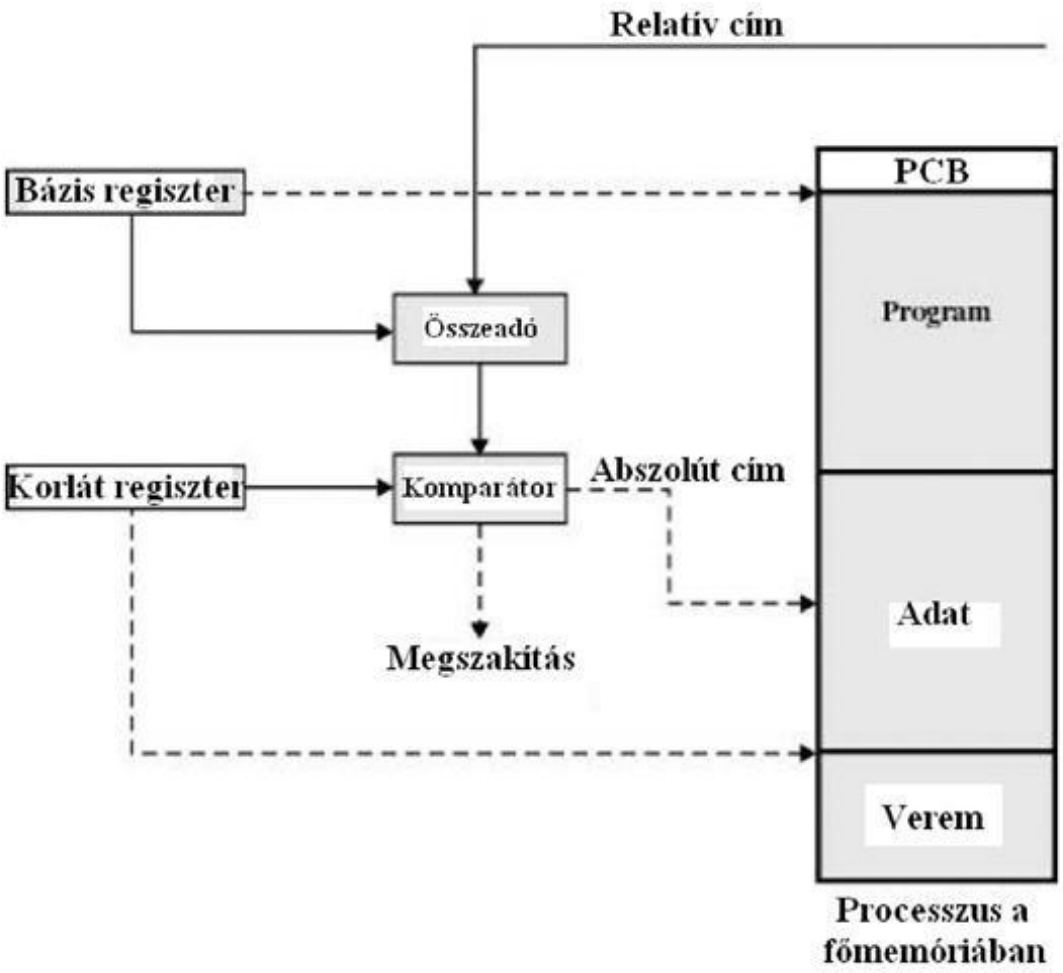
- Megoldás: „buddy” rendszer, mint kompromisszum
  - – az összes elérhető memória egy 2U méretű egyszerű blokk
  - – ha a processzus által kért méret 2U-1 < s ≤2U, akkor az egész blokk lefoglalásra kerül, máskülönben
    - ezt a blokkot két egyenlő részre osztjuk (2db 2U-1 méretű blokk)
    - az eljárást addig folytatjuk, amíg a legkisebb olyan blokkot kapjuk, ami nagyobb vagy egyenlő s-sel



Példa „Buddy” rendszerre

3. Relokáció

- Egy program memórába való betöltődése során az abszolút memóriacímek meghatározásra kerülnek
- Egy processzus futás során különböző partíciókra kerülhet (swapping miatt) ami egyben különböző abszolút memóriacímet is jelent
- A tömörítés szintén okozhatja processzusok más partícióba kerülését, ami szintén az abszolút memóriacímek megváltozását jelenti
- Ezek miatt fontos a következő memóriacímeket bevezetni:
  - – Logikai cím
    - olyan memóriacím, mely független az aktuális memóriakiosztástól (CPU által generált cím – virtuális cím)
    - a fizikai címre történő átfordítása szükséges
  - – Relatív cím
    - egy ismert ponthoz viszonyított pozíciót meghatározó cím
  - – Fizikai cím (abszolút cím)
    - főmemóriabeli abszolút cím (memóriakezelő egység által generált)

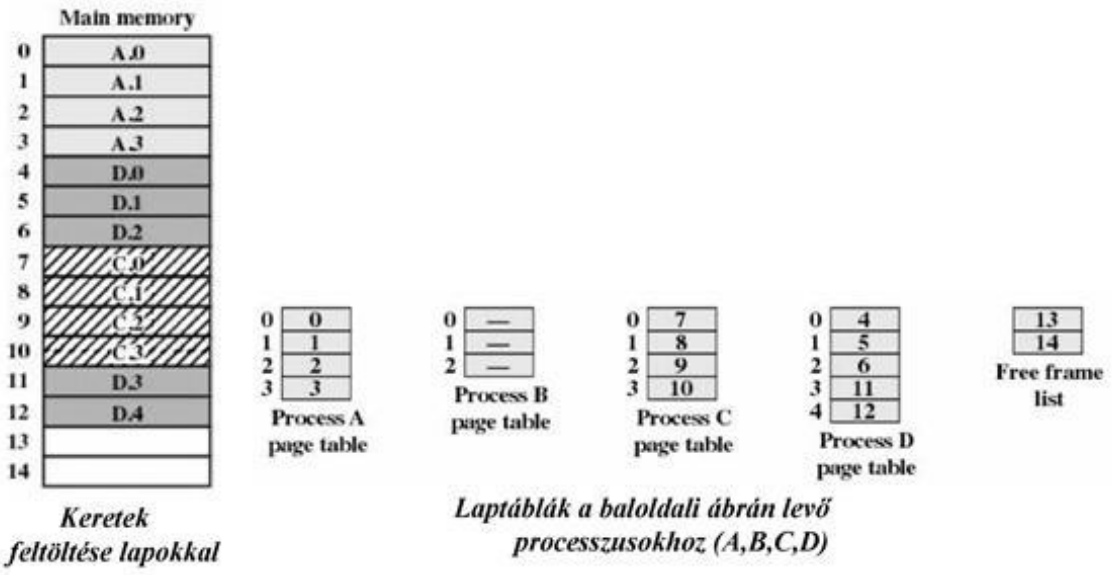


Hardveres támogatás a relokációhoz

4. Lapozás és szegmentáció

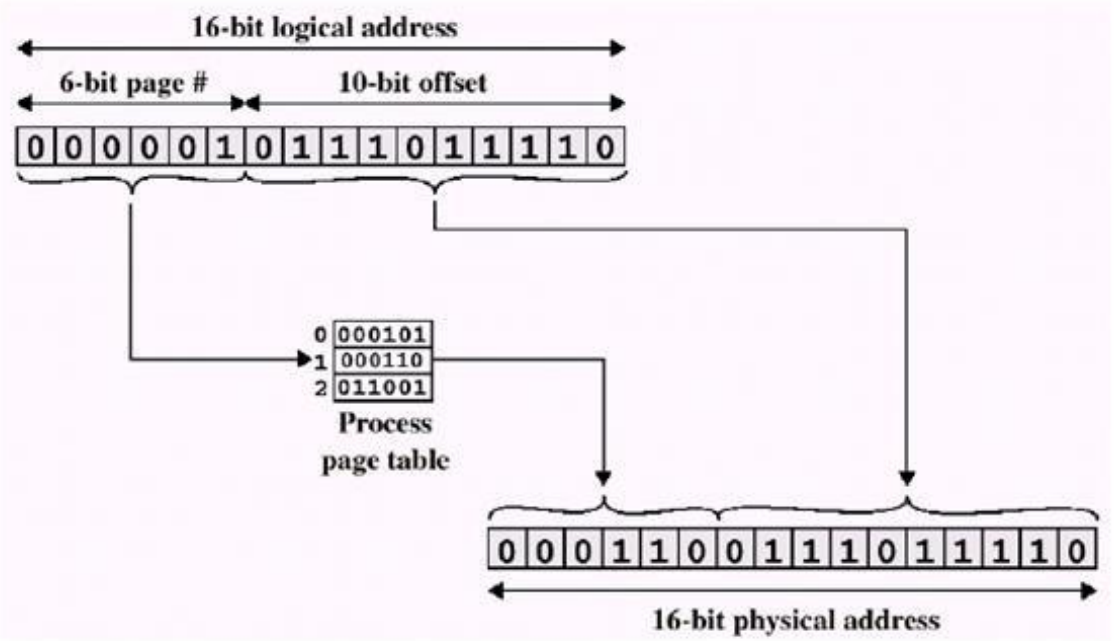
Lapozás (Paging)

- A külső fragmentáció problémájának egy megoldását kapjuk, ha a memóriát és a processzusokat kis, egyenlő méretű egységekre osztjuk (processzusdarab: lap – page ; memóriadarab: keret – frame)
- Az operációs rendszer minden processzushoz egy ún. laptáblát (page table) tart fent
  - – tartalmazza a processzus lapjaihoz tarozó keretek helyzetét (ábra)
  - – *logikai cím*: lap sorszáma + lapon belüli relatív cím
  - – *fizikai cím:* keret memóriabeli kezdőcíme + kereten belüli kezdőcím



Keretek feltöltése lapokkal

Laptáblák a baloldali ábrán levő processzusokhoz (A,B,C,D)

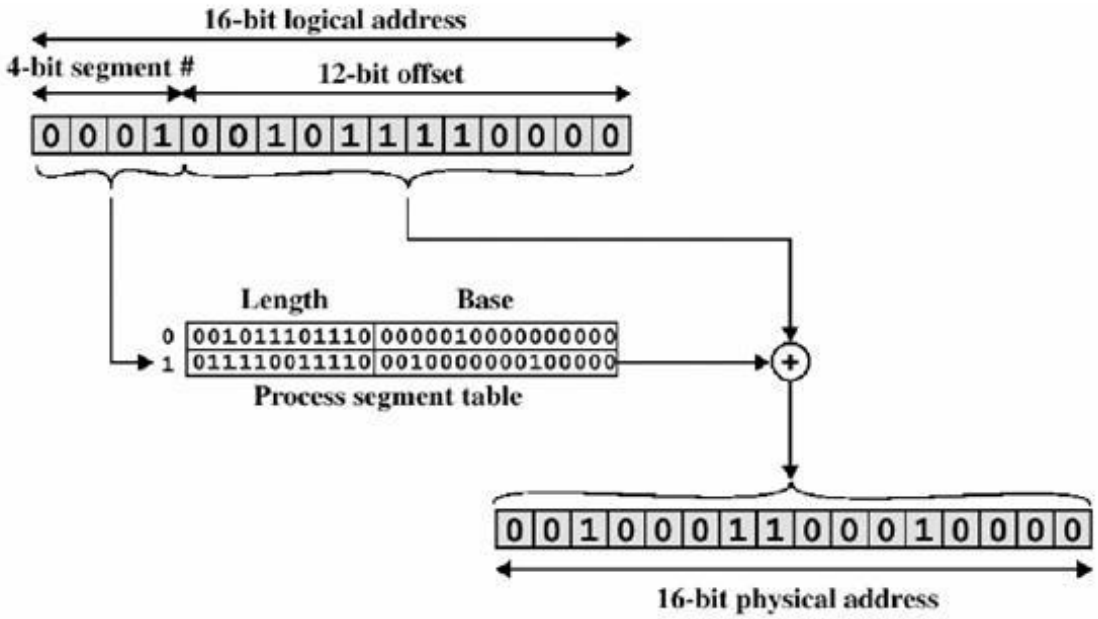


Logikai cím fizikai címmé való fordítása (példa)

Szegmentáció

- Programozói szemléletet tükröző memóriafelosztási séma
- A programokat szegmensekre bontjuk, melyeknek nem kell azonos méretűnek lenniük, de egy maximális szegmensméretnél kisebbnek
- A program tehát szegmensek együttese, a szegmens egy programozói logikai egység:
  - – főprogram, eljárás, függvény, lokális változók, globális változók, közös változók, verem, tömbök
- A logikai cím két részből áll: szegmens szám + offset
- Minden processzushoz tartozik egy szegmenstábla:

- – két dimenziós, felhasználó által definiált címeket egy dimenziós fizikai címekké alakít; a táblában minden bejegyzés tartalmaz egy bázist (a szegmens fizikai kezdőcímét adja meg), mérethatárt (amely a szegmens hosszát mondja meg)
- – Szegmens táblázat bázis regiszter (STBR): a szegmens tábla memóriabeli helyére (kezdőcím) mutat (pointer).
- – Szegmens táblázat hossz regiszter (STLR): a szegmens tábla maximális bejegyzéseinek számát adja meg. az s szegmens szám akkor legális, ha s < [STLR]



Logikai cím fizikai címmé való fordítása (példa)

Szegmentáció lapozással

- Ötlet: a lapozás a külső fragmentációt, a szegmentálás a belső fragmentációt csökkentheti!
- INTEL példa:
  - Egy processzus által használható szegmensek maximális száma: 16K (!)
  - Egy szegmens mérete: 4 GB, lapméret: 4K= 4096 bájt
  - A szegmensek egyik fele privát, ezek címét (adatait) az LDT (Local Descriptor Table) tartalmazza
  - A többi (az összes processzusok által) közösen használt szegmens, ezek címét a GDT (Global Descriptor Table) tartalmazza.
  - Mindkét táblában egy-egy bejegyzés 8 byte, az adott szegmens leírója (kezdőcím és hossz).
  - Logikai cím: szelektor + offset, ahol az offset egy 32 bites érték, a szelektor <s, g, p> alakú, ahol s: szegmens szám, g: GDT, vagy LDT, p: protection (védelem) jelzése
  - A processzor 6 szegmens regisztere egy-egy szegmens egyidejű gyors megcímzését teszi lehetővé.

8. fejezet - Virtuális memória

1. Virtuális memória alapfogalmak

- Egy processzus logikai címtartománya ténylegesen nagyobb lehet, mint a rendelkezésre álló fizikai címtartomány. (Az overlay segíthet, de nehézkes)
- Megoldás: korlátozhatnánk a végrehajtható program méretét a fizikai memória méretére, de ez nem jó megoldás: a programok gyakran tartalmaznak olyan (kód)részeket amelyek rendkívüli eseteket kezelnek. Statisztikailag tekintve ezek olyan ritkák, hogy ez a kódrész szinte sohasem hajtódik végre.
- (Statikus) tömböknek, táblázatoknak, listáknak sokszor olyan sok memóriát allokálnak, amennyire általában nincs szükség. A program bizonyos ágai csak ritkán aktivizálódnak.
- **A virtuális memória koncepciója a felhasználó/programozó memóriaszemléletének teljes szeparálását jelenti a fizikai memóriától.**

A lapozás és a szegmentáció előnyei:

- Több processzus is tartózkodhat egyszerre a főmemóriában
  - – minden processzusnak csak egy része kerül betöltésre
  - – a főmemóriában tartózkodó sok processzus esetén nagyon valószínű, hogy bármely időpillanatban lesz „futásra kész” processzus
- Egy processzus logikai mérete nagyobb lehet, mint a főmemória fizikai mérete

Memória típusai:

- Valós memória
  - – főmemória, melyben a processzusok végrehajtásra kerülnek
- Virtuális memória
  - – a memória a merevlemezen helyezkedik el
  - – hatékony multiprogramozást tesz lehetővé és mentesíti a felhasználót a főmemória méretének korlátai alól, a felhasználó a valós memóriánál nagyobb memóriát érzékel

Probléma:

- vergődés (*thrashing*)
  - – a memóriából olyan processzus kerül ki, melyre azután azonnal szükség van
  - – előfordulhat, hogy a processzoridő nagy részét a blokkok „kicserélgetése” foglalja le (felhasználói utasítások végrehajtása helyett)

Megoldás:

- lokalitás elv:
  - – egy processzuson belül a program és az adathivatkozások klasztereket alkotnak
  - – rövid idő alatt csak kevés számú processzusblokkra lehet szükség
  - – jóslások tehetők arra vonatkozóan, hogy a program mely blokkjaira lesz szükség a jövőben
  - – ezekkel együtt a virtuális memória hatásosan működhet

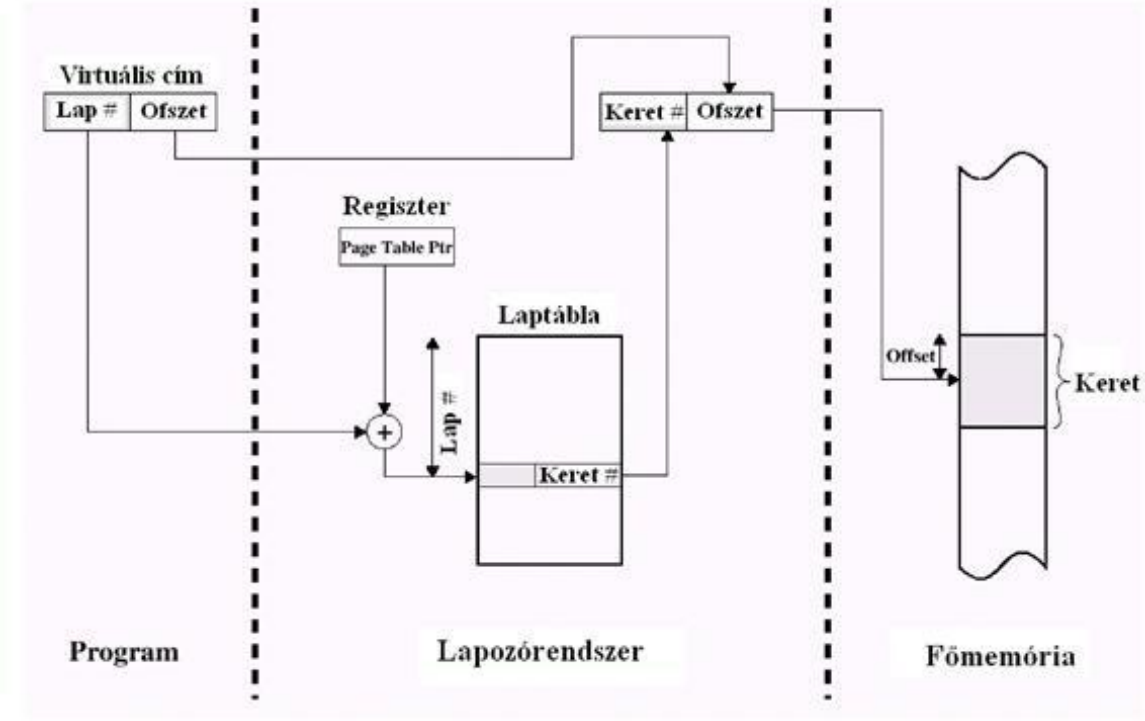
A virtuális memória használatához szükséges feltételek

- a hardvertámogatás kell a lapozáshoz és a szegmentációhoz
- az operációs rendszer rendelkezzen olyan résszel, amely kezeli a lapok és/vagy szegmensek mozgatását a másodlagos és a főmemória között.

