Operációs rendszerek I.-II.

(Előadásvázlat)

Szerkesztette: Csige Lóránt

Számítógépes rendszerek: szerkezeti jellemzők

1. Fő szerkezeti elemek
2. A processzor regiszterei
3. Utasításvégrehajtás
4. Megszakítások
5. Tárrendszerhierarchia
6. Gyorsítótár (cache)
7. I/O kommunikációs technikák

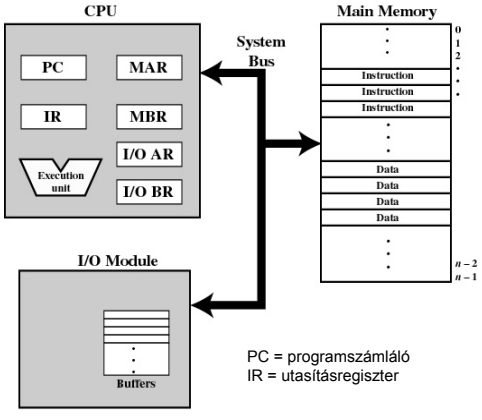
Fő szerkezeti elemek

- Processzor
- Fő memória: *memória*nak, illetve *operatív tár*nak, *főtár*nak is nevezik
- I/O egységek
  - másodlagos memória eszközök, másodlagos tárak, háttértárak
  - kommunikációs eszközök
  - terminálok
- Rendszerbusz
  - kommunikáció a processzor(ok), a memória és az I/O egységek között

Operációs rendszer:

- Egy vagy több processzor hardver erőforrásainak kihasználását optimalizálja
- A rendszer használóinak egy sor szolgáltatást biztosít
- Kezeli a másodlagos memóriát és az I/O eszközöket

Fő szerkezeti elemek



1. ábra Számítógép összetevői (komponensek)

CPU regiszterei:

- MAR - **M**emory **A**ddress **R**egister
  - a következő írás/olvasás memóriacíme
- MBR - **M**emory **B**uffer **R**egister
  - memóriába küldendő adatok tárolása
  - memóriából olvasott adatok tárolása
- I/OAR - **I/O A**ddress **R**egister
  - kijelöl egy bizonyos I/O eszközt
- I/OBR - **I/O B**uffer **R**egister
  - a processzor és a I/O eszközök közötti kommunikáció adattárolására
- Programszámláló- és utasításregiszter

A processzor regiszterei

- Regiszterek: az operatív tárnál gyorsabb és kisebb memória, mely a feldolgozás alatti ideiglenes adattárolásért felelős:
  - Felhasználó által látható regiszterek
    - lehetővé teszi a programozó számára, hogy csökkentse az operatív tárra való hivatkozások számát
  - Vezérlő- és állapotregiszterek
    - a processzor használja önmaga vezérléshez
    - az operációs rendszer egyes rutinjai használják a programok futásának vezérléséhez

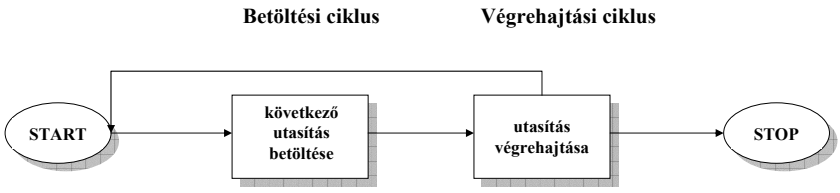
Programozó számára látható regiszterek

- 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 verem 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)

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

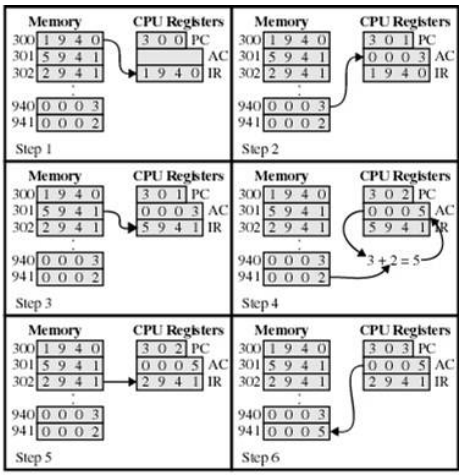
Utasításvégrehajtás



2. ábra Egyszerű utasításciklus

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

Egy program végrehajtása

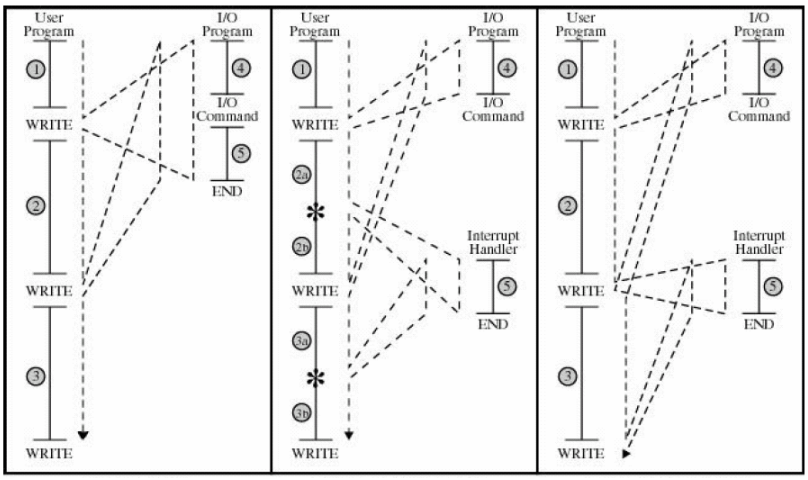


3. ábra Példa egy program végrehajtására

Megszakítás

- 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:
  - Pl.: processzor más utasításokat hajthat végre amíg egy I/O művelet folyamatban van

Megszakítás példa



a) Nincs megszakítás ***b) Megszakítás, rövid I/O*** ***c) Megszakítás, hosszú I/O***

4. ábra Programok végrehajtásának folyamata megszakítással és anélkül

Megszakításkezelő

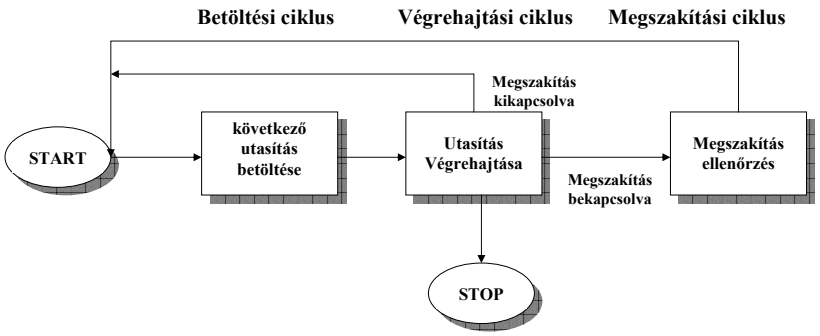
- Program, amely meghatározza a megszakítás okát és végrehajtja azokat az eljárásokat, amelyek ebben az esetben szükségesek (a vezérlést megszakításkor kapja meg)
- A *megszakítás (interrupt)* átadja a vezérlést a *megszakítás-feldolgozó rutinnak*. Ez általában a *megszakítási vektor* segítségével történik, 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 (op. rendszer része!) közvetlen feladatai:
  - „maszkolás„
  - a CPU állapotának megőrzése
  - a megszakítás okának, körülményeinek részletesebb elemzése
  - a megszakított programhoz történő visszatérés megszervezése

A megszakítások osztályai

- Program
  - aritmetikai túlcsordulás
  - nullával való osztás
  - nem létező művelet végrehajtásának megkísérlése
  - felhasználói memóriaterületen kívülre való hivatkozás (szegmentáció)
  - „rendszerhívás” (szoftver által direkt módon kiváltott megszakítás)
- Időzítő, időadó (timer)
- I/O
- Hardverhiba

Megszakítás-ciklus

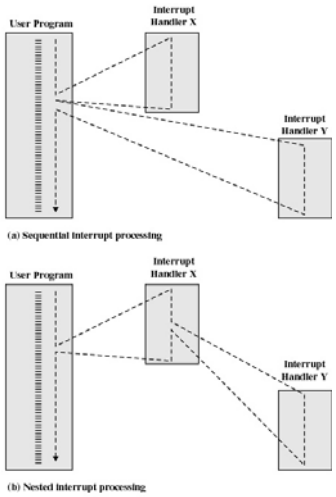
- a processzor „megvizsgálja van-e megszakítás”
- ha nincs, lehívja a program szerint soron következő utasítást a memóriából
- ha egy megszakítás függőben van, felfüggeszti a program végrehajtását, és elindítja a megfelelő megszakításkezelőt.



5. ábra Utasításciklus megszakítással

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



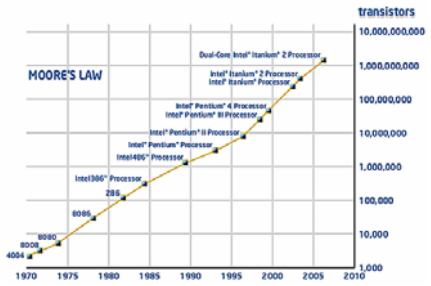
6. ábra Többszörös megszakítás

Megszakításkezelés

Sorrend és prioritás:

- A megszakítások letilthatók (maszkolás), amíg a processzor befejez egy feladatot, és függőben marad 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

Multiprogramozás



- *Moore* törvényének egy megfogalmazása : „az integrált áramkörökben lévő tranzisztorok száma minden 18. hónapban megduplázódik”
- A háttértárak sebessége azonban messze nem nő ilyen ütemben!

