Gábor Dénes
Főiskola

# ELŐADÁS VÁZLATOK

## OPERÁCIÓS RENDSZEREK

**105**

**Vezetőtanár:**
**KNAPP GÁBOR**

**2003**



## *Előszó*

**A tárgy célja**
- Informatikai célok
- Általános mérnöki célok
- Menedzsment aspektusok

**Az alkalmazott módszerek**
- Megközelítés
- Esettanulmányok
- Számítási példák
- Számonkérés

**Eredmény**



## *Cél 1. → Módszer → Eredmény*

**Konkrét célok („informatikus")**
- Az operációs rendszerek lehetőségeinek és korlátainak megismerése és megértése, értelmezése.
- Egyszerűbb telepítési, konfigurálási lépéseket elvégezése (főleg a gyakorlatok anyaga)
- Kommunikációs képesség kialakítása a
  - speciális szakemberekkel (a probléma pontos megfogalmazása, a szakkifejezések helyes használata, a hiba lehetséges okainak behatárolása) és a
  - felhasználókkal (mit lehet elvárni az operációs rendszertől és mi számít hibának). (az informatikus szakma interfész jellegű!)



## *Cél 2. → Módszer → Eredmény*

**Általános célok („mérnök")**
- Mérnöki precizitás és koncentráció a precizitás a számítási feladatoknál.
  - Megértés << Megtanulás << Alkalmazás !!!
- Sarokpontokból (axiomákból) kiinduló, illetve ezekre visszavezető analizáló/szintetizáló gondolkodásmód.
- Egészséges kételkedés (szkepticizmus)
  - Egy állítást csak akkor fogadjunk el, ha az tudományos alapokból kiindulva tudományos módszerekkel levezethető
  - (Ha ezt nem tudjuk megtenni, attól még az állítás lehet igaz, de kételkedni kell. Egy tekintély (tanár stb.) szava önmagában nem elég!)



## *Cél 3. → Módszer → Eredmény*

**Általános célok („vezető", „menedzser")**
- Az elvégzendő tevékenységek és a rendelkezésre álló erőforrások összehangolása az eredményesség és hatékonyság érdekében.
- Nincs jó és rossz. Az adott körülmények között optimumot keresünk.
- Egyensúly a szervezetlenség és a túlszabályozás között
- Erőforrások maximális kihasználása (erőforrásból mindig kevés van!)
- Gazdálkodás az idővel (Tevékenységek ütemezése)
- Kockázatok kezelése (Hatékonyság vs. Biztonság)



## *Cél → Módszer 1. → Eredmény*

**Megközelítés**
- Kívülről haladunk befelé (a felhasználói felülettől a rendszermag komponensei felé), mert
  - Taktikai ok: Így jobban szinkronban lehetünk a párhuzamosan folyó gyakorlatokkal
  - Stratégiai ok: Az operációs rendszerek a feladatok egyre nagyobb hányadát vállalják ugyan magukra, de egyre jobban „elrejtőznek", háttérbe vonulnak, így a felhasználók többsége inkább csak a felülettel találkozik.
- Az Elmélet és a Gyakorlat adminisztratív szempontból független

## *Cél → Módszer 2. → Eredmény*

**Esettanulmányok**

- Az elméletben tanultakat gyakorlati példákkal, elsősorban a Linux ismertetésével illusztráljuk
  - Az operációs rendszerek belső felépítése a 70-es évek közepén, a Unix megjelenésével kialakult
  - A Linux gyorsan terjedő, nyílt operációs rendszer, így belső felépítéséről is sokat lehet tudni.
- Célunk – többek között, de nem utolsó sorban – az, hogy bemutassuk, hogy
  - A gyakorlat mindig bonyolultabb, összetettebb
  - A hatékonyság és biztonság között kompromisszumot kell kötni.



## *Cél → Módszer 3. → Eredmény*

**Számítási feladatok**

- A fontosabb algoritmusokat számítási példák segítségével is feldolgozzuk
- A feladatok a számonkérés meghatározó részét képezik
  - Mélyebb tudást követelnek, precizitásra ösztökélnek
  - Felmérhető, hogy a hallgatók mennyire képesek alkalmazni a tanultakat
  - Az alternatív algoritmusok mennyiségi összevetésére is alkalmasak



## *Cél → Módszer 4. → Eredmény*

**Számonkérés**

- A második előadástól zárthelyik (összesen 3 db)
  - 1. Öt, néhány mondatban megválaszolandó kérdés
    - Az egyik kérdés gondolkodtató, nem mechanikusan megválaszolható
  - 2. Három kérdés + Kettő számítási feladat
  - 3. Kettő kérdés + Kettő számítási feladat
- Ötven százalékos eredmény felett megajánlott jegy!
- A zárthelyik egyre nehezebbek!
- A konzultációkon tárgyalt anyagrészeket a ZH kérdések alapján foglaljuk össze.



## *Cél → Módszer → Eredmény*

**Kommunikációs készség kialakítása (specialisták, felhasználók, döntéshozók felé),**

**A szakkifejezések ismerete és értelmezése**

**Az alkalmazáshoz szabott operációs rendszer kiválasztása, korlátainak felismerése és kezelése**

**Egyszerűbb paraméterezési eljárások**

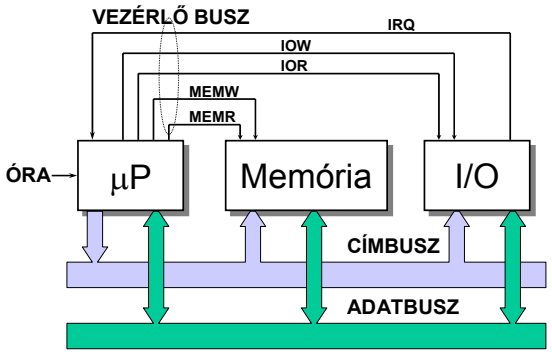
**„Mérnökibb” gondolkodásmód**



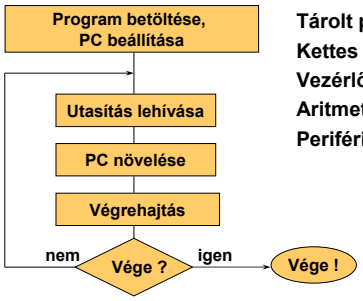
## *Bevezetés*

- **A számítógépek felépítése (ismétlés)**
- **Múlt, jelen, jövő**
- **Alapfogalmak**
  - **Folyamatok, Erőforrások**
  - **Az operációs rendszerek felépítése, feladata**





## Neumann-ciklus, elvek




**Tárolt program**
**Kettes számrendszer**
**Vezérlőegység**
**Aritmetikai-Logikai egység**
**Perifériák**



## Periféria – Memória átvitel módjai

**Lekérdezéses módszer (polling)**
- A CPU a periféria egy adott port-ját folyamatosan olvassa, ha változást észlel, az új adatot elhelyezi a memóriában
- Könnyű programozni, de a CPU kihasználás siralmas

**Megszakításos (interrupt)**
- A periféria elég intelligens ahhoz, hogy szóljon, ha kiszolgálási igénye van (megszakítást kér)
- A CPU csak az átvitel idejére foglalt
- Nehezebb programozni

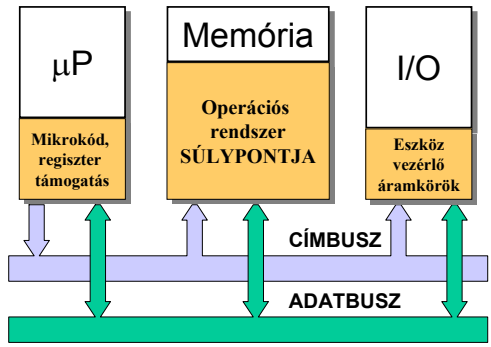
**Közvetlen memória átvitel (DMA)**
- Megszakítással indul
- A CPU felprogramozza a DMA vezérlőt (honnan, hova, mennyit)
- A DMA vezérlő önállóan intézi az átvitelt
- Nagy mennyiségű adatnál célszerű, kicsit nehezebb programozni



## Periféria – Memória átvitel összehasonlítása

| | Kezdeményező | Végrehajtó |
|---|---|---|
| Polling | CPU | CPU |
| Interrupt | Periféria | CPU |
| DMA | Periféria | DMA |



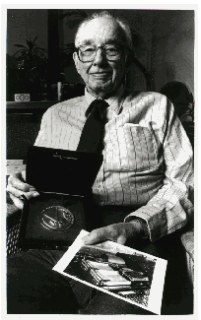





## A logikai modell

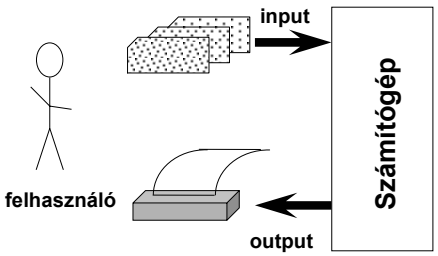
**Alkalmazások (Word, Excel)**
**Magas szintű nyelvek (Pascal, C)**
**Alacsony szintű nyelvek (Assembly)**
**Operációs rendszer**
**Hardver (Memória, busz, perifériák)**
**CPU (mikroprogram, regiszterek)**
**Logikai áramkörök (kapuk, összeadó)**





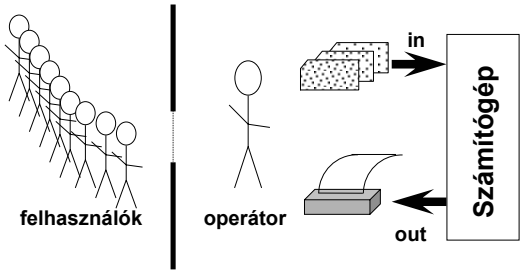
*John Vincent Atanasoff (1903-1995)*

## „Open shop”



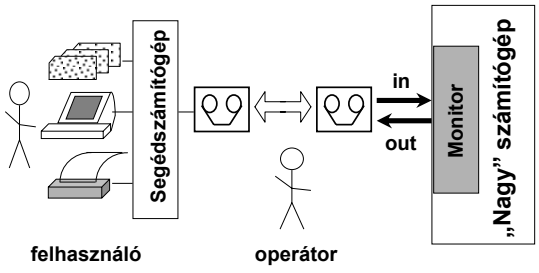



## „Closed shop”






## Mágnesszalag alkalmazása



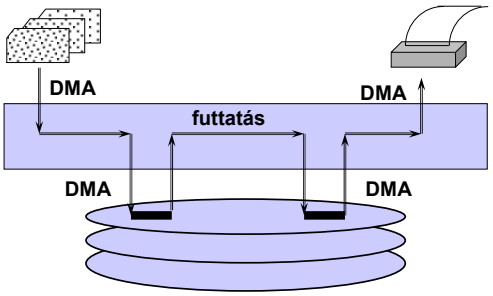



## Parancsnyelv (pl. JCL)

```
$ JOB OK    # a munka neve
$ FTN       # a FORTRAN fordító indítása
-
-           # FORTRAN programsorok
-
$ LOAD      # a lefordított program betöltése
$ RUN         futtatás
-
-           # a program bemenő adatai
-
$ END       # a munka vége
```

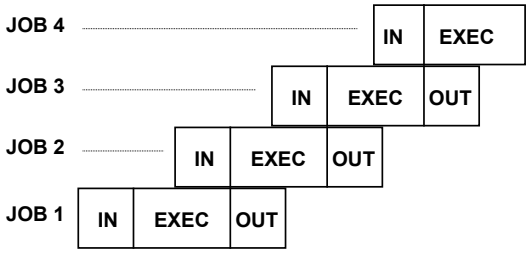


## „Spooling” rendszer

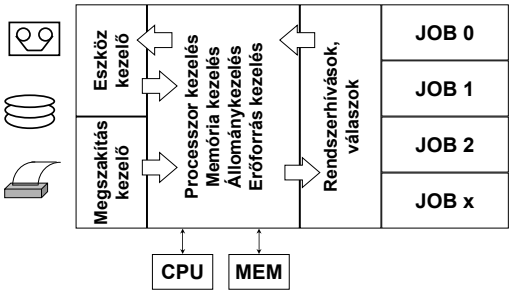





## Átlapoló rendszer (pipelined)

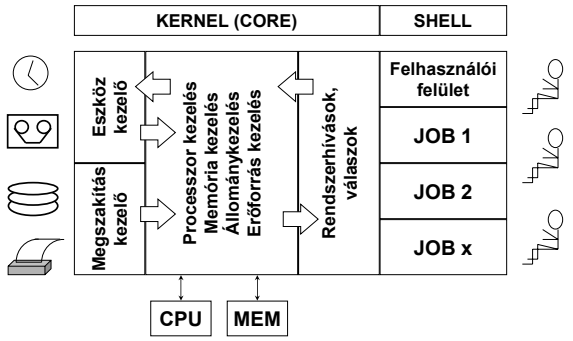


## Multiprogramozás






## Időosztásos, interaktív rendszerek






## Operációs rendszerek feladatai

**Eszközkezelők**
**Megszakítás kezelő**
**Rendszerhívások, válaszok**
**Erőforrás kezelő**
**Memóriakezelő**
**Állomány- és lemezkezelés**
**Felhasználói felület**



## Az interaktív rendszerek új szempontjai

**„Emberi” válaszidő**
**Időosztás (time sharing)**
**Felhasználói felület**
- Kiegészített parancsnyelv
- Barátságos felület

**Felhasználói adminisztráció**



## Feldolgozási módok

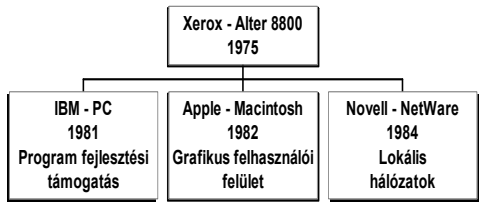
**Kötegelt**
- Legfontosabb a hardver kihasználása
- Minimális adminisztráció
- Az ütemezést a folyamatok maguk vezérlik (azaz környezetváltás a befejezéskor vagy olyan erőforrás igénylésénél, amely éppen nem érhető el – kb. kooperatív multitasking)

**Időosztásos**
- A felhasználók a fontosak (rövid válaszidő)
- Interaktív rendszerek (preemptív multitasking)
- Biztonsági és kommunikációs probklémák

**Valósidejű**
- Véges időkorláton belül kiszolgál max. terhelés esetén is
- Folyamatvezérlés
- Esemény a kezdeményező

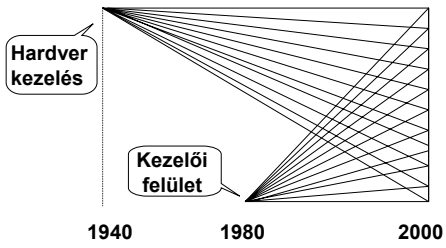


## Személyi számítógépek



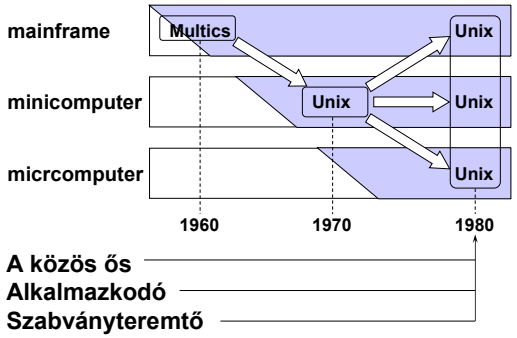

**Legfontosabb a felhasználó !**

## Tendencia






## A Unix jelentősége




A közös ős
Alkalmazkodó
Szabványteremtő



## Többprocesszoros rendszerek

Nagyobb átbocsátó képesség
Erőforrás megtakarítás
Megbízhatóság növekedése
(Fault tolerant system)

Szimmetrikus (SMP)
Aszimmetrikus

A hardver és szoftver együttműködése szükséges!