2. Lapozás

- minden processzusnak saját laptáblája (Page Table) van
- minden laptáblabejegyzés tartalmazza a főmemóriában található megfelelő lap keretszámát
- egy bit szükséges annak jelzésére, hogy a lap a főmemóriában van, vagy nem
- egy másik, ún. „módosító bit” szükséges annak jelzésére, hogy a lap tartalma megváltozott-e a főmemóriába való utolsó betöltődése óta
- ha nem történt változás, a lap kimentésekor a lemezre való kiírása nem szükséges

Memóriakezelés lapozással



Címfordítás egy lapozó rendszerben

Laptáblák:

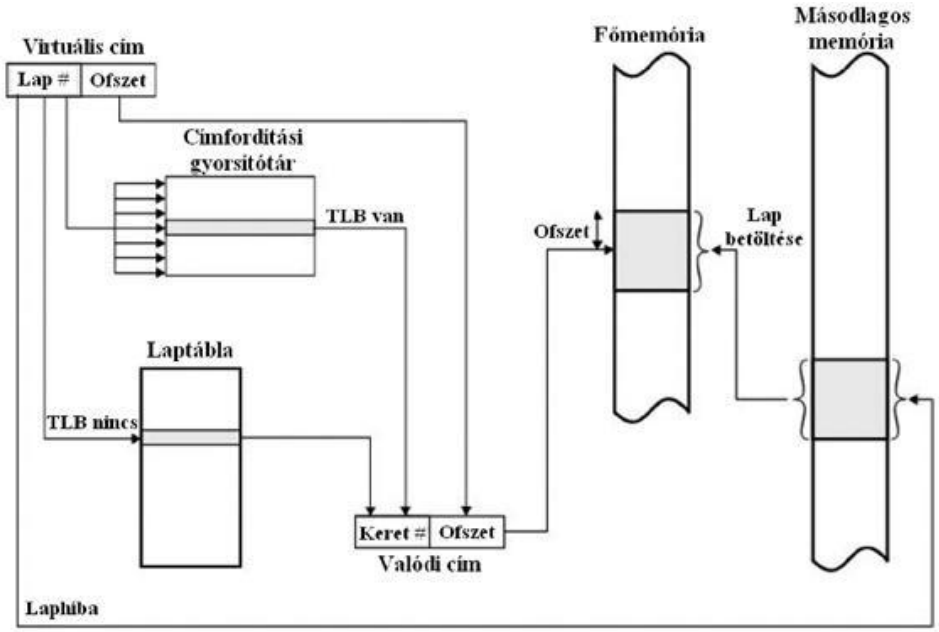
- az egész laptábla túl sok főmemóriát foglalhat le
- laptáblák szintén tárolhatók a virtuális memóriában, így amikor egy processzus fut, laptáblájának csak egy része van a főmemóriában

Címfordítási gyorsítótár (Translation Lookaside Buffer):

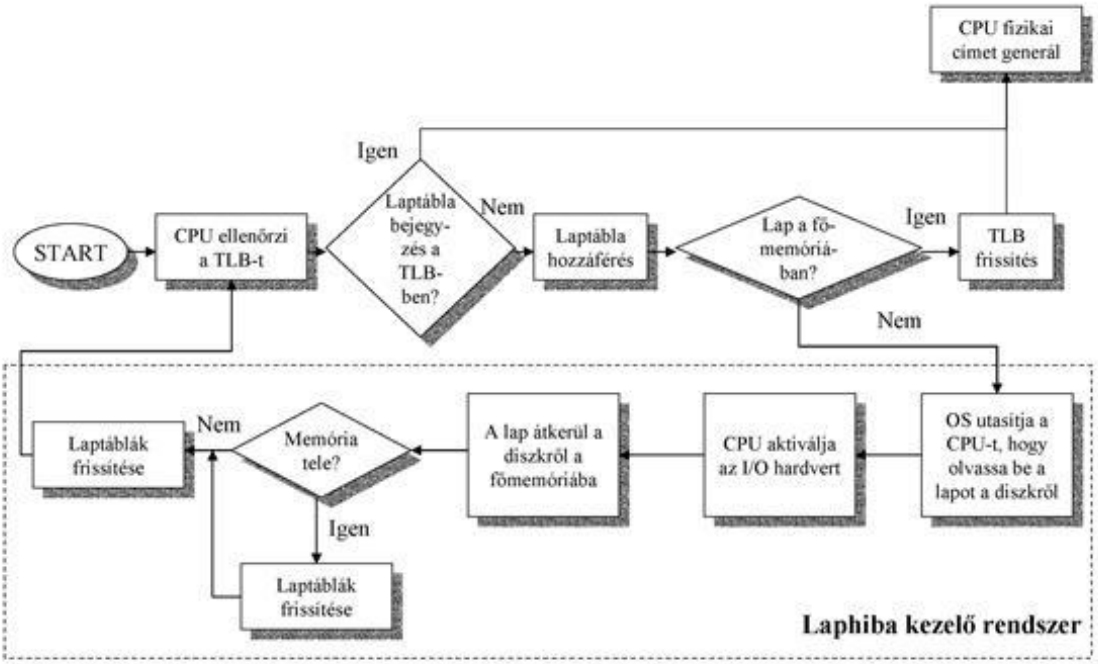
- minden virtuális memóriahivatkozás két fizikai memóriahozzáférést igényel:
  - – behozni a megfelelő laptáblát
  - – behozni az adatot
- a memóriahozzáférési idő ezen duplázásának kivédésre egy nagy sebességű cache memóriát használunk a laptáblabejegyzésekhez – TLB - Translation Lookaside Buffer
- ez tartalmazza a legutóbb használt laptábla bejegyzéseket
- úgy működik, ahogyan egy memória gyorsítótár (cache)

Lapozás (Paging) algoritmusa

- ha adott egy virtuális cím, a processzor megvizsgálja a TLB-t
- ha laptábla bejegyzést talál, a keretsorszámot (címet) kinyeri és megalkotja a valós címet
- ha laptábla bejegyzést nem talál a TLB-ben, a lapsorszámot használja a processzus laptáblájának idexelésére
- először ellenőrzi, hogy a lap a főmemóriában van-e már
  - – ha nincs, egy laphiba történik
- a TLB egy újabb lapbejegyzéssel történő frissítése a következő lépés



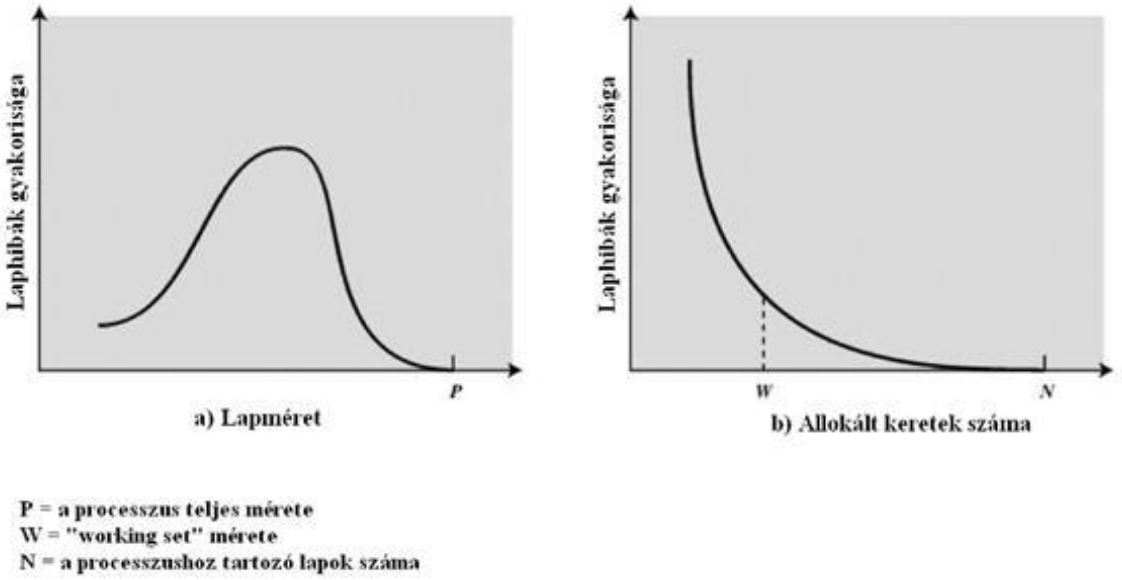
A TLB használata



A lapozás és a TLB működése

Lapméret:

- minél kisebb a lapméret, annál kisebb a belső töredezettség
- minél kisebb a lapméret, annál több lap szükséges egy processzushoz
- minél több lap tartozik egy processzushoz, annál nagyobb lesz a laptábla mérete
- a másodlagos memóriát nagy blokkokból álló adatok mozgatására tervezték, így a nagy lapméret előnyösebb
- minél kisebb a lapméret, annál több lap található a főmemóriában
- a végrehajtás előrehaladtával egyre több olyan lap lesz a memóriában, mely a processzus azon részeit tartalmazza, amely részekre történő hivatkozások a legfrissebbek, így a laphibák száma csökken.
- több lapméret használata rugalmasságot biztosítana TLB hatékony használatához
- nagy méretű lapok használatosak a programutasításokhoz
- kis méretű lapok használatosak a szálakhoz
- a legtöbb operációs rendszer egy lapméretet támogat



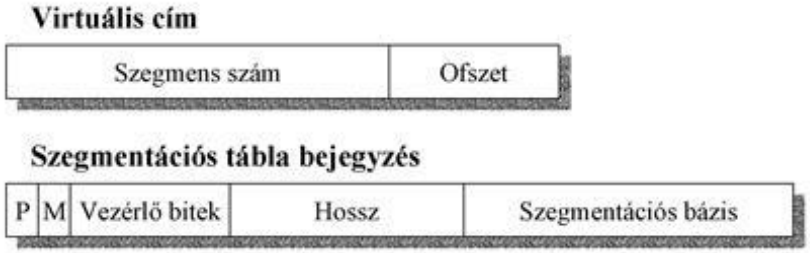
Egy program lapozási viselkedése

3. Szegmentáció

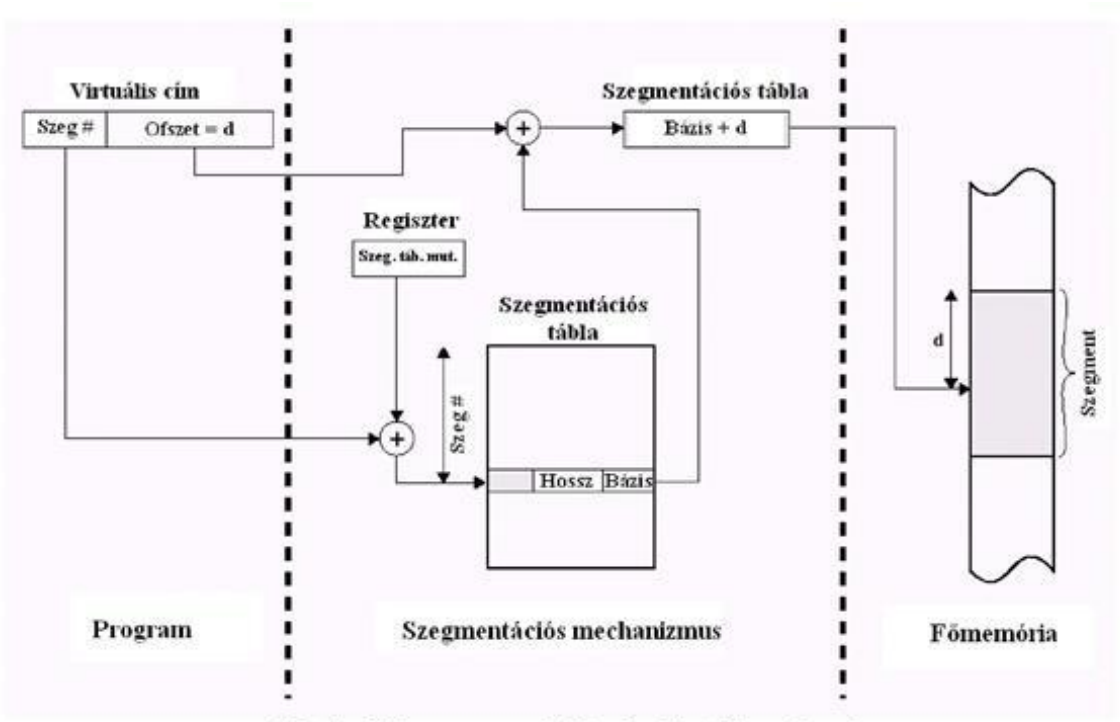
- szegmentáció: lehet különböző méretű (dinamikusan változtatható) memória blokkokkal is dolgozni
- egyszerűsíti a növekvő adatszerkezetek kezelését
- lehetővé teszi programok változtatását és független újrafordítását
- alkalmas a processzusok közötti adatmegosztásra és a védelem megoldására

Szegmentációs tábla:

- minden bejegyzése tartalmazza a megfelelő szegmens főmemóriabeli kezdőcímét, illetve a szegmens hosszát
- egy bit szükséges annak eldöntésére, hogy a szegmens a főmemóriában van-e már
- egy másik bit is kell annak meghatározására, hogy a szegmens memóriába való betöltése után megváltozott-e



Memóriakezelés szegmentációval



Címfordítás szegmentációval működő rendszerben

4. Szegmentáció lapozással az INTEL architektúrában

- Szegmentáció és lapozás: a lapozás láthatatlan, a szegmentáció látható a programozó számára
- a lapozás a külső töredezettséget csökkenti
- a szegmentáció lehetővé teszi az adatszerkezetek növelését, a moduláris felépítést, illetve támogatja a megosztás (és a védelem) megvalósítását
- minden szegmens több, azonos méretű lappá van tördelve

Virtuális cím

Szegmens szám	Lapszám	Ofszet

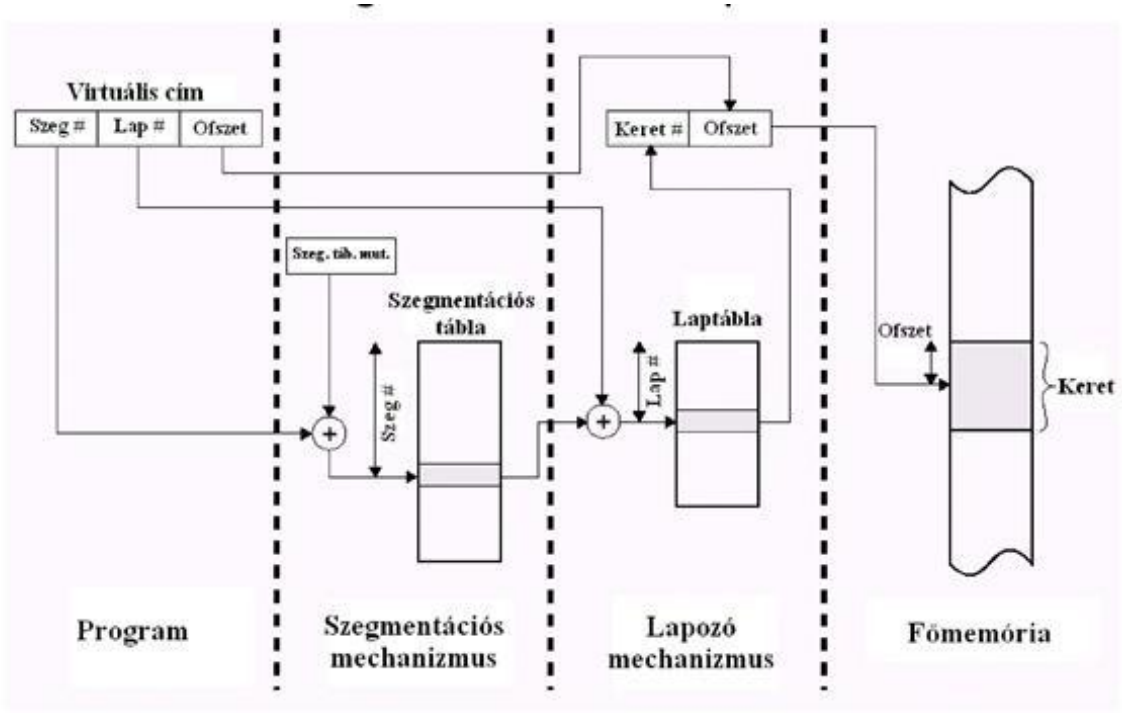
Szegmentációs tábla bejegyzés

Vezérlő bitek	Hossz	Szegmentációs bázis

Laptábla bejegyzés

P	M	Vezérlő bitek	Keretszám

Memóriakezelés lapozás és szegmentáció kombinációjával



Címfordítás lapozás és szegmentáció kombinációjával működő rendszerben

5. Virtuális memóriakezelési stratégiák

- **Behozási stratégia (Fetch Policy)**
  - – azt határozza meg, mikor és mennyi lapot kell a memóriába betölteni
  - – Demand paging esetén csak akkor töltünk be egy lapot a főmemóriába, ha hivatkozás történik rá
    - a processzus első indításakor sok laphiba történik
  - – Prepaging a szükségesnél több lapot hoz be
    - hatékonyabb olyan lapokat behozni, melyek a diszken szomszédosak egymással
- Elhelyezési stratégia (Placement Policy)
  - – meghatározza, hogy a valós memória mely részére tegyen egy adott processzus darabot
  - – szegmentáció esetén:
    - best-fit, next-fit, first-fit, ...
  - – lapozás esetén:
    - megfontolás nélkül ...

Áthelyezési stratégia (Replacement policy)

- Keretzárolás (Frame locking)
  - – ha a keret zárolva van, tartalmát nem lehet kicserélni
  - – pl: egy operációs rendszer kernele
  - – vezérlőrendszerek

- – I/O pufferek
- – minden keretnek megfelel egy „zárolva” bit (lock bit)

- Lapcserélési algoritmusok
  - – optimális stratégia
    - azt a lapot dobjuk ki, melyre való hivatkozás a jövőben a legkésőbb történne
    - lehetetlen implementálni: nem ismerhetjük a jövő eseményeit...
  - – legrégebben használt (Least Recently Used - LRU)
    - laphiba esetén a legrégebben használt lapot dobjuk ki
    - a lokalitási elvből következik, hogy annak valószínűsége, hogy erre a lapra a közeljövőben szükségünk lesz, a lehető legkisebb
    - minden lapot meg kell címkézni az utolsó rá való hivatkozás idejével
    - ennek hatékony implementálása nem egyszerű feladat!
  - first-in, first-out (FIFO)
    - a processzushoz tartozó kereteket körkörös pufferként kezeljük
    - a lapokat körkörösen (round-robin) dobjuk ki
    - ezt a stratégiát a legegyszerűbb megvalósítani
    - a memóriában legrégebb óta tartózkodó lap kerül kidobásra
    - Probléma: előfordulhat, hogy ezekre a lapokra nagyon hamar szükség lesz újból (v.ö. ciklusmag)
  - – "óra algoritmus" (második esély algoritmus)
    - bevezetünk egy jelző bitet minden laphoz (use bit)
    - amikor a lap betöltődik a memóriába, a use bit 0 értéket kap
    - a lapra való hivatkozás után a use bit 1 értéket kap
    - amikor lapot kell cserélni, az első olyan lap lesz dobva, melynek use bit értéke 0
    - a kicserélésre való keresés során minden 1 értékű use bit értékét 0-ra állítjuk
  - – lap pufferelés (Page buffering)
    - a kidobott lapok a következő listák egyikébe kerülnek felvételre
      - – nem módosított lapok listája
      - – módosított lapok listája
  - – mostanában nem használt (Not Recently Used – NRU)
    - – hivatkozás bit (reference bit - R), módosított bit (modified bit - M)
    - – óramegszakítás: R bit törlése periodikusan
    - – véletlenszerűen veszünk ki egy lapot a legalacsonyabb nemüres osztályból:
      - 0. osztály: nem mostanában hivatkozott, nem módosított

    - 1. osztály: nem mostanában hivatkozott, módosított
    - 2. osztály: mostanában hivatkozott, nem módosított
    - 3. osztály: mostanában hivatkozott, módosított
  - – nem gyakran használt (Not Frequently Used - NFU)
    - – hivatkozás bit (R) és szoftveres számlálók
    - – óramegszakítás: periódikusan hozzáadni az R (0 vagy 1) bitet a számlálóhoz
    - – a számlálók jelzik, hogy egy lap milyen gyakran kerül hivatkozásra
    - – laphiba esetén a legkisebb számlálóval rendelkező lapot választjuk ki cserére
    - – NFU és aging együtt jobb megoldás!