→ Megoldás: multiprogramozás

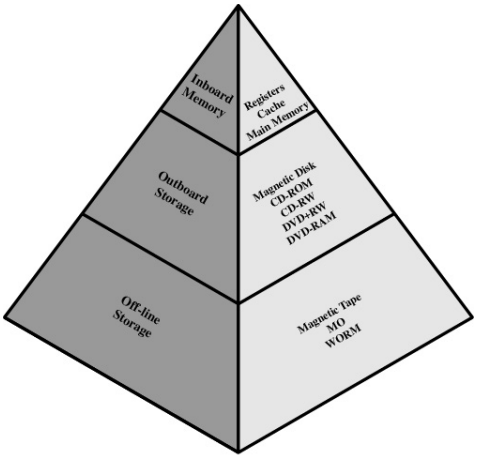
7. ábra Moore törvénye

Multiprogramozás kivitelezése:

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

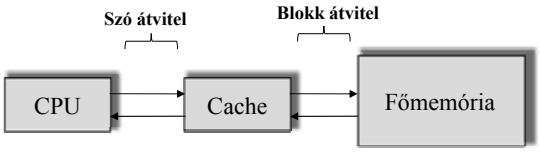
Tárrendszer-hierarchia

- Csökkenő bitköltség
- Növekvő kapacitás
- Növekvő hozzáférési idő
- Csökkenő memória-processzor kommunikációs frekvencia



8. ábra Tárrendszer hierarchia

Cache memória (gyorsítótár)

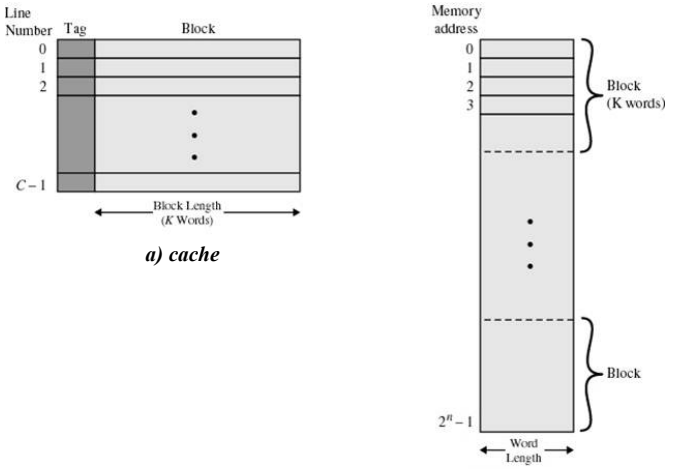


- az operációs rendszer számára láthatatlan
- növeli a memória sebességét
- a processzor sebessége gyorsabb a memóriánál

9. ábra A cache és a fő memória

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

Cache/főmemória szerkezet

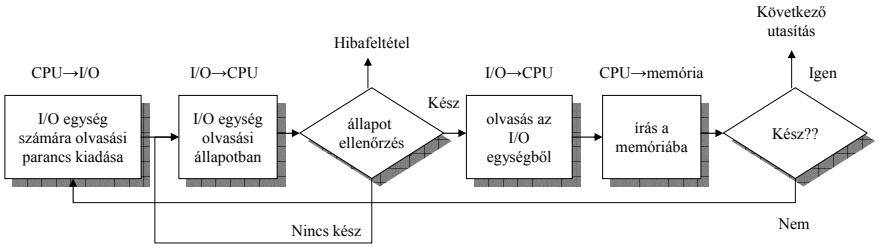


a) cache ***a) főmemória***

10. ábra A cache és a főmemória szerkezete

Programozott I/O

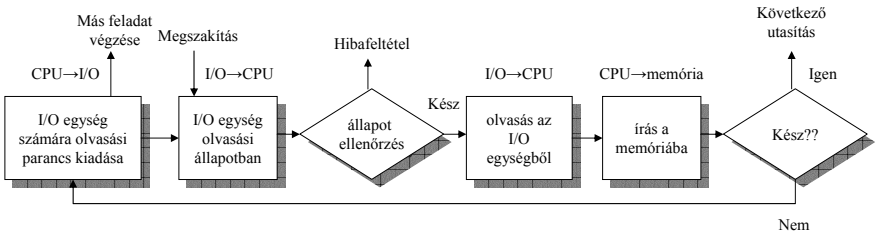
- az I/O modul végzi el a művelet, nem a processzor
- az I/O állapotregiszter bit értékeinek beállítása is megtörténik
- megszakítás nem lehetséges!
- a processzor ellenőrzi a művelet állapotát, amíg az be nem fejeződik



11. ábra Programozott I/O

Megszakításvezérelt I/O

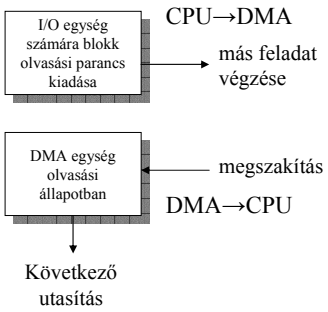
- ha egy I/O egység kész adatok cseréjére, a processzornak megszakítást küld
- a processzor más munkákkal foglalkozhat, így nincs haszontalan várakozás
- még így is sok processzoridőt fogyaszt, mert minden olvasás és írás a processzoron keresztül történik



12. ábra Megszakításvezérelt I/O

Közvetlen memóriahozzáférés (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 foglalkozhat más feladatok elvégzésével

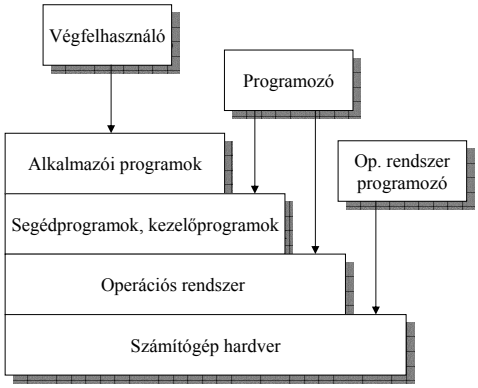


13. ábra Közvetlen memóriahozzáférés

Operációs redszerek: Áttekintés

1. Az operációs rendszer (szolgáltatásai)
2. Az operációs rendszerek fejlődése
3. Főbb elemek
4. Modern rendszerek jellemzői
5. A Windows 2000 és a Unix

Számítógépes rendszer rétegei



1. ábra A számítógépes rendszer rétegei

Operációs rendszerek

Olyan program, mely vezérli a felhasználói programok és alkalamzások futását, interfész a felhasználó és a hardver között.

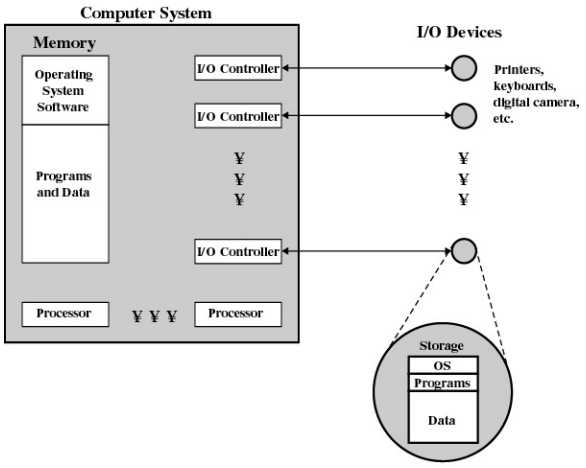
Célok:

- Komfort: a számítógép használatát kényelmesebbé teszi
- Hatásosság: a számítógépes rendszer erőforrásainak optimális kihasználását teszi lehetővé
- Testreszabás lehetősége: lehetőséget teremt a fejlesztésre, tesztelésre és új rendszerfüggvények bevezetésére anélkül, hogy összeakadnánk egyéb szolgáltatásokkal

Az op. rendszer szolgáltatásai

- Programok fejlesztése, készítése: szerkesztés és hibakeresés
- Programvégrehajtás (Processzuskezelés – *process management*)
- I/O rendszer kezelése
- File-okhoz való hozzáférés vezérlése (*file management*)
- Másodlagos tár kezelése
- Rendszerhozzáférés
- Hibaészlelés és válasz
  - belső és külső hardverhibák: memóriahiba, eszközhiba
  - szoftver hibák: számolási túlcsordulás, tiltott memóriaterületekhez való hozzáférés
- Könyvelés (*accounting*)
  - statisztika gyűjtése a rendszerről
  - teljesítmény monitorozása
  - felhasználók kezelése
- Védelem

Erőforráskezelés



2. ábra Az operációs rendszer, mint erőforráskezelő

Az operációs rendszerek fejlődése

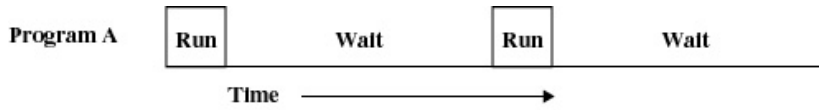
- A programok feldolgozásának lépései (fordítás, szerkesztés, futtatás)
- Soros feldolgozás
  - nincs operációs rendszer; a felhasználó közvetlenül a hardvert éri el, a programozó egyben operátor is, egyfelhasználós rendszer
  - problémák: drága erőforrások alacsony hatásfokú kihasználása (hosszú beállítási 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*)

A kötegelt feldolgozás (*Simple Batch*)

- 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 adatokon milyen fordítót használjon)
- szükséges hardverjellemzők:
  - memóriavédelem: a monitort tartalmazó memóriaszegmens megváltoztatásának letiltása
  - időzítés: *job*ok meggátolása abban, hogy kisajátítsák a rendszert
  - lefoglalt utasítások: csak a monitor által használható utasítások
  - megszakítások: rugalmasságot biztosít a felhasználói szoftverek vezérléséhez

A kötegelt feldolgozás (I.)