## Elosztott rendszerek (hálózatok)

Rugalmasság
Erőforrásmegosztás
Sebességnövekedés
Megbízhatóság
Kommunikáció

DE ! Biztonsági kérdések, hozzáférés szabályozása



## Folyamatok (task, process)

1. „Életre kelt” programok
2. Olyan programok, amelyeknek van folyamatleíró táblája



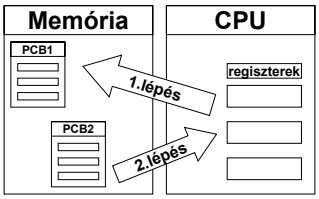
## Folyamatleíró blokk (PCB, TSS)

Folyamat azonosítója
Programszámláló állása
Folyamat állapota
Regiszterek tartalma
Memóriaterület adatai
Perifériák, állományok állapota

## Környezetváltás (context switching)

**A CPU egy másik folyamat/szál utasításainak végrehajtásába kezd**

**Átkapcsoláskor el kell menteni minden olyan adatot, amely az éppen félbeszakított folyamat folytatásához szükséges ⇨ PCB**






## Szálak és Folyamatok

**Folyamatok**

- Környezetváltáskor minden paraméter mentendő a PCB-be
- Saját erőforrásokkal rendelkezik, biztonságos
- Szálakat hoz(hat) létre

**Szálak**

- Gyors (környezetváltáskor minimális adatot kell menteni – PC, regiszterek)
- Veszélyes, mert nincs saját erőforrása (közösen használják a szülő folyamat erőforrásait)
- Csak jól tesztelt, biztonságos környezetben!



## Erőforrások

**Minden, ami egy folyamat végre-hajtásához szükséges**
(memória, processzor,perifériák állományok stb.)



## Erőforrások típusai

| Megszakítható (preemptív) | Nem megszakítható (non-preemptív) |
|---|---|
| **A folyamatoktól az erőforrás a folyamat vagy erőforrás károsodása nélkül elvehető** | **Az erőforrás használat félbeszakítása esetén a folyamat vagy erőforrás sérülhet!** |



## Operációs rendszer (Erőforrás szemlélet)

**A folyamatok egy olyan csoportja, amely a felhasználói folyamatok között elosztja az erőforrásokat**



## Operációs rendszer (Felhasználói szemlélet)

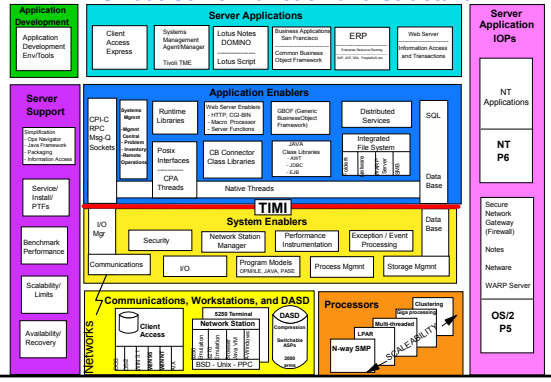
**A folyamatok egy olyan csoportja, amely megkíméli a felhasználókat a hardver kezelés nehézségeitől, kellemesebb alkalmazói környezetet biztosít**

# Az operációs rendszer szerkezete

| | |
|---|---|
| Felhasználói programok | |
| Program készítési támogatás | |
| Felhasználói folyamatok kiszolgálása | |
| Rendszerhívások | Válaszok |
| Rendszermag (KERNEL) | Processzorkezelés |
| | Memóriakezelés |
| | Állománykezelés |
| Eszközmeghajtók | Megszakítás kezelés |
| Hardver | |



## AS/400e server Function and Structure






## Megszakítások fajtái

**Megszakítás (Interrupt)**
- A perifériák jelzése a processzor számára (pl adatátvitel vége)

**Kivételek (Exception)**
- A processzorműveletek során keletkező hibák esetén (pl rossz címszámítás, osztás 0-val)

**Nem maszkolható megszakítások (NMI)**
- súlyos hardver (pl RAM, tápfeszültség) hiba

**Csapda (Szoftver megszakítás, Trap)**
- a felhasználói folyamatoktól érkező rendszerhívások



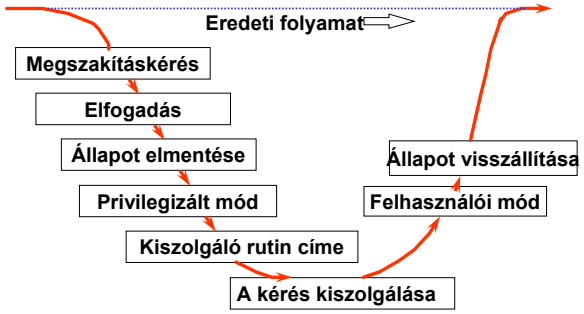
## Operációs rendszerek minőségi követelményei

| Hatékonyság | Biztonság |
|---|---|
| – Sebesség | – Hozzáférés védelem |
| – Felhasználó barátság | – Stabilitás |
| – Könnyű fejleszthetőség | – Megbízhatóság |
| – Skálázhatóság | – Hibatűrés |
| – Menedzselhetőség | – Bizalmasság |
| – Sok funkció | – Naplózás |
| – Rugalmasság | |

**Ellentétes követelmények!**
**(azonos költség esetén)**

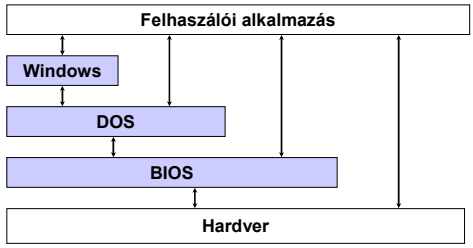


## Megszakítás-kezelés lépései

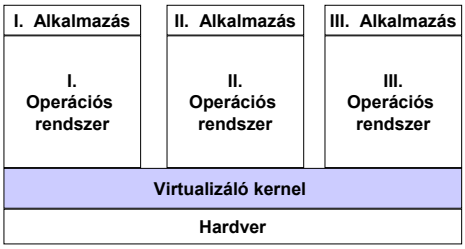





## Példa: DOS+Windows

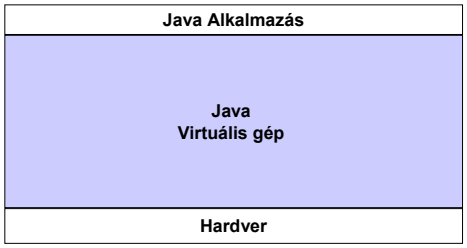


## Virtuális gépek: Az IBM VM






## Virtuális gépek: JAVA






## Informatikai rendszerek jellege

**Jelenleg digitális számítógépeken alapulnak**

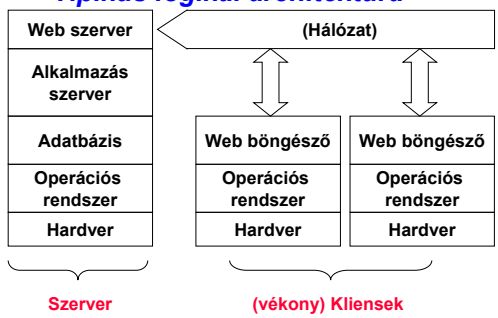
- (Nem törvényszerű, de most ez a legolcsóbb, leghatékonyabb. Vannak azonban olyan feladatok, amelyek más módon oldhatók meg kedvezőbben.)

**Többfeladatos, elosztott (hálózati) rendszerek**

- Leggyakrabban kliens-szerver architektúra
- A hardver erősödésével a maximális hatékonyság helyett a kényelem és a biztonsági kérdések fontossága nő. (komponens alapú fejlesztés)
- A kliens egyre inkább „vékony”, tehát a kliensen csak egy böngésző fut, az alkalmazás és az adatok szerver oldalon vannak

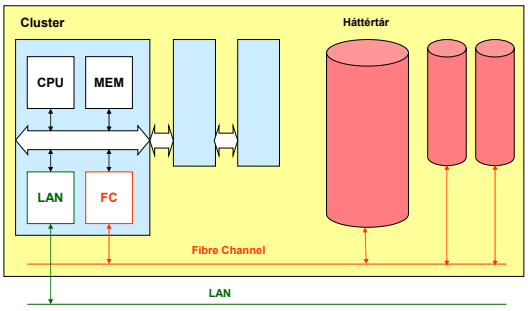


## Tipikus logikai architektúra






## Korszerű szerver architektúra






## Összefoglalás

**Folyamatok, PCB fogalma**
**Erőforrás fogalma, alaptípusai**
**Operációs rendszerek funkciói**
**Kommunikáció:**

- Megszakítások
- Rendszerhívások

## A felhasználói felület

- A felhasználó és a rendszermag
- A programozói felület
- Segédprogramok, alrendszerek
- Jellemzők



## Felhasználói szintek (szempontok)

**Végfelhasználó**
- Grafikus felület
- Hatékonyság, Biztonság
- Egyre több funkció a háttérben történjen (automatikusan)

**Fejlesztő (rendszer vagy szoftver)**
- Komponens alapú alkalmazásfejlesztés
- Magas szintű formalizmus
- Objektumorientált megközelítés

**Üzemeltető (rendszer menedzser)**
- Hibatűrés
- Központi menedzsment



## Fejlesztői szerepek

**Az informatikai szakmák hasadnak, specializálódnak**

**access provider** (hozzáférést biztosító - ISP, ASP)
- szakmunkások: digitalizáló, operátor, rendszergazda

**content provider** (tartalom szolgáltató - infolaikus)
- újságíró, esztéta, nyelvész, jogász

**content builder** (alkalmazásfejlesztő - )
- látványtervező, interaktív fejlesztő, forgatókönyv író stb.
- programfejlesztő, adatmodellező, adatbázis tervező stb.



## A felhasználói felület részei

**A programindítás eszközei**
**A rendszermag szolgáltatásai**
**A programozói felület**
**Alapvető segédprogramok**



## A perifériák beállítási lehetőségei

| | |
|---|---|
| **„Kézi” beállítások** | **Config.sys** |
| **Automatikus beállítások** | **Plug and Play** |
| **Félautomata beállítások** | **Scan for Devices** |



## A memória beállítási lehetőségei

**Bufferek száma, mérete (hálózat, lemez)**
**Leíró táblák száma (pl. FCB)**
**Lemezgyorsító tár (cache)**
**Felhasználási módok (EMS, HMA,XMS)**

## A programozói felület

**API (Application Programming Interface)**
**SDK (Software Development Kit)**

**Rendszerhívások = Függvények**
**DOS - kb. 100 alapvető függvény**
**Windows - kb. 1000 magasszintű fv.**

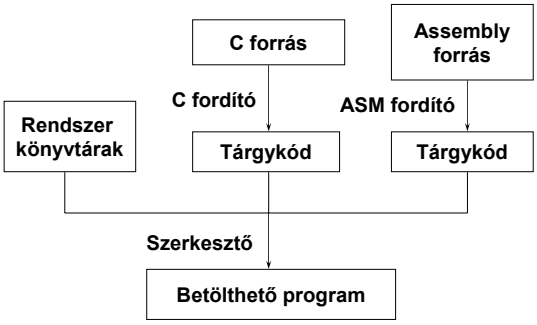


## Programkészítés

**Forráskód (algoritmus)**
**Tárgykód (könyvtárak)**
**Betölthető kód (program)**



## Programok készítése






## Strukturált programfejlesztő környezet

**Editor (ASCII szövegszerkesztő)**
- Forráskód szerkesztése

**Compiler(ek) (Fordító)**
- Tárgykód előállítása

**Linker (Kapcsolatszerkesztő)**
- Címek összehangolása

**Loader (Betöltő)**
- Memóriába töltés (ha az OS loader-ét használnánk, nem lenne lehetséges a nyomkövetés, hiszen az OS maga védené az új folyamatot)

**Debugger (Hibakereső)**

**A DOS-nál még Az Operációs rendszer részei !**



## Parancsértelmező

**Elnevezések**
**(A lényegesnek tartott funkció szerint)**
- Shell (DOS-Shell, Bourne-shell)
- Command interpreter (command.com)
- Monitor (Novell)
- Manager (Program, Fájl, Nyomtató)



## A parancsértelmező feladatai

**Programindítás**
**Fájlkezelés**
**Programkörnyezet beállítása**
**Folyamatok futásának ellenőrzése**
**Vezérlési szerkezetek**

## Programindítás

**Közvetlen indítás (név beírása, load, run)**
**Közvetett elérés**
**Keresési útvonalak**
**Láncolt (kötegelt) futtatás (*.bat, *.ncf)**
**Automatikus programbetöltés**



## A program környezete

**Paraméterek**
**Kapcsolók**
**Átirányítási adatok**
**Környezeti változók**



## A program környezete

**Az alkalmazás az operációs rendszer egy „gyerek” folyamatának tekinthető**

- Kicsit hasonlít a helyzet a szubrutin híváshoz (de ezek a „szubrutin-folyamatok” párhuzamosan futnak!!!)
- Paramétereket kell átadni neki, ezek alkotják a program környezetét (kb. a parancsikon tulajdonságai)

**A program környezete**

- Globális változók: keresési útvonal, parancssor, OS változók, munkaterület
- Aktuális paraméterek:
  - Paraméterek (a művelet „tárgya”)
  - Kapcsolók (a művelet „jelzői”)
  - Átirányítások (a művelet „helyhatározója”)
  - (az állítmány maga a program, az alany az operációs rendszer)
  - (Kedves operációs rendszer,) másold át az x.y fájl tartalmát binárisan a képernyőre!
  - COPY X.Y /b >>CON



## Alrendszerek

**Állománykezelő**
**Programfejlesztői**
**Adatbázis kezelő**
**Kommunikációs**

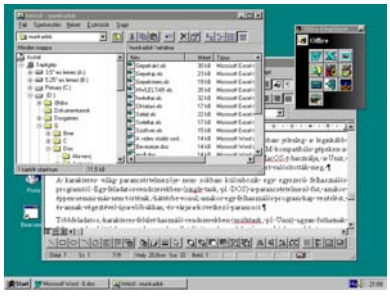


## Felhasználói felület jellemzői

**Könnyű legyen megtanulni**
**Méretezhető legyen**
**Lehessen visszavonni**
**Törölni lehessen a műveleteket**
**Többszintű súgó rendszer**
**Hasonlítson az élő nyelvhez**
**Minden parancsra legyen válasz**
**Hasonló funkciók hasonló módon**



## A Windows95 néhány ablaka



## Ablakozó rendszerek

**Követelmények**

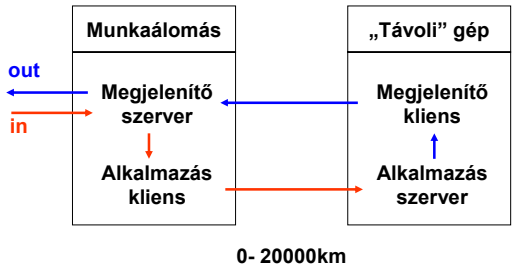
- A multitask környezet miatt megoldandó az események címzettjeinek felismerése (aktív ablak, fókusz)
- Eszközfüggetlen működés biztosítandó
- Az adatforgalom a kliens és szerver között csökkentendő

**Megoldás**

- Szerver-kliens architektúra
- X-szerver, X-kliens, X.11, X-window

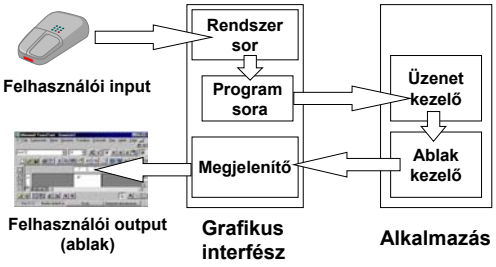


## X-window architektúra






## Üzenetvezérlés






## Eszközfüggetlen működés

- A megjelenítő szerver az alkalmazás szabványos, eszközfüggetlen üzenetei alapján állítja össze a megjelenítendő képet
- Az alkalmazásnak nem kell ismernie a munkaállomás perifériáit, a megjelenítésről a helyi szerver gondoskodik (GUI)
- A GUI az alapvető objektumokat maga állítja elő az alkalmazástól kapott paraméterek alapján