Rezidens készlet (szet) felügyelet

- Rezidens szet mérete
  - – Fix kiosztású
    - a processzusok fix számú lapot kapnak
    - amikor egy laphiba történik, ezen processzus egyik lapja lesz cserélve
  - – Változtatható kiosztású
    - a processzusokhoz rendlet lapok száma a processzus élete során változik
- Változtatható kiosztású, globális
  - – a legegyszerűbb megvalósítani, így a legtöbb operációs rendszer adoptálta
  - – az operációs rendszer egy listát tart a szabad keretekről
  - – laphiba esetén szabad keretet adunk a processzus rezidens szetjének
  - – ha nincs szabad keret, egy másik processzustól cserél egyet
- Változtatható kiosztású, lokális
  - – új processzus esetén a keretek lefoglalása az alkalmazás típusától illetve más kritériumoktól függ
  - – laphiba esetén az okozó processzus rezidens szetjéből választunk lapot
  - – kiosztás újraértékelése időről időre

Tisztítási stratégia

- A tisztítási stratégia feladata annak eldöntése, mikor szükséges egy módosított lapot a másodlagos tárra menteni
- Igényelt tisztítás (demand cleaning)
  - – egy lap csak akkor lesz kiírva, amikor kiválasztjuk cserére
- Előtisztítás (precleaning)
  - – lapok kimentése még mielőtt lapkeretjeikre szükség lenne, a lapok kiírása kötegekben (batch) történik
- A legjobb közelítés: page buffering

- – a kicserélt lapokat két listában helyezzük el
  - módosított lapok és nem módosított lapok
- – a módosított listában lévő lapokat periodikusan kiírjuk és a nem módosított lapok listájába tesszük
- – a nem módosított lapok listájában levő lapok hivatkozásuk esetén újra visszanyerhetők, vagy véglegesen elvesznek (törlődnek) ha a lapkeretüket egy másik lap kapja meg

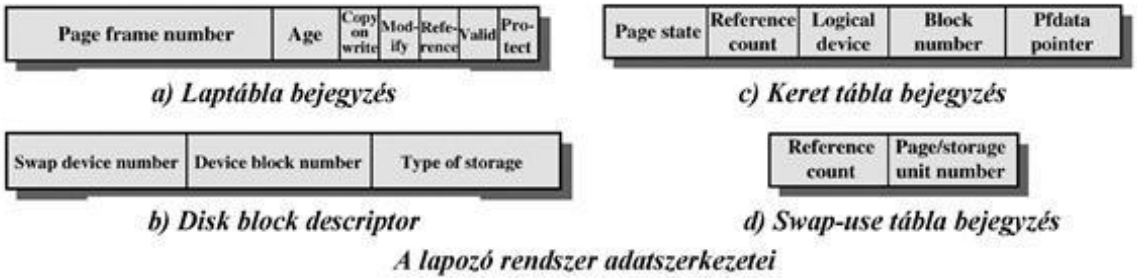
Betöltésvezérlés

- Betöltésvezérlés:
  - meghatározza azon processzusok számát, melyek a főmemóriába kerülnek
  - ha túl kevés processzus, túl sok alkalommal lesz minden processzus blokkolt és túl sok idő fog elmenni cserével (swapping)
  - túl sok processzus vergődéshez vezethet
- Processzus felfüggesztése:
  - legkisebb prioritású processzust
  - hibázó processzust
    - – ezen processzusnak nincs működő része a főmemóriában, így úgyis blokkolva van
  - utolsó aktivált processzust
  - a legkisebb rezidens szettel rendelkező processzust
    - – ezen processzus újratöltése igényli a jövőben a legkisebb erőfeszítést
  - legnagyobb processzust
    - – a legtöbb szabad keretet igényli
  - legnagyobb hátralevő végrehajtási idővel rendelkező processzust

6. A Unix és a Windows 2000 virtuális memóriakezelése

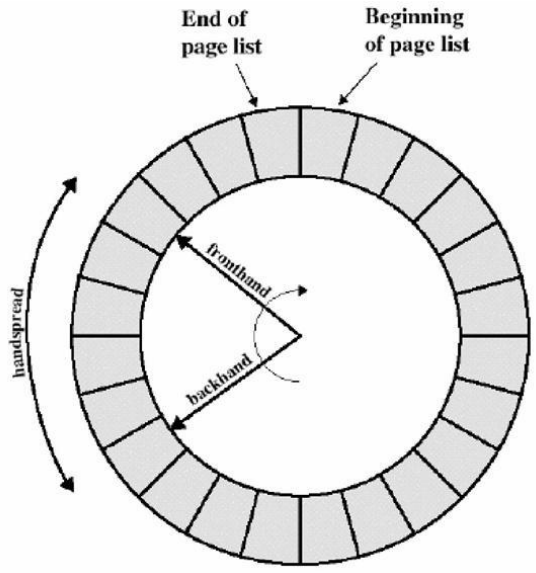
A Unix memóriakezelése

- Unix régebbi verziói: változtatható méretű partícionálás használata
- Lapozó rendszer felhasználói processzusokhoz
  - – laptábla: egy laptábla egy processzushoz illetve egy bejegyzés a processzus minden táblájához
  - – Disk block descriptor: a virtuális memória lemezen levő másolatát írja le
  - – Keret tábla (page frame data table): a valós memória kereteit írja le
  - – „Swap-use” tábla: minden egyes csereeszközhöz
- Kernel memory allocator (kernel számára memória lefoglalása)



A lapozó rendszer adatszerkezetei

- A Unix memóriakezelése az óra stratégia továbbfejlesztett változatát alkalmazza („két-karú” óra stratégia)
- – reference bit használata
  - értékét 0-ra állítjuk, mikor a lapot először behozzuk
  - értékét 1-re állítjuk, mikor a lapra hivatkozás történik
- – az elülső kar végigpásztázza a lapokat és a referencia bitet 0-ra állítja
- – kicsivel később a hátulsó kar szintén végigpásztázza a lapokat és összegyűjti a 0 referencia bittel rendelkezőket
- Kernel Memory Allocator
  - – a lapozó rendszer itt nem használható, a legtöbb blokk kisebb a tipikus lapméretnél
  - – dinamikus memóriahozzárendelés

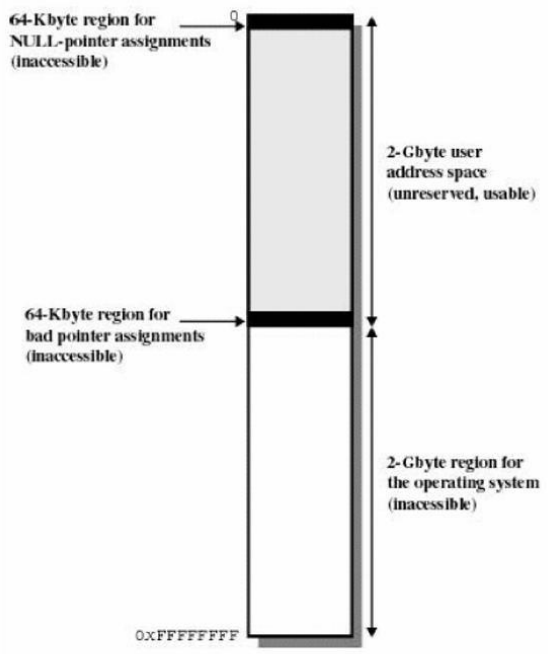


Két-karú órastratégia

A Windows 2000 memóriakezelése

- W2K virtuális címtér
  - – minden processzusnak egy 32 bites címtere van
    - 2 GB felhasználó processzusoknak
    - 2 GB a rendszernek, melyet az összes processzus megosztva használhat

- W2K lapozórendszer
  - – egy lap a következő állapotokban lehet:
    - elérhető
    - lefoglalt
    - bizományban



Windows virtuális címtere

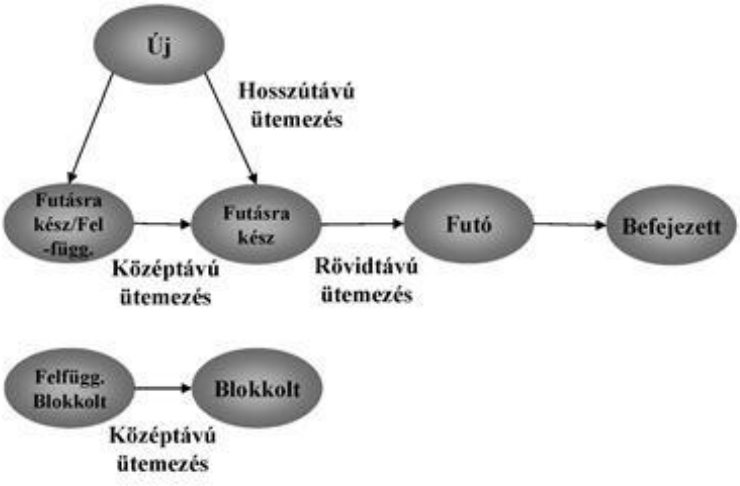
9. fejezet - Egy- és többprocesszoros folyamatütemezés

1. Egyprocesszoros ütemezés

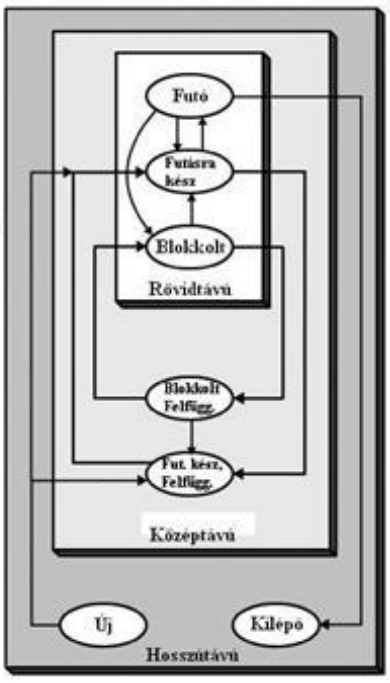
Az ütemezés célja: válaszidő csökkentése, processzor hatásfokának növelése

Ütemezés típusai:

- Hosszútávú ütemezés (Long term scheduling – Job Scheduler)
  - – meghatározza, hogy mely processzusok kerülnek készenléti állapotba
  - – a multiprogramozás fokát határozza meg
  - – minél több processzus van, annál kevesebb futási idő jut egy processzusra
- Középtávú ütemezés (Medium term scheduling)
  - – a csereszolgáltatás (swapping) része, felfüggesztendő processzusok kiválasztása
  - – a multiprogramozás felügyeletéért felelős
- Rövidtávú ütemezés (Short term scheduling – CPU Scheduler)
  - – a diszpécser (dispacther) adja át a vezérlést a kiválasztott processzusnak
  - – leggyakrabban használt ütemezési típus
  - – hívása egy külső esemény bekövetkezésének hatására történik
    - például: óra megszakítás, I/O megszakítás, rendszer hívások, szignálok



Ütemezés és a processzus állapotai



Ütemezési szintek

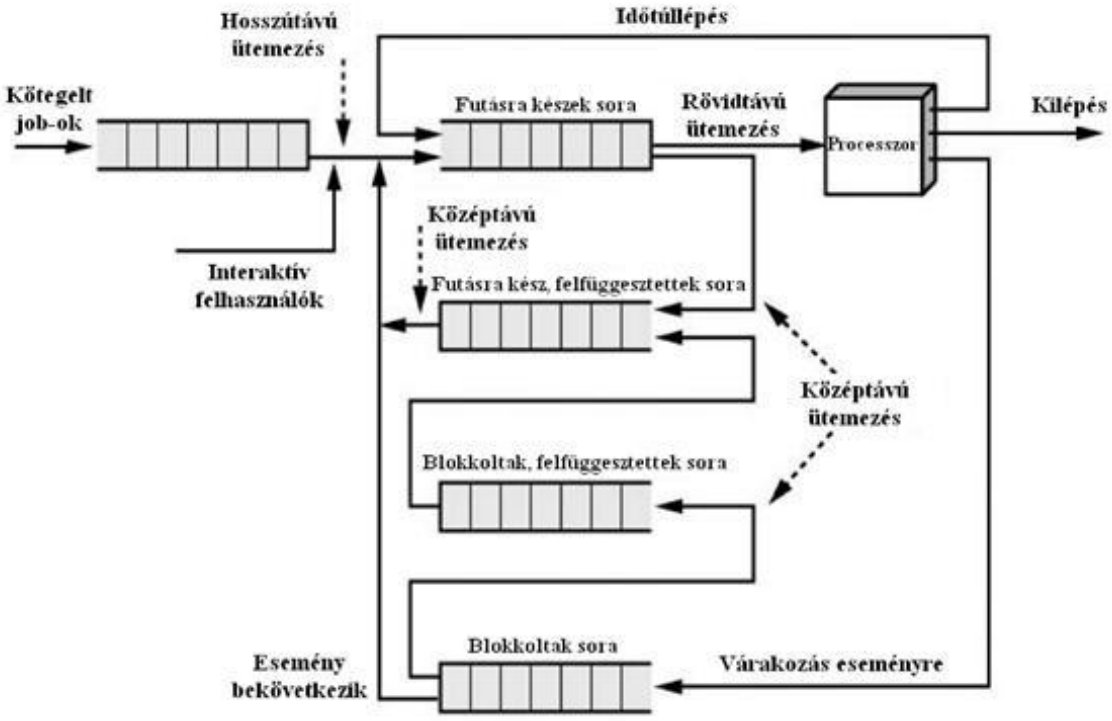
2. Ütemezési algoritmusok

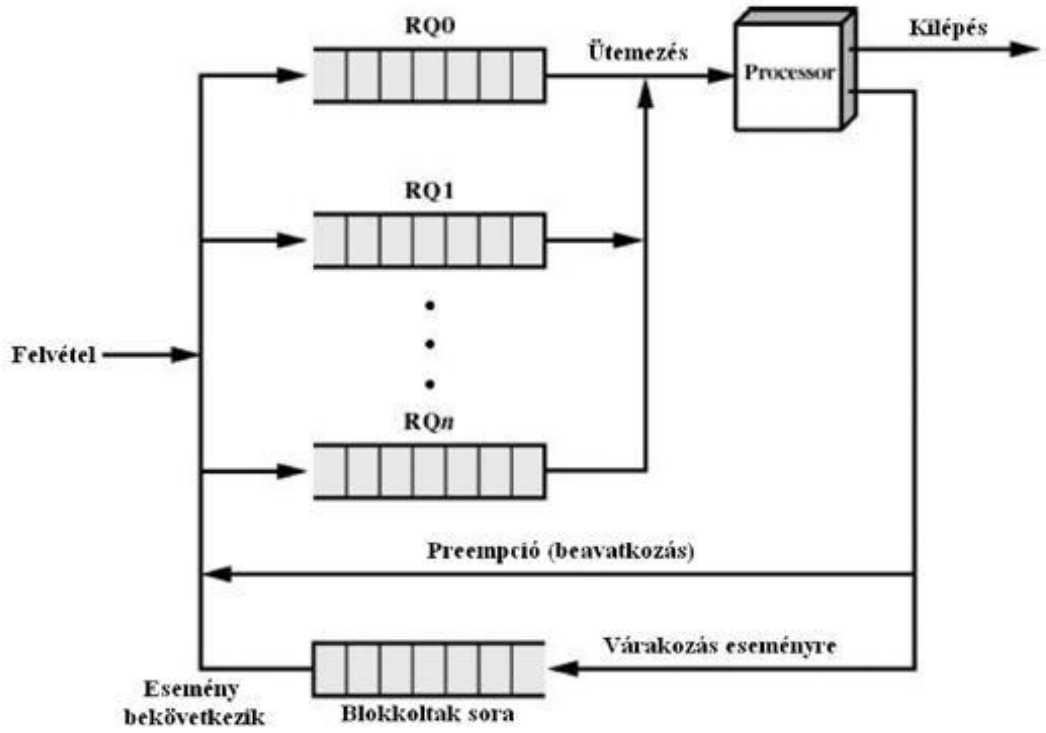
Rövidtávú ütemezési feltételek:

- Felhasználó szemszögéből
  - – a válaszidő csökkenjen (a kérelem benyújtása és az első válasz között eltelt idő)
- Rendszer szemszögéből
  - – a CPU az idő minél nagyobb részében legyen elfoglalt
- Teljesítménnyel kapcsolatos
  - – átbocsátó képesség (egységnyi idő alatt befejezett processzusok száma) növekedjen, illetve végrehajtási idő (memóriába kerülés ideje + várakozási idő + CPU + I/O idő) csökkenjen
  - – **átlagos várakozási idő (készenléti sorban eltöltött idő) csökkenjen**

Prioritási sorrend szerinti kiszolgálás:

- az ütemező mindig a nagyobb prioritású processzust választja
- több készenléti sor használata (minden prioritási szinthez)
- alacsony prioritásúak éhezést, éhhalált szenvedhetnek!
  - – megoldás: „kora” alapján egy processzus megváltoztathatja a prioritását (aging)





Prioritás szerinti ütemezési sorok

3. Ütemezési stratégiák

Döntési helyzetek, módok:

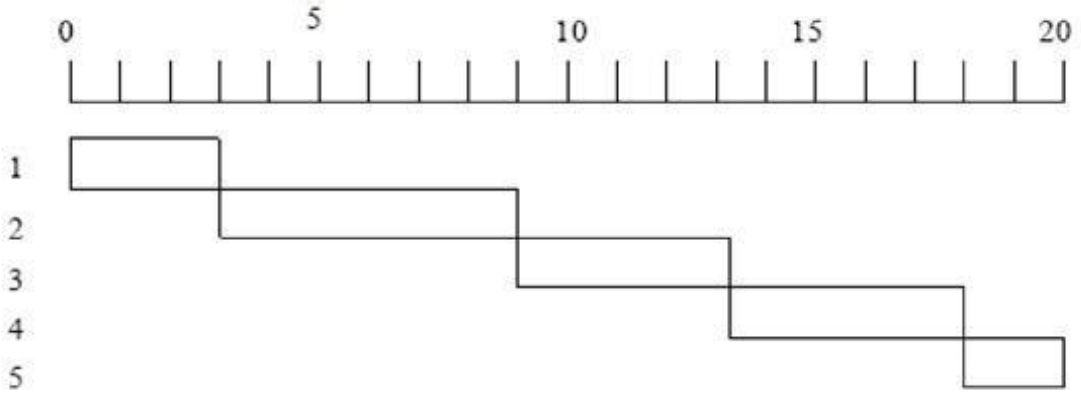
- Nem beavatkozó (nem preemptív)
  - – a processzus maga mond le a CPU-ról (futó állapotból várakozó állapotba kerül
  - –I/O eseményre vár
  - – vagy megáll
- Beavatkozó (preemptív)
  - – egy futó processzust az operációs rendszer megszakít és készenlétei állapotba helyez, vagy várakozó állapotból készenléti állapotba küld
  - – jobb szolgáltatást tesz lehetővé, hiszen egy processzus sem sajátíthatja ki a CPU-t túl sok ideig

Processzus ütemezési példa a különböző stratégiák bemutatásához

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

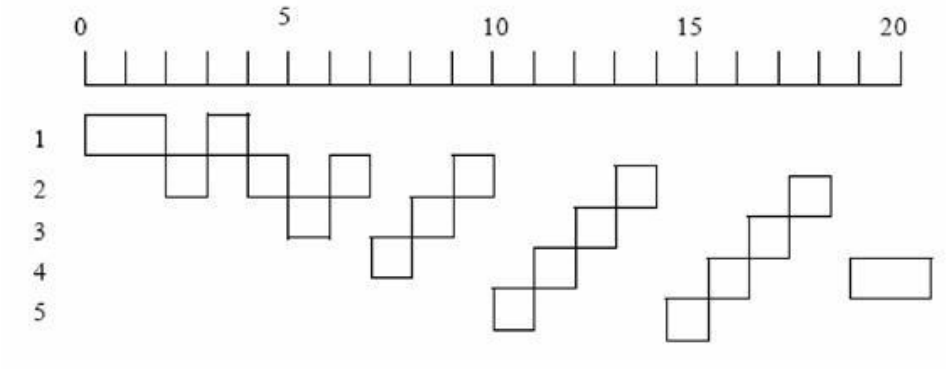
Igénybejelentési sorrend szerinti kiszolgálás- First come first served (FCFS):

- minden processzus a készenléti sorba kerül
- mikor az aktuális processzus végrehajtási megszűnik, a készenléti sorban legrégebb óta váró processzus lesz kiválasztva végrehajtásra
- egy rövid processzus túl sokáig várhat végrehajtása előtt...
- az átlagos várakozási idő nagyon szórhat! (konvoj hatás)



Körleosztásos - Round Robin:

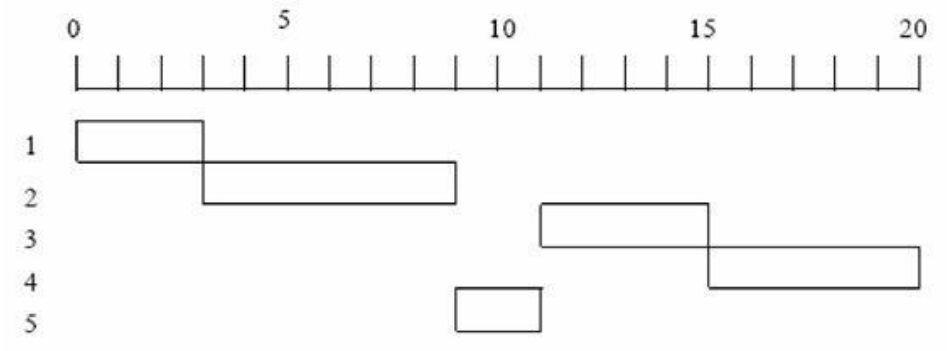
- beavatkozás egy óra alapján: minden processzus sorban egy meghatározott ideig (q) használhatja a CPU-t (q = 10-100 millisec.)
- egyenlő időközönként óramegszakítás generálódik
- megszakítás esetén az éppen futó processzus a készenléti sorba kerül és a következő processzus kerül futó állapotba
- n processzus esetén a várakozási idő: (n-1)/q