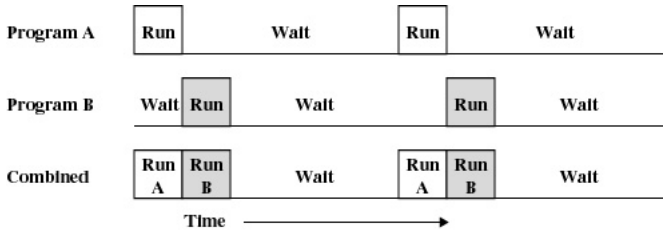
- Egyfeladatos feldolgozás:
  - a processzornak várnia kell egy I/O utasítás befejeződésére mielőtt továbblép, az I/O és a CPU műveletek nem fedhetik át egymást
  - probléma: I/O lassú a processzorhoz képest (pl. kártyaolvasó lassú), CPU nem megfelelően kihasznált



3. ábra Uniprogramozás

A kötegelt feldolgozás (II.)

- 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)



4. ábra Multiprogramozás

A kötegelt feldolgozás (III.)

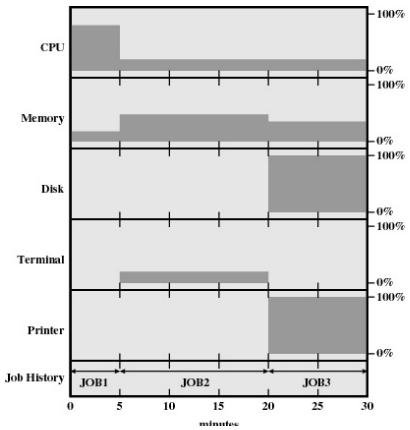
• Uniprocesszálás és multiprocesszálás összehasonlítása:

	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

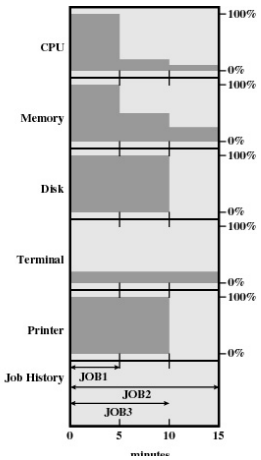
A multiprogramozás által az operációs rendszerekkel szemben támasztott követelmények

•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.

A kötegelt feldolgozás (IV.)



a) Uniprocesszálás



b) Multiprocesszálás

4. ábra Kihasználtság uniprocesszálás és multiprocesszálás esetén

A kötegelt feldolgozás (V.)

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

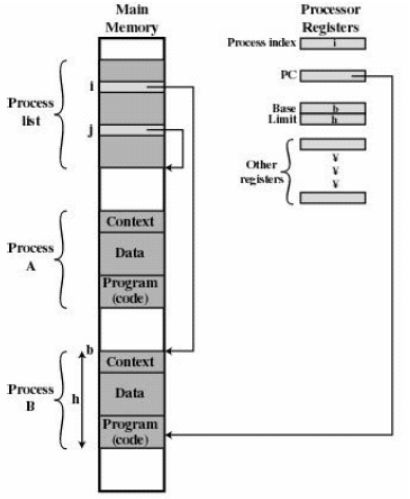
Időosztásos rendszerek

- A kötegelt rendszerek hátránya: nincs interaktivitás!
- a CPU váltakozva áll olyan *job*oknak 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.

Folyamatok (Processzusok - *Process*)

- *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 op. 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

Processzusok



5. ábra Egy processzus tipikus végrehajtása

Memóriakezelés

- Az operatív memóriát bájtokból (szavakból) álló (absztrakt) tömbnek fogjuk tekinteni, 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

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 op. rendszer a következőkért felelős a másodlagos tár kezelését illetően:
  - Szabadhely kezelés
  - Tárhozzárendelés
  - Lemezelosztás, ütemezés (*scheduling*)

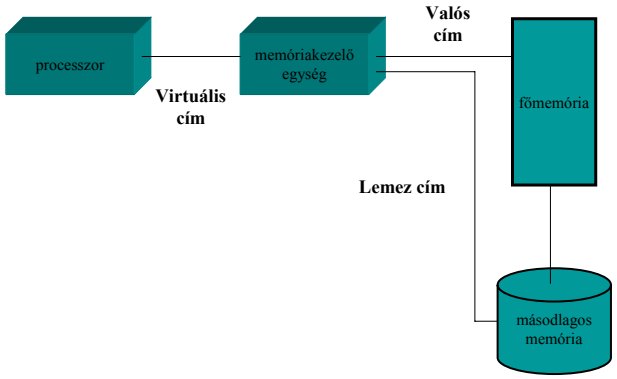
Fájlrendszer kezelés:

- Az információ 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.

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 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

Virtuális memóriacímzés



7. ábra A virtuális memória címzése

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
  - *igazolás*a 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

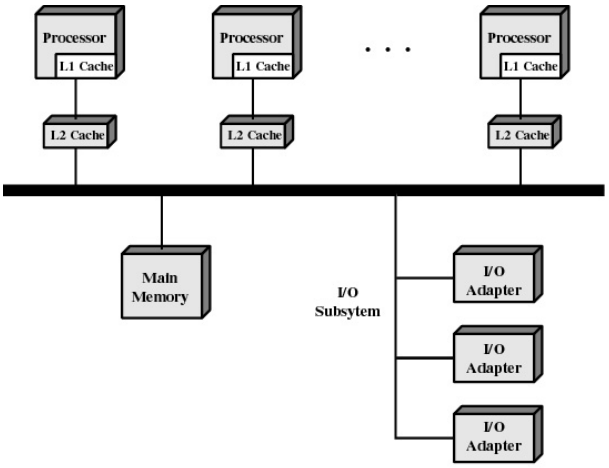
Modern rendszerek jellemzői

- 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

Modern rendszerek jellemzői

- Párhuzamos rendszerek
  - szimmetrikus multiprocesszálás
    - több processzor, melyek ugyanazon főmemórián és I/O rendszeren osztoznak
    - minden processzor az op. 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

Modern rendszerek jellemzői



8. ábra Szimmetrikus multiprocesszálás

Modern rendszerek jellemzői

- 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.
  - „*hard*” („merev" valós idejű) rendszerek.
    - a másodlagos tár korlátozott, vagy teljesen hiányzik; az adatokat az operatív memóriában (RAM), vagy akár ROM-ban tárolják.
    - 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) igen hasznosak.

A Windows 2000 (áttekintés)

- a 32 bites mikroprocesszorok teljesítményének kiaknázására fejlesztették
- 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

A Windows 2000 (áttekintés)

Réteges szerkezet:

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

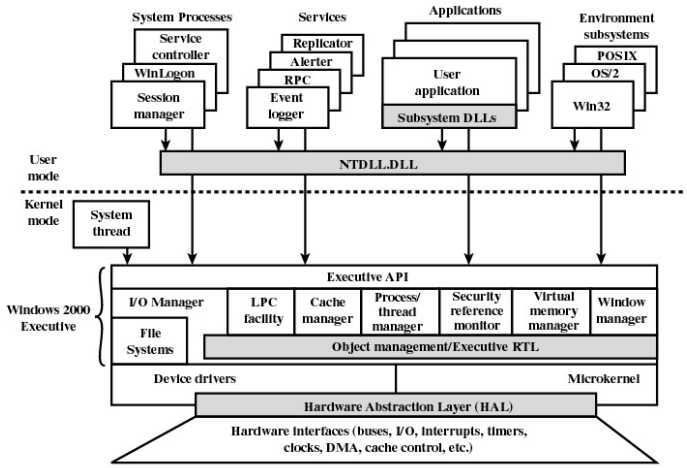
Adminisztratív modulok:

- 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:

- 1. rendszert támogató processzusok (bejelentkezés, session manager)
- 2. szerver processzusok, 3. környezeti alrendszerek processzusai, 4. felhasználó alkalmazások

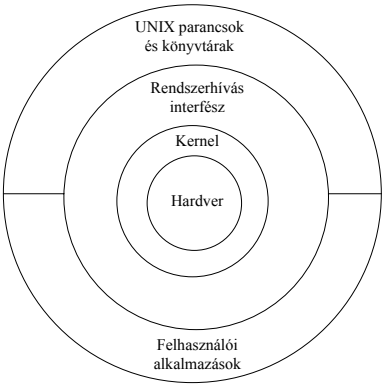
A Windows 2000 (áttekintés)



9. ábra Windows 2000 architektúra

A Unix (áttekintés)

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



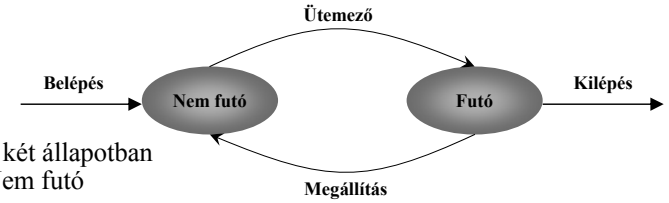
10. ábra Unix architektúra

Processzusleírás és -vezérlés

1. Processzusállapotok
2. Processzusleírás
3. Processzusvezérlés
4. A UNIX processzuskezelése

Két állapotú processzus modell

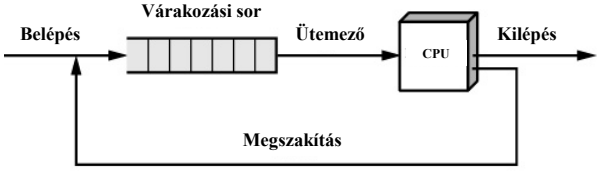
- Folyamat (*processzus*): végrehajtás alatt álló program



- A processzus két állapotban lehet: Futó, Nem futó

1. ábra A két állapotú processzus modell

- Nem futó processzus várakozása:



2. ábra A várakozási sor (queue)

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 op. 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 op. rendszer által létrehozott processzusok valamilyen szogáltatásnyújtás érdekében (pl. nyomtatásvezérlés)
    - egy már létező processzus is létrehozhat processzust (egymással kapcsolatban álló processzusok kommunikációját meg kell oldani!)