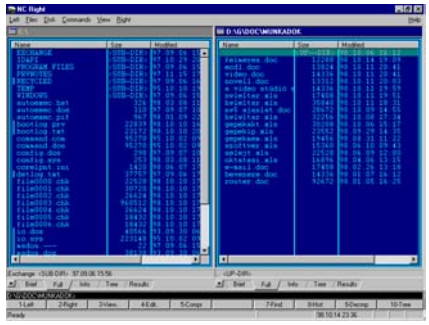


## Adatforgalom csökkentése

- Nem mozognak az átviteli csatornán nagyméretű képernyőtartalmak, csak az ezek összeállítására vonatkozó utasítások
- A GUI az alapvető objektumokat maga állítja elő az alkalmazástól kapott paraméterek alapján
- Az ideiglenesen nem használt felületek tartalma tárolódik az X-szerveren
- Lehetőség szerint csak a változásokra vonatkozó utasítások mozogjanak



## A Norton Commander



## Összefoglalás

**Felhasználói felületek**
- célja
- beállítási lehetőségek
- programindítás

**Programkészítés lépései**

**Grafikus, üzenetvezérelt felületek**



## Fájlok, katalógusok

- **Fájlnevek, jellemzők**
- **Közvetett hivatkozások**
- **Katalógus szerkezetek**
- **Hozzáférési jogok**
- **Fájlok elhelyezése**
- **Műveletek állományokkal, katalógusokkal**



## File = Állomány

**Adatok egy olyan csoportja,
melyre együttesen,
egy névvel hivatkozhatunk**



## Fájltípusok

**Ideiglenes rendszerállományok (pl. virtuális memória)**

**Adminisztratív állományok (pl. katalógusok)**

**Felhasználói állományok**



## Tipikus fájlnevek

| | |
|---|---|
| **DOS** | **EZEREGY.DOC** |
| **UNIX** | **Az.Ezeregy.Ejszaka.Mesei.DOC** |
| **Win 95 (hosszú)** | **Az ezeregy éjszaka meséi.doc** |
| **Win 95 (rövid)** | **AZEZER~1.DOC** |
| **VMS** | **EZEREGY.DOC;4** |



## Helyettesítő karakterek

| | JÓ | NEM JÓ |
|---|---|---|
| **LEVI?.TXT** | **LEVI1.TXT**<br>**LEVI2.TXT** | **LEVI10.TXT** |
| ***NI.DOC** | **DANI.DOC**<br>**ZOKNI.DOC** | **DANO.DOC** |
| **[TD]OBOZ**<br>***(unix!)*** | **TOBOZ**<br>**DOBOZ** | **KOBOZ**<br>**doboz** |

## Fájl jellemzők

Fáljnév
Méret
Utolsó módosítás ideje
Attribútumok
[Hozzáférési jogok]
Fizikai elhelyezkedés



## Jogosultságok típusai

Olvasás (Read - R)
Írás (Write - W)
Létrehozás (Create - C)
Végrehajtás (eXecute - X)
Törlés (Erase - E)
Jellemzők módosítása (Modify - M)
Hozzáférés módosítása (Access control - A)



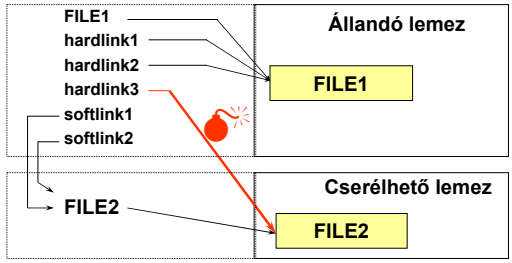
## *Katalógus = Directory*

**(“*könyvtár*”)**

**Olyan speciális állomány, melynek tartalma a fájlok nevét és jellemzőit tartalmazó rekordok listája**

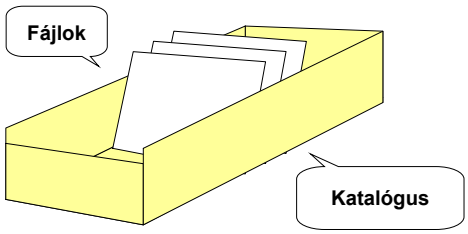


## Láncolás



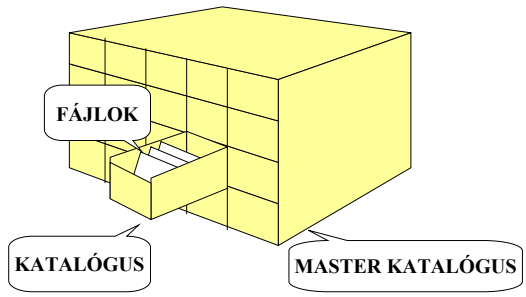



## Egyszintű katalógus

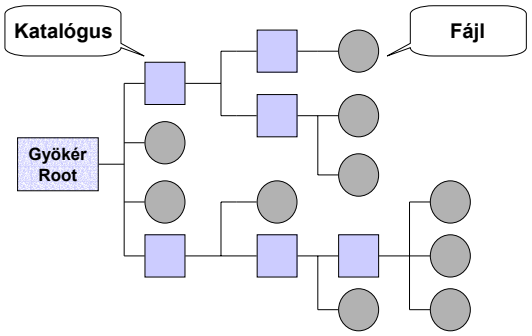





## Kétszintű katalógus



## Fa struktúra

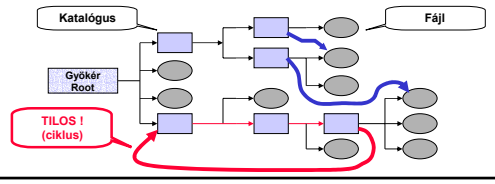





## Aciklikus gráf

**A hierarchikus katalógus struktúra általánosítása**
**Hasonló a közvetett file eléréshez (kb. parancsikon)**
**Rugalmasabbá teszi a szervezést, de bonyolítja fájl rendszert**
**Többszempontú csoportosítást tesz lehetővé**
- Alkalmazások, adatbázisok szintjén már megjelent
- Lotus notes, Media 360

**Körbejárható útvonal értelmetlen, kialakulása zavart okoz, így nem megengedett**





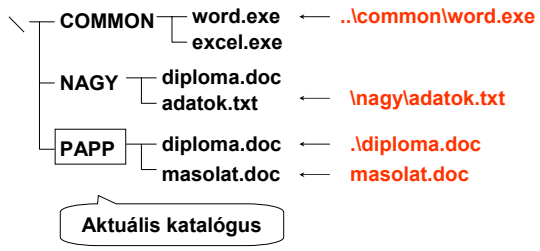

## Címzés a fájlrendszerben

Abszolút címzés
- **kiindulópont a gyökér, a cím '/'-rel kezdődik**

Relatív címzés
- **kiindulópont az aktuális katalógus**
- **'..' a szülő, '.' az aktuális katalógus neve**



## Példák fájl elérésekre



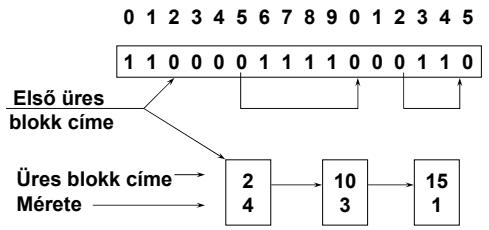



## Fájlok elhelyezkedése

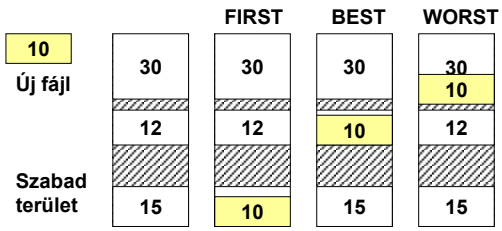
**Szabad helyek nyilvántartása**
**Folytonos: first, worst, best**
**Láncolt: FAT**
**Indexelt: INODE**



## Szabad helyek nyilvántartása

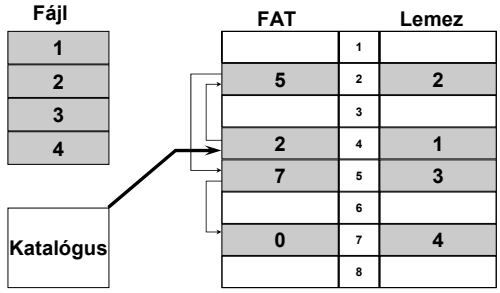


## Folytonos elhelyezés



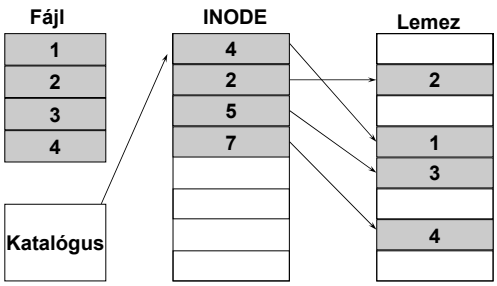



## Láncolt elhelyezés



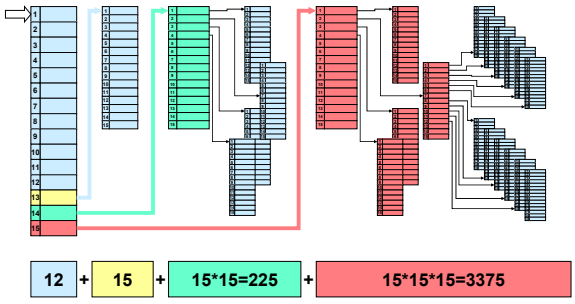



## Indextábla alkalmazása






## Az i-node tábla alkalmazása nagy fájloknál




Linuxnál a magasabb szintű táblák 256 eleműek! (15 ⇨256)



## Műveletek állományokkal

**Létrehozás**
**Keresés katalógusban**
**Írás, olvasás**
- **megnyitás**
- **pozicionálás**
- **írás-olvasás**
- **lezárás**

**Törlés**



## PCB és FCB kapcsolata

**PCB (Process Control Block)**
- A folyamat folytatásához szükséges adatok

**FCB (File Control Block)**
- A fájl művelet folytatásához szükséges adatok
- Tartalma
  - Logikai azonosító
  - Aktuális pozíció (hol tartok az olvasással?)
  - Megnyitási mód (írás, olvasás, bináris, szöveg)
  - Zárolási adatok (ne piszkáljon bele senki!)
  - Változás jelző stb. (el kell menteni, mert megváltozott a tartalom!)

**A PCB tartalmazza az folyamat által használt fájlok FCB-ire mutató pointereket is !**

## Fájlrendszerek jövője (vízió!)

**Jelen**

- Katalógus (leíró információ) ⇨ Fájl (tartalom)
- A katalógus célja a kereshetőség, rendszerezés
- Kereshetőségi szempontok: Név, Idő, Méret
  - KEVÉS, többszempontú kereshetőség kell
  - A katalógos elemeinek a bővítése operációs rendszer szinten csak időleges megoldás
  - Egyéni bővíthetőség esetén elveszik a hordozhatóság

**Jövő (megoldás)**

- ÖNLEÍRÓ állományok, azaz a rendszer a struktúráját SZABVÁNYOS módon hordozza (XML)
- Ez nem más, mint az ADATBÁZIS (relációs, OO)!
- A fájlrendszerek ELTŰNNEK, vagy legalábbis az adatbázis koncepció felé tartanak.



## Összefoglalás

**Fájlok szerepe, jellemzői**

**Elnevezések,**

- célja
- beállítási lehetőségek
- programindítás

**Programkészítés lépései**

**Grafikus, üzenetvezérelt felületek**

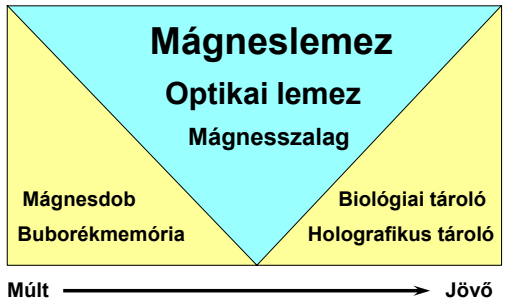


## Lemezkezelés

- **Háttértárolók felépítése**
  - Szalag, Lemez, CD
- **Fizikai lemezkezelés**
  - Ütemezés, Címszámítás, Adatátvitel
- **Az adattárolás optimalizálása**
  - Blokkméret
  - Tömörítés, redundancia

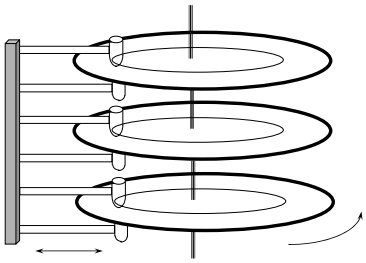


## Háttértárak



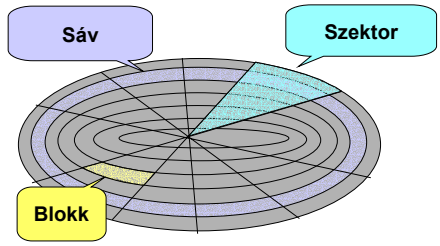



## Merevlemezek felépítése





## Lemezfelületek felosztása



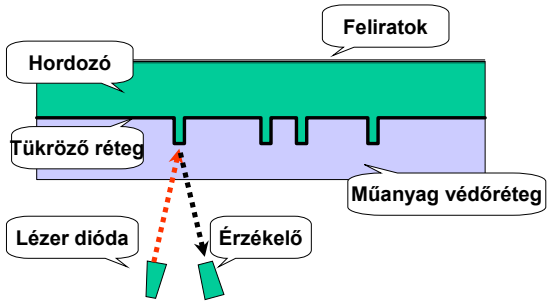
## Mágneslemezek jellemzői

Technikai jellemzői
- **Kapacitás: 1-10GB**
- **Elérési idő: 10 ms**
- **Adatátviteli sebesség: 2-10 Mb/s**

Alkalmazása
- **Virtuális memória**
- **Programok tárolása**
- **Adatok tárolása**



## Optikai lemezek felépítése





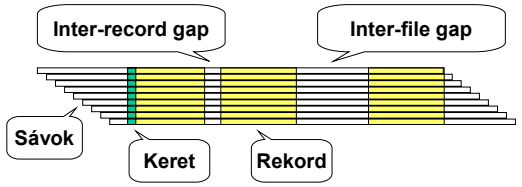

## Optikai lemezek jellemzői

Technikai jellemzői
- **Kapacitás: 650MB-14GB**
- **Elérési idő: 300 ms-tól**
- **Adatátviteli sebesség: 150 kb/s-tól**

Alkalmazása
- **Archiválás**
- **Program kereskedelem**
- **Nagy adatbázisok, lexikonok**



## Mágnesszalagok felépítése




**Kapacitás: 2-16 GB**
**Keresési idő: 0-5 perc**
**Adatátviteli idő: 2-10 Mb/sec**
**Újra felhasználható !**



## Mágnesszalagok jellemzői

Technikai jellemzői
- **Kapacitás: 2-16 GB**
- **Keresési idő: 0-5 perc**
- **Adatátviteli idő: 2-10 Mb/sec**
- **Újra felhasználható !**

Alkalmazása
- **Nagy mennyiségű, összefüggő adat**
- **Archiválás, adatmentés**
- **Nagy tömegű adat átvitele**
- **Sok adat átmeneti tárolása**



## Háttértárak összehasonlítása

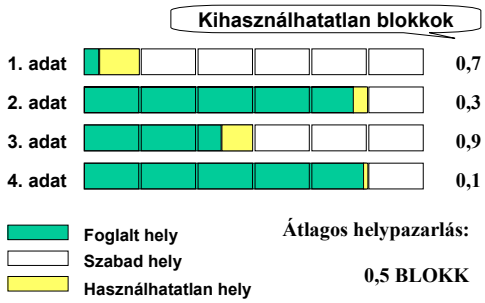
| | Kapacitás | Elérés | Átvitel | Ár/MB |
|---|---|---|---|---|
| HDD-IDE | 2..72 GB | 8 ms | 100 MB/s | 2000 Ft/GB |
| HDD-SCSI | 9..72 GB | 5 ms | 80 MB/s | 8000 Ft/GB |
| HDD-FC | 30..72 GB | 10 ms | 400 MB/s | 3000 Ft/GB |
| CD-ROM | 650 MB | 100 ms | 6 MB/s | 300 Ft/GB |
| CD-RW | 650 MB | 100 ms | 6 MB/s | 500 Ft/GB |
| DVD-ROM | 8/16 GB | 100 ns | 16 Mbps | n.a. |
| DVD-RAM | 2,5/5 GB | 10 ms | 2 MB/s | 2000 Ft/GB |
| Streamer | 4..20 GB | n.a. | 1 MB/s | 1000 Ft/GB |
| DAT | 2 GB | n.a. | 2 MB/s | 200 Ft/GB |
| DLT | 20/40 GB | n.a. | 10 MB/s | 1200 Ft/GB |