Rövidebb igény először - Shortest Process Next:

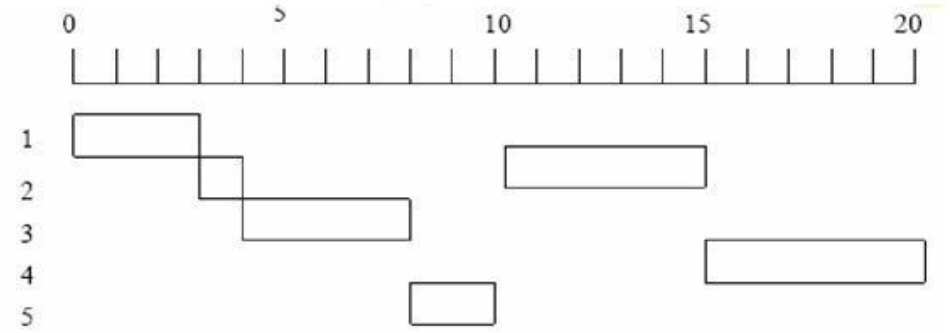
- nem preemptív ütemezés
- a legkisebb várható processzorfoglalási idővel rendelkező processzus kerül kiválasztásra
- hosszabb processzusok háttérbe szorulhatnak, éhezés!
- ha a megjósolt feldolgozási idő nem helyes, az operációs rendszer megszakíthatja a processzust (preemptív ütemezés)

- elméletileg minimalizálja az átlagos várakozási időt



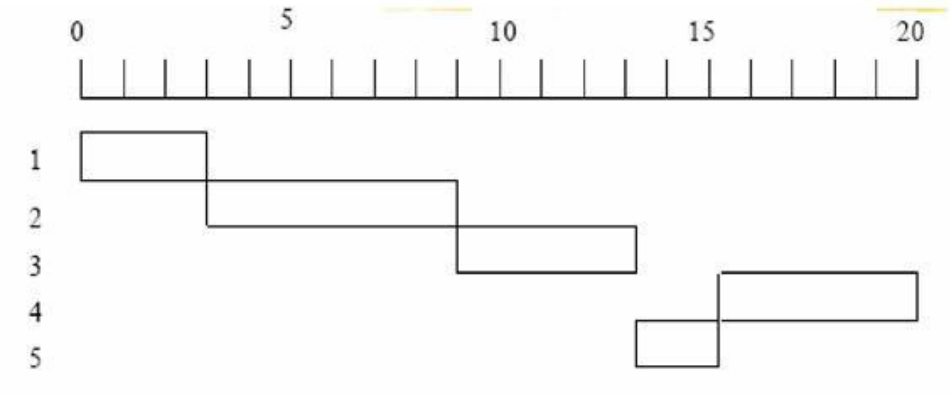
Rövidebb megmaradó idő- shortest remaining time next:

- a rövidebb igény először preemptív változata
- a feldolgozási idő becslése szükséges



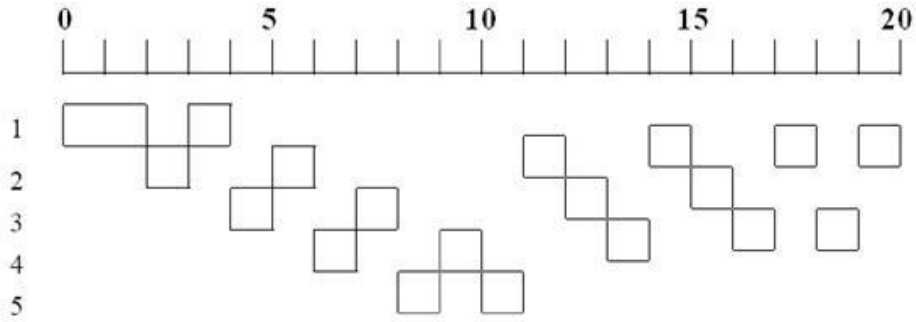
Magasabb válaszarány először:

- a legnagyobb R=(w+s)/s arányú processzus választása következőnek (w: proceszorra való várakozással töltött idő; s: várható kiszolgálási idő)



Visszacsatolt ütemezés - Feedback Scheduling

- a hosszabban futó processzusok „büntetése”



4. A Unix egyprocesszoros folyamatütemezése

- Az ütemezés kernel illetve felhasználói módban eltér:
  - – felhasználói módban: preemptív prioritásos ütemezés, időben változó prioritások; egyenlő prioritású processzusok esetén körleosztásos ütemezés
  - – kernel módban: nem preemptív ütemezés, rögzített prioritású processzusok
- A prioritások minden századik óraciklusban újraszámolásra kerülnek
- A prioritást meghatározó tényezők (felhasználói mód):
  - – kedvezési szám (nice number): a felhasználó által meghatározott szám
  - – CPU használatra vonatkozó szám: öregítés (aging) illetve egyenletes CPU használat biztosítása
- Korrekciós faktorok használata:
  - – rendszer terheltségének figyelembevétele
  - – sok processzus: lassú öregítés (illetve fordítva)
  - – várakozó processzusok számával fordítottan arányos érték

5. Többprocesszoros folyamatütemezés

- Lazán csatolt többprocesszoros rendszer
  - – minden processzornak saját memóriája és I/O csatornái vannak
  - – speciális funkcióval rendelkező processzorok (pl.: I/O processzorok)
  - – egy fő (master) processzor által vezérelt
- Szorosan csatolt többprocesszoros rendszer
  - – a processzorok osztoznak a főmemórián
  - – operációs rendszer által vezérelt

Független párhuzamosság:

- külön alkalmazás vagy job
- nincs szinkronizáció
- több, mint egy processzor használható
  - – az átlagos válaszidő így kisebb lesz

Durva-szemcsés párhuzamosság:

- processzusok közötti gyenge szinkronizáció
- egyprocesszoros multiprogramozott rendszereknél használatos
  - – kis változtatással multiprocesszoros rendszerek is támogatják

Közép-szemcsés párhuzamosság:

- párhuzamos feldolgozás vagy multitaszking egy alkalmazáson belül
- egy alkalmazás szálak összességéből áll, melyek általában gyakran kölcsönhatnak, kommunikálnak egymással

Finom-szemcsés párhuzamosság:

- különösen nagy fokú párhuzamosságot igénylő alkalmazásoknál
- speciális terület...

A processzusok processzorokhoz való rendelése

- processzorok, mint közös erőforrások; a processzusok processzorokhoz való rendelése igény szerint történik
- a processzusok véglegesen egy processzorhoz vannak jelölve
  - – dedikált rövid-idejű sor minden processzornak
  - – kevesebb „overhead”
  - – egyik processzor üresen járhat, míg a másik processzor elmaradásban van
- globális sor: ütemezés minden elérhető processzor bevonásával
- mester/szolga (Master/slave) architektúra
  - – a kulcsfontosságú kernelfüggvények mindig egy kiegészítő speciális processzoron futnak
  - – a mester felelős az ütemezésért
  - - a szolga szolgáltatáskérelmet küld a mesternek
  - – hátrányok: a mester hibája megbéníthatja a rendszert
- egyenrangú architektúra
  - – operációs rendszer bármelyik processzoron futhat
  - – minden processzor önütemezést végez
  - – operációs rendszer bonyolultabbá válik:
    - meg kell bizonyosodni, hogy több processzor nem választja ki ugyanazt a processzust

Processzus ütemezés

- egyszerű sor minden processzusnak
- többszörös sor használata a prioritásokhoz
- miden sor a közös processzorokhoz rendelve
- speciális ütemezési elvek kevésbé fontosak több processzor esetén

Szálak ütemezése

- egy alkalmazás olyan szálak gyűjteménye lehet, melyek együttműködve illetve konkurens módon hajtódnak végre ugyanazon a címtéren
- külön processzoron futó szálak jelentősen növelik a teljesítményt
- terhelés-megosztás (Load sharing)
  - – a processzusokat nem rendeljük külön-külön a processzorokhoz, globális sor alkalmazása
- csoportos ütemezés
  - – összefüggő szálak futásának ütemezése úgy, hogy az egymással párhuzamosan dolgozó processzorokon egyidőben fussanak
- ajánlott processzorhozzárendelés
  - – a szálak hozzárendelése egyedi processzorokhoz
- dinamikus ütemezés
  - – a szálak számának változtatása a végrehajtás folyamata közben

Terhelés-megosztás

- a terhelés egyformán van elosztva a processzorok között
- nincs szükség központi ütemezőre
- globális sor használata

Hátrányai:

- a központi sorhoz kölcsönös kizárás kell
  - – torlódás léphet fel, ha több, mint egy processzor néz munka után egy időben
- felfüggesztett szálak futtatásának folytatása kis valószínűséggel történik ugyanazon a processzoron
  - – gyorsítótár használata kevésbé hatékony
- ha az összes szál a globális sorban van, egy program összes szálja nem szerezhet hozzáférést a processzorhoz egy időben

Csoportos ütemezés

- egy egyszerű processzus szálainak szimultán ütemezése
- hasznos az összes olyan alkalmazásnál, ahol a teljesítmény drasztikusan csökken, ha az alkalmazás valamelyik része nem fut
- a szálakat gyakran kell egymáshoz szinkronizálni

Dinamikus ütemezés

- a szálak számának dinamikus változtatása rendszereszközök segítségével
- az operációs rendszer szervezi a processzusok betöltését
  - – üresen járó processzorok hozzárendelése processzusokhoz
  - – újonnan érkező processzusok kiosztása olyan processzorhoz, mely olyan job-ok által van használva, amelyek aktuálisan több processzort is használnak

- – a kérés fenntartása, míg egy processzor elérhető nem lesz
- – processzorok jobokhoz való rendelése FCFS alapon

6. Valós idejű rendszerek folyamatütemezése

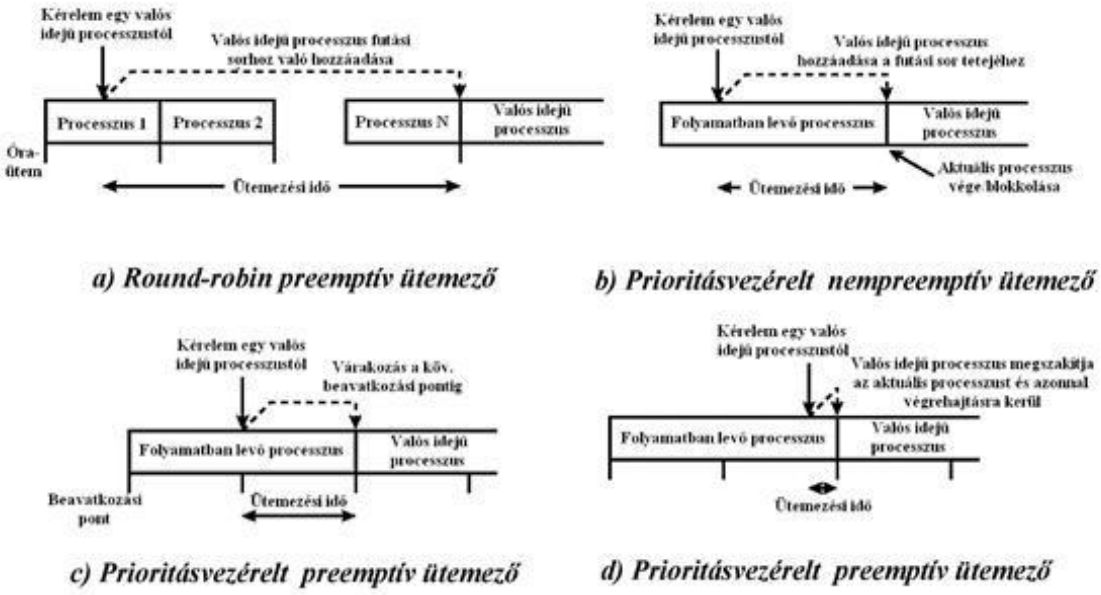
Valós idejű rendszerek:

- a rendszer pontosságát, jóságát nem csak a számítások eredménye határozza meg, hanem az ehhez szükséges idő is
- processzusok megpróbálnak külvilági eseményekre hatást gyakorolni illetve külső folyamatokat vezérelni
- a processzusoknak az események „történésével” lépést kell tartaniuk (real-time)
- Például:
  - – Kísérletek vezérléséhez
  - – Folyamatirányító rendszerek
  - – Robotika – Légiforgalom irányítás
  - – Telekommunikáció
  - – Hadászati vezérlőrendszerek

Valós idejű operációs rendszerek jellemzői:

- Determinisztikus
  - – műveletek végzése előre meghatározott időközönként
  - – mértékét az is meghatározza, hogy mennyi időt késleltet az operációs rendszer, míg egy megszakítást elfogad
- Befolyásolhatóság
  - – mennyi idő szükséges az operációs rendszernek, hogy egy megszakítást kiszolgáljon:
    - megszakítás végrehajtásának megkezdéséhez szükséges idő
    - megszakítás feldolgozásához/végrehajtásához szükséges idő
- Felhasználói vezérlés
  - – a felhasználó állapítja meg a prioritásokat
  - – lapozás meghatározása
  - – milyen processzusoknak kell folyamatosan a főmemóriában lenni
  - – a használható lemez algoritmus megállapítása
  - – processzusok jogainak beállítása
- Megbízhatóság
  - – a teljesítmény csökkenésének katasztrofális következményei is lehetnek
  - – feladat a problémák javításának megpróbálása illetve negatív hatásának minimalizálása
  - – magas prioritású feladatok előnyben

- Valós idejű operációs rendszerek további jellemzői:
  - gyors context switch
  - kis méret
  - külső megszakításokra való gyors reagálás képessége
  - multitaszking, olyan processzusok közötti kommunikációs eszközökkel, mint például szemaforok és szignálok
  - fájlok, melyek nagy adatgyűjtési sebességet érhetnek el: – speciális soros hozzáférésű állományok használata
  - prioritás alapú beavatkozó ütemezés
  - azon időszakok minimalizálása, amikor új megszakítás nem lehetséges
  - feladatok határozott idejű késleltetése
  - speciális riasztások és időtúllépések



Valós idejű processzusok ütemezése

Valós idejű ütemezés: algoritmus osztályok

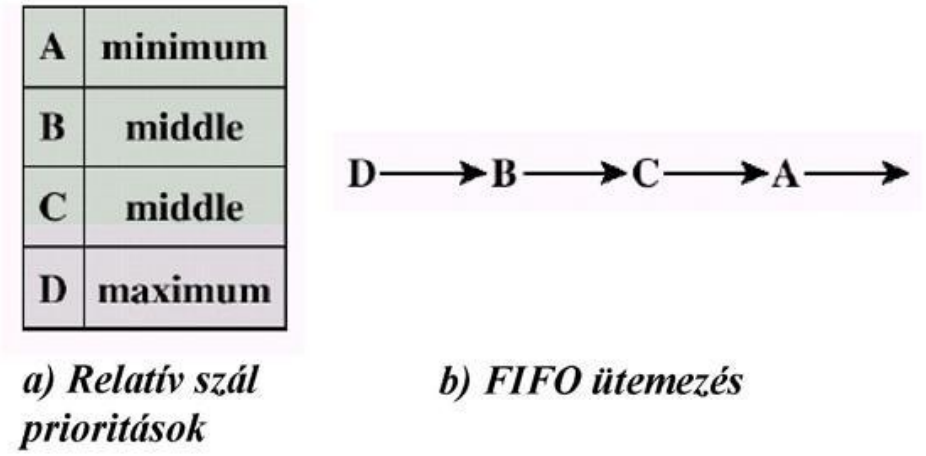
- Statikus, táblázat-vezérelt megközelítés
  - – előzetes végrehajthatósági tervet készít, az ütemezés ennek alapján történik
- Statikus, prioritás-vezérelt preemptív megoldás
  - – a szituáció elemzése statikus, de az eredmények alapján az ütemezést hagyományos, prioritás alapú ütemező végzi
- Dinamikus, terv-alapú megközelítés
  - – új taszk indítása esetén az indítást csak akkor engedi, ha az újratervezett ütemezési terv alapján az időzítési elvárások tarthatók
- Dinamikus, „best effort” megközelítés

- – nem végzünk megvalósíthatósági elemzést, a rendszer mindent megtesz, hogy a határidőket tartsa

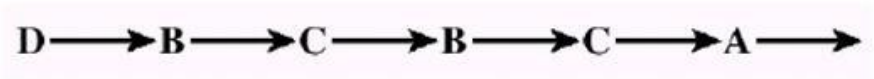
7. A Linux, a Unix és a Windows 2000 ütemezési tulajdonságai

Linux ütemezés

- ütemezési osztályok
  - – SCHED_FIFO: first in – first – out típusú valós idejű szálak
  - – SCHED_RR: Körkörös típusú valós idejű szálak
  - – SCHED_OTHER: Más, nem valós idejű szálak
- minden osztályban több prioritás használata



a) Relatív szál prioritások *b) FIFO ütemezés*

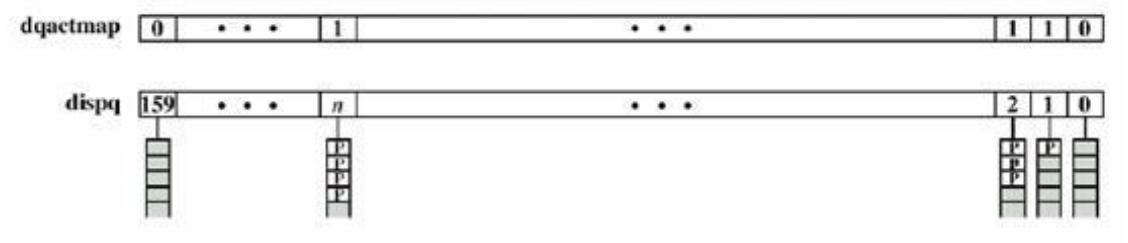


c) Körkörös ütemezés

Linux ütemezésére példa

Unix ütemezés

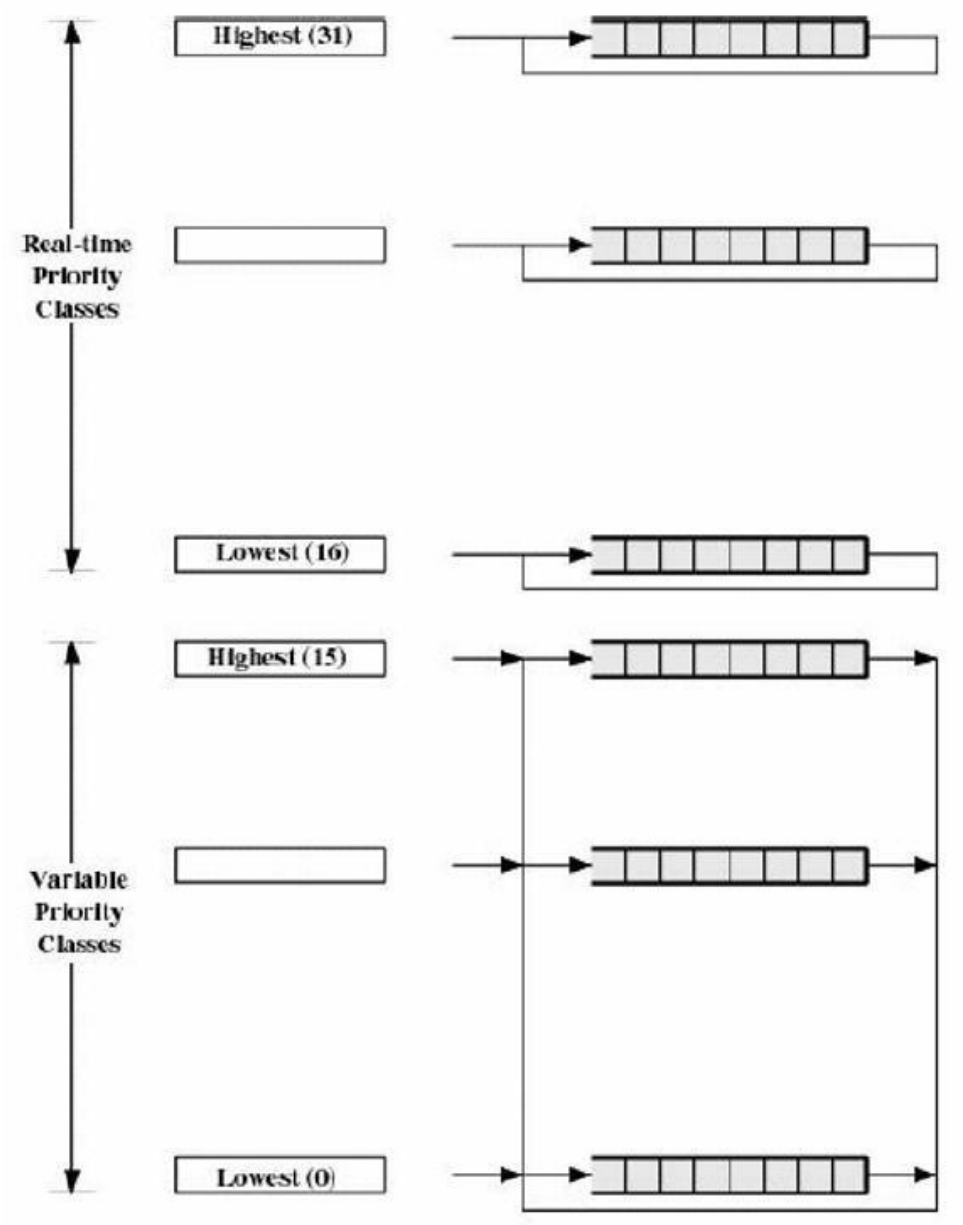
- leginkább előnyben a valós idejű processzusok
- következő a kernel-módú processzusok
- legkevésbé előnyben a felhasználó-módú processzusok