Processzusmegszakítás

- Kötegelt munka kiadja a *Halt* utasítást
- Egy felhasználó kijelentkezik
- Alkalmazásból való kilépés
- Bizonyos hibafeltételek teljesülése

- Normális processzusbefejezés
- Időhatár túllépése
- Memória nem áll rendelkezésre
- Határok megsértése *(Bounds violation)*
- Védelmi hiba: például írás csak olvasható fájlba
- Számolási hiba
- Időtúllépés
- I/O hiba

Okai:

- Érvénytelen utasítás: adat „végrehajtása”
- Privilegizált utasítás: az utasítás az op. rendszer által lefoglalt
- Használhatatlan adatsor
- Operációs rendszer közbeavatkozása
- Szülő processzus és így az utód processzus is megszakad
- Szülő processzus által történő megszakítás

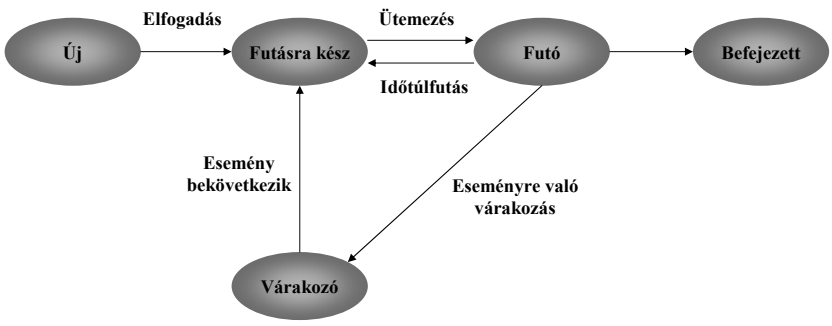
Öt állapotú modell

- 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 szkennelnie a listát a nem blokkolt legrégebbi processzus után keresve
  - Nem futó processzusok kettéválasztásának szükségessége:
    - Futásra kész (*Ready*) állapot és blokkolt (*Blocked*) állapot

A processzusok öt állapota:

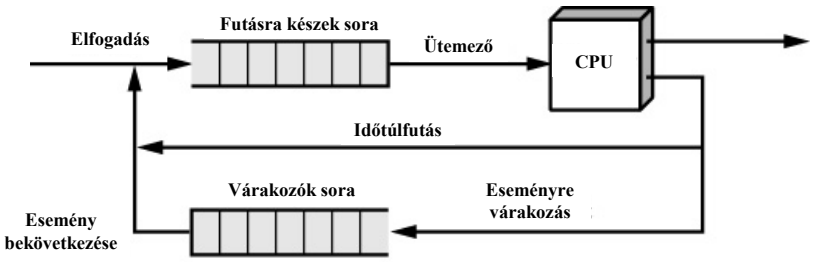
- Futó (*Running*)
- Futásra kész (*Ready*)
- Blokkolt vagy várakozó (*Blocked*)
- Új (*New*): újonnan létrehozott processzus, mely nincs még a főmemóriában
- Befejezett (*Exit*): processzus, melyet az op. rendszer kivon a végrehajtandó processzusok közül

Öt állapotú modell



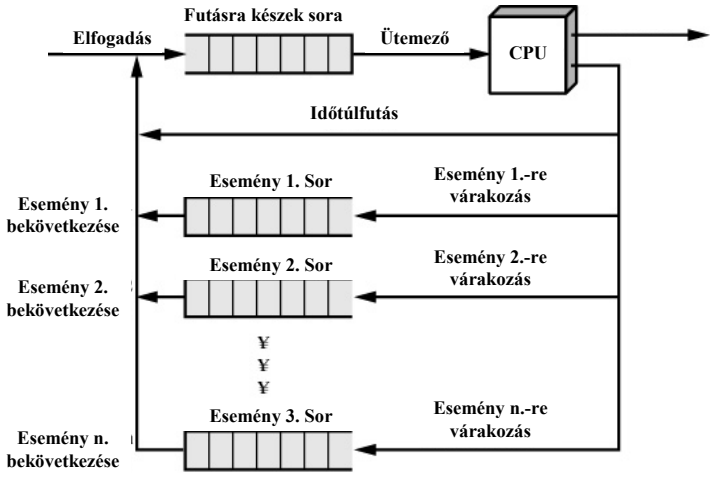
3. ábra Az öt állapotú folyamat modell

Várakozási sor használata



4. ábra Egyszeres várakozási sor

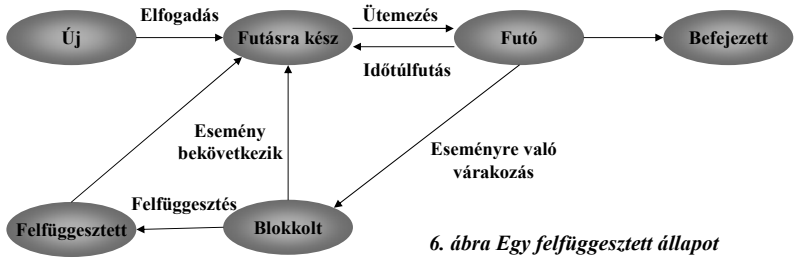
Várakozási sor használata



5. ábra Többszörös várakozási sor

Processzusfelfüggesztés

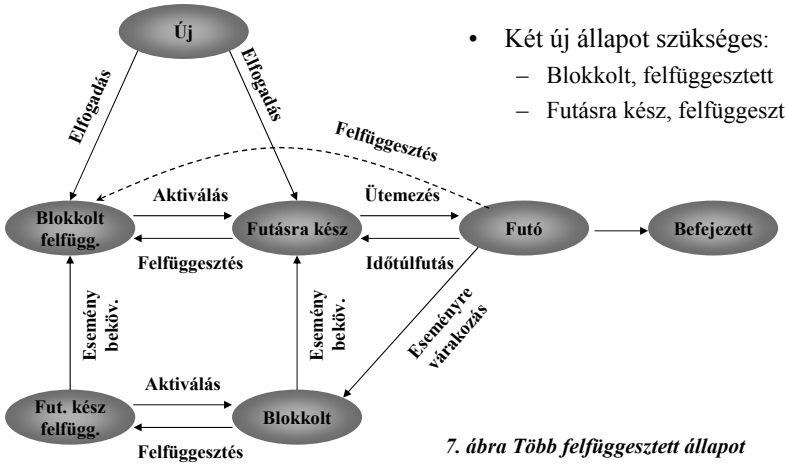
- A processzor sokkal gyorsabb, mint az I/O rendszer, így előfordulhat, hogy az összes processzus I/O-ra vár (a processzor üresjáratban van....)
- Ezen processzusok memóriából lemezre történő mozgatásával memória szabadítható fel új processzusok számára *(swap in, swap out)* - *SWAPPING*
- A processzus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



6. ábra Egy felfüggesztett állapot

Két felfüggesztett állapot

- Probléma: egy felfüggesztett processzus időközben futásra késszé válhat



- Két új állapot szükséges:
  - Blokkolt, felfüggesztett
  - Futásra kész, felfüggesztett

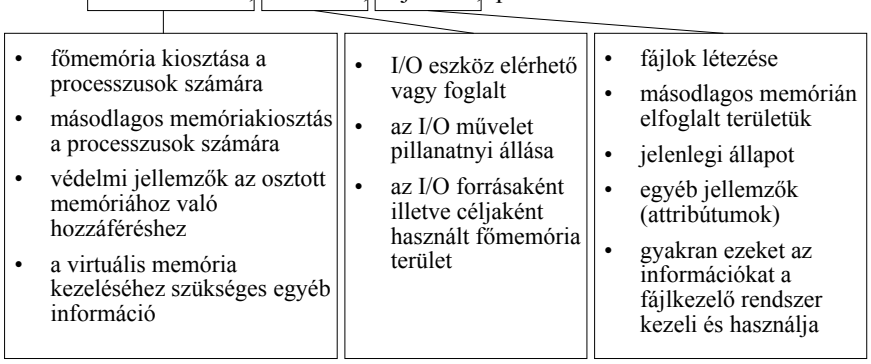
7. ábra Több felfüggesztett állapot

Processzusfelfüggesztés okai

- Swapping:
  - az operációs rendszernek főmemóriát kell felszabadítani, hogy egy készen álló processzust be tudjon tölteni
- Egyéb operációs rendszerhez köthető okokból:
  - pl. 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:
  - szülő processzus felfüggesztheti az utód processzust annak vizsgálata illetve megváltoztatása céljából

Processzusleírás

- Az operációs rendszernek információra van szüksége a processzusok és erőforrások pillanantnyi állapotáról
- Az op. rendszer az által felügyelt egységekhez táblázatokat rendel
- Négy ilyen táblázat (op. rendszer függő):
  - Memória tábla, I/O tábla, fájl tábla, processszustábla



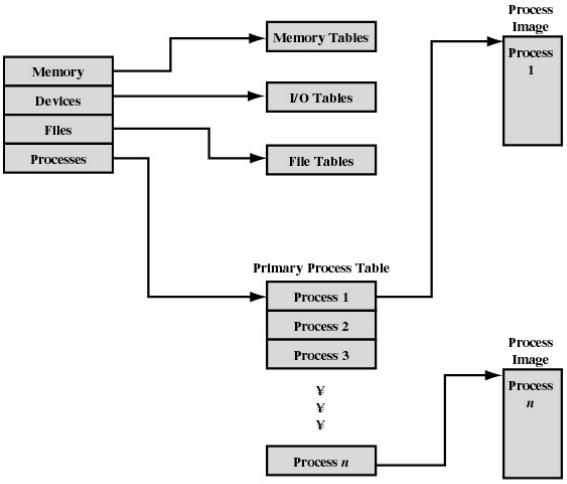
A processzustábla

- Hol található meg a processzus?
- Jellemzők, melyek szükségesek a kezelésükhöz:
  - 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 op. rendszer számára a processzus vezérléséhez szükséges adatok

A processzustábla



8. ábra Operációs rendszert vezérlő táblák

A processzusvezérlő blokk elemei

- Processzusazonosítás
  - processzusazonosító: egyedi numerikus azonosító
    - az elsődleges processzustábla egy indexe is lehet
  - szülőprocesszus azonosítója
  - felhasználóazonosító
- Processzor Állapot Információ (*Processor State Information*)
  - Felhasználó által látható regiszterek állapota
  - Vezérlő- és státuszregiszterek állapota: processzorregiszterek, melyek a processzor működését vezérlik
    - *Programszámláló:* a következő meghívandó 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.