## Mágneslemezek

Blokkméret
Tömörítés
Adatvédelem



## Helykihasználás




Átlagos helypazarlás:

0,5 BLOKK



## Foglaltsági tábla

**A lemez kapacitása 1 GB = $2^{30}$ bájt**

| Blokkméret | Foglaltsági tábla mérete | |
|---|---|---|
| 512 = $2^9$ bájt | $2^{30}/2^9/2^3 = 2^{18}$ | 256 kB |
| 2048 = $2^{11}$ bájt | $2^{30}/2^{11}/2^3 = 2^{16}$ | 64 kB |
| 64k = $2^{16}$ bájt | $2^{30}/2^{16}/2^3 = 2^{11}$ | 2 kB |



## Blokkok elhelyezkedése

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. lemez | 1 | 21 | 5 | 25 | 9 | 29 | 13 | 33 | 17 |
| 2. lemez | 2 | 22 | 6 | 26 | 10 | 30 | 14 | 34 | 18 |
| 3. lemez | 3 | 23 | 7 | 27 | 11 | 31 | 15 | 35 | 19 |
| 4. lemez | 4 | 24 | 8 | 28 | 12 | 32 | 16 | 36 | 20 |

**4 lemez, 9 szektor, 1:2 interleave**



## A beolvasott adatok továbbítása

**Beolvasási sebesség**
- A mágnesesen rögzített jelek megtalálása
- Mechanikai korlátok (aláfordulási idő, fejmozgás)

**Átviteli sebesség**
- Az adatok átvitele a memória felé
- Elektronikus (interfész, buszrendszer) korlátok

**Sebességkülönbség esetén**
- Beolvasási > Átviteli → Közbeékelő (interleave) technika
- Átviteli > Beolvasási → Szalag (stripping) technika
- Beolvasási ≠ Átviteli → Buffer (csak „burst” jelleggel)



## Tömörítési eljárások

**Kisebb helyfoglalás** ↔ **Nagyobb számításigény**
**Gyorsabb adatátvitel** ↔ **Kisebb adatbiztonság**

Futás hossz kódolás: Sok azonos karakter esetén
Különbségi kódolás: Lassan változó minta esetén
Huffmann-kódolás: Erősen eltérő gyakoriságú karakterek esetén

**Gyakorlatban**: **Microsoft Drive Space / Double Space**
**Novell NetWare Compressed Volume**

## Huffmann-kódolás

Eredeti szöveg: **KEREKES SZEKEREK MENNEK**

| Statisztika, kódolás: | |
|---|---|
| 8 db E | 00 |
| 5 db K | 01 |
| 2 db R | 10 |
| 2 db S | 1100 |
| 2 db N | 1101 |
| 2 db space | 1110 |
| 1 db M | 11110000 |
| 1 db Z | 11110001 |

**Hatékonyság:**

**Eredeti szöveg**:
184 bit

**Kódolt szöveg**
70 bit



## Adatbiztonságot javító módszerek

Adatszintű védelem
- **paritásbit - egyetlen bithiba**
- **hibajavító kód - független hibák**
- **CRC - összefüggő hibák**

Eszközszintű védelem
- **lemeztükrözés - lemez megkettőzése**
- **RAID - adatok redundáns elosztása**



## Az adatátvitelhez szükséges adatok

Eszköz típusa
Eszköz sorszáma
Adat kezdőcíme az eszközön
Adat kezdőcíme a memóriában
Adat mennyisége
Átvitel iránya: Írás vagy olvasás
Visszatérési folyamat



## A lemez által várt adatok

Fej sorszáma
Szektor sorszáma
Cilinder sorszáma
Adat kezdőcíme a memóriában
Adat mennyisége
Átvitel iránya: Írás vagy olvasás



## Címtranszformáció

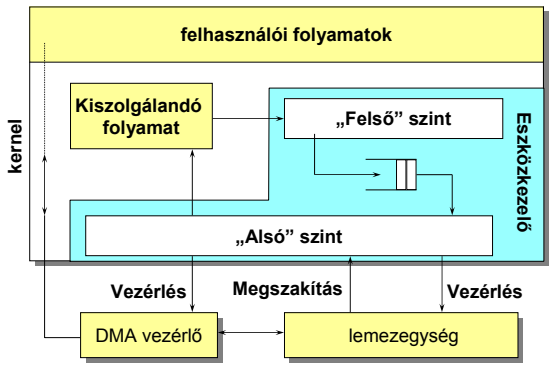
Blokk logikai címe ⇨
- h - Fej sorszáma
- s - Szektor sorszáma
- c - Cilinder sorszáma

**b = h * C* S + c * S + s**

**C - cilinderek száma**
**S - szektorok száma**

**Ennek az inverze kell !!!**





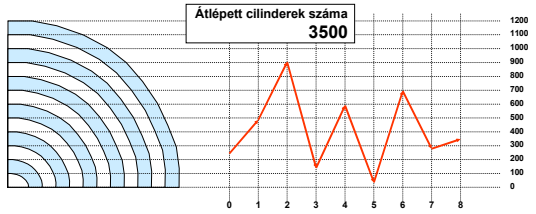
## Lemezütemezési algoritmusok

| Algoritmus | | Várakozási idő | Várakozási idő szórása |
|---|---|---|---|
| Sorrendi | FCFS | nagy | kicsi |
| Legrövidebb idejű | SSTF | kicsi | nagy |
| Pásztázó | SCAN | közepes | közepes |
| Egyirányú pásztázó | C-SCAN | közepes | kicsi |



## Lemezütemezés - FCFS

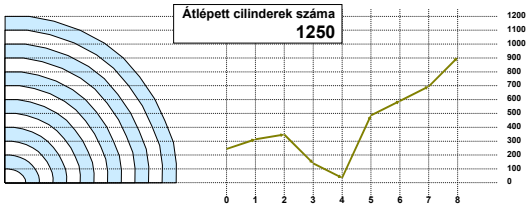
| Eredeti sor | 250 | 500 | 900 | 150 | 600 | 50 | 700 | 300 | 350 |
|---|---|---|---|---|---|---|---|---|---|
| Átrendezett | 250 | 500 | 900 | 150 | 600 | 50 | 700 | 300 | 350 |






## Lemezütemezés - SSTF

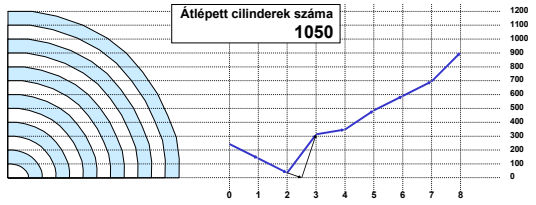
| Eredeti sor | 250 | 500 | 900 | 150 | 600 | 50 | 700 | 300 | 350 |
|---|---|---|---|---|---|---|---|---|---|
| Átrendezett | 250 | 300 | 350 | 150 | 50 | 500 | 600 | 700 | 900 |






## Lemezütemezés – SCAN (LOOK)

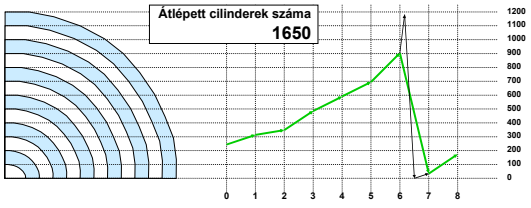
| Eredeti sor | 250 | 500 | 900 | 150 | 600 | 50 | 700 | 300 | 350 |
|---|---|---|---|---|---|---|---|---|---|
| Átrendezett | 250 | 150 | 50 | 300 | 350 | 500 | 600 | 700 | 900 |





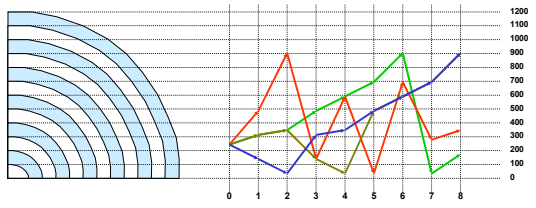

## Lemezütemezés – C-SCAN (C-LOOK)

| Eredeti sor | 250 | 500 | 900 | 150 | 600 | 50 | 700 | 300 | 350 |
|---|---|---|---|---|---|---|---|---|---|
| Átrendezett | 250 | 300 | 350 | 500 | 600 | 700 | 900 | 50 | 150 |






## Lemezütemezés – Összehasonlítás



## Adatátviteli módok

**Szinkron: kiszolgálás közben a folyamat várakozik**
**Aszinkron: a folyamat fut tovább**

**Lemezgyorsító (Disk cache)**

**Körkörös átmeneti tár (Buffer pool)**



## Szinkron és aszinkron átvitel

**Szinkron**

**A kérés (rendszerhívás) átadása után**

- A folyamat ***felfüggesztődik***,
- ***Elveszti*** erőforrásait (átvitel az OS területére)
- A felelősség a biztonságért az ***operációs rendszeré***

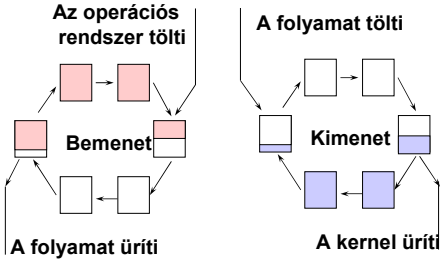
**Aszinkron**

**A kérés (rendszerhívás) átadása után**

- A folyamat ***tovább fut***,
- ***Megtartja*** erőforrásait (átvitel az FOLYAMAT területére történik)
- A biztonságért a ***folyamat*** a felelős



## Körkörös átmeneti tárolók






## Tárolórendszer követelményei

**Hagyományos szempontok**
**Kapacitás (tera/peta bájt nagyságrendű)**
**Adatátviteli sebesség (>100 Mbyte/sec)**
**Elérési idő (minimális, de nagyon változó)**

**Hagyományos, de egyre fontosabb szempontok**
**Megbízhatóság, rendelkezésre állás**
**Rugalmas konfigurálhatóság, bővíthetőség**



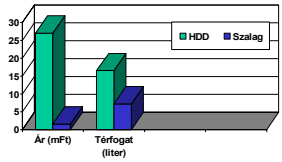
## Hagyományos rendszerek korlátai

**Korlátozott kapacitás**
**Rögzített konfiguráció, nem rugalmas**
**Mentés helyi hálózaton (LAN)**
**Meghatározott szállítók (nem nyílt rendszer)**
**Platform függőség**



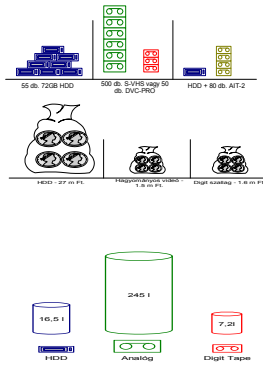
## Szalag vagy Lemez?

| 2 TB kapacitáshoz | Mennyiség (db) | Ár (mFt) | Térfogat (liter) | Kapac/Ár/Térf |
|---|---|---|---|---|
| HDD | 55 db | 27 mFt | 16,5 liter | 4,5 |
| Digitális szalag | 80 db | 1,6 mFt | 7,2 liter | 173 |

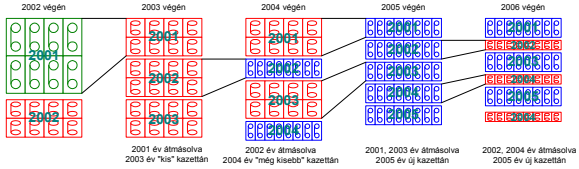



Mérőszám:
Kapacitás/Ár/Térfogat

**Egyéb szempontok: áramfelvétel, melegedés, sebesség**
**Nagy kapacitásigény gazdaságosan csak szalagos tárolóval !**








## *Hierarchikus tároló rendszer*

**HSM (Hierarchical Storage Management)**

**számítógép által „látott” tárterület megegyezik a szalagok összterületével.**

**Az elérési idő kedvező esetben megegyezik a mágneslemezek elérési idejével.**

**Gyorsítótár (cache) szükségessége**

**Szalagcserélési idő: 20-40 mp**

**A gyakran és/vagy nemrégiben használt anyagoknál nem megengedhető.**



## *A megbízhatóság növelése*

**A megbízhatóság növelésének módja:**

- **garantált minőségű (és ezért igen drága) elemek**
- **olcsóbb elemeket használunk ugyan, de az egyes összetevőket többszörözzük**

**Követelmények:**

- **Automatikus hibadetektálás**
- **Önjavítás (hot-swap)**



## *RAID*

**(Redundant Array of Inexpensive Disks)**

**RAID 1 - a lemeztükrözés**

- Minden adatból fizikailag kettő van
- Sérülés esetén az épen maradt lemez működik
- kétszeres háttértár kapacitást kell kiépíteni

**RAID 3 vagy RAID 5**

- A logikai adat blokkokat fizikailag több lemezegységen elosztva tároljuk,
- a szétosztott blokkokhoz ellenőrző összeget rendelünk, amelyet egy további fizikai eszközön tárolunk



## *RAID-3/5 példa*

**Tegyük föl, hogy van négy darab számunk (A, B, C, D), amit szeretnénk nagy biztonsággal megőrizni.**

| 1. lemez | 2. lemez | 3. lemez | 4. lemez | 5. lemez (paritás) |
|---|---|---|---|---|
| A = 35 | B = 12 | C = 20 | D = 71 | P = 138 |

**Bármely lemezegység hibája esetén a**

**P = A + B + C + D**

**összefüggés megfelelő átrendezésével annak tartalma visszaállítható.**

## RAID rendszerek összehasonlítása

**RAID-1 (tükrözés)**
- A leggyorsabb, a legnagyobb helyigényű
- A legdrágább megoldás

**RAID-3 (paritás külön eszközön)**
- Nagy állományok, kevés tranzakció
- Olcsóbb, mint a tükrözés

**RAID-5 (elosztott paritás)**
- Kis állományok, sok tranzakció,
- A lemezek terhelése egyenletesebb
- RAID-3-nál lassabb



## Fibre channel (FC)

| | |
|---|---|
| **Sebesség** | **266 Mb/mp – 4 Gb/mp** |
| **Legnagyobb áthidalható távolság (Cu)** | **30 m** |
| **Legnagyobb áthidalható távolság (optikai)** | **10 km** |
| **Támogatott protokollok** | **IP, SCSI, IPI, HIPPI, audio/video** |

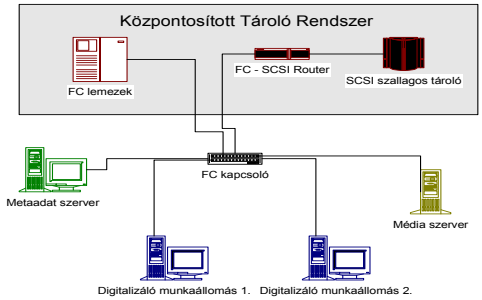


## FC – Ethernet - ATM

| | Fibre Channel | Ethernet | ATM |
|---|---|---|---|
| **Alkalmazás** | **Tárolórendszer, Hálózat, Videó, Számítógép cluster** | **Hálózat** | **Hálózat, Videó** |
| **Topológia** | **Pont-pont Kapcsolt, HUB** | **Pont-pont Kapcsolt, HUB** | **Kapcsolt** |
| **Sebesség (jelenleg)** | **1,06 Gb/s** | **1,25 Gb/s** | **622 Mb/s** |
| **Sebesség (tervek)** | **2,12 / 4,24 Gb/s** | **–** | **1,24 Gb/s** |
| **Garantált átvitel** | **van** | **nincs** | **nincs** |
| **Keret mérete** | **Változó, 0-2kB** | **Változó, 0-1,5 kB** | **Állandó, 53 bájt** |
| **Fizikai közeg** | **Rézvezeték, optikai** | **Rézvezeték, optikai** | **Rézvezeték, optikai** |
| **Protokollok** | **Hálózati, SCSI, Video** | **Hálózati** | **Hálózati, Video** |