Unix ütemezési sorok

Windows 2000 ütemezés

- a prioritások szerinti két osztály
  - – valós-idejű: minden szálnak fix, meg nem változtatható prioritása van
  - – változtatható: a szálak prioritása élettartamuk alatt változtatható
- minden osztályon belül 16 prioritási szint van
- ütemezés: prioritás vezérelt preemptív ütemezés



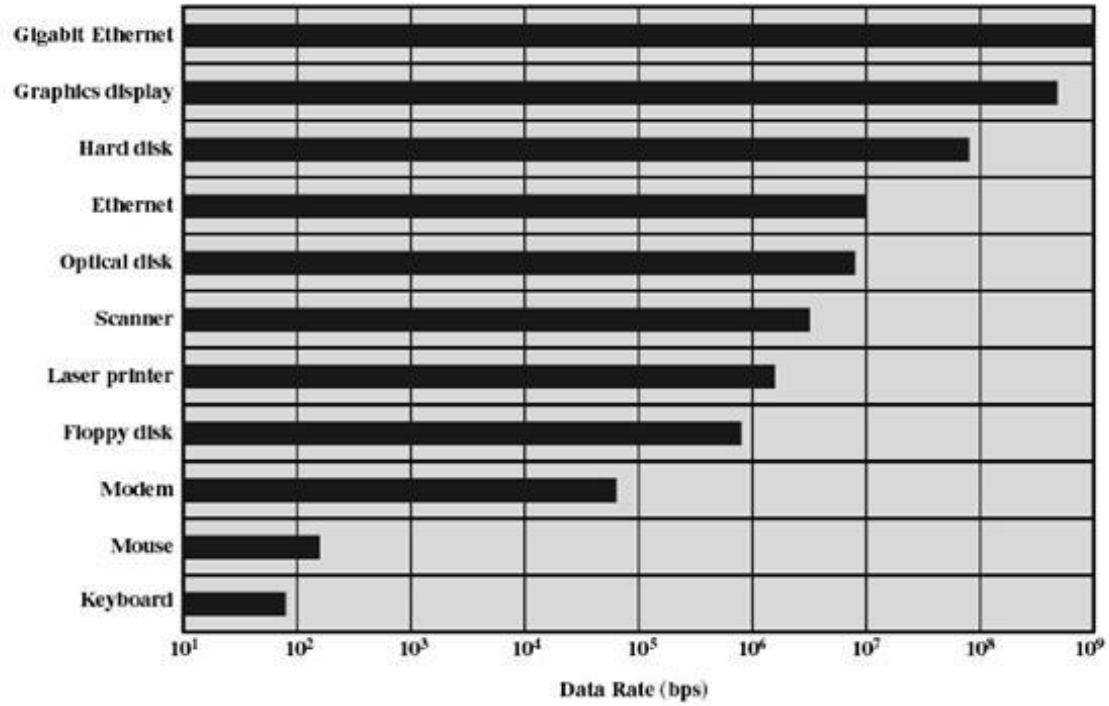
10. fejezet - I/O kezelés és lemezütemezés

1. I/O eszközök

- Felhasználó által olvasható
  - – a felhasználóval történő kommunikációra használják
  - – nyomatók
  - – kijelző terminálok
    - kijelző, billentyűzet, egér
- Gép által olvasható
  - – elektronikus eszközzel történő kommunikációhoz
  - – lemez-(disk) és szalag- (tape) meghajtók
  - – érzékelők
  - – kontrollerek
- Kommunikáció
  - – távoli eszközökkel történő kommunikáció eszközei
  - – digitális vonalvezetők
  - – modemek

I/O eszközök közötti különbségek

- Adatátviteli sebesség (Data rate)
  - – több nagyságrend is lehet az adattovábbítási sebességek közötti különbség



I/O eszközök bitrátái

- Felhasználási terület (Application)
  - – lemezen való fájltárolás esetén fájlkezelő szoftverre van szükség
  - – virtuális memória lemezen való kialakításához speciális hardver és szoftver szükséges
  - – rendszeradminisztrátor által használt terminálnak nagyobb prioritása lehet
- Vezérlés összetettsége
- Adatátvitel egysége (Unit of transfer)
  - – átvitel bájtok folyamaként (pl. terminál I/O), vagy
  - – nagyobb blokkokban (lemez I/O)
- Adatábrázolás
  - – kódolási elképzelés
- Hibalehetőségek (Error conditions)
  - – az eszközök különbözőképpen reagálnak a hibákra

2. Az I/O megvalósítása

- Programozott I/O
  - – a processzor I/O utasítást ad ki egy processzus nevében
  - – a művelet befejeződéséig a processzus várakozó státuszba kerül, majd végrehajtása folyatódik (wait loop)
- Megszakításvezérelt I/O
  - – a processzor I/O utasítást ad ki egy processzus számára

- – a processzor folytatja a rá következő utasítások végrehajtását (ha az I/O nem szükséges a folytatáshoz, ha igen, akkor egy másik processzus kerül végrehajtásra)
- – az I/O egység az I/O művelet befejezésekor egy megszakítást küld a processzornak

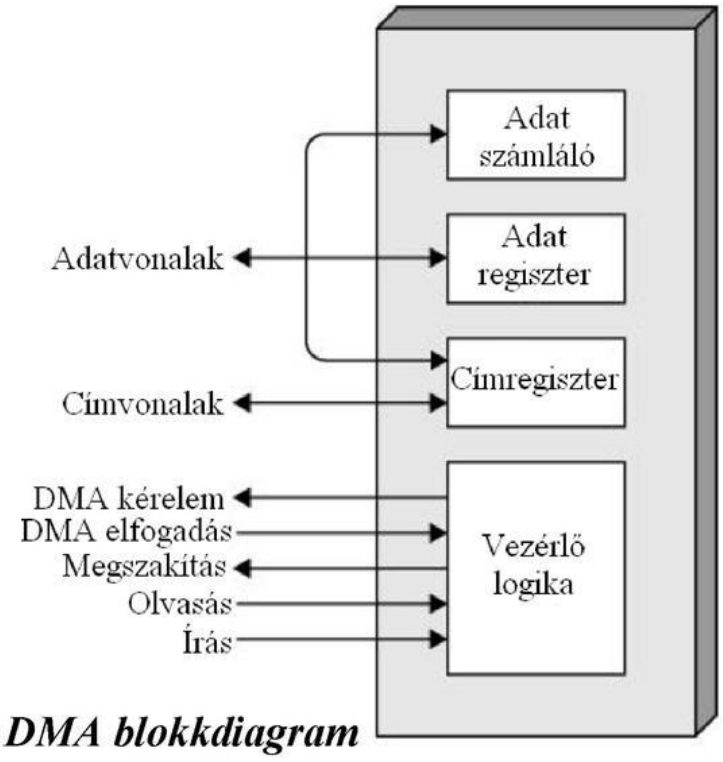
- Közvetlen memóriahozzáférés (DMA)
  - – a DMA egység vezérli az I/O eszköz és főmemória közötti adattranszfert
  - – a processzor a DMA egységnek küld egy adatblokkra vonatkozó adatmozgatási kérelmet, mely csak az egész blokk transzferje után küld megszakítást a processzornak

Az I/O szolgáltatás fejlődése

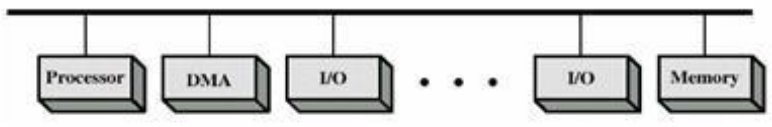
- A processzor közvetlenül vezérli a perifériákat
- Kontroller vagy I/O egység hozzáadása
  - – a processzor megszakítás nélküli programozott I/O–ot használ
  - – a processzornak nem szükséges kezelnie a külső eszközök részleteit
- Kontroller vagy I/O egység megszakítással
  - – a processzor nem tölti várakozással az időt amíg egy I/O művelet befejeződik
- Közvetlen memóriahozzáférés (Direct Memory Access – DMA)
  - – adatblokkok mozgatása a memóriába a processzor bevonása nélkül
  - – a processzor csak a művelet elején és végén van bevonva
- I/O egység, mint különálló processzor
- I/O processzor
  - – I/O egységnek saját helyi memóriája van

Közvetlen memóriahozzáférés (DMA)

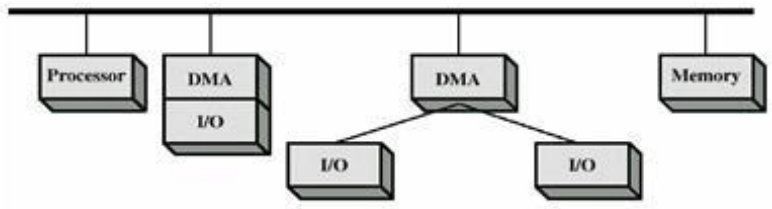
- A rendszerbuszon való adatmozgatáshoz ún. ciklusloPást (Cycle stealing) hajt végre a DMA egység, ilyenkor a processzor működése ideiglenesen felfüggesztődik, a CPU szünetet tart egy buszciklus erejéig
- A ciklusloPás miatt a CPU működése lassabb
- A szükséges buszciklusok csökkentését lehet elérni a DMA és I/O modulok egyesítésével (így köztük nincs rendszerbusz)
- Továbblépés: I/O busz használata az I/O modulok és DMA modul között (vö. DMA blokkdiagram ábra)
- A rendszer irányítását átveszi a CPU-tól, hogy adatot mozgasson a memóriába illetve memóriából a rendszerbuszon keresztül



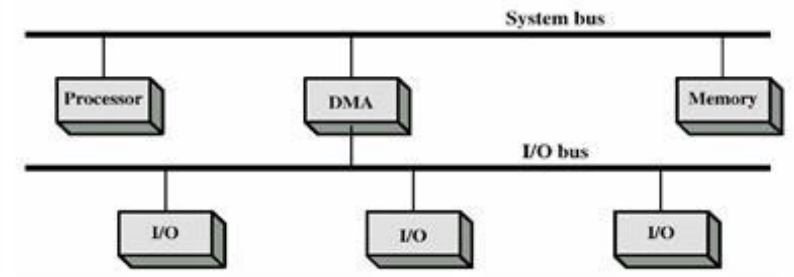
DMA blokkdiagram



a) egy busz, leválasztott DMA



b) egy busz, integrált DMA-I/O



c) I/O busz

DMA szerkezetek

I/O kivitelezés

- Hatékonyság
  - – a legtöbb I/O eszköz a főmemóriához és CPU-hoz képest nagyon lassú
  - – multiprogramozás használatával lehetővé válik, hogy processzusok I/O-ra várakozzanak, miközben más processzusok végrehajtás alatt állnak
  - – ma már léteznek olyan gyors perifériák, amelyek kiszolgálása jelentős teljesítmény-optimalizálást igényel

- – a csereszolgáltatás (swapping) lehetővá teszi, hogy további, készen álló, várakozó processzusok a processzornak munkát adjanak

- Általánosság
  - – az I/O eszközök sokszínűségének ellenére egységes periféria-kezelési megoldás szükséges
  - – az eszközök I/O működésének részleteit alacsony szintű rutinokkal kell eltakarni, hogy a processzusok számára már csak általános műveletek maradjanak: olvasás, írás, megnyitás, bezárás, felfüggesztés, feloldás, stb..
  - – ezt az operációs rendszer szintjén kell megtenni

3. I/O pufferelés

- A pufferelés okai
  - – a processzusoknak meg kell várniuk az I/O befejeződését ahhoz, hogy folytatódjanak
  - – bizonyos lapoknak a főmemóriában kell maradni az I/O alatt
- Blokkos eszközök
  - – az információ adott méretű blokkban van tárolva
  - – az egyes blokkok írhatók, olvashatók, az összes többi blokktól függetlenül
  - – egy blokk egyszerre kerül átvitelre
  - – lemezeknél és szalagoknál használatos
- Karakteres eszközök
  - – információ átvitele, mint bájtok folyama, nincs blokkszerkezet
  - – terminálokhoz, printerekhez, kommunikációs portokhoz, egérhez, és a másodlagos tárakon kivüli eszközökhöz használatos I/O pufferelés

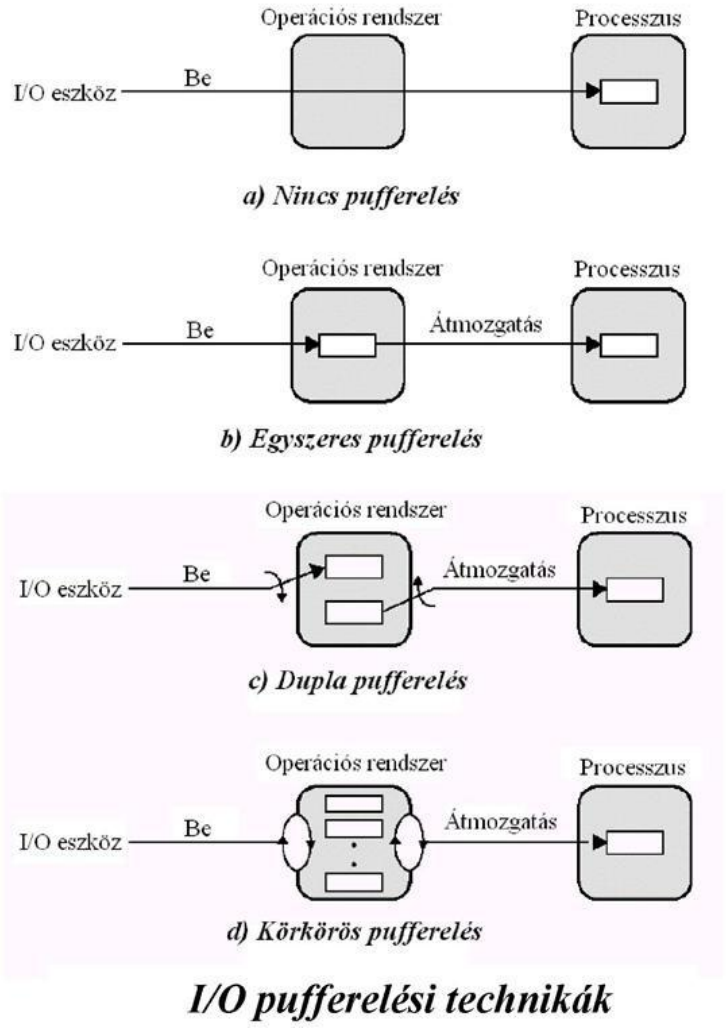
Egyszeres pufferelés

- az operációs rendszer a főmemóriában egy puffert rendel az I/O kéréshez
- blokkos eszközök esetében:
  - – lépései:
    - – a bevitel egy pufferba történik
    - – az adatblokk a pufferból akkor kerül a „felhasználóhoz”, amikor szükség van rá
    - – ezután egy másik/következő blokk pufferba mozgatása következik (read ahead)
  - – a felhasználói processzus feldolgoz egy blokk adatot, míg a következő blokk beolvasásra kerül
  - – a swapping megoldható, mert az input adat az operációs rendszer saját területére (pufferébe) kerül, nem pedig a felhasználói program pufferébe
  - – az operációs rendszer nyilvántartja a rendszer pufferek felhasználói folyamatokhoz történő hozzárendelését
- karakteres eszközök esetében:
  - – egyszerre egy sor használata

- – felhasználói adatbevitel: egyszerre egy sor, kocsi-vissza (carriage return ~ CR) jellel a sor végén
- – a terminálra való kivitel: egyszerre egy sor

Dupla és körkörös pufferelés

- dupla pufferelés:
  - két puffert használunk, az egyiket az operációs rendszer, a másikat a felhasználói processzus
  - a processzus adatot írhat illetve olvashat az egyikbe(ből), míg az operációs rendszer üríti és tölti a másikat
- körkörös pufferelés:
  - n darab puffer hozzárendelése egy processzushoz
  - minden egyes puffer egy körkörös puffer egyik egysége
  - a leggyorsabb,
  - akkor használatos, amikor az I/O műveleteknek a processzussal lépést kell tartani

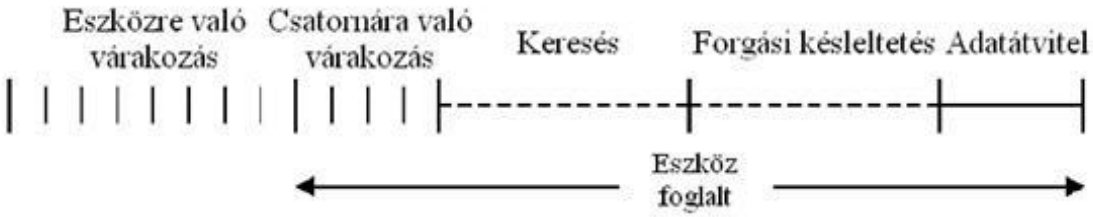


I/O pufferelési technikák

4. Lemezütemezés

Lemez teljesítményét jellemző paraméterek:

- íráshoz és olvasáshoz az író-olvasó fejnek a kívánt cilinderre, sávra és a kívánt szektor elejére kell helyeződni (pozícionálás)
- keresési idő (seek time)
  - – az író/olvasó fej kívánt cilinder- sávpozícióba mozgatásának ideje
- forgási késleltetés (rotational latency) ideje
  - – várakozás, amíg a kérdéses szektor az író/olvasó fej elé fordul
- hozzáférési idő
  - – a keresési idő és forgási késleltetés összege, a fej írásra/olvasásra kész
- az adatátvitel megkezdődik, amikor a szektor a fej alá kerül