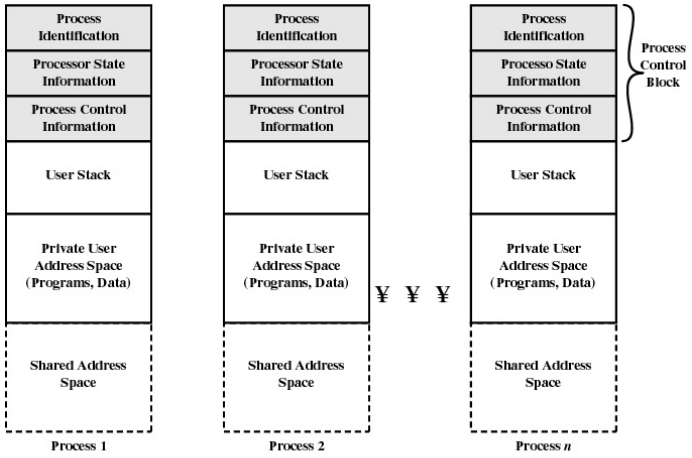
A processzusvezérlő blokk elemei

- Processzusvezérlő Információ (*Process Control Information*)
  - Ütemezési és állapot információ: ez az információ szükséges az op. 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. Pl: 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

A processzusvezérlő blokk elemei

- Processzusvezérlő Információ
  - processzusok közötti kommunikáció
    - Több jelző illetve üzenet is rendelhető két független processzus kommunikációjához. Ezen információk egy része vagy egésze a processzusvezérlő blokkban tarthatók fent.
  - processzus privilégiumok
    - a processzusoknak privilégiumok adhatók; a számára elérhető memóriát és a végrehajtható utasítások típusait határozza meg
  - memóriakezelés
    - ez a rész laptábla mutatókat tartalmazhat mely a processzushoz rendelt virtuális memóriát írja le
  - erőforrás felhasználás
    - a processzus által vezérelt erőforrásokat (pl. megnyitott fájlok) jelezheti. A processzor illetve más erőforrás felhasználásának történetét is tartalmazhatja; ez az információ az ütemezőrendszer számára lehet fontos.

A processzusvezérlő blokk elemei



9. ábra Operációs rendszert vezérlő táblák

Processzusvezérlés

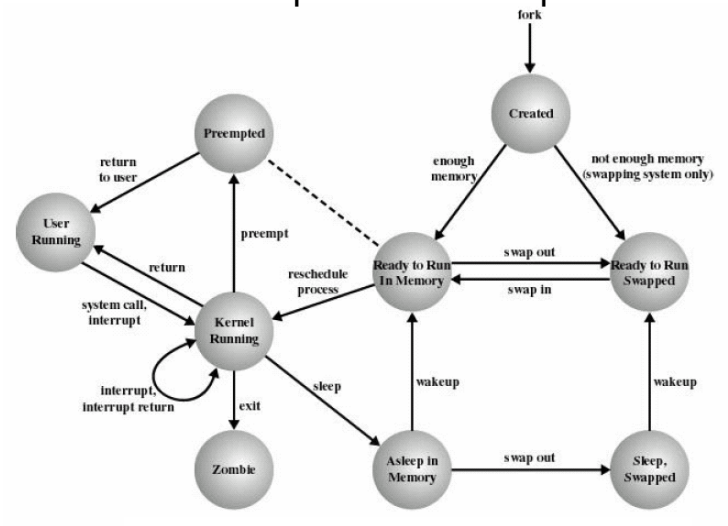
- Végrehajtás módjai:
  - Felhasználói mód
    - csökkentett privilégiumokkal járó mód
    - felhasználói programok tipikusan ebben a módban kerülnek végrehajtásra
  - Kernel mód
    - több privilégiummal rendelkező mód
    - teljes felügyelet a processzor (és összes utasítása), a regiszterek és a memória 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

Processzusvezérlés (processzusváltás)

Processzusváltás okai:

- Megszakítás
  - Óramegszakítás
    - a processzus a maximális időszeleten túlfut
  - I/O megszakítás
  - Memóriahiba
    - a memóriacím a virtuális memóriában lévő szóra mutat amit először a főmemóriába kell áthozni, csak ezután futhat tovább a processzus
- Csapda
  - hibaesemény
  - a processzus „Kilépés” állapotba történő mozgatását jelentheti
- Rendszergazda hívás
  - op. rendszer függvényének hívása

A UNIX processzusállapotai



10. ábra A UNIX processzusállapotai és átmenetei

Szálak, mikrokernelek

1. Processzusok és szálak
2. Mikrokernelek
3. A UNIX processzuskezelése
4. Windows 2000 és Linux processzusai, száljai

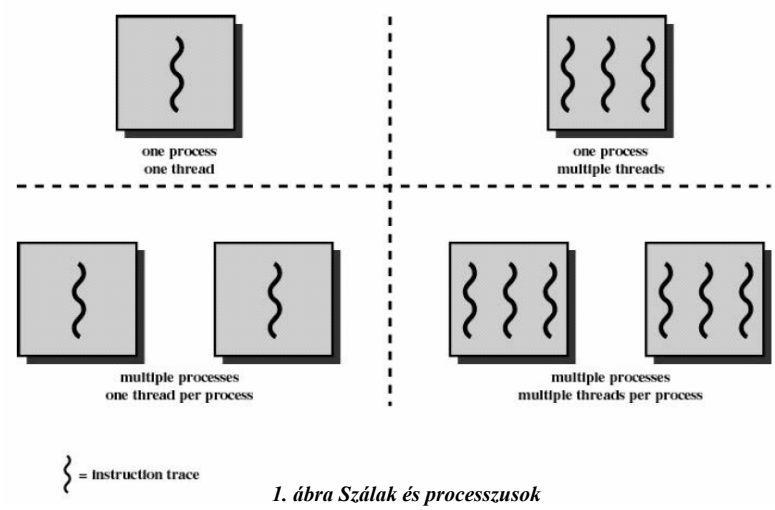
Processzusok és szálak (*Thread*)

- 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 op. rendszer egymástól függetlenül kezelheti őket:
  - Processzus
    - erőforráskiosztás alapegysége
    - virtuális címtartomány, főmemória, I/O eszközök és fájlok
  - Szál (vagy könnyűsúlyú processzus)
    - processzor kiszolgálás, ütemezés alapegysége
    - ütemezés és kiszolgálás op. 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 op. rendszer támogathat 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

Többszörös szálak *(Multithreading)*



1. ábra Szálak és processzusok

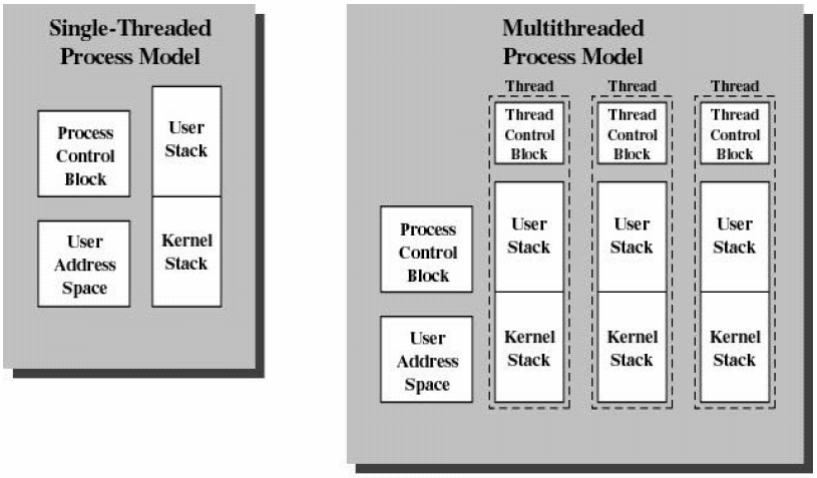
Szálak

- 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 van
  - 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

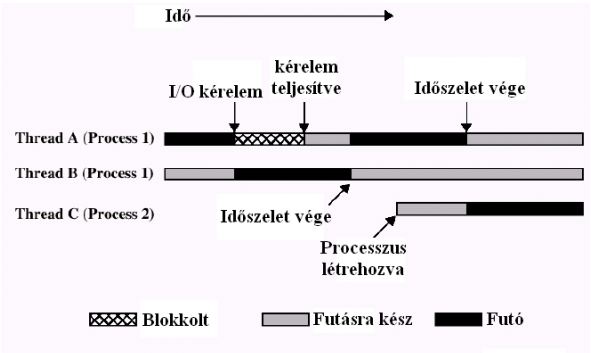
Szálak



2. ábra Egyszeres és többszörös szálak

Szálak

- Műveletek melyek egy szál állapotát megváltoztatják
  - Származtatás: másik, új szálat származtatni
  - Blokkolás, Unblokkolás
  - Befejezés: erőforrások felszabadítása (regiszterek, vermek)