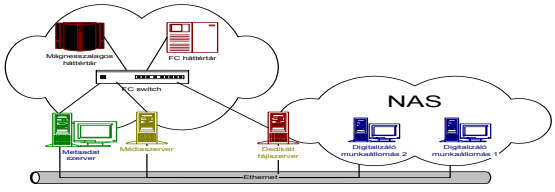


## SAN (Storage Area Network)






## NAS (Network Attached Storage)






## SAN-NAS összehasonlítás

| *SAN* | *NAS* |
|---|---|
| **Szerver-háttértár kapcsolat** | **Szerver-kliens kapcsolat** |
| **Sok platform** | **Tetszőleges platform** |
| **Nagy megbízhatóság, sávszélesség** | **Nem garantált, osztott sávszélesség** |
| **A kapcsolódó szerverek száma néhány száz** | **Tetszőleges számú kliens** |
| **Maximális kiterjedés 10-20 km** | **Tetszőleges távolság áthidalható** |
| **Közvetlen háttértárat igénylő alkalmazások: média szerver, adatbázis szerver** | **Állomány szerver alapú alkalmazások** |
| **Csatorna orientált átvitel** | **Csomag orientált átvitel** |

## SAN-NAS együttműködés

**A két rendszer működhet:**

- egymás mellett (függetlenül)
- egymást kiegészítve

**A NAS szerver háttértára lehet a SAN része**

- A kliensek kérései kielégíthetők
  - Engedélyek a LAN-on keresztül (ethernet kapcsolat)
  - Adatok a SAN-on keresztül (fibre channel kapcsolat)



## Összefoglalás

- **Fizikai lemezkezelés**
  - Ütemezés: FCFS, SSTF, Scan változatok
  - Logikai-fizikai cím konverzió
  - Blokkméret optimalizálás
- **Tömörítés**
  - RLE, DE, Huffmann
- **Adatvédelem**
  - Szoftver: paritás, CRC
  - Hardver: tükrözés, RAID



## Erőforráskezelés

**Erőforrások**

- Erőforrás foglalási gráf
- Holtpont, Kiéheztetés

**Holtpont kezelő stratégiák**

- Megelőzés
- Felszámolás
- Közösen használt erőforrások problémái



## Erőforráskezelés alapfogalmai

- **Feladat**
  - **a számítógéprendszer erőforrásainak elosztása (a futó folyamatok igényei alapján)**
  - **illetve az erőforrások használatáért vívott versenyhelyzetek kezeléséről**
- **Céljai:**
  - **a rendszer működését gazdaságossá tenni**
  - **elkerülni a holtponthelyzetek kialakulását és/vagy felszámolni a kialakult holtponthelyzeteket**



## Erőforrások csoportosítása

**hardver erőforrások (processzor, memóriák, I/O csatornák, perifériák)**

**szoftver erőforrások (programok, adatállományok - egyre fontosabbak)**

**"hagyományos" erőforrások (nyomtatók, szövegszerkesztők)**

**az operációs rendszer által létrehozott erőforrások (lapok, pufferek, floppy blokkok, adatállományok, tartalomjegyzékek)**

**elvehető erőforrások (processzor, memória)**

**nem elvehető erőforrások (index állomány, nyomtató)**



## Gazdaságosság

**Valamilyen költségfüggvény minimalizálása**

- kihasználtság
- válaszidő

**Egyéb szempontok (ellentmondóak)**

- átlagos átfutási idő minimalizálása
- a lassú válaszok számának minimalizálása
- a (hardver) kihasználtság maximalizálása
- maximális kihasználtság adott válaszidőkorlát mellett

**Tendencia: a kihasználtság háttérbe szorul (HW ára csökken!) a válaszidővel szemben**

## Az erőforrások lefoglalása

**Statikus lefoglalás**

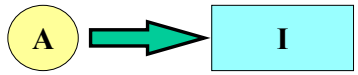
– a folyamat indulása előtt lefoglalja az összes szükséges erőforrást
- **pazarló**
- **kiéheztetés**
- **ha egyszer elindult, erőforrás korlát miatt nem áll le**

**Dinamikus lefoglalás**

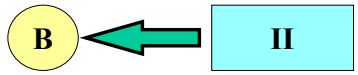
– a folyamat csak akkor igényel erőforrást, amikor éppen szüksége van rá
- **hamar elindulhat egy folyamat, de versenyzés miatt lassabban fut(hat)**
- **teljesítőképesség, átbocsátóképesség nő**
- **nagy probléma: holtpont**



## Állapot átmeneti gráf



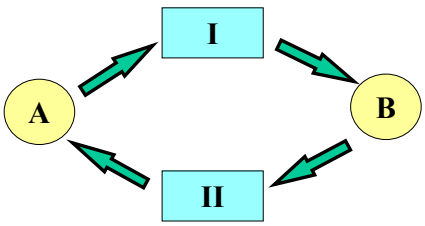

**Az „A" folyamat igényli a „I" erőforrást**




**A „B" folyamat birtokolja a „II" erőforrást**



## Holtponti állapotgráf






## HOLTPONT - Deadlock

**Több folyamat egy olyan erőforrás felszabadulására vár, amit csak egy ugyancsak várakozó folyamat tudna előidézni**

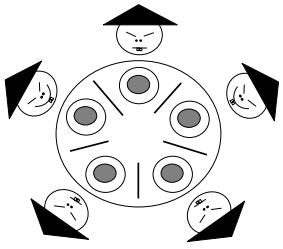


## KIÉHEZTETÉS - Starvation

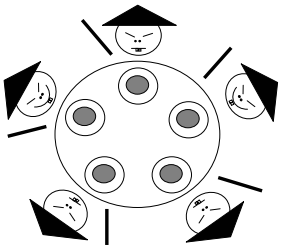
**Egy folyamat - az erőforráskezelő stratégiája miatt - beláthatatlan ideig nem jut erőforráshoz**



## Vacsorázó bölcsek - példa



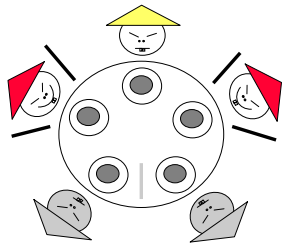
## Vacsorázó bölcsek - Holtpont



**Egyikük sem rendelkezik az ÖSSZES szükséges erőforrással**



## Vacsorázó bölcsek - Kiéheztetés



**Összesen ugyan van elegendő erőforrás, de szerencsétlen esetben egyesek mégis éheznek**



## Holtpontkialkulás lehetőségének feltételei

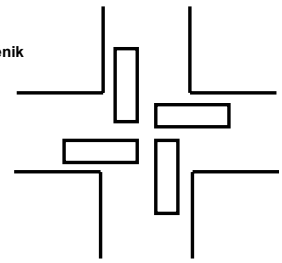
1. Kölcsönös kizárás van
2. Várakozás közben lekötés történik
3. Rablás nincs
4. Ciklikus várakozás van

**A feltételeknek EGYSZERRE kell teljesülniük a holtpont kialakulásához, vagyis HA LEGALÁBB AZ EGYIK FELTÉTEL NEM TELJESÜL, NEM ALAKULHAT KI HOLTPONT!**



## Holtpontkialkulás lehetőségének feltételei - illusztráció

1. Kölcsönös kizárás van
2. Várakozás közben lekötés történik
3. Rablás nincs
4. Ciklikus várakozás van





## Egy „igazi” holtpont



## Holtpont kezelési stratégiák

- **A holtpont megelőzése**
  - A négy feltétel (legalább) egyikét nem engedjük teljesülni
  - **NEM MINDIG LEHETSÉGES**
    **(a korlátozó feltételek miatt)**
- **A kialakult holtpont felismerése és megszüntetése**

**A megelőzés “olcsóbb”**

*A holtpont megelőzése*

## *Kölcsönös kizárás*

**Ezzel a feltétellel nem tudunk mit kezdeni, vannak olyan erőforrások, melyeket egyszerre csak egy folyamat használhat (pl. nyomtató)**



*A holtpont megelőzése*

## *Várakozás közbeni lekötés*

**Egy folyamat csak akkor igényelhet újabb erőforrást, ha nincs más lekötött erőforrása**

**Megoldások:**

- Statikus erőforrás-kezelés
- Ha egy folyamat erőforrást igényel, el kell engedje az összes korábban lefoglalt erőforrását

**Hátrányok:**

- rossz az erőforrások kihasználtsága
- kiéheztetés



*A holtpont megelőzése*

## *Erőforrásrablás*

1. **Ha egy folyamat egy olyan erőforrást igényel, amit nem tud megkapni azonnal (vagyis várakoznia kell), akkor elvesszük tőle az ÖSSZES lefoglalva tartott erőforrását, akkor folytatódhat, ha visszaszerezte az összes régit és megszerezte az újat is**
2. **Ha egy folyamat olyan erőforrást igényel, amit más VÁRAKOZÓ folyamat foglal, ezt attól elveszi; ha az igényelt erőforrást egy nem várakozó folyamat használja, akkor az IGÉNYLŐ folyamat kerül várakozó állapotba és TŐLE rabolhatnak a többiek**

**E módszerek CSAK az elvehető erőforrásokra alkalmazhatók**



*A holtpont megelőzése*

## *Ciklikus várakozás 1.*

**1. módszer**

**Minden erőforráshoz egy (a többitől különböző) sorszámot rendelünk**

**A folyamatok az erőforrásokat csak azok sorszámaik szerint növekvő sorrendjében igényelhetik (ezért célszerű, ha a sorszámokat az erőforrások természetes felhasználási sorrendje szerint osztjuk ki)**

**Másképp fogalmazva: ha egy folyamat egy bizonyos sorszámú erőforrást igényel, fel kell szabadítania az összes általa lefoglalt, az igényeltnél NAGYOBB sorszámú erőforrást**

**Hátrány: csökken a rendszer áteresztőképessége**



*A holtpont megelőzése*

## *Ciklikus várakozás 2.*

**2. módszer**

**Az operációs rendszer csak akkor engedi meg egy új erőforrás lefoglalását, ha ennek teljesítése után a rendszer ún. BIZTONSÁGOS állapotban marad**

**Biztonságos állapot: az összes folyamat erőforrás igénye valamilyen sorrendben kielégíthető**

**Nem biztonságos állapot:**

**holtpont kialakulhat (nem biztos, hogy kialakul!)**

**Hátrányok:**

**Olyan pluszinformáció szükséges (a maximális igény), amely sokszor nem tudható előre**

**Bonyolult algoritmus**



*Holtpont megelőző stratégiák*

## *Egyetlen foglalási lehetőség*

- **Csak az a folyamat foglalhat erőforrást, amelyik egyetlen egy fölött sem rendelkezik**

| **Előny:** | **Hátrány:** |
|---|---|
| – **nincs holtpont** | – Rossz erőforrás kihasználás,<br>– kiéheztetés veszélye |

*Holtpont megelőző stratégiák*

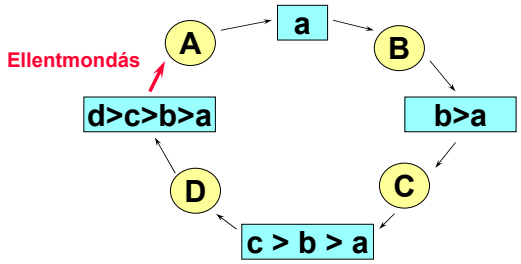
## *Rangsor szerinti foglalás*

- **Egy folyamat csak olyan osztályból igényelhet erőforrást, melynek sorszáma magasabb, mint a már birtokolt erőforrások sorszáma**

| • **Előny:** – **Nincs holtpont** | **Hátrány:** – **Pazarló** – **Önkényes** |
|---|---|



## *Rangsor szerinti foglalás bizonyítás*






*Holtpont megelőző stratégiák*

## *Bankár algoritmus*

- **Sohase elégítsünk ki egy igényt, ha az bizonytalan állapotot eredményez**

| **Előny:** – **nincs holtpont** – **az előzőeknél hatékonyabb** | **Hátrány:** – **Sok számolást igényel** – **Előzetes feltevéseken alapul** |
|---|---|



## *Bankár algoritmus - példa A.1*

**Aktuális állapot:**
**Összesen 12 db erőforrás**
**A és B folyamatok**

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| A | 4 | 6 | 2 |
| B | 4 | 11 | 7 |
| Szabad | 4 | | |

**Várakozó „C” folyamat - BEENGEDHETŐ ?**

| | | | |
|---|---|---|---|
| C | 2 | 8 | 6 |



## *Bankár algoritmus - példa A.2.*

**Tegyük föl, hogy beengedjük...**

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| A | 4 | 6 | 2 |
| B | 4 | 11 | 7 |
| C | 2 | 8 | 6 |
| Szabad | 2 | | |

**Az „A” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad erőforrások számánál**



## *Bankár algoritmus - példa A.3.*

**Az „A” lefutott, foglalt erőforrásai felszabadultak...**

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| B | 4 | 11 | 7 |
| C | 2 | 8 | 6 |
| Szabad | 6 | | |

**Az „C” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad erőforrások számánál**

## Bankár algoritmus - példa A.4.

Az „C” lefutott, foglalt erőforrásai felszabadultak...

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| B | 4 | 11 | 7 |
| Szabad | 8 | | |

Az „B” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad erőforrások számánál

Az állapot a „C” beengedése után is biztonságos

**ENGEDJÜK BE !**



## Bankár algoritmus - példa B.1

Aktuális állapot:

Összesen 12 db erőforrás

A és B folyamatok

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| A | 4 | 6 | 2 |
| B | 4 | 11 | 7 |
| Szabad | 4 | | |

Várakozó „C” folyamat - **BEENGEDHETŐ ?**

| | | | |
|---|---|---|---|
| C | 2 | 9 | 7 |



## Bankár algoritmus - példa B.2.

Tegyük föl, hogy beengedjük...