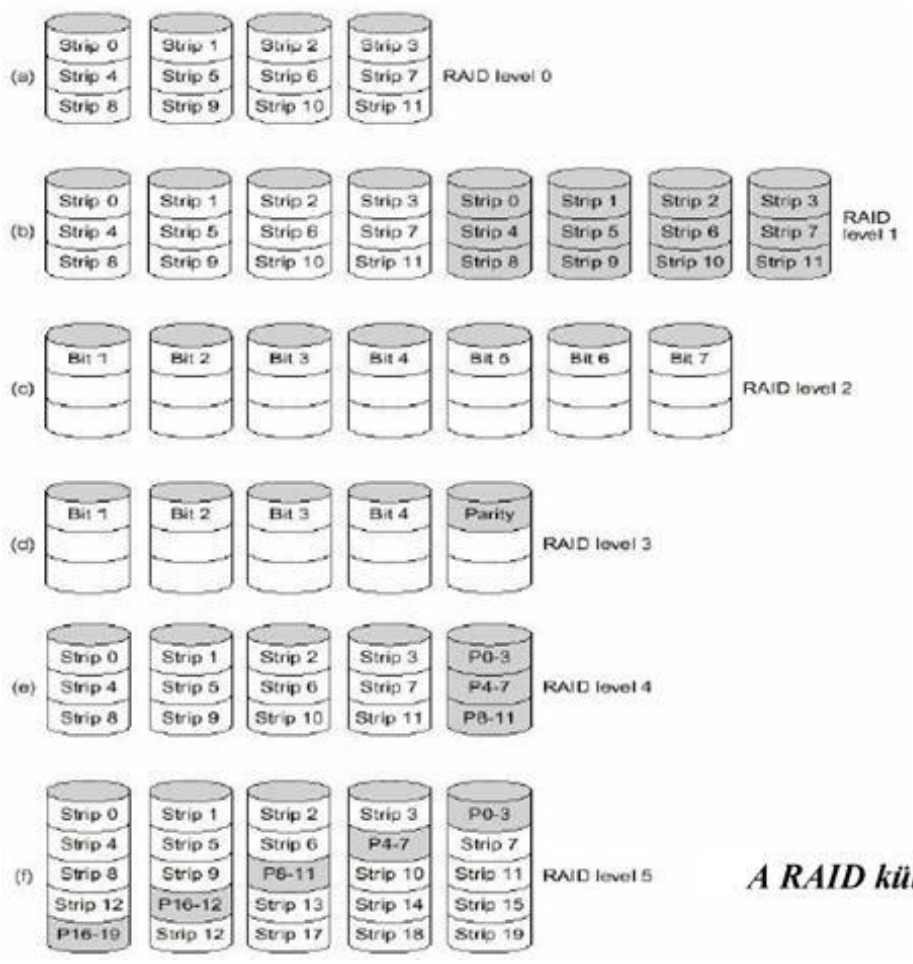
Lemez I/O átvitel

Ütemezési elvek

- a keresési idő függvényében változik a teljesítmény
- egy lemez esetében általában sok I/O kérelem történik, ha a kérelmeket véletlen sorrendben szolgáljuk ki, a legrosszabb teljesítményt érjük el
- First-in, first-out (FIFO) – kiszolgálás a kérelmek beérkezésének sorrendjében – korrekt ütemezés, kevés számú folyamatnál hatékony is lehet – sok processzus esetén a véletlen ütemezés teljesítményéhez közelít
- Prioritásos – a cél nem a lemezhasználat optimalizálása, a processzusok prioritásától függ.... – a rövidebb, kötegelt jobok prioritása nagyobb, mindig a nagyobb prioritású kérést szolgálja ki először
- Last-in, first-out – mindig a legfrissebb kérést szolgálja ki először – éhezés lehetősége: egy job soha nem térhet vissza a sor elejére
- Legkisebb elérési idő először (Shortest Seek Time First - SSTF) – mindig a legrövidebb kiszolgálási időt igénylő (amihez a legkevesebb fejmozgás fejmozgás szükséges) kérést szolgálja ki, sajnos nem optimális
- Pásztázás (SCAN) – cél a hatékonyság növelése éhezés elkerülése mellett – a fej egy irányba mozog csak, kielégítve az összes esedékes kérelmet, amíg eléri az utolsó track-et abba az irányban – ezután megfordul és ellentétes irányba is pásztázik – a lemez középső részeit favorizálja, ill. nagy mennyiségű kérés „leragaszthatja”
- Egyirányú pásztázás (Circular SCAN) – a szkennelést csak egy irányra korlátozza – az utolsó szektor elérése után a fej a diszk ellenkező végére ugrik és a szkennelés újból megkezdődik
- N-step-SCAN – a leragadás megoldása – a diszk kérelmi sort (queue) N nagyságú részekre (subqueue) osztjuk – egyszerre egy ilyen résznek a feldolgozása történik pásztázással – ha N nagy, akkor ez nem más, mint a SCAN, ha N=1, akkor pedig FIFO
- FSCAN – a leragadás megoldása – két sor (queue), amíg az egyikből dolgozik, a kérések a másikba gyűlnek

5. RAID

- mágneslemezekkel kapcsolatos problémák
  - – CPU lényegesen gyorsabb a diszknél
  - – nagy kapacitású lemezek hibájának magas kockázata
  - – diszkek mérete sohasem elég nagy...
- RAID = **R**edundant **A**rray of **I**nexpensive / **I**ndependent **D**isks
- RAID koncepciója: nagy kapacitású és teljesítményű drága diszkek helyett kisebb (olcsóbb) diszkeket használva érjük el célunkat, azaz:
  - – a kapacitás, teljesítmény és megbízhatóság növelését
- a RAID jellemzői:
  - – több diszk funkcionális összekapcsolása úgy, hogy azok egymástól függetlenül és parallel működnek:
    - az operációs rendszer számára egy diszknek látszanak
    - az adatot szétosztjuk a diszkek között, a lemezhibák ellen paritás információ tárolásával védekezzünk
    - a szabvány 5+1 szintet definiál
  - – a különböző megoldások a szükséges tárolóterület overhead-ben, a megoldás teljesítményigényében és a biztonság szintjében térnek el



A RAID különböző szintjei

6. Lemez gyorsítótár

- központi memória puffer a diszk szektorainak
- a diszk néhány szektorának másolatát tartalmazza
- amikor egy I/O kérelem jelentkezik, először ellenőrzésre kerül, vajon a kívánt szektor benne van-e a gyorsítótárban
- blokkcsere algoritmusok:
  - – legrégebben használt (Least Recently Used - LRU)
    - az a blokk lesz cserélve, amelyik a legrégebb idő óta a gyorsítótárban van és nem történt rá hivatkozás
    - a gyorsítótár blokkok halmazából épül fel
    - a legutoljára hivatkozott blokk (illetve egy új blokk) a halom tetejére kerül
    - a halom alján levő blokk lesz eltávolítva új blokk behozatalakor
  - – legritkábban használt (Least Frequently Used – LFU):
    - az a blokk lesz cserélve, amelyikre a legkevesebb hivatkozás történt
    - minden blokkhoz egy számláló tartozik, értéke minden hozzáférés alkalmával eggyel nő, a legkisebb számhoz tartozó blokk lesz cserélve

11. fejezet - Állomány-(fájl)-kezelés

1. Áttekintés: a fájl, mint absztrakt periféria

- A számítógépek az adatokat különböző fizikai háttértárakon tárolhatják, a számítógép kényelmes használhatósága érdekében **az operációs rendszerek egységes logikai szemléletet vezetnek be az adattárolásra és adattárakra: az operációs rendszer elvonatkoztatva a tároló eszköz és a tárolt adat fizikai tulajdonságaitól, egy logikai tároló egységet (adatállomány – fájl – file) használ.**
- A fájlokat az operációs rendszer képezi le a konkrét fizikai tároló berendezésre. A fájlokat tartalmazó fizikai tároló berendezések általában nem törlődnek.
- Felhasználói szemszögből: a fájl összetartozó adatok egy kollekciója, amelyeket egy másodlagos tárban tárolunk. **A fájl a felhasználó számára az adattárolás legkisebb allokációs egysége: felhasználói adatot a háttértáron csak valamilyen fájlban tárolhatunk.**
- Az operációs rendszer támogatást nyújthat a fájl tartalmának kezelésében, a fájl szerkezetének (adatszerkezet) létrehozásában.

Szerkezeti elemek

- Mező – az adat alapvető egysége – egy értéket tartalmaz – hosszával és típusával jellemezhető
- Rekord – összetartozó mezők gyűjteménye – egy egységként kezelhető • például: a vállalat egy dolgozójának rekordja
- Fájl – hasonló rekordok gyűjteménye – önálló egység – egyedi fájlnevek – hozzáférés korlátozható
- Adatbázis – összetartozó adatok gyűjteménye – az elemek között kapcsolatok léteznek

Alapvető műveletek fájlokkal

- Retrieve_All, Retrieve_One, Retrieve_Next, Retrieve_Previous, Insert_One, Delete_One, Update_One, Retrieve_Few

Fájlkezelő rendszer

- a fájlokhoz való hozzáférést biztosítja a felhasználók számára
- a programozónak nem szükséges fájlkezelő szoftvert fejlesztenie, ez az operációs rendszer egyik szolgáltatása

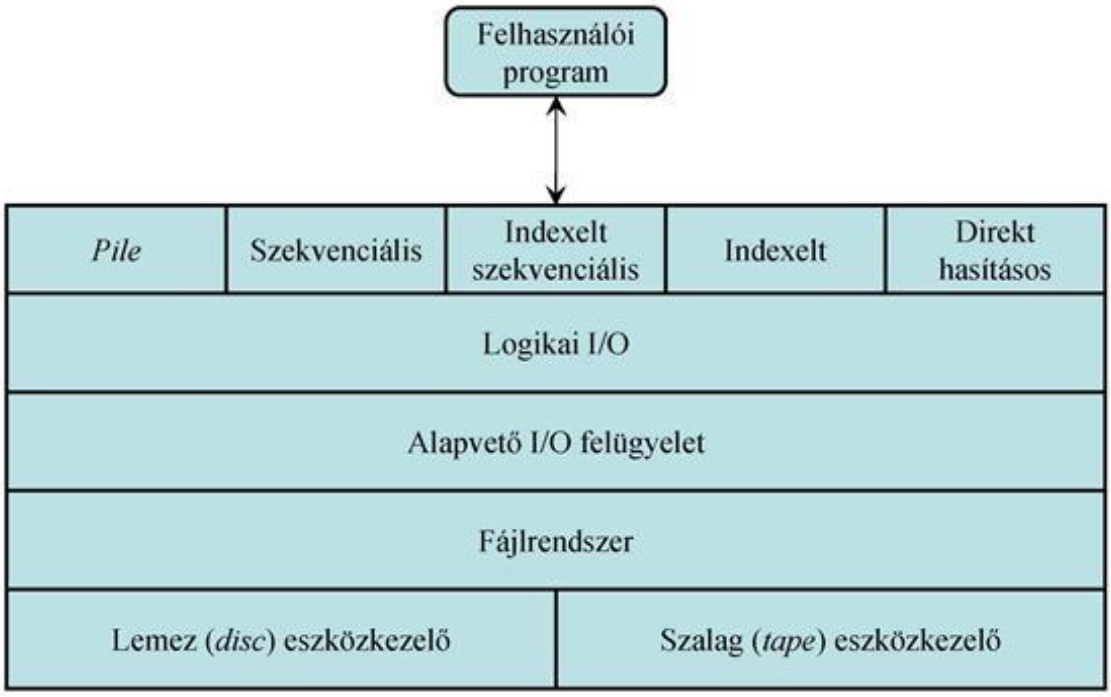
Célok, elvárások

- felhasználók (alkalmazások) adattárolási -kezelési igényeinek kielégítése
- a fájlban levő adat érvényességének garantálása
- a teljesítmény optimalizálása a rendszer és a felhasználó szemszögéből is
- I/O támogatás biztosítása különböző tárolóeszközök számára
- adatvesztés és sérülés lehetőségének minimalizálása ill. kizárása
- egységes programozói I/O interfész biztosítása
- I/O támogatás biztosítása többfelhasználós rendszeren

Minimális szükségletek

- minden felhasználó képes legyen fájlokat létrehozni, törölni, olvasni és megváltoztatni

- minden felhasználónak felügyelt hozzáférése lehet más felhasználó fájljaihoz
- minden felhasználó megszabhatja milyen hozzáféréseket biztosít saját fájljaihoz
- minden felhasználó átszervezheti a fájljait a problémának megfelelően
- minden felhasználónak tudnia kell adatot mozgatni fájlok között
- minden felhasználó képes legyen elmenteni és visszaállítani fájljait (sérülés esetén)
- minden felhasználó képes legyen fájljait szimbolikus nevekkel elérni



Fájlrendszer szoftver architektúra

Fájlrendszer architektúra

- Eszközkezelők
  - – legalacsonyabb szint
  - – perifériákkal való közvetlen kommunikáció (eszközfüggő)
  - – I/O műveletek megkezdéséért felelős az adott eszközön
  - – I/O kérelmeket dolgoz fel
- Fizikai I/O
  - – alacsony (blokk) szintű műveleteket végez
  - – a blokkok elsődleges memóriában való elhelyezésével foglalkozik
- I/O felügyelő
  - – a fájl I/O elkezdéséért és bejezéséért felelős
  - – a hozzáférés ütemezésével foglalkozik (teljesítményfokozás)

  - – az operációs rendszer része
- Logikai I/O
  - – lehetővé teszi az alkalmazások és a felhasználó számára a rekordokhoz való hozzáférést
  - – általános célú rekord I/O műveleteket szogáltat
  - – a fájlokat jellemző alapvető adatokat tartja karban

Fájlkezelési funkciók:

- egy kiválasztott fájl azonosítása és helyének meghatározása
- könyvtár használata az összes fájlhoz helyüknek és attribútumaiknak leírásához
- osztott rendszeren a felhasználói hozzáférés vezérlése
- fájlhozzáférés blokkolása
- szabad tárhely kezelése

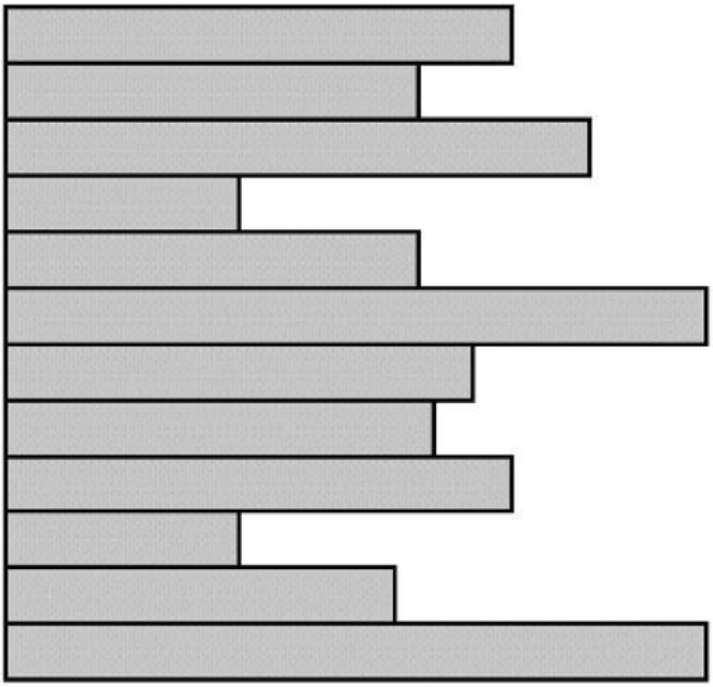
2. Fájlszervezés és hozzáférés

Fájlhozzáférés követelmények:

- Gyors hozzáférés – egy egyszerű rekordeléréshez szükséges – kötegelt módban (batch mode) nem szükséges
- Egyszerű frissítés – egy CD-ROM fájlt nem lehet frissíteni, így ez nem teljesül mindig
- Gazdaságos tárhelyhasználat – felesleg adatok minimalizálása – redundanciával gyorsabb hozzáférés érhető el
- Egyszerű fenntartás
- Megbízhatóság

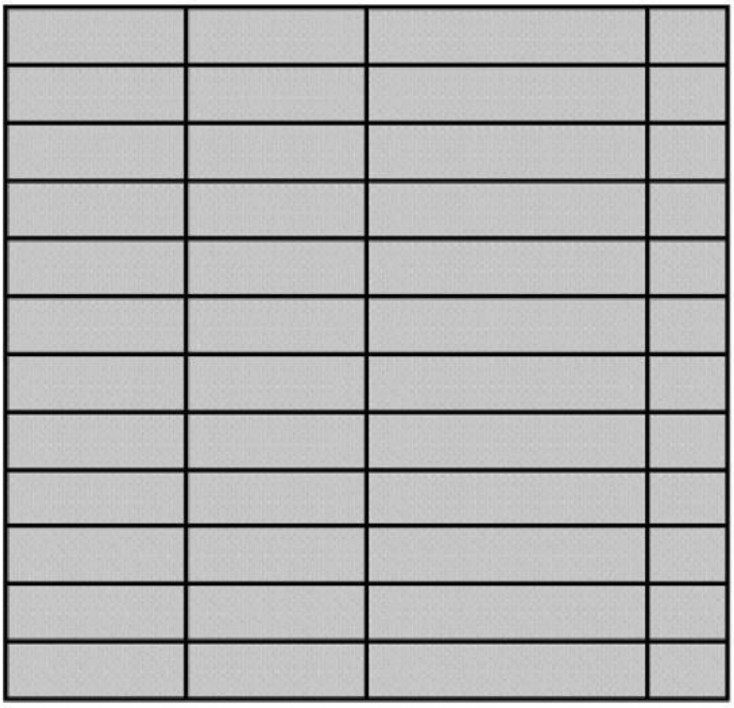
Fájlszervezés

- pile
  - – adatgyűjtés érkezési sorrendben (struktúrálatlanul)
  - – a cél: nagy mennyiségű adatot felhalmozni és elmenteni
  - – rekordoknak különböző mezőik lehetnek
  - – nincs szerkezete
  - – a rekordhoz való hozzáférés fárasztó kereséssel jár....



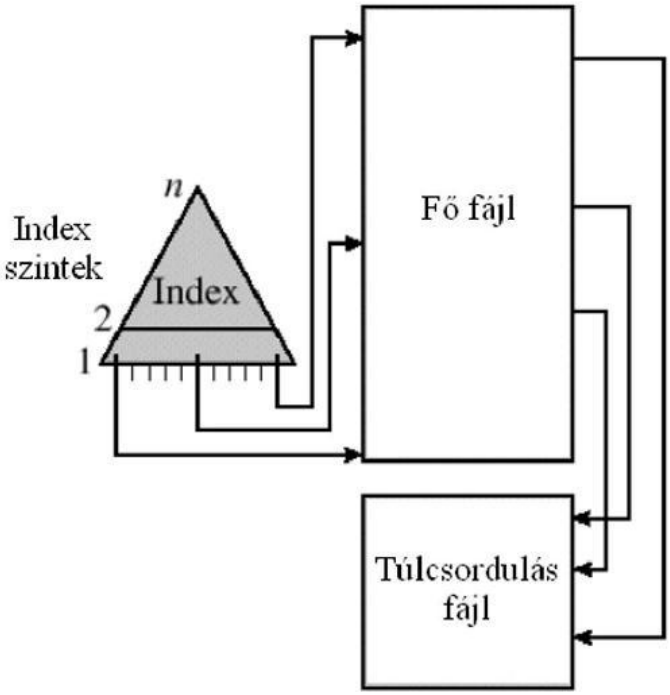
Változó hosszúságú rekordok
Változó mezőkészlet
Kronologikus sorrend

- szekvenciális
  - – a rekordokat egyetlen sorrendben, a fájl első rekordjától az utolsó felé haladva éri el, mely sorrend megegyezik a rekordok létrehozásának sorrendjével
  - – a rekordok mérete és formátuma azonos,
  - – kulcsmező használata
    - egyértelműen meghatározza a rekordot
    - a rekordok fizikailag egymás után következnek, vagy rekordmutatók használatával egy láncolt lista határozza meg a rekordok sorrendjét.
  - – akkor alkalmazzuk, ha a fájlt használó program a rekordok összességének feldolgozását igényli

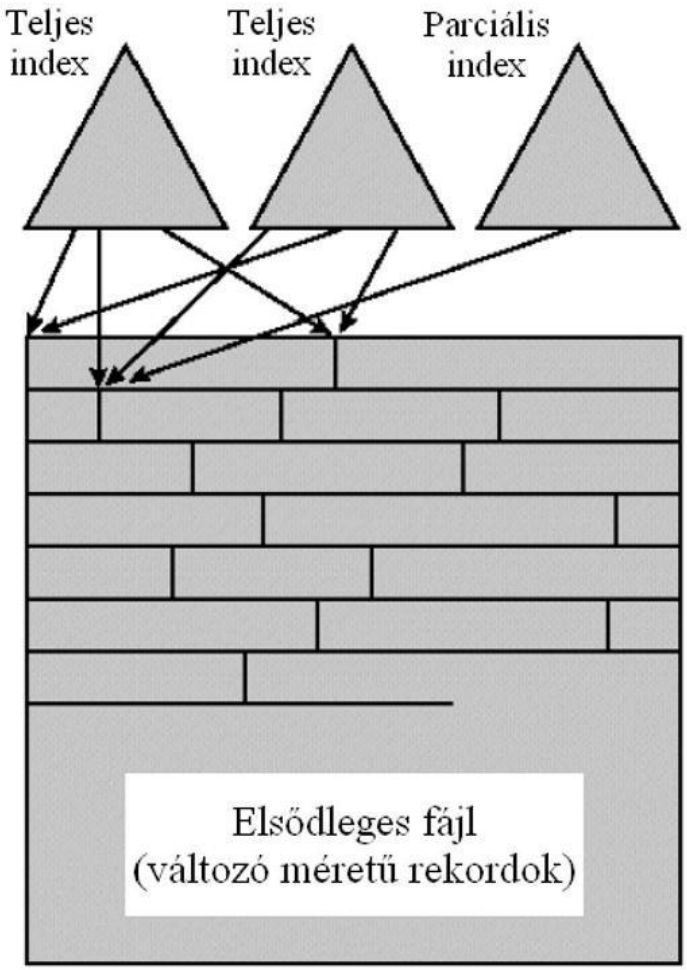


Fix hosszúságú rekordok
Fix mezőkészlet rögzített sorrendben
Szekvenciális sorrend (kulcsmező szerinti)