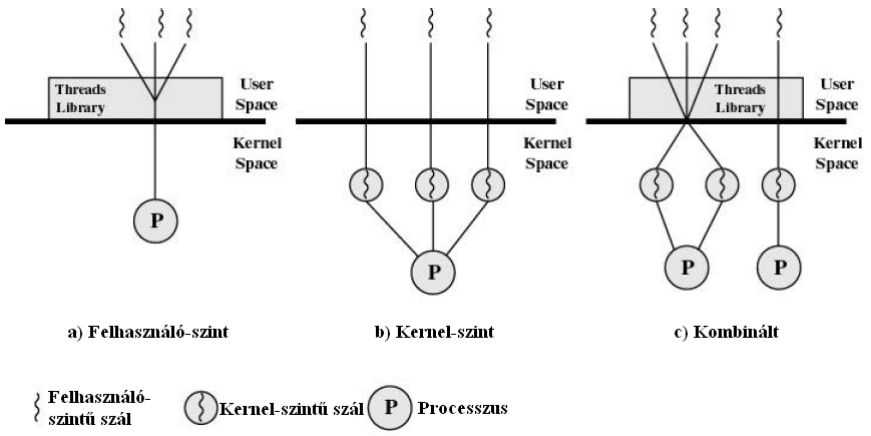


3. ábra Multithreading egy processzoron

Szálak megvalósítása

- Felhasználói-szintű szálak (*User Level Thread* - ULT)
  - a szálak kezelését az alkalmazások 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
  - Pl.: Solaris

Szálak megvalósítása

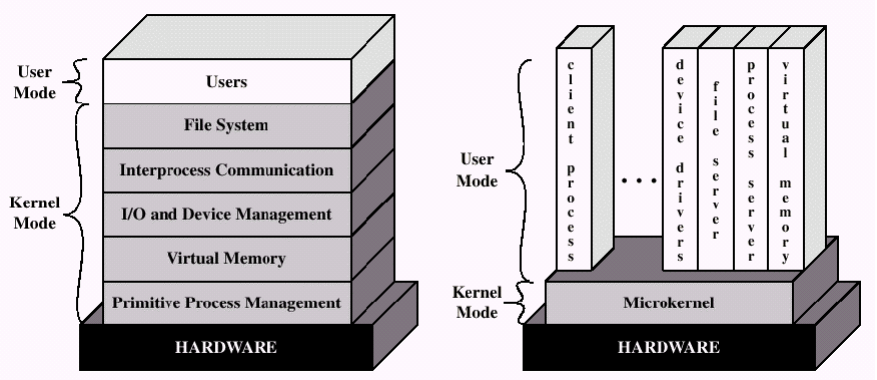


4. ábra Felhasználó-szintű és kernel-szintű szálak

Mikrokernel

- Kis operációs rendszermag
- Csak az alapvető operációs rendszerfüggvényeket, szolgáltatásokat tartalmazza:
  - alacsony szintű memóriakezelés
    - hozzárendelni minden virtuális oldalt egy fizikai oldalhoz
  - processzusok közötti kommunikáció
    - üzenet *(message)* az alapvető forma
    - 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

Mikrokernel



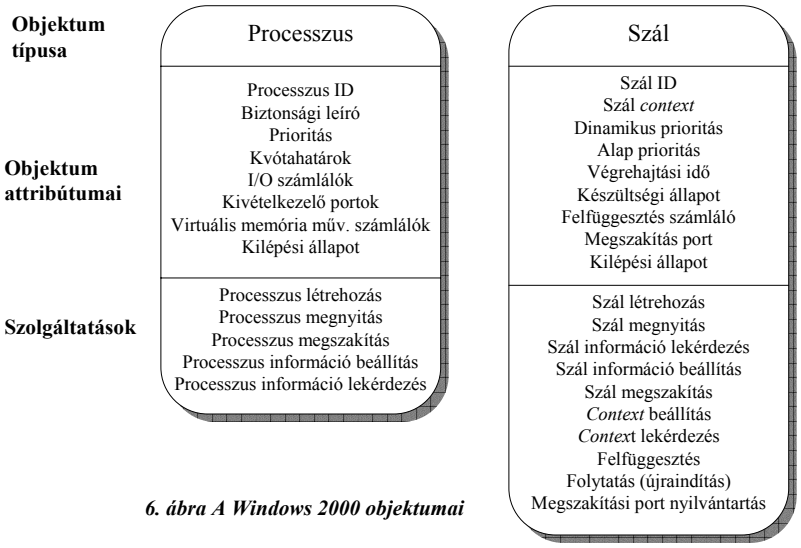
a) Hagyományos kernel ***b) Mikrokernel***

5. ábra Kernel architektúra

A mikrokernel előnyei

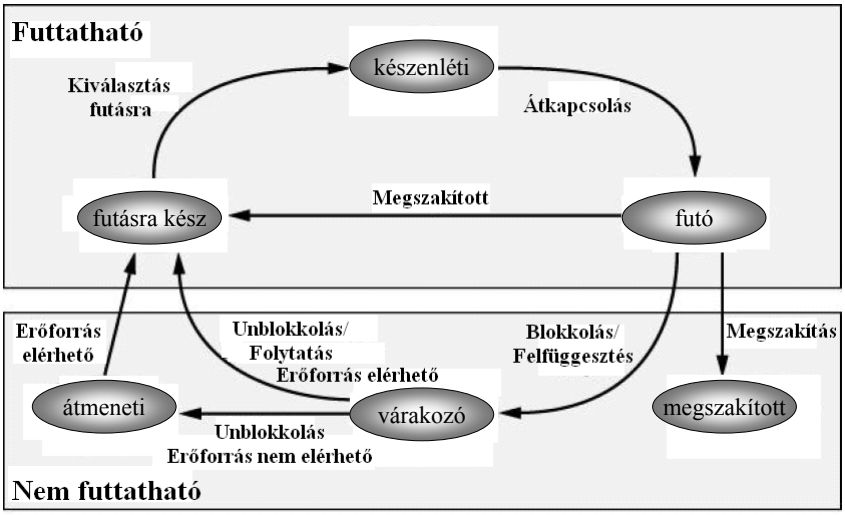
- Egységes felületet biztosít a processzusok számára
  - a processzusoknak nem kell különbséget tenniük kernel-szintű és felhasználó-szintű szolgáltatások között
- Kiterjeszthető
  - új szolgáltatások könnyen hozzáadhatók
- Rugalmas
  - új szolgáltatások hozzáadhatók, létező szolgáltatások kivehetők, 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

A Windows 2000 objektumai



6. ábra A Windows 2000 objektumai

A Windows 2000 objektumai



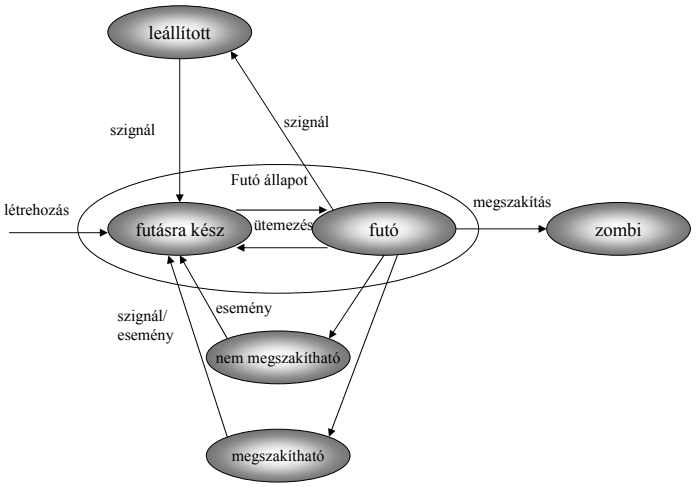
7. ábra A Windows 2000 szálállapotai és azok átmenetei

Linux processzusok, 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

Linux processzusok, szálak



6. ábra A Linux processzus/szál modell

Folyamatszinkronizáció

1. Konkurencia (versenyhelyzetek)
2. Kölcsönös kizárás (megvalósítás és hardvertámogatás)
3. Szemaforok (termelő-fogyasztó, alvó borbély, vacsorázó filozófusok problémája)
4. Monitorok
5. Processzusok közötti kommunikáció (írók-olvasók problémája)

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!

Versenyhelyzetek

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

Versenyhelyzetek

- 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

Kölcsönös kizárás megvalósítása:

- Dekker algoritmusa: kölcsönös kizárás 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

```
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

Kölcsönös kizárás megvalósítása

```
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

- 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!
- Spec. 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, de több processzoros esetre is
  - egyszerű ezért könnyű az ellenőrzés
  - több kritikus szakasz használatát is támogatja
- Hátrányok
  - az „aktív várakozás” jelentősen fogyasztja a processzoridőt
  - Éhezés (*Starving*) 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

Szemaforok

- A szemaforok (**S**) speciális, egész típusú (*integer*) változók, melyeket processzusok megjelölésére használnak
  - nemnegatív kezdőértéket kaphat
  - „*Wait*” művelet csökkenti a szemaforok értékét
    WAIT(**S**): S:=S-1; **if** S<0 **then** BLOCK(**S**);
  - „*Signal*” művelet növeli a szemaforok értékét
    SIGNAL(**S**): S:=S+1; **if** S≥0 **then** WAKEUP(**S**);
- Egy processzus felfüggesztésre kerül, amíg meg nem kapja a jelölést
- A „wait” és „signal” műveletek nem megszakíthatók
- A *block*(S) felfüggeszti a hívó processzus végrehajtását, és a szemaforon várakozó processzusok sorához adja

Termelők/fogyasztók problémája

- Egy vagy több termelő adatot generál melyet egy bufferbe tesz
- Egy egyszerű fogyasztó ezeket az adatokat egyenként veszi ki a tárból
- Egyszerre egy termelő vagy fogyasztó érheti el a tárat

```
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;
}
```


Termelők/fogyasztók problémája

```
/* 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);
}
```

1. ábra A termelők/fogyasztók probléma egy megoldása szemaforokkal

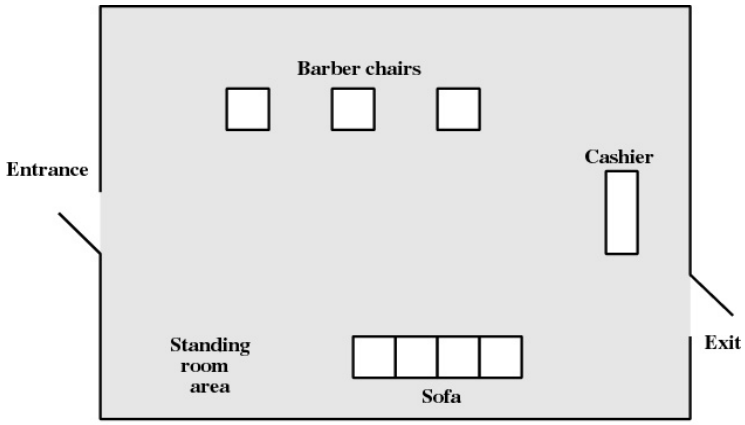
Alvó borbély probléma

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!