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| A | 4 | 6 | 2 |
| B | 4 | 11 | 7 |
| C | 2 | 9 | 7 |
| Szabad | 2 | | |

Az „A” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad erőforrások számánál



## Bankár algoritmus - példa B.3.

Az „A” lefutott, foglalt erőforrásai felszabadultak...

| | Foglal | Max. igény | Várható igény |
|---|---|---|---|
| B | 4 | 11 | 7 |
| C | 2 | 8 | 7 |
| Szabad | 6 | | |

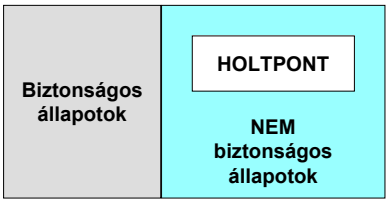
Az állapot „C” beengedése után

**NEM BIZTONSÁGOS**



## Biztonságos - nem biztonságos

Egy rendszer biztonságos, ha létezik olyan sorrend, mely szerint haladva a folyamatok igénye kielégíthető






## Bankár algoritmus

1. Felírjuk a folyamatok által maximálisan igényelt erőforrások számait tartalmazó mátrixot (MAX. IGÉNY)
2. Felírjuk a folyamatok által lefoglalva tartott erőforrások számait tartalmazó mátrixot (FOGLAL)
3. A MAX. IGÉNYből FOGLALt kivonva kapjuk a még kielégítetlen igényeket leíró mátrixot (IGÉNY)
4. Erőforrás fajtánként összeadjuk a lefoglalva tartott erőforrások számait, majd ezeket kivonva az egyes erőforrások összdarabszámából kapjuk a pillanatnyilag rendelkezésre álló erőforrás készletet (KÉSZLET)

# Bankár algoritmus - folytatás

5. Megnézzük, hogy a KÉSZLETből kielégíthető-e valamelyik folyamat igénye

6a. Ha nincs ilyen folyamat, a rendszer NEM BIZTONSÁGOS állapotban van

6b. Ha van ilyen folyamat, kielégítjük igényét

7. A kiválasztott folyamat a lefutása után felszabadítja az összes általa használt erőforrást, azaz KÉSZLET új értékét megkapjuk, ha előző értékéhez hozzáadjuk a folyamat által eredetileg lefoglalva tartott erőforrások számát (FOGLAL megfelelő sorát)

8. Visszamegyünk az 5. lépésre



## Bankár algoritmus - Több erőforrás

# A feladat megfogalmazása

Egy rendszerben az alábbi erőforrások vannak:
E1: 10 darab E2: 5 darab E3: 7 darab
A rendszerben 5 folyamat van: F1, F2, F3, F4, F5
Biztonságos-e holtpontmentesség szempontjából a következő állapot?

| | MAX. IGÉNY | | | FOGLAL | | |
|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 5 | 3 | 0 | 1 | 0 |
| F2 | 3 | 2 | 2 | 3 | 0 | 2 |
| F3 | 9 | 0 | 2 | 3 | 0 | 2 |
| F4 | 2 | 2 | 2 | 2 | 1 | 1 |
| F5 | 4 | 3 | 3 | 0 | 0 | 2 |



## Bankár algoritmus - Több erőforrás

# IGÉNY meghatározása

**[IGÉNY] = [MAX.IGÉNY] - [FOGLAL]**

| | MAX. IGÉNY | | | FOGLAL | | | IGÉNY | | |
|---|---|---|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 |
| F2 | 3 | 2 | 2 | 3 | 0 | 2 | 0 | 2 | 0 |
| F3 | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 |
| F4 | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 |
| F5 | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 |



## Bankár algoritmus - Több erőforrás

# KÉSZLET meghatározása

(minden erőforrásra)

Σ ERŐFORRÁS
- Σ FOGLAL
KÉSZLET

| | FOGLAL | | |
|---|---|---|---|
| | E1 | E2 | E3 |
| F1 | 0 | 1 | 0 |
| F2 | 3 | 0 | 2 |
| F3 | 3 | 0 | 2 |
| F4 | 2 | 1 | 1 |
| F5 | 0 | 0 | 2 |
| Σ FOGLAL | 8 | 2 | 7 |

E1: 10 darab
E2: 5 darab
E3: 7 darab

| | E1 | E2 | E3 |
|---|---|---|---|
| Σ ERŐFORRÁS | 10 | 5 | 7 |
| Σ FOGLAL | 8 | 2 | 7 |
| KÉSZLET | 2 | 3 | 0 |



## Bankár algoritmus - Több erőforrás

# IGÉNY kielégítése - 1. lépés

| | IGÉNY | | | FOGLAL | | |
|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 4 | 3 | 0 | 1 | 0 |
| F2 | 0 | 2 | 0 | 3 | 0 | 2 |
| F3 | 6 | 0 | 0 | 3 | 0 | 2 |
| F4 | 0 | 1 | 1 | 2 | 1 | 1 |
| F5 | 4 | 3 | 1 | 0 | 0 | 2 |

Kielégíthető

Felszabadul

KÉSZLET : (2, 3, 0)

ÚJ KÉSZLET : (5, 3, 2)



## Bankár algoritmus - Több erőforrás

# IGÉNY kielégítése - 2. lépés

| | IGÉNY | | | FOGLAL | | |
|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 4 | 3 | 0 | 1 | 0 |
| F2 | 0 | 2 | 0 | 3 | 0 | 2 |
| F3 | 6 | 0 | 0 | 3 | 0 | 2 |
| F4 | 0 | 1 | 1 | 2 | 1 | 1 |
| F5 | 4 | 3 | 1 | 0 | 0 | 2 |

Kielégíthető

Felszabadul

KÉSZLET : (5, 3, 2)

ÚJ KÉSZLET : (5, 3, 4)

*Bankár algoritmus - Több erőforrás*

## *IGÉNY kielégítése - 3. lépés*

| | IGÉNY | | | FOGLAL | | |
|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 4 | 3 | 0 | 1 | 0 |
| F2 | 0 | 2 | 0 | 3 | 0 | 2 |
| F3 | 6 | 0 | 0 | 3 | 0 | 2 |
| F4 | 0 | 1 | 1 | 2 | 1 | 1 |
| F5 | 4 | 3 | 1 | 0 | 0 | 2 |

Kielégíthető

Felszabadul

**KÉSZLET : (5, 3, 4)**

**ÚJ KÉSZLET : (7, 4, 5)**



*Bankár algoritmus - Több erőforrás*

## *IGÉNY kielégítése - 4. lépés*

| | IGÉNY | | | FOGLAL | | |
|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 4 | 3 | 0 | 1 | 0 |
| F2 | 0 | 2 | 0 | 3 | 0 | 2 |
| F3 | 6 | 0 | 0 | 3 | 0 | 2 |
| F4 | 0 | 1 | 1 | 2 | 1 | 1 |
| F5 | 4 | 3 | 1 | 0 | 0 | 2 |

Kielégíthető

Felszabadul

**KÉSZLET : (7, 4, 5)**

**ÚJ KÉSZLET : (7, 5, 5)**



*Bankár algoritmus - Több erőforrás*

## *IGÉNY kielégítése - 5. lépés*

| | IGÉNY | | | FOGLAL | | |
|---|---|---|---|---|---|---|
| | E1 | E2 | E3 | E1 | E2 | E3 |
| F1 | 7 | 4 | 3 | 0 | 1 | 0 |
| F2 | 0 | 2 | 0 | 3 | 0 | 2 |
| F3 | 6 | 0 | 0 | 3 | 0 | 2 |
| F4 | 0 | 1 | 1 | 2 | 1 | 1 |
| F5 | 4 | 3 | 1 | 0 | 0 | 2 |

Kielégíthető

Felszabadul

**KÉSZLET : (7, 5, 3)**

**ÚJ KÉSZLET : (10, 5, 7)**



*Bankár algoritmus - Több erőforrás*

## *BIZTONSÁGOS !*

Vagyis találtunk legalább egy (ebben a példában több is van) sorrendet, amelyben a folyamatok erőforrás igénye kielégíthető:

**F2, F5, F4, F1, F3**

Tehát a rendszer **BIZTONSÁGOS ÁLLAPOTBAN** van.



## *A holtpont felismerése*

**Folyamatos adatgyűjtés az erőforrások szétosztásáról és a ki nem elégített igényekről**

**Ezen adatokból a holtponthelyzetet detektálni képes algoritmus ismételt futtatása**

- ha egy igényt nem lehet azonnal kielégíteni
  - gyakran kell futtatni, sok plusz idő
- előre megbecsült időnként - ezalatt holtpont (akár több is) kialakulhat
  - nagy felélesztési veszteség



## *Holtpontból való felélesztés*

**Megszüntetjük a holtponti helyzetben lévő folyamatokat - nagy veszteség**

**Finomítás: EGYENKÉNT szüntetünk meg holtponti helyzetben lévő folyamatokat, amíg a holtpont fennáll - kisebb veszteség**

*Holtpontból való felélesztés*

## *Áldozatkijelölési szempontok*

**Melyikkel hány erőforrást nyerek**
**Hány további erőforrást igényel még**
**Mennyi már elhasznált CPU időt ill. I/O munkát vesztek**
**Mennyi van még hátra a futásából**
**Ismételhető / nem ismételhető folyamat-e**
**A folyamat prioritása**
**Megszüntetése hány további folyamatot érint**



*Holtpontból való felélesztés*

## *Erőforrás rablás*

**A holtponti helyzetben lévő folyamatoktól erőforrásokat rablok el**

**Veszteség, ha nem elvehető erőforrást kell elvenni**

**Áldozatkijelölés az előzőhöz hasonló szempontok szerint**



## *Holtpontból való felélesztés*

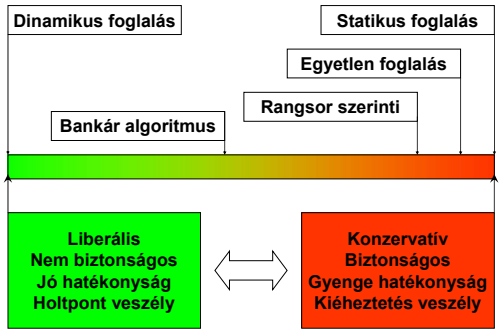
**Sokszor elég, ha a folyamatot nem szüntetjük meg teljesen, hanem egy előző állapotából folytatjuk - ellenőrző pontok (check point)**

– nagy tárigény
– sok plusz időt igényel a folyamatok állapotának periodikus mentése

**Ugyanaz a folyamat nem választható akárhányszor áldozatul (kiéheztetés)**

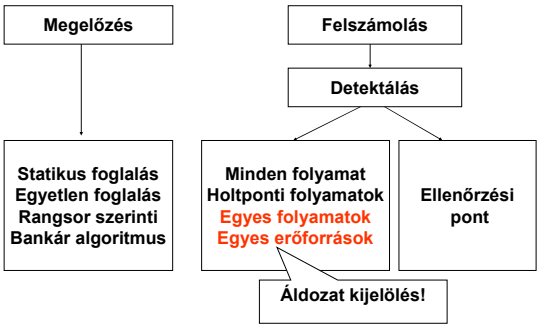


## *Erőforráskezelő stratégiák összehasonlítása*






## *Holtpont kezelési módszerek*



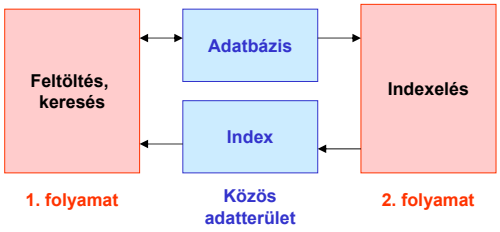



## *Az erőforrás-kezelés problémái*

**ALULSZABÁLYOZÁS - nem tudjuk kihasználni a HW lehetőségeit (nem működtet párhuzamosan eszközöket, feleslegesen várakoztat stb.)**

**TÚLSZABÁLYOZÁS - túl sok adminisztrációt végez és ezzel csökkenti a hasznos időt és tárméretet**

## Kölcsönös kizárás (példa)




**A közös adatterületet egyszerre csak egy folyamat használhatja!**



### Közösen használt erőforrások
## Termelő / fogyasztó probléma


**A közös adatterületet (KÖZÖS ERŐFORRÁS) egyszerre csak egy folyamat használhatja (KÖLCSÖNÖS KIZÁRÁS)**



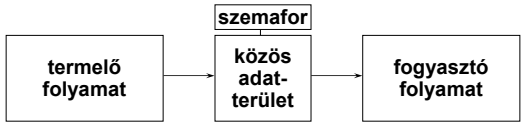
### Közösen használt erőforrások
## Vezérlés: Szemafor


**Kölcsönös kizárás igénye nemcsak közös memória esetén lép fel; pl. nyomtató közös használata**

**Mielőtt a folyamat használni kezdené a közös erőforrást, ellenőriznie kell, hogy az szabad-e. (Ezt az adott közös erőforráshoz rendelt szemafor jelzi.) CSAK akkor kezdheti el használni, ha a szemafor szabadot jelzett, ellenkező esetben várakoznia kell!**



### Közösen használt erőforrások
## Termelő és fogyasztó programja




1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az erőforrás használata (írás a közös memóriába)
6. a szemafor szabadra állítása

1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az erőforrás használata (olvasás a közös memóriából)
6. a szemafor szabadra állítása



### Közösen használt erőforrások
## A program közös elemei


1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az erőforrás használata (írás a közös memóriába)
6. a szemafor szabadra állítása

1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az erőforrás használata (olvasás a közös memóriából)
6. a szemafor szabadra állítása



## P és V primitívek

**Primitív: megszakíthatatlan (oszthatatlan) művelet**

**P primitív: FOGLALTTÁ ÁLLÍTÁS**
- 1. a szemafor olvasása
- 2. a beolvasott érték vizsgálata
- 3. ha szabad: a szemafor foglaltra állítása
- 4. ha foglalt: vissza 1-re

**V primitív: SZABADDÁ ÁLLÍTÁS**
- A szemafor szabadra állítása

***Közösen használt erőforrások***

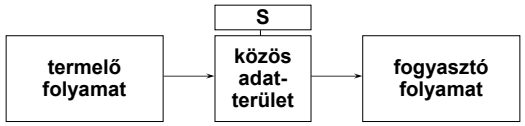
## *Termelő és fogyasztó programja -Primitívekkel*


**P(S);**
**ÍRÁS A MEMÓRIÁBA**
**V(S);**

**P(S);**
**OLV. A MEMÓRIÁBÓL**
**V(S);**



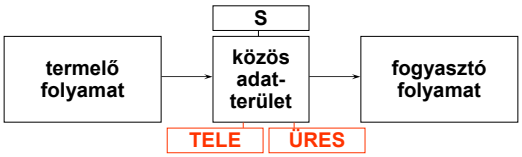
## *Postaláda kezelés*




**Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db.) üzenet írható**



## *Postaláda kezelés*




**Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db.) üzenet írható**

**Újabb szemaforok a vezérléshez:**

- TELE
- ÜRES



## *Postaláda kezelés*

**Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db.) üzenet írható**

**3 db. szemafor kell a vezérléséhez**

- S: a kölcsönös kizárást megvalósító szemafor (bináris; 0=foglalt; 1=szabad; kezdeti értéke: szabad)
- TELE: a tele helyek száma (nem bináris; értéke 0 és N között lehet; kezdeti értéke:0)
- ÜRES: az üres helyek száma (nem bináris; értéke 0 és N között lehet; kezdeti értéke:N)



## *P és V primitívek*

**Primitív: megszakíthatatlan (oszthatatlan) művelet**

**P primitív: a paraméterül kapott szemafor értékének EGGYEL CSÖKKENTÉSE (bináris szemafor esetén ez a FOGLALTTÁ ÁLLÍTÁS)**

**V primitív: a paraméterül kapott szemafor értékének EGGYEL NÖVELÉSE (bináris szemafor esetén ez a SZABADDÁ ÁLLÍTÁS)**

**Feltétel:**

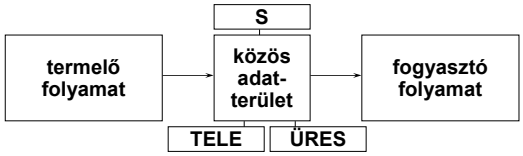
- SZABAD = 1
- FOGLALT = 0



## *Bináris és nembináris szemaforok*

| | Bináris | Nembináris |
|---|---|---|
| P | $S_{N+1} = 0$ | $S_{N+1} = S_N - 1$ |
| V | $S_{N+1} = 1$ | $S_{N+1} = S_N + 1$ |

## Postaláda kezelés programja




P(ÜRES);
P(S);
ÍRÁS A MEMÓRIÁBA
V(S);
V(TELE);

P(TELE);
P(S);
OLV. A MEMÓRIÁBÓL
V(S);
V(ÜRES);



## Összefoglalás

- Az erőforráskezelés alapfogalmai
- Holtpont, kiéheztetés
- Holtpont feltételei, megelőzése
  - Egyetlen foglalási lehetőség
  - Rangsor szerinti foglalás
  - BANKÁR ALGORITMUS
- Holtpont felismerése, felszámolása
- Közösen használt erőforrások
  - Termelő-fogyasztó probléma
  - Postaláda kezelés
  - Szemaforok, primitívek

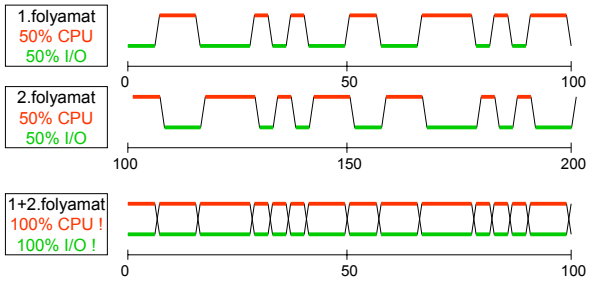


## Folyamat- és processzorkezelés

- Folyamatok létrehozása
- Műveletek folyamatokkal
- Speciális állapotok
- Processzor ütemezés



## Processzorkezelés - bevezető




Az összes futási idő (majdnem) a felére csökken
A CPU és az I/O eszköz kihasználtsága 100%
DE plusz tevékenységeket kell végezni



## Operációs rendszer (Erőforrás szemlélet)

**A folyamatok egy olyan csoportja, amely a felhasználói folyamatok között elosztja az erőforrásokat**



## Folyamatok

1. „Életre kelt” programok
2. Olyan programok, amelyeknek van folyamatleíró táblája

## Folyamatleíró/vezérlő blokk

*(Process Control Block - PCB)*

**Egyértelműen azonosítja a folyamatot, minden információ megtalálható benne, ami az operációs rendszer számára szükséges**

- A folyamat azonosítója, szülő, gyerekek
- A folyamat állapota (új, futásra kész, aktív, várakozó stb.)
- A PC + regiszterek tartalma (a következő utasítás címe)
- Erőforrások állapota (ki nem elégített I/O igények, a folyamathoz rendelt I/O eszközök, megnyitott állományok)
- CPU ütemezési információ (prioritás, CPU igény)
- Elszámolási adatok, statisztikák (rendszerben eltöltött idő)
- Memóriaterület adatai, címtranszformációs táblák



## Várakozási sor (queue)

**A folyamatok közül egyidejűleg csak egy futhat, tehát a többinek valamire várnia kell (I/O eszköz, CPU, memória stb.), tehát egy várakozási sorba kell állnia**.