- indexelt szekvenciális
  - – direkt hozzáférési eljárás, amely a kulcs szerinti kereséshez indexeket használ
  - – index: kulcsértékeket és rekordmutatókat tartalmazó táblázat. Az index lehet egyszintű vagy többszintű. Az indexek külön fájlba, ún. indexfájlba kerülnek.
  - – az egyszintű indexben illetve a többszintű index legalsó szintjén a kulcsértékek mellett a rekordmutatókat találjuk, míg a többszintű index felsőbb szintjein a kulcsértékek mellett az alattuk lévő szint táblázataira találunk utalásokat.
  - – új rekordok hozzáadása egy overflow fájlhoz, amit firssítéskor hozzáfűzünk a fő fájlhoz
  - – a teljesítmény növeléséhez többszintű indexeket lehet használni ugyanahhoz a kulcsmezőhöz
  - – olyan adatbázisokhoz is alkalmazzuk, ahol gyakoriak az összetett feltételű keresések



- indexelt
  - – a különböző kulcsmezőkhöz többszintű indexet használunk
  - – új rekord hozzáadása esetén az összes indexfájlt frissíteni kell
  - – olyan alkalmazásoknál használatos, ahol az információ időzítése kritikus
    - légiközlekedés foglalórendszere
- direkt hasításos (hash) fájlok
  - – direkt hozzáférési eljárás, melynek során egy kulcs értékéből az ún. hasítófüggvény határozza meg a rekordmutatót. Ha az így kijelölt helyen nincs a keresett rekord, az eljárás szekvenciális kereséssel folytatódik.
  - – kulcsmező szükséges minden rekordhoz
  - – alkalmazás: ha a tárolandó adatmennyiséghez képest legalább 3-4- szeres terület áll rendelkezésre
  - – probléma: kulcsütközés

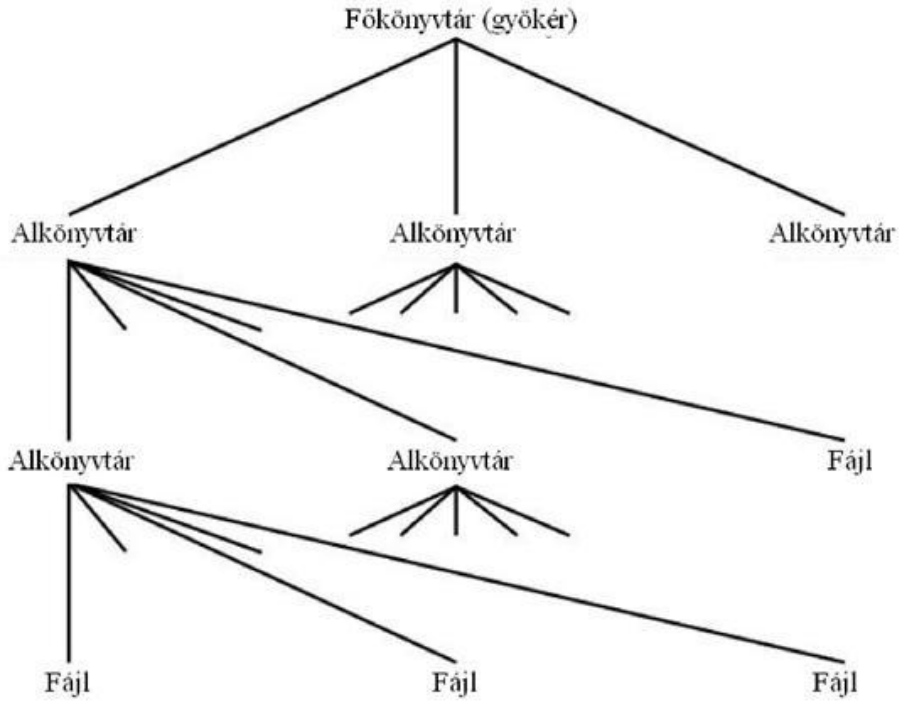


3. Könyvtárak (Directory - fájljegyzék megoldások)

- Tartalom
  - – fájlokkal kapcsolatos információkat tartalmaz (név, kiterjesztés, hely, tulajdonos, ...), ezek az állomány *attribútumai*
  - – a fájljegyzék maga is egy fájl, melynek tulajdonosa az operációs rendszer is lehet
  - – a fájlnevek és fájlok közötti kapcsolatot biztosítja
- Könyvtárszerkezet
  - Egyszintű könyvtár / fájljegyzék
    - – bejegyzések listája, minden fájlhoz egy
    - – szekvenciális állomány, ahol a fájlnevek szolgálnak kulcsként
    - – nem nyújt segítséget a fájlok rendezéséhez (csoportosítási problémák)
    - – nem lehet két különböző fájlnak ugyanaz neve! (elnevezési problémák)
  - Kétszintű könyvtár
    - – egy-egy jegyzék minden felhasználónak és egy főkönyvtár (user/master directory)

- – a főkönyvtár minden felhasználóhoz tartalmaz bejegyzést (hozzáférési jogok)
- – minden felhasználói jegyzék egy egyszerű listája a felhasználó fájljainak
- – névadási probléma megoldva, de csoportosítás továbbra sem lehetséges

- Fa-szerkezetű könyvtár
  - – főkönyvtár, alatta (benne) felhasználói könyvtárak
  - – egy fájljegyzék bizonyos elemei lehetnek újabb fájljegyzékek (alfájljegyzék), így fájljegyzékeknek egy hierarchikus rendszere jön létre
  - – a fájlok a főkönyvtárból kiindulva különböző ágakon haladva találhatók meg
    - ez lesz a fájl elérési útja (path)
  - – több fájlnak is lehet azonos neve, amíg az elérési útjuk eltérő
  - – munkakönyvtár (current directory) váltása cd()
  - – a fájlok a munkakönyvtárhoz képest is hivatkozhatók (relative path)



- Általános gráf - szerkezetű könyvtár
  - – linkek / aliasnevek használata
  - – függő link
  - – pásztázás
  - – ciklusfigyelés

4. Fájlmegosztás

Többfelhasználós rendszerben a fájlok megoszthatók a felhasználók között.

Hozzáférési jogok típusai:

- – nincs
  - a felhasználó még a fájl létezéséről sem tud
  - a felhasználó számára nem engedélyezett azon könyvtár olvasása, mely tartalmazza a fájlt
- – ismeret
  - a felhasználó csak a fájl létezéséről tud, illetve tudja, hogy ki a fájl tulajdonosa
- – végrehajtás
  - a felhasználó betöltheti és futtathatja a programot, de nem másolhatja
- – olvasás
  - a fájl minden célból olvasható, így futtatható és másolható is
- – hozzáfűzés
  - a fájlhoz adat hozzáfűzhető, de a fájl eredeti tartalma nem törölhető és módosítható
- – frissítés
  - a fájl módosítható, törölhető, létrehozható, újraírható, stb.
- – védelem megváltoztatása
  - a felhasználó a hozzáférési jogokat megváltoztathatja
- – törlés
  - a felhasználó törölheti a fájlt
- – tulajdonos
  - az összes elöbbi joggal rendelkezik
  - jogokat határozhat meg más felhasználó számára a következő csoportosítással
    - – egy bizonyos felhasználó
    - – felhasználók egy csoportja (user group)
    - – mindenki (publikus fájlok esetén)

Szimultán hozzáférés

- – a felhasználó lezárhatja a fájlt frissítés megakadályozása céljából
- – a felhasználó lezárhat egyedi rekordokat frissítés közben
- – a megosztott hozzáférés problémái: kölcsönös kizárás és holtpont

5. Másodlagostár-kezelés

- fájl allokáció: másodlagos tárhely fájloknak való kiosztása
- szabad tárhely kezelés: nyomonköveti a kiosztásra alkamas tárhelyet

Előfoglalás

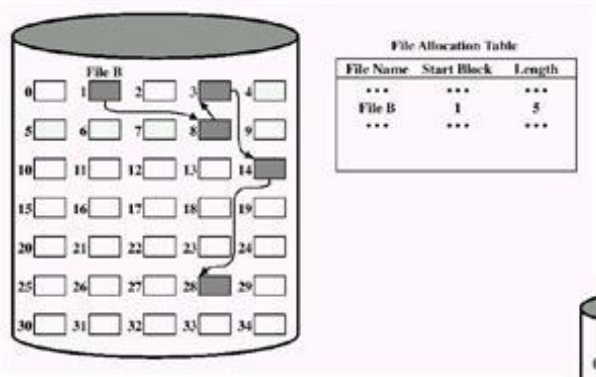
- a fájl létrehozásakor szükség van a lehető legnagyobb várható fájlméretre
- nehéz elég pontosan megjósolni a potenciális maximális fájlméretet
  - – fájlméret túlbecslése célravezető

Háttértár kiosztási módszerek

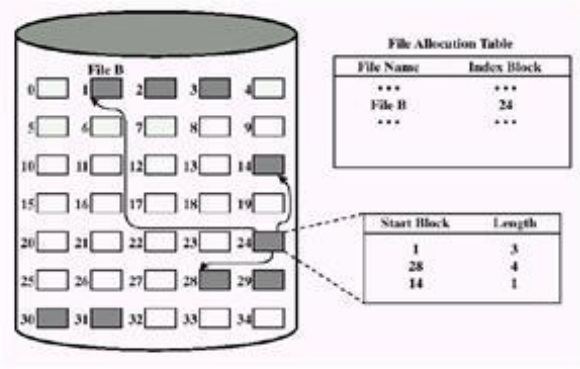
- folytonos kiosztás
  - – minden fájl egymást követő blokkok sorozatát foglalja el
  - – a helyfoglalás katalógusbejegyzése: kezdő blokk és elfoglalt blokkok száma
  - – algoritmusok szükségesek a megfelelő méretű szabad helyek megkeresésére
  - – algoritmusok közös hibája: külső töredezettség veszélye
  - – állományok általában nem bővíthetők
- láncolt kiosztás
  - – minden állomány blokkok lácolt listája, ezek a lemezen tetszőleges helyen helyezkednek el
  - – minden blokk tartalmaz egy mutatót a lánc következő blokkjára
  - – a fájl allokációs tábla bejegyzése az első és az utolsó blokkra mutat
  - – nincs külső töredezettség, és a fájlok egyszerűen bővíthetők
  - – szekvenciális fájlok esetén biztosít nagy hatékonyságot
- indexelt kiosztás – mutatókat indexblokkokba tömöríti, az indexblokk i-edik eleme az állomány i-edik blokkjára mutat, a fájlallokációs tábla az indexblokk címét tárolja

Szabad hely nyilvántartása

- – Bit tábla használata: diszk minden blokkjához egy bitet rendelünk, a bit értéke mutatja az adott blokk foglaltságát
- – Láncolás: láncolt lista a szabad blokkokról
- – Indexelés: indextábla a szabad blokkokról
- – Szabad blokkok listája : külön területen, a diszken tárolva



Láncolt kiosztás

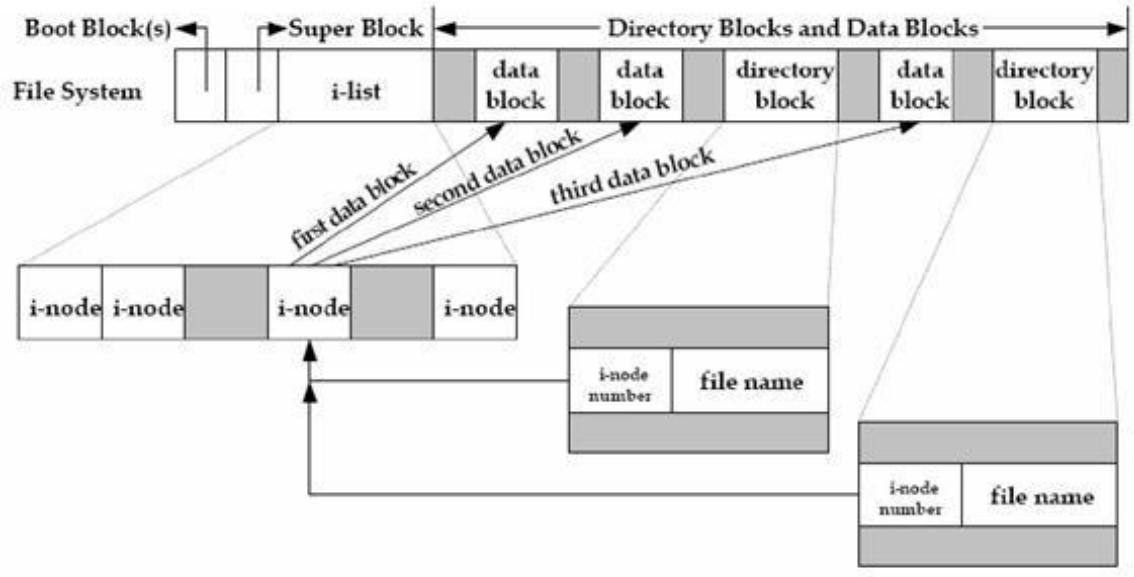


Indexelt kiosztás

6. A Unix és a Windows 2000 fájlkezelése

Unix fájlkezelés: "a Unixban minden fájl!"

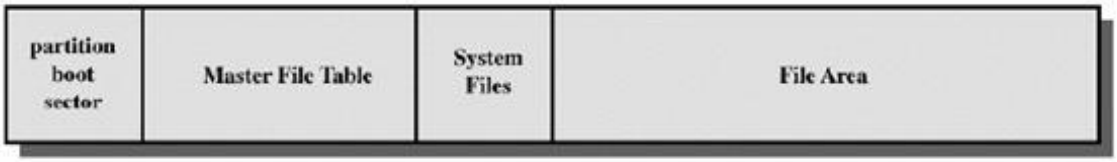
- fájltípusok
  - – hagyományos: tartalommal a felhasználók illetve programok töltik fel
  - – könyvtár: fájlnevekből álló listát tartalmaz illetve inode-okra (I-bög) történő hivatkozásokat (mutatókat)
  - – speciális: perifériák eléréséhez használatos
  - – névvel rendelkezőt pipe-ok (csőhálózat)
- az inode tartalmazza:
  - − a fájl tulajdonosátnak azonosítóját, a fájl típusát, hozzáférési jogokat, utolsó hozzáférés illetve módosítás idejét, fájlra mutató linkek számát, fájlméretet, a fájl által elfoglalt lemezblokkok táblázatát (többszintű indexeit)



UNIX fájlrendszer

Windows 2000 fájlkezelés

- NTFS fájlrendszer tulajdonságai
  - – rendszerösszeomlás esetén visszaállítható
  - – nagyfokú biztonság
  - – nagy lemezek, nagy fájlok támogatása
  - – többszörös adatfolyam definiálása egy fájlhoz
  - – általános indexelés: minden fájlhoz több jellemzőt is rendel



Az NTFS kötet (volume)

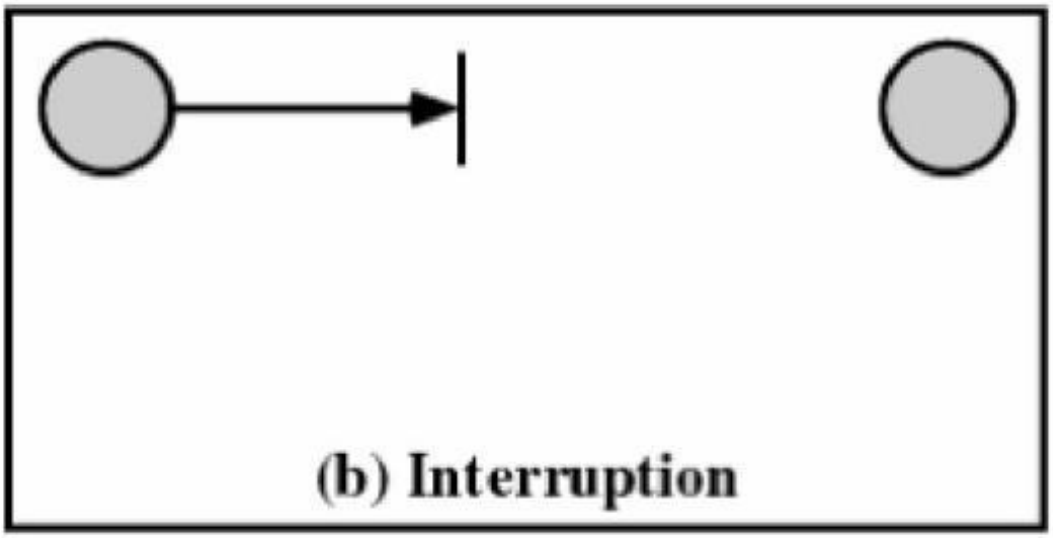
12. fejezet - Operációs rendszerek védelmi kérdései

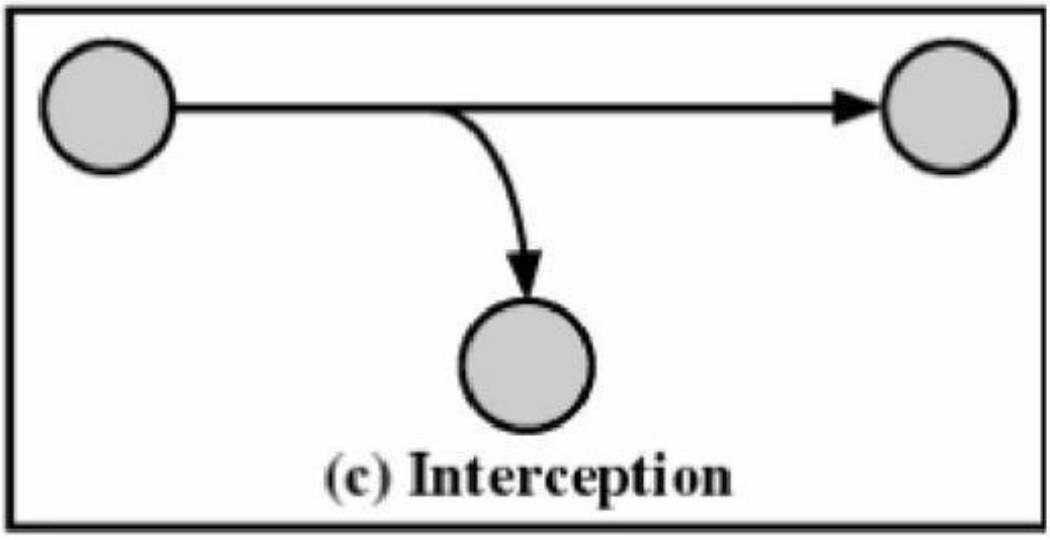
1. Biztonsági elvárások

- „Bizalmasság” (confidentiality)
  - – a számítógépen tárolt információt csak az arra jogosultak tudják olvasni
- Integritás (integrity) épség
  - – eszközök / adatok módosítását csak az arra jogosultak végezhetik (írás, olvasás, törlés...)
- Elérhetőség (availibility)
  - – az eszközök legyenek elérhetőek az arra jogosultak számára
- Azonosítás (authentication)
  - – a rendszer képes legyen megerősíteni / ellenőrizni a felhasználó azonosságát

2. Biztonsági veszélyforrások

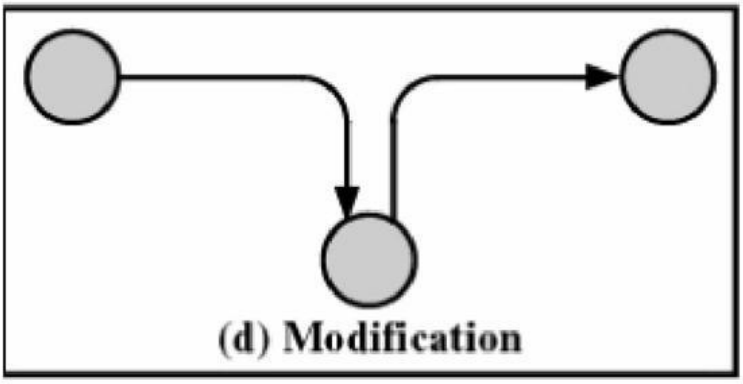
- Megszakítás (Interruption)
  - – egy rendszereszköz elérhetetlenné vagy használhatatlanná válik
  - – támadás az elérhetőség ellen – hardverrombolás – kommunikációs vonal elvágása
  - – a fájlkezelő rendszer megbénítása
- Lehallgatás (Interception)
  - – illetéktelen csoportok hozzáférést szereznek egy eszközhöz
  - – támadás a bizalmasság ellen
  - – „kábelcsapolás” a hálozatban adatlopási céllal
  - – tiltott fájl és programmásolás