Alvó borbély probléma



2. ábra Az alvó borbély probléma

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);
    }
}
```

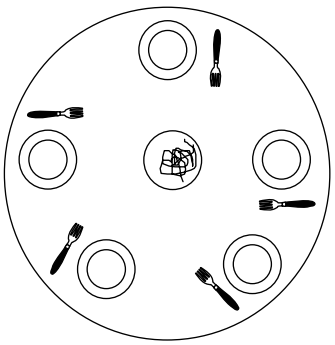
```
void cashier()
{
    while (true)
    {
        wait(payment);
        wait(coord);
        accept_pay();
        signal(coord);
        signal(receipt);
    }
}
```

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

3. ábra Az alvó borbély probléma egy megoldása

A vacsorázó filozófusok probléma

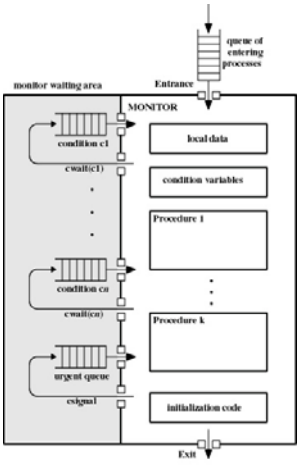
- Egy köralakú asztal mellett öt filozófus ül, mindegyik előtt van egy tányér rizs és a szomszédos tányérok között egy-egy evőpálcika.
- Evéshez a filozófus a saját tányérja melletti két evőeszközt használhatja úgy, hogy ezeket egymás után kézbe veszi.
- Ha befejezte az étkezést, visszateszi az eszközöket,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



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 *állapotváltozó*kat (condition típus) használ két rajtuk elvégezhető művelettel (WAIT, SIGNAL). Ezek az állapotváltozók nem számlálók, mint a szemaforok!

Monitorok



4. ábra Egy monitor szerkezete

```
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.
```

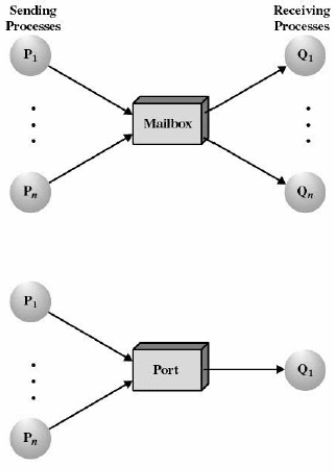
5. ábra A vacsorázó filozófusok probléma megoldása monitorral

Processzusok közötti kommunikáció (IPC)

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

- Az IPC két művelete:
  - **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 közötti kommunikáció (IPC)



6. ábra 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!

Az „olvasók-írók” probléma