Funkció: A folyamatok (PCB-k) egy olyan sora, melyben minden elem utal az őt követő elemre

Megvalósítás: láncolt adatstruktúra (lista)

Tipikus: FIFO (First In First Out)

Műveletek: hozzáfűzés, beékelés

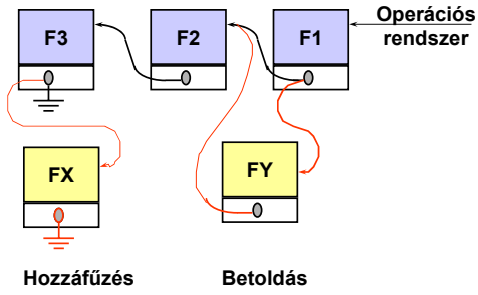


## Lista deklarációja

```
type
  PCB = record
          ProgramCounter: integer;
          ProcessorState: integer;
          Registers: array 1:16 of integer
          {egyéb adatok}
        end;
  PCBpointer = ^PCB;
  Lista = record
          process: PCB;
          NextProcess: PCBpointer;
        end;
```

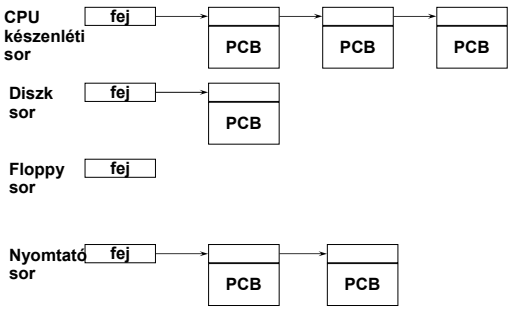


## Lista (sor) bővítése






## PCB-k használata - példa



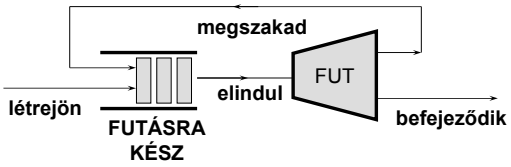



## Folyamatok állapotai

A várakozási sorok megfelelnek a folyamatok állapotainak.

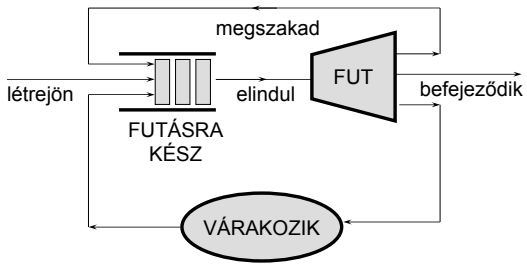
- **FUTÁSRA KÉSZ** (már csak a CPU hiányzik)
  - Megszakított (pl. kivétel, I/O művelet)
- **FUT**
- **VÁRAKOZIK** (további erőforrásra)
  - Holtponti (ha reménytelenül vár valamire)
- **FELFÜGGESZTETT** (erőforrás nélkül, csak PCB)

## Folyamatok állapotai I.



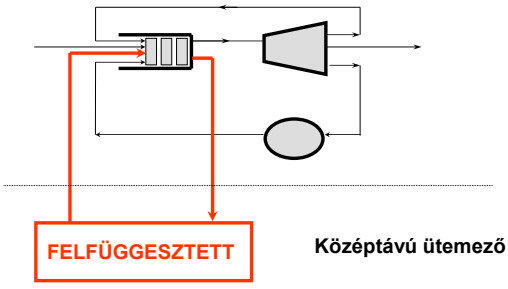



## Folyamatok állapotai II.





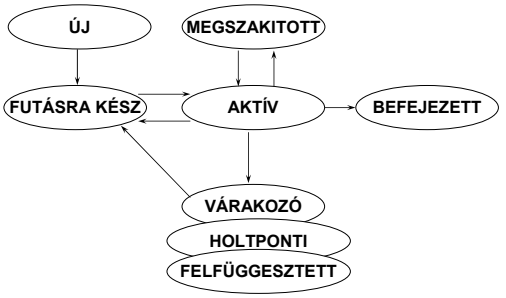

## Folyamatok állapotai III.




Középtávú ütemező

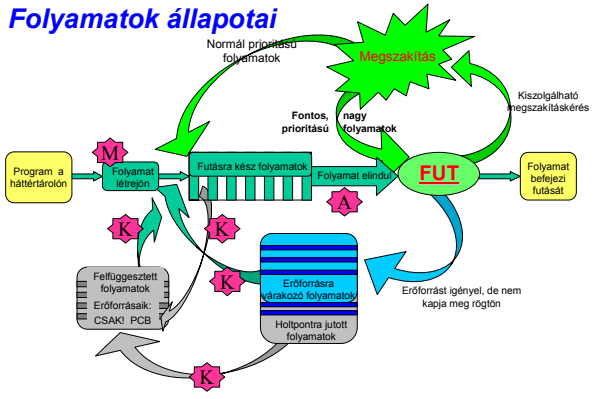


## A folyamatok állapotai



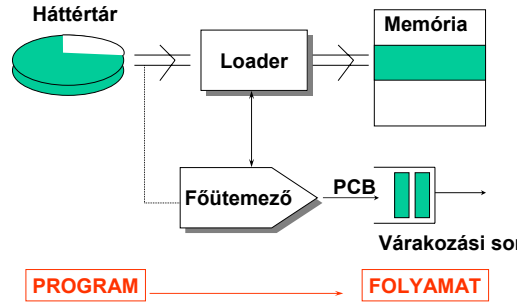



## Folyamatok állapotai






## A Főütemező



## Magas szintű ütemező

**A diszken várakozó folyamatok közül "enged be" újat a processzor várakozási sorába**

**A kiválasztott folyamathoz elkészíti a PCB-t**

**Viszonylag ritkán fut - nem sebesség kritikus**

**Interaktív rendszereknél háttérbe szorul**

**Optimális munkakeveréket töltsön be**

- CPU orientált folyamatok (viszonylag ritka)
- I/O orientált folyamatok



## Közbenső szintű ütemező

**Nagyobb rendszerekben található meg**

**Olyan szempontokat figyel, amelyre az alacsony szintű ütemezőnek nincs ideje**

- pl. folyamatok futási sebességének figyelése
  - kiéheztetést tudja detektálni
  - észreveheti, ha egy folyamat aránytalanul lassan fut
- a megbomlott egyensúly helyreállítása
  - folyamatok prioritásának megváltoztatása
  - folyamatok ideiglenes / végleges felfüggesztése

**"Globális optimum"**



## Alacsony szintű ütemező

**A CPU-ra várakozó folyamatok közül választja ki azt, amelyik a következő időben használja a CPU-t**

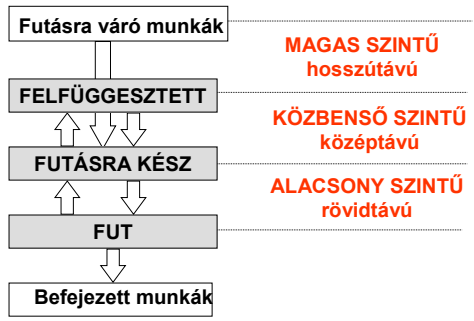
**Aktivizálja a kiválasztott folyamatot (regiszterek visszaállítása a PCB-ből, vezérlésátadás) - diszpécser**

**Gyakran fut - sebességkritikus (különösen időosztásos és valósidejű rendszereknél)**

**Gyors kell legyen => viszonylag kevés szempontot vehet figyelembe a választáshoz ("lokális optimum")**



## Ütemezési szintek






## Ütemezési statisztikák

| | |
|---|---|
| $T_{ÉRKEZÉS}$ | A folyamat a futásra kész sorba érkezik |
| $T_{KEZDET}$ | A futás kezdőidőpontja |
| $T_{BEFEJEZÉS}$ | A folyamat elhagyja a rendszert |
| $T_{IGÉNY}$ | A folyamat processzoridő igénye |

$$T_{VÁRAKOZÁS} = T_{BEFEJEZÉS} - T_{ÉRKEZÉS} - T_{IGÉNY}$$

$$T_{VÁLASZ} = T_{KEZDET} - T_{ÉRKEZÉS}$$

$$T_{ÁTFUTÁS} = T_{BEFEJEZÉS} - T_{KEZDET}$$



## Az ütemezési módszerek minősítése

**Szimuláció mintavétel alapján**

- Egy rendszer statisztikai adatait gyűjtjük, ezek az adatsorok szolgálnak mintasorozatként

**Szimuláció valószínűségi modell alapján**

- Az összegyűjtött adatok alapján valószínűségi modellt állítunk fel (Poisson-eloszlás), ennek alapján generálunk mintasorozatot

**Analitikus számítások**

**Tapasztalat**

## Ütemezési algoritmusok

**Optimalizálási szempontok**

**CPU kihasználtság**

**A rendszer átbocsátóképessége (pl. a befejezett feladatok száma óránként)**

**Fordulási idő (feladat leadása és befejezése közti idő)**

**Várakozási idő (a CPU-ra várakozási sorban töltött idő - tkp. EZT befolyásolják az ütemezési algoritmusok)**

**Válaszidő (interaktív rendszereknél nagyon fontos)**



## Előbb jött - előbb fut
### First Come First Served - FCFS

**A folyamatok érkezési sorrendjükben kapják meg a processzort**

**Előny:**
- a legegyszerűbb stratégia, biztos

**Hátrány:**
- a folyamatok várakozási, fordulási ideje nagymértékben függ a folyamatok érkezési sorrendjétől
- „lassú kamion” - effektus, csorda hatás
- nagy várakozási idő, rossz hatékonyság



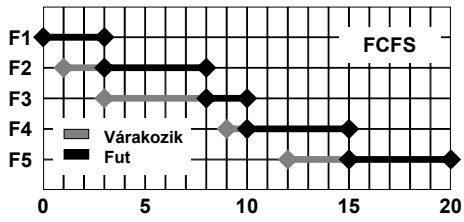
## FCFS példa

| | Érkezés | Igény |
|---|---|---|
| F1 | 0 | 3 |
| F2 | 1 | 5 |
| F3 | 3 | 2 |
| F4 | 9 | 5 |
| F5 | 12 | 5 |

| Kezdés | Befejezés | Várakozás |
|---|---|---|
| 0 | 3 | 0 |
| 3 | 8 | 2 |
| 8 | 10 | 5 |
| 10 | 15 | 1 |
| 15 | 20 | 3 |
| Átlagos idő | | 11/5 |



## FCFS időzítés






## Átlagos várakozási idő számítás - FCFS

**Határozza meg az alábbi terhelés esetén az átlagos várakozási idő értékét!**

| PR. | ÉRK. IDŐ | CPU IGÉNY |
|---|---|---|
| F1 | 0 | 14 |
| F2 | 7 | 8 |
| F3 | 11 | 36 |
| F4 | 20 | 10 |



## Átlagos várakozási idő számítás - FCFS

| PR. | ÉRK. IDŐ | CPU IGÉNY |
|---|---|---|
| P1 | 0 | 14 |
| P2 | 7 | 8 |
| P3 | 11 | 36 |
| P4 | 20 | 10 |

| PR. | ÉRK. IDŐ | CPU IGÉNY | KEZD. IDŐPONT | BEF. IDŐPONT | VÁRAKOZÁSI IDŐ (KEZD. - ÉRK.) |
|---|---|---|---|---|---|
| P1 | 0 | 14 | 0 | 14 | 0 |
| P2 | 7 | 8 | 14 | 22 | 7 |
| P3 | 11 | 36 | 22 | 58 | 11 |
| P4 | 20 | 10 | 58 | 68 | 38 |
| | | | | | 56 |

ÁTLAGOS VÁRAKOZÁSI IDŐ: 56 / 4 = 14

## A legrövidebb előnyben
### Shortest Job First - SJF

**A CPU-t egy folyamat befejeződése után a legrövidebbnek adja oda (ha több ilyen van, FCFS szerint választ közülük), általában kötegelt rendszereknél van ilyen**

**Előny:**
- a legrövidebb az átlagos várakozási idő

**Hátrány:**
- KIÉHEZTETÉS (hosszú folyamatokkal mostohán bánik)
- Tudni kell ELŐRE a folyamat hosszát
  - kötegelt rendszereknél programozói becslés
  - időosztásos rendszereknél statisztikai becslés
  - mi történjen, ha a becslés rossz?