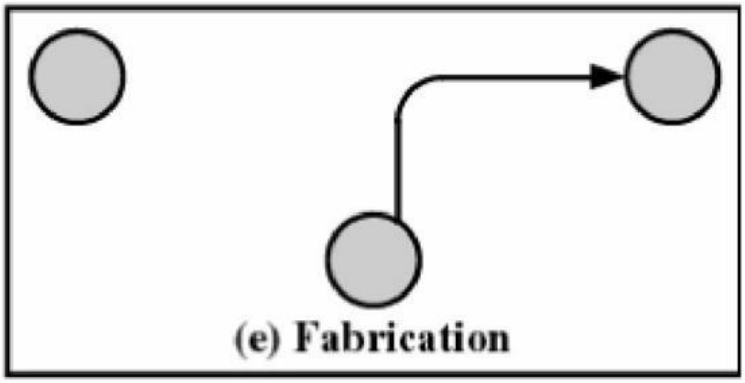




- Módosítás (Modification)
  - – illetéktelenek hozzáférést szereznek és befolyásolják is a rendszert
  - – támadás az integritás ellen
  - – pl.: érték megváltoztatása egy adatállományban
  - – egy programkód megváltoztatása
  - – hálózaton átvitt üzenet tartalmának módosítása

- Gyártás (Fabrication)
  - – illetéktelenek hamis objektumokat helyeznek a rendszerbe
  - – támadás az azonosítás ellen
  - – hamis üzenetek küldése hálózaton
  - – rekordok hozzáadása fájlhoz





3. Rendszereszközök fenyegetései

- Hardver
  - – véletlen és szándékos károk elszenvedésének veszélye
- Szoftver
  - – törlés, változtatás veszélye
  - – biztonsági mentések fentarthatják a nagyfokú elérhetőséget
- Adat

- – fájlok megváltoztatásának veszéle
- – diszkrécióval, elérhetőséggel és integritással kapcsolatos biztonsági kérdések
- – adatbázisok statisztikai analízise egyéni információk meghatározását teszi lehetővé: titkosság sérülésének veszélye

- Kommunikációs vonalak és hálózatok – passzív támadások
  - – üzenet tartalmának kikerülése (telefonbeszélgetés, elektronikus levelezés, fájlátvitel)
  - – forgalom elemzés
    - küldő és fogadó azonosítása, üzenetek hossza és küldésének gyakorisága: a kommunikáció természetére vonatkozó információk...
- Kommunikációs vonalak és hálózatok – aktív támadások
  - – színlelés (Masquerade): egy személy más személynek adja ki magát
  - – ismétlés: egy adategység (üzenet) elfogása és későbbi újraküldése
  - – üzenetek módosítása: egy szabályszerű üzenet részének megváltoztatása, késleltetése vagy újrarendelése
  - – szolgáltatás megtagadása: a kommunikációs eszközök kezelésének és használatának akadályozása
    - a hálózat megbénítása vagy túlterhelése üzenetekkel

4. Védelem

- – Nincs védelem
  - megfelelő, ha érzékeny eljárások különböző időkben futnak
- – Izoláció
  - minden processzus más processzustól elkülönítve működik (nincs megosztás és kommunikáció)
- – Megosztás teljes hozzáféréssel vagy semmilyen hozzáféréssel
  - egy objektum tulajdonosa meghatározza, hogy az publikus vagy privát
- – Megosztás a hozzáférés korlátozásával
  - az operációs rendszer ellenőrzi minden hozzáférésnél a jogosultságokat
  - operációs rendszer, mint őr
- – Megosztás dinamikus lehetőségek szerint
  - objektumok megosztási jogainak dinamikus létrehozása
- – Objektum limitált használata
  - nem csak a hozzáférés korlátozása, de a végrehajtható műveletek korlátozása is
  - példa: egy felhasználó statisztikai kivonatokat nyerhet egy adatbázisból, de nem tudja az egyes adatértékeket meghatározni; vagy dokumentumhoz való hozzáférés engedélyezése, de nyomtatásának tiltása

5. Hozzáférés vezérlése (Access control)

Felhasználó-orientált:

– bejelentkezés

- szükséges hozzá egy felhasználóazonosító (User ID) és egy jelszó
- a rendszer csak akkor engedi be a felhasználót, ha az ID ismert és a hozzátartozó jelszó helyes
- a felhasználók szándékosan vagy véletlenül felfedhetik a jelszavukat
- a heckerek a jelszavak megtalálásának szakértői.....
- az ID/jelszó fájl könnyen megszerezhető

Adat-orientált:

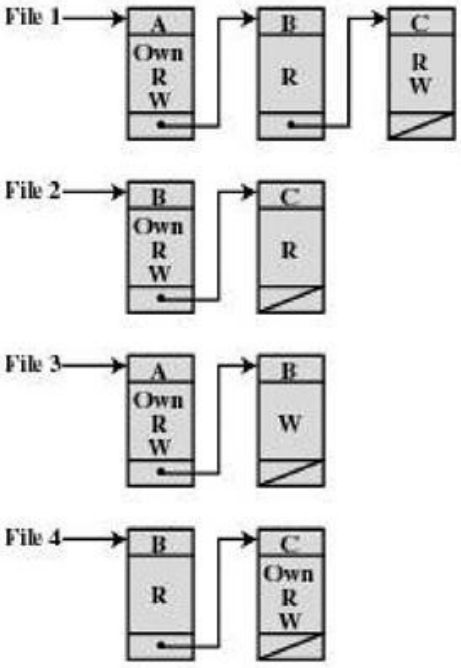
- – minden felhasználóhoz egy felhasználói profil tartozik, mely meghatározza a megengedett műveleteket és fájlhozzáféréseket
- – az operációs rendszer kikényszeríti ezen szabályok betartását
- – az adatbázis kezelő rendszer vezérli a rekordok hozzáférését (hozzáférési mátrix – access matrix)

6. Adatorientált hozzáférésvezérlés

- Hozzáférési mátrix:
  - Alany – felhasználók illetve alkalmazások
  - Objektum – fájlok, programok, memóriaszegmensek
  - Hozáférési jogok – olvasás, írás, futtatás
- Hozzáférést vezérlő lista:
  - – oszlopokból álló mátrix
  - – minden objektumhoz megadja a felhasználókat illetve, hogy az egyes felhasználóknak milyen hozzáférési jogaik vannak

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

Hozzáférési mátrix (példa)



Hozzáférést vezérlő lista (példa)

7. Betolakodók (Hacker)

- a behatoló célja, hogy hozzáférést nyerjen a rendszerhez illetve, hogy kiszélesítse jogait egy rendszeren
- egy behatoló által megszerezni kívánt védett információ: a jelszó
- jelszómegszerzési taktikák:
  - – a szabvány, számítógéppel szállított jelszavak kipróbálása
  - – az összes rövid jelszó kipróbálása (igen időigényes...)
  - – a szótár szavainak kipróbálása illetve a legvalószínűbb jelszavak tesztelése
  - – információk gyűjtése a felhasználóról és ezeket használni jelszóként
  - – telefonszámok, szobaszámok, személyi szám, születési idő, rendszámtábla, stb...
  - – Trójai faló használata a rendszer korlátozásainak megkerülésére
  - – a távoli felhasználó és gazdarendszer közötti vonal megcsapolása

8. Jelszóvédelem

- Jelszó: igazolhatja a felhasználó személyazonosságát (autentikáció), s ezzel együtt meghatározhatja, hogy a felhasználó fel van-e hatalamazva a rendszer eléréséhez, illetve hogy a felhasználó milyen jogosultságokkal rendelkezik
- Számítógép által generált jelszavak
  - – a felhasználók által nehezen megjegyezhetők (ezért aztán leírják....)
  - – a felhasználók nem nagyon szeretik....
- Reaktív jelszóellenőrzési stratégia
  - – a rendszer időközönként egy saját jelszófeltörőt futtat, hogy megtalálja a könnyen kitalálható jelszavakat
  - – a rendszer törli azokat a jelszavakat melyeket kitalált és értesíti a felhasználókat
  - – ennek természetesen erőforrás ára van
  - – a hacker ezt a saját gépén tudja használni jelszófájl másolatán .....
- Proaktív jelszóellenőrző
  - – a rendszer a jelszóváasztáskor ellenőrzi, hogy a jelszó megengedhető-e
  - – a rendszer útmutatásával a felhasználó egy nehezen kitalálható, de könnyen megjegyezhető jelszót találhat magának

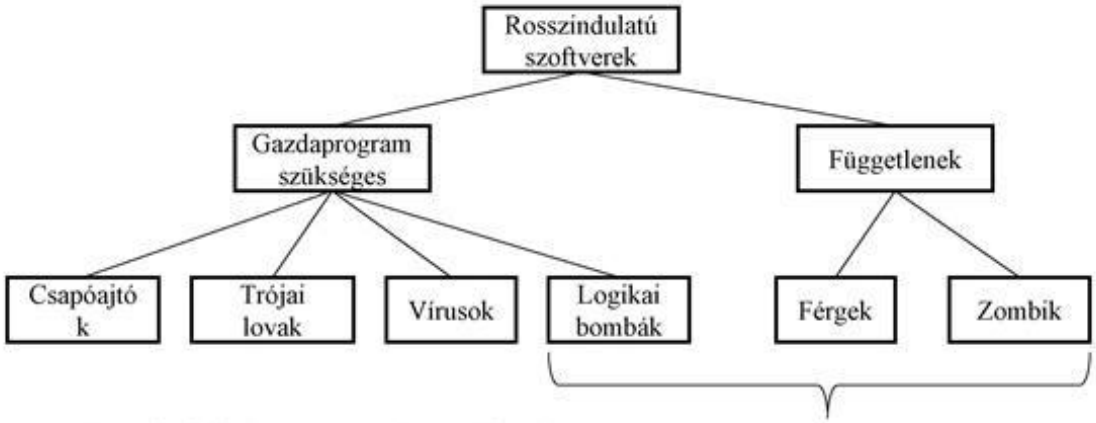
9. Behatolás észlelése

- feltételezzük, hogy egy behatoló viselkedése különbözik egy legitim felhasználóétól
- statisztikus anomáliák észlelése
  - – legitim felhasználók viselkedésével kapcsolatos adatok gyűjtése
  - – statisztikai tesztekkel észlelhetjük, ha a viselkedés nem „normális”
- szabály-alapú észlelés
  - – szabályok felállítása a korábbi használat jellemzőitől való eltérés észlelésére
  - – szakértő rendszer keresi a gyanús viselkedést
- felülvizsgálati jelentések
  - – natív felülvizsgálati feljegyzések
    - minden operációs rendszer tartalmaz könyvelő, jelentéskészítő szoftvereket, melyek információt gyűjtenek és jegyeznek a felhasználói aktivitásról
  - – detektáló-specifikus felülvizsgálati feljegyzések
    - behatolás detektáló rendszer számára szükséges információ gyűjtése

10. Rosszindulatú programok

- Gazdaprogramra van szükségük
  - – programtöredékek, mely felhasználó programok, rendszerprogramok nélkül nem létezhetnek
- Független

- – csak önmagukat tartalmazó programok, melyeket az operációs rendszer ütemez és futtat



Rosszindulatú programok osztályozása

- Csapóajtó (Trapdoor):
  - – egy program belépési pontja, mely megengedi a csapóajtó ismerőjének, hogy hozzáférést nyerjen
  - – programozók használják debughoz és teszteléshez
    - elkerülik az általában szükséges beállításokat és hitelesítéseket
      - programok aktíválásának módszere, ha valami probléma van a hitelesítéssel
- Logikai bomba:
  - – kód beágyazása egy szabályszerű programba, ami „robban”, ha bizonyos feltételek teljesülnek:
    - – bizonyos fájlok jelenléte vagy hiánya
    - – a hét egy meghatározott napja, stb.
- Trójai falovak:
  - – rejtett kódot tartalmazó hasznos programok, melyek hívásuk esetén káros függvényeket hajhatnak végre
  - – olyankor használják, ha egy felhasználó számára nem megengedett bizonyos feladatok direkt végrehajtása (fájl engedélyekkel való játszadozás....)
- Vírusok:
  - – programok, melyek „megfertőznek” más programokat úgy, hogy módosítják
    - a módosításba beletartozik a vírusprogram másolása is
    - egy fertőzött program továbbfertőzhet más programokat
- Férgek:
  - – terjedéshez a hálózati kapcsolatokat használják
  - – elektronikus levelezőrendszer
    - egy féreg saját maga egy másoltát küldi el levélben egy másik rendszernek
  - – távoli végrehajtási képesség
    - a féreg magának egy másolatát el tudja indítani egy távoli gépen

- – távoli bejelentkezési képesség • féreg azon képessége, hogy bejelentkezik egy távoli rendszerbe és ott különböző utasításokat használva lemásolja magát rendszerről rendszerre

- Zombik:
  - – programok, melyek átveszik az irányítást egy internetre kötött gép felett

11. Vírusok típusai

- Parazita
  - – hozzácsatolja magát végrehajtható fájlokhoz és sokszorozódik
  - – amikor egy fertőzött program végrehajtódik, más fertőzhető programot keres
- Memóriarezidens vírusok
  - – egy memóriarezindens rendszerprogram részeként megtelepedik a főmemóriában
  - – ha egyszer a memóriába kerül, minden végrehajtásra kerülő programot megfertőz
- Boot szektor vírusok
  - – a boot rekordot fertőzi meg
  - – akkor terjed, ha a rendszer a fertőzött diszkről indul
- Stealth
  - – úgy készítik, hogy az antivírus programok elől hatékonyan el tudjon rejtőzni
- Polimorf vírusok
  - – minden fertőzéskor mutálódik, így hagyományos észlelésük lehetetlen
  - – a mutációs motor egy kulcsot generál, mellyel titkosítja a vírus többi részét

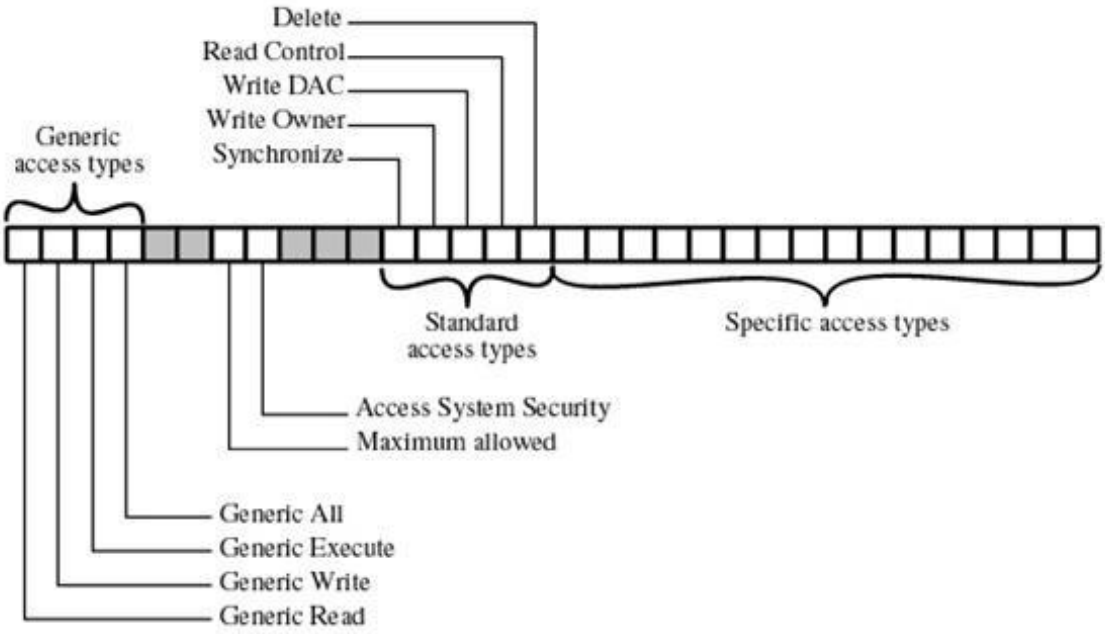
12. Windows 2000 biztonság

- Hozzáférés vezérlési séma
  - – név/jelszó
  - – minden objektumhoz a felhasználó jogait megadó hozzáférés-jelölés van rendelve
- Hozzáférés jelölése:
  - – Biztonsági azonosító
    - minden felhasználót egyértelműen azonosít a hálózat minden gépén
  - – Csoport azonosítók
    - azon csoportok listája, melyhez a felhasználó is hozzátartozik
  - – Privilégiumok
    - azon biztonsági szempontpól kockázatos rendszerszolgáltatások listája, melyet ez a felhasználó meghívhat
  - – Alapértelmezett tulajdonos
    - ha a processzus egy objektumot létrehoz ez írja le, hogy ki lesz a tulajdonos

- – Alapértelmezett hozzáférést vezérlő lista
  - kezdeti lista azon objektumhoz, melyet a felhasználó hozott létre

Biztonsági leíró (Security Descriptor):

- Flags
  - – egy biztonsági leíró típusát és tartalmát definiálja
- Tulajdonos
  - – az objektum tulajdonosa bármilyen általános műveletet végrehajthat a biztonsági leírón
- Rendszerhozzáférést vezérlő lista (System Access Control List - SACL)
  - – azt is meghatározza, hogy egy adott objektumhoz milyen műveletek esetén kell ellenőrző üzeneteket generálni
- Tetszőleges hozzáférést vezérlő lista (DACL)
  - – meghatározza melyik felhasználó és csoport férhet hozzá egy bizonyos objektumhoz és milyen műveletet hajthatnak rajta (vele) végre



Hozzáférési maszk

13. fejezet - Ajánlott irodalom

- Silberschatz, Abraham, Operating system concepts, [Abraham Silberschatz, Peter B. Galvin],4th ed. Reading, Mass. : Addison-Wesley, c1994, xvi, 780 p., ISBN 0 201 59292 4
- Andrew S. Tanenbaum, Albert S. Woodhull, Operációs rendszerek; [ford. Dévényi Károly, Gombás Éva stb.] Budapest : Panem ; Upper Saddle River, NJ : Prentice-Hall, 1999, 980 p., ISBN 963 545 189 X
- Kóczy A., Kondorosi K., (szerk), Operációs rendszerek mérnöki megközelítésben, Panem Kiadó, Budapest, 2000., ISBN 963 545250 0
- Nutt, Gary J., Operating systems : a modern perspective / Gary J. Nutt. - 1. print. Reading, Mass. [u.a.] : Addison-Wesley, 1997. - XXII, 630 S. , ISBN 0-8053-1295-1
- Stallings, William: Operating systems: internals and design principles, 4th ed. Upper Saddle River, N.J. : Prentice Hall, cop. 2001, xviii, 779 p., ISBN: 0130329866
- Bacon, J., Harris, T., Operating systems, concurrent and distributed software design, Addison Wesley (Pearson), Harlow, 2003., ISBN 0-321-11789-1
- Frisch, Aeleen, Windows NT rendszeradminisztráció, ford. Mogyorósi István , [Budapest] : Kossuth ; [cop.] 1999, 459 p., ISBN 963 09 4094 9
- Petersen, Richard, Linux : referenciakönyv : könnyen is lehet , [ford. Szilágyi Erzsébet, Vankó György, Varga Imre] ; [a 21. fejezet szerzõi Mayer Gyula, Sudár Csaba és Wettl Imre] Budapest : Panem ; Maidenhead : McGraw-Hill, 1998, ISBN 963-545-177-6
- M. Beck et al., LINUX KERNEL PROGRAMMING, 3rd ed., Addison Wesley (Peason), London, 2002., ISBN 0 201 71975 4.
- B. W. Kernighan, Rob Pike, A UNIX operációs rendszer, [ford. Turi Gabriella, Kovács Tibor] ; [a verseket ford. Tandori Dezsõ], 3. kiad. - Budapest , Mûszaki Könyvkiadó, 1994, 362 p., ISBN 963 16 0498 5