```
/* 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);
}
```

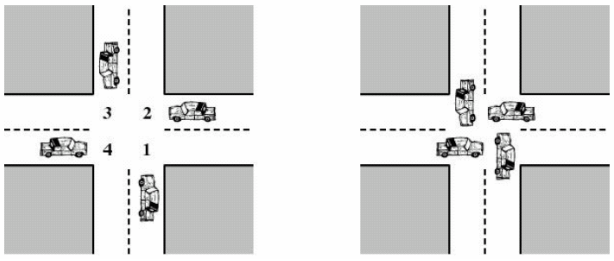
7. ábra Az írók-olvasók probléma egy megoldása (prioritás az olvasóknál)

Holtpont és éhezés

1. A holtpont fogalma
2. Holtpont megelőzés
3. Holtpont elkerülés
4. Holtpont detektálás
5. A UNIX konkurenciakezelése

A holtpont

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

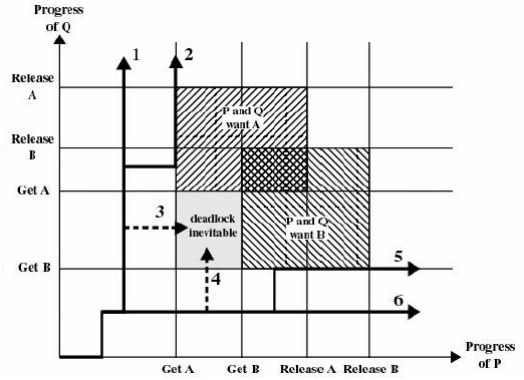


a) Holtpont lehetséges ***a) Holtpont***

1. ábra Holtpont (illusztráció)

A holtpont

- Példa: két processzus (*P, Q*), két erőforrás (*A, B*), mindkét processzus igényt tart mindkét erőforrásra. A 2. ábra a hat lehetséges végrehajtási útvonalat mutatja (egyprocesszoros rendszerben egyszerre egy processzus végrehajtása lehetséges!)
- a 3. és 4. útvonalnál a holtpont elkerülhetetlen!



2. ábra Példa holtpontra

A holtpont kialakulásának négy feltétele

•**Kölcsönös kizárás**: legalább egy – többek által igényelt – erőforrás nem megosztható, azaz egyszerre csak egy processzus használhatja

•**Tartani és várni** (hold & wait): Valamelyik processzus már lefoglalt egy erőforrást, és arra vár, hogy továbbiakat lefoglaljon

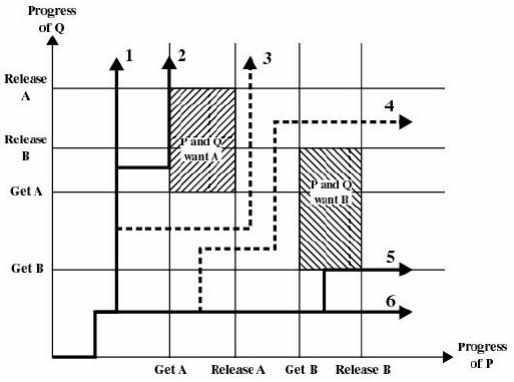
•**Nincs preempció**: az erőforrást a foglalótól nem lehet kívülről – operációs rendszer beavatkozással – elvenni

•**Körkörös várakozás** (circular wait): a körben állók mindegyike a következő által foglalt erőforrásra vár

Megjegyzés: a négy feltételnek egyszerre kell teljesülnie (de nem függetlenek)

A holtpont

- 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.



3. ábra Példa holtpont elkerülésére

A holtpont

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
- pl: 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 (*4. ábra* – 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)

4. ábra Két processzus újrahasználható erőforrásokért versenyez

A holtpont

Fel/el-használható erőforrások:

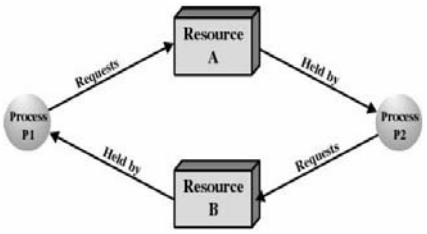
- processzus által létrehozott és megsemmisített erőforrások
- pl: megszakítások, szignálok, üzenetek és I/O pufferekben lévő információk
- két processzus (*P1, P2*) egymástól vár üzenetet, majd annak megkapása után üzenetet küld a másiknak. Így holtpont állhat elő, hiszen a *Receive* blokkolttá válik (*5. ábra*)

5. ábra Két processzus felhasználható erőforrások által okozott holtponthelyzete

A holtpont

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 (*3.ábra*)



5. ábra Körkörös várakozás

Holtpont megelőzés

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

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 ($R_1,...,R_m$), rendelkezésre álló erőforrások (*Available*) vektora ($V_1,...,V_m$), allokációs (*Allocation*) mátrix ($A_{11},....A_{nm}$), illetve az összes processzus összes erőforrásra vonatkozó igényeinek (*Claim*) mátrixa ($C_{11},...,C_{nm}$)
- Így: egy új processzus akkor indítható el, ha $R_i \geq C_{(n+1)i} + \sum^{n}_{k=1} C_{ki}$ az összes *i*-re
- ez nem optimális stratégia, ugyanis a *legrosszabbat* feltételezi: az összes processzus egyszerre akarja megszerezni az összes, számára szükséges erőforrást

Holtpont elkerülése

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 számú erőforrás foglalható csak le
  - processzus nem léphet ki, amíg erőforrást foglal éppen le

Holtpont észlelése

Holtpont detektálási algoritmus:

- allokációs mátrix (A), erőforrás vektor, elérhetőségi vektor
- kérelem mátrix Q bevezetése, ahol q_{Ij} 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 $Q_{ik} \leq W_k$, ahol $1 \leq k \leq m$. Ha ilyen nincs, szakítsuk meg az algoritmust!
  4. ha van, jelöljük meg a processzust és állítsuk be az új W-t: $W_k = W_k + A_{ik}$, ahol $1 \leq k \leq m$, 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

Holtpont észlelése

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.)

A UNIX konkurenciakezelése

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 következő elemekből áll: 1. a szemafor aktuális értéke, 2. a legutóbb szemaforon működő processzus azonosítója, 3. azon processzusok száma, mely arra vár, hogy a szemafor értéke nagyobb legyen, mint jelenlegi értéke, 4. azon processzusok száma, mely arra vár, hogy a szemafor értéke zérus legyen
- Szignálok
  - hasonlatosak a hardver megszakításhoz, de prioritás nélküliek

Memóriagazdálkodás

1. Memóriakezelés
2. Memóriafelosztás
3. Relokáció
4. Lapozás és szegmentáció

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

Memóriakezelés

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 mozgatni a háttértárra (*swap*) és vissza, de a memóriába való visszamozgatása általában eltérő helyre történik (*relocation*)
  - a memória hivatkozásokat a kódba kell fordítani 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

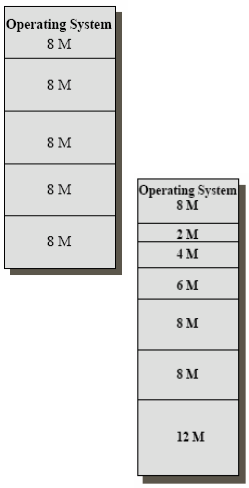
Memóriakezelés

- Megosztás (*sharing*)
  - több processzus számára engedélyezett ugyanazon memóriaszegmens elérése
  - jobb, 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

Memóriafelosztás

Fix partícionálás:

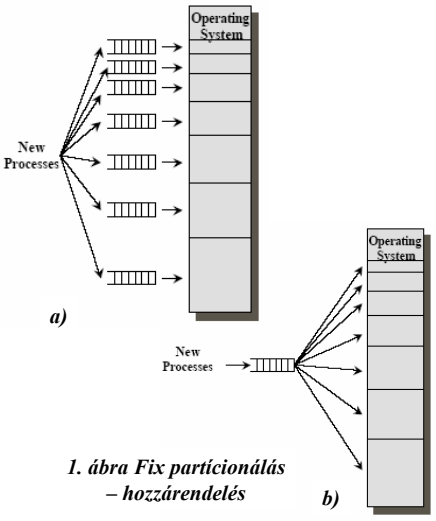
- A memória felosztás fix határokkal rendelkező régiókra
- Egyenlő méretű partíciók kialakítása
  - bármelyik olyan processzus melynek mérete kisebb vagy egyenlő a partíció méretével, betölthető egy elérhető partícióba
  - ha az össze partíció tele van, az operációs rendszer kicserélheti egy partícióban 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



Memóriafelosztás

Elhelyezési algoritmus:

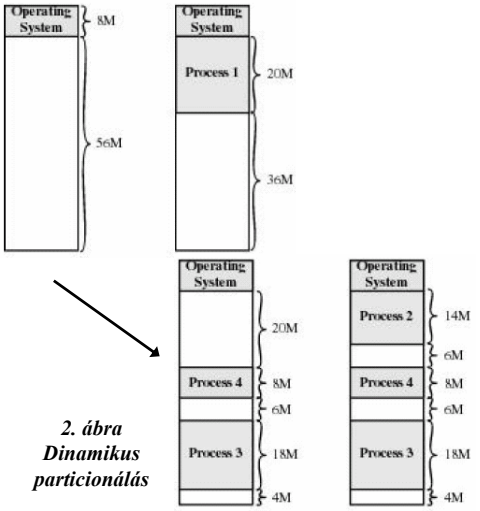
- Azonos méretű partíciók
  - azonos méret miatt bárhova mehet
- Nem azonos méretű partíciók
  - minden processzushoz a lehető legkisebb alkalmas partíciót választani (belső töredezettség minimalizálása)
  - minden partícióhoz külön bemeneti sor *(1/a. ábra)*
  - az összes partícióhoz csak egy bemeneti sor *(1/b. ábra)*



1. ábra Fix partícionálás – hozzárendelés

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....



2. ábra Dinamikus particionálás

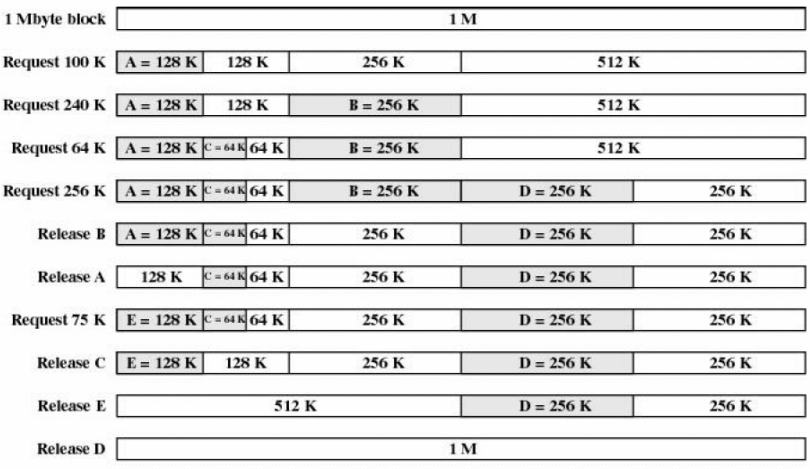
Dinamikus particionálás

- Az op. 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 2^U méretű egyszerű blokk
  - ha a processzus által kért méret $2^{U-1} < s \leq 2^U$, akkor az egész blokk lefoglalásra kerül, máskülönben
    - ezt a blokkot két egyenlő részre osztjuk (2db 2^{U-1} méretű blokk)
    - az eljárást addig folytatjuk, amíg a legkisebb olyan blokkot kapjuk, ami nagyobb vagy egyenlő *s*-sel

„Buddy” rendszer

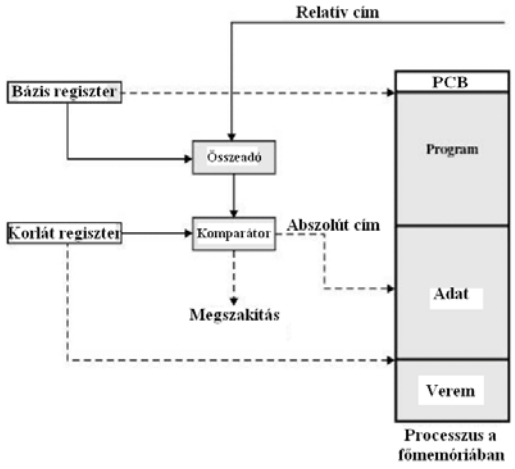


3. ábra Példa „Buddy” rendszerre

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)

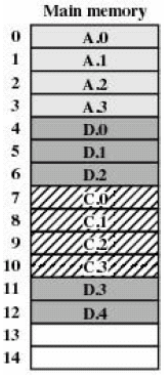
Relokáció



4. ábra Hardveres támogatás a relokációhoz

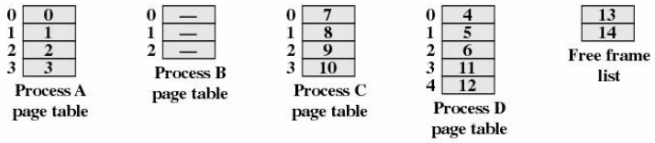
Lapozás (*Paging*)

- A külső fragmentáció problémájának egy megoldását kapjuk, ha a memóriát és a processzusokat kis, egyenlő méretű egységekre osztjuk (processzusdarab: lap – *page* ; memóriadarab: keret – *frame*)



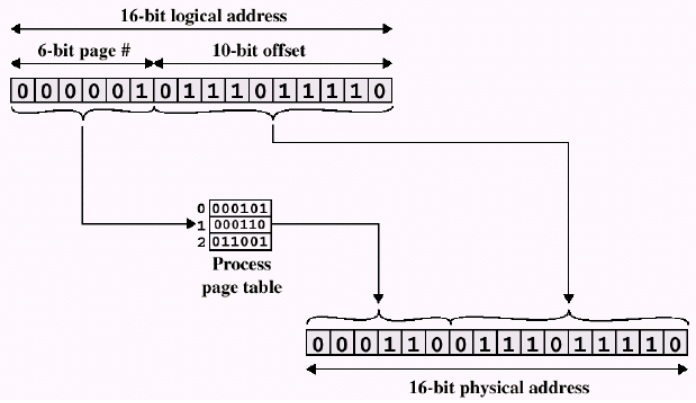
5. ábra Keretek feltöltése lapokkal

- Az op. rendszer minden processzushoz egy ún. laptáblát (*page table*) tart fent
  - tartalmazza a processzus lapjaihoz tarozó keretek helyzetét (*6.ábra*)
  - logikai cím: lap sorszáma + lapon belüli relatív cím
  - fizikai cím: keret memóriabeli kezdőcíme + kereten belüli kezdőcím



6. ábra Laptáblák az 5. ábrán levő processzusokhoz (A,B,C,D)

Lapozás (*Paging*)

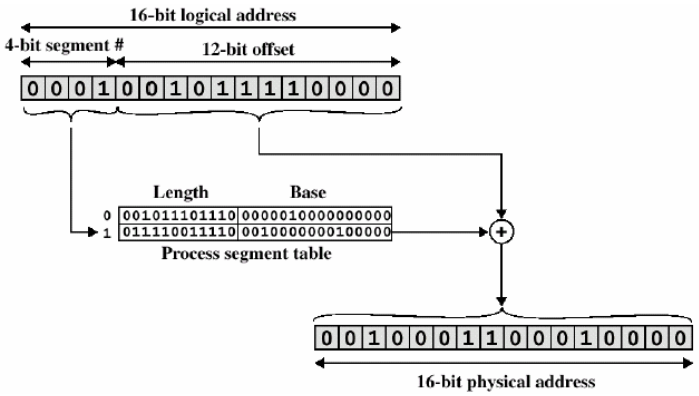


7. ábra Logikai cím fizikai címmé való fordítása (példa)

Szegmentáció

- Felhasználói 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 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]

Szegmentáció



8. ábra 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.