## Preemptív / nem preemptív algoritmusok

**Az SJF-nek és a prioritásos algoritmusoknak vannak preemptív formái is**

**Preempció: ha egy rövidebb (SJF) vagy egy magasabb prioritású (prioritásos algoritmus) folyamat érkezik, az MEGSZAKÍTJA az éppen futó folyamatot**

**Előny: kisebb átlagos várakozási időt ad (SJF), illetve jobban preferálja a magasabb prioritású folyamatokat (prioritásos algoritmus)**

**Hátrány: a megszakításkor környezetváltásra van szükség**



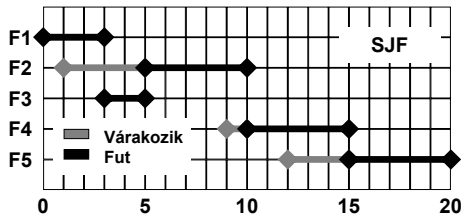
## SJF - példa

| | Érkezés | Igény |
|---|---|---|
| F1 | 0 | 3 |
| F2 | 1 | 5 |
| F3 | 3 | 2 |
| F4 | 9 | 5 |
| F5 | 12 | 5 |

| Kezdés | Befejezés | Várakozás |
|---|---|---|
| 0 | 3 | 0 |
| 5 | 10 | 4 |
| 3 | 5 | 0 |
| 10 | 15 | 1 |
| 15 | 20 | 3 |
| Átlagos idő | | 8/5 |



## SJF időzítés






## Átlagos várakozási idő számítás - SJF

**Határozza meg az alábbi terhelés esetén az átlagos várakozási idő értékét!**

| PR. | ÉRK. IDŐ | CPU IGÉNY |
|---|---|---|
| F1 | 0 | 14 |
| F2 | 7 | 8 |
| F3 | 11 | 36 |
| F4 | 20 | 10 |



## Átlagos várakozási idő számítás - SJF

| PR. | ÉRK. IDŐ | CPU IGÉNY |
|---|---|---|
| P1 | 0 | 14 |
| P2 | 7 | 8 |
| P3 | 11 | 36 |
| P4 | 20 | 10 |

| PR. | ÉRK. IDŐ | CPU IGÉNY | KEZD. IDŐPONT | BEF. | VÁR. IDŐ | VÁRÓ PROC. | LEG-RÖVIDEBB |
|---|---|---|---|---|---|---|---|
| P1 | 0 | 14 | 0 | 14 | 0 | P2(8), P3(36) | P2 |
| P2 | 7 | 8 | 14 | 22 | 7 | P3(36), P4(10) | P4 |
| P4 | 20 | 10 | 22 | 32 | 2 | P3(36) | P3 |
| P3 | 11 | 36 | 32 | 68 | 21 | - | - |
| | | | | | 30 | | |

ÁTLAGOS VÁRAKOZÁSI IDŐ: 30 / 4 = 7,5

## Körbenjáró
### Round Robin - RR

**A folyamatokat egy zárt körbe szervezzük, és minden folyamat egy előre rögzített IDŐSZELET - re kapja meg a processzort, majd visszaáll a sor végére**

**Tipikusan az interaktív rendszerek stratégiája**

**Előny:**
- egyszerű algoritmus, kis *válaszidő*
- nincs kiéheztetés, demokratikus

**Hátrány:**
- az időszelet lejártakor a folyamat állapotát el kell menteni - időveszteség



## RR - példa

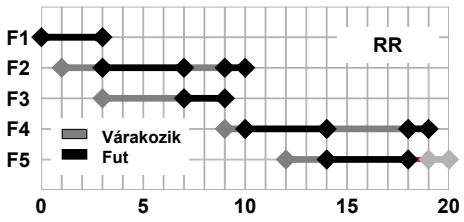
| | Érkezés | Igény |
|---|---|---|
| F1 | 0 | 3 |
| F2 | 1 | 5 |
| F3 | 3 | 2 |
| F4 | 9 | 5 |
| F5 | 12 | 5 |

| Befejezés | Várakozás |
|---|---|
| 3 | 0 |
| 10 | 4 |
| 9 | 4 |
| 19 | 5 |
| 20 | 3 |
| Átlagos idő | 16/5 |

**Időszelet = 4 egység**



## RR időzítés






## Átlagos várakozási idő számítás - RR

**Határozza meg az alábbi terhelés esetén az átlagos várakozási idő értékét, ha az időszelet 10!**

| PR. | ÉRK. IDŐ | CPU IGÉNY |
|---|---|---|
| F1 | 0 | 14 |
| F2 | 7 | 8 |
| F3 | 11 | 36 |
| F4 | 20 | 10 |



## Átlagos várakozási idő számítás - RR

| PR. | ÉRK. IDŐ | CPU IGÉNY |
|---|---|---|
| P1 | 0 | 14 |
| P2 | 7 | 8 |
| P3 | 11 | 36 |
| P4 | 20 | 10 |

ÁTLAGOS VÁRAKOZÁSI IDŐ: 44 / 4 = 11

| PR. | ÉRK. IDŐ | CPU IGÉNY | KEZD. IDŐPONT | BEF. IDŐPONT | VÁR. IDŐ | MARAD. IDŐ | VÁRÓ PROC. |
|---|---|---|---|---|---|---|---|
| P1 | 0 | 14 | 0 | 10 | 0 | 4 | P2, P1 |
| P2 | 7 | 8 | 10 | 18 | 3 | - | P1, P3 |
| P1* | (10) | 4 | 18 | 22 | 8 | - | P3, P4 |
| P3 | 11 | 36 | 22 | 32 | 11 | 26 | P4, P3 |
| P4 | 20 | 10 | 32 | 42 | 12 | - | P3 |
| P3* | (32) | 26 | 42 | 52 | 10 | 16 | P3 |
| P3* | (52) | 16 | 52 | 62 | 0 | 6 | P3 |
| P3* | (62) | 6 | 62 | 68 | 0 | - | - |
| | | | | | 44 | | |



## Prioritásos módszerek

**Minden folyamathoz egy prioritási értéket rendelünk, és a CPU-t a legmagasabb prioritású folyamat kapja meg**

- külső prioritás - a folyamat "fontosságától" függő érték
- belső prioritás - az operációs rendszer határozza meg pl. erőforrás igény alapján

**Előny:**
- a prioritásokkal sokféle szempont érvényesíthető

**Hátrány:**
- KIÉHEZTETÉS (kis prioritású folyamaté)
- Megszüntetése: öregedés (prioritás növelése adott időnként)

## Összefoglalás

**Folyamatok állapotai, a PCB**
**Várakozási sorok (queue)**
**Ütemezési szintek**
**magas szintű, közbenső szintű, alacsony szintű**
**Ütemezési stratégiák**
**FCFS, SJF, RR**



## Memóriakezelés

- Valóságos tárkezelés
- Virtuális memória
- Tárvédelem, szegmentálás
- Gyorstárak, Tároló hierarchia

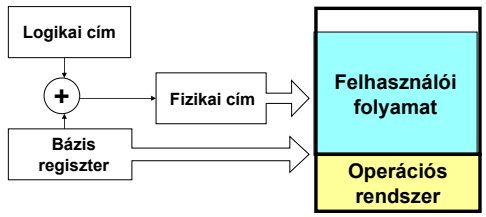


## Abszolút címzésű rendszerek

**a tárat két részre osztják**
– operációs rendszer rezidens része
– felhasználó területe
**a program helye már írásakor ismert**
**az utasítások közvetlenül címezhetik a teljes tárat**
**rugalmatlan (mi van, ha az operációs rendszer mérete nő?)**
**multiprogramozásra alkalmatlan**

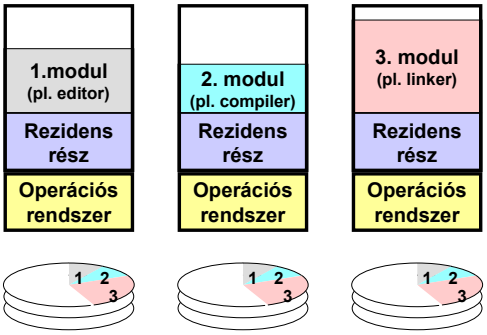


## Áthelyezhető címzés



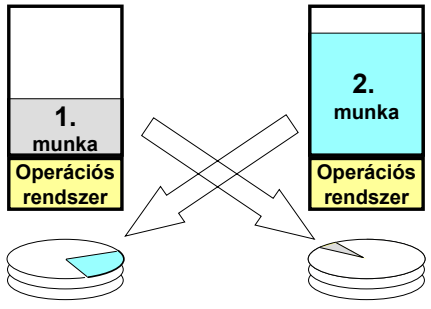



## Átlapoló (overlay) technika

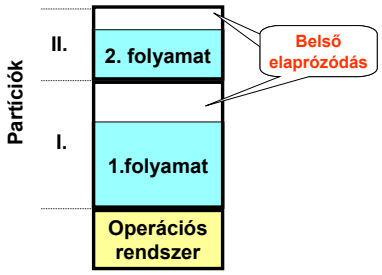





## Tárcsere (swapping)

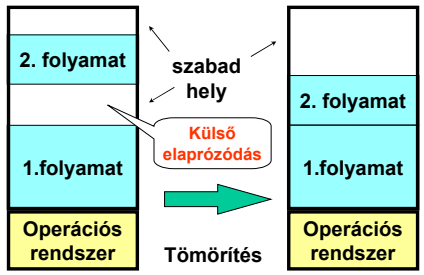


## Particionált rendszer



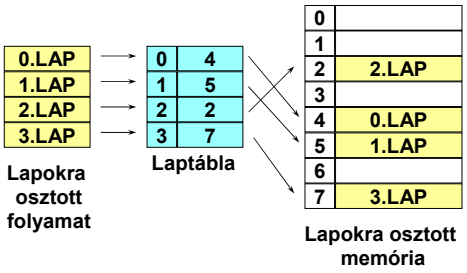



## Rugalmas partíciók



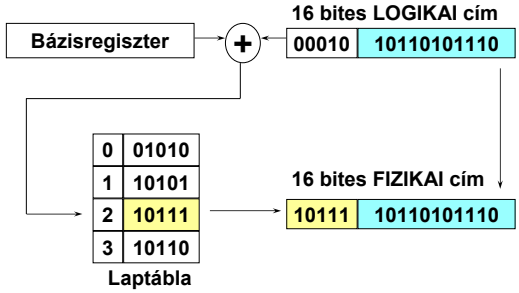



## Lapozási technika



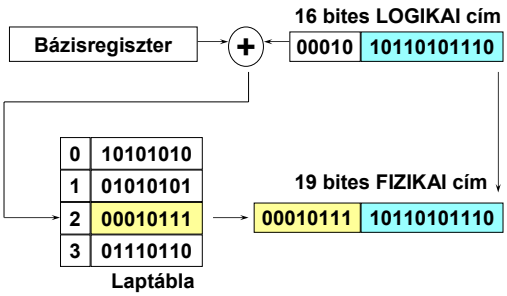



## Fizika cím számítása lapozásnál






## Memóriabővítés laptábla segítségével






## Előzmények

Áthelyezhető: nem mindig ugyanoda töltődik
Átlapoló (overlay): nem minden rész kell egyszerre
Tárcsere (swapping): memóriakép a háttértáron
Lapozás: nem folytonos elhelyezés

## Virtuális tárkezelés

**A címtartomány háttértáron (virtuális tár, virtuális / logikai címtartomány)**

**Az operatív memória mérete sokkal kisebb a háttértárénál**

**A virtuális címtartomány csak egy kis része fér be az operatív tárba, ahonnan futtatni tudunk**

**általában nincs szükség minden programrészletre**

**Előnyök:**

- ma már a mikroprocesszorok címtartománya is túl nagy ahhoz, hogy teljesen kiépítsük
- multiprogramozás során nem kell bent tárolni a teljes folyamatokat => több folyamat futhat párhuzamosan
- gyorsabb a betöltés és a felfüggesztés



## Lapszervezésű virtuális tár

**Felosztjuk a virtuális és az operatív tárat is egyforma méretű (kis, néhány kByte-os) egységekre, ún. LAPOKRA**

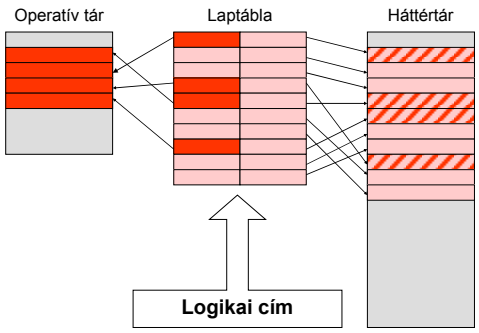
**Az operatív memóriában csak az éppen szükséges lapokat tartjuk bent**

**Laptábla tartja nyilván, hogy mely lapok vannak bent az operatív tárban**

**A processzor által kiadott (logikai) címet át kell alakítani operatív tárbeli (fizikai) címmé**



## Virtuális tárkezelés






## Lényeg, előnyök

A folyamat lapjai közül nem mindegyik van a memóriában

a többi a háttértáron várakozik

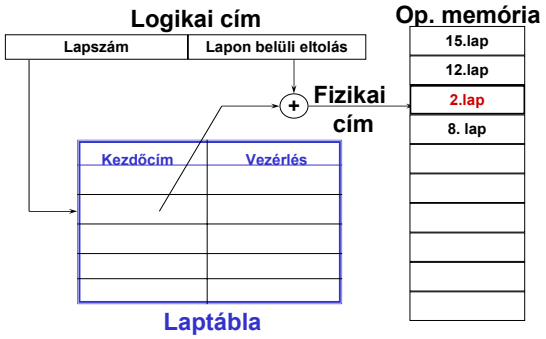
csak a folyamat által igényeltek kerülnek be (igény szerinti lapozás)

Előny:

- **a valós memóriánál nagyobb program**
- **több folyamat futhat egyszerre**
- **gyorsabb a betöltődés is**



## Címszámítás lapszervezésű virtuális tár esetén






## Laphiba (page fault)

**Ha a folyamat olyan memóriacímre hivatkozik, ami olyan lapon található, amely nincs betöltve az operatív memóriába**

**Nem igazi hiba! Virtuális tárkezelésnél természetes!**

**Szükséges lépések:**

- Ellenőrizzük, hogy az adott folyamat használhatja-e az adott címet
- A lapot behozzuk a háttértárról az operatív tárba (természetesen előtte egy lapot lehet, hogy ki kell menteni) és módosítjuk a laptáblát
- Meg kell ismételni annak az utasításnak a végrehajtását, ahol a laphiba fellépett

## A lapkezelés algoritmusa

A laptábla-vezérlés mezőjének tartalma:

Bent van-e a lap az operatív memóriában (present)

Módosítottuk-e (dirty)

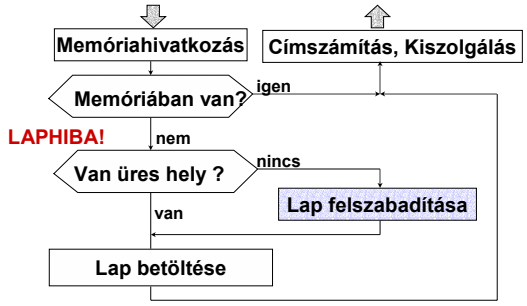
– programlap / adatlap

Információ a lapcseréhez (pl. milyen régóta van bent az adott lap)

Védelmi információ



## A virtuális memóriakezelés






## Lapozási stratégiák
## FIFO - Előbb be, előbb ki

A behozott lapok számait egy FIFO sorban tároljuk

Laphiba esetén a FIFO elején álló lapot cseréljük le

A behozott lap sorszámát a FIFO végére írjuk

Előny: egyszerű

Hátrány: sok laphibát okoz



## FIFO - első példa

Azt a lapot kell lecserélni, amely

- legrégebben van a tárban

| Igényelt lap | 6 | 8 | 3 | 8 | 6 | 0 | 3 | 6 | 3 | 5 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.lap | 6 | 6 | 6 | | | 0 | | 0 | | 0 | 3 |
| 2.lap | | 8 | 8 | | | 8 | | 6 | | 6 | 6 |
| 3.lap | | | 3 | | | 3 | | 3 | | 5 | 5 |
| | * | * | * | | | * | | * | | * | * |

Laphibák száma: 3 + 4



## FIFO - második példa

| 5 | 4 | 3 | 2 | 4 | 5 | 1 | 7 | 4 | 5 | 4 | 3 | 6 | 7 | 3 | 4 | 5 | 4 | 3 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 5 | 5 | 5 | | | | 1 | 1 | 1 | 1 | | 3 | 3 | 3 | | 3 | 5 | | 5 |
| - | 4 | 4 | 4 | | | | 4 | 7 | 7 | 7 | | 7 | 6 | 6 | | 6 | 6 | | 3 |
| - | - | 3 | 3 | | | | 3 | 3 | 4 | 4 | | 4 | 4 | 7 | | 7 | 7 | | 7 |
| - | - | - | 2 | | | | 2 | 2 | 2 | 5 | | 5 | 5 | 5 | | 4 | 4 | | 4 |

FIFO:

5 4 3 2 1 7 4 5 3 6 7 4 5 3

Laphibák száma: 4 + 10



## OPT - optimális stratégia

Az új lapot mindig annak a helyére hozzuk be, amelyre a legkésőbb fogunk (újra)hivatkozni

Előny: ez adja a minimális laphiba számot

Hátrány: a gyakorlatban megvalósíthatatlan, mert nem tudhatjuk előre a hivatkozások sorrendjét

Megvalósíthatatlansága miatt csak az egyéb stratégiák jóságának vizsgálatához való referenciaként használják

## OPT - első példa

**Azt a lapot kell lecserélni, melyre**

- **a legkésőbb lesz szükség**

| Igényelt lap | 6 | 8 | 3 | 8 | 6 | 0 | 3 | 6 | 3 | 5 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.lap | 6 | 6 | 6 | | | 6 | | | | 6 | |
| 2.lap | | 8 | 8 | | | 0 | | | | 5 | |
| 3.lap | | | 3 | | | 3 | | | | 3 | |
| | * | * | * | | | * | | | | * | |

**Laphibák száma: 3 + 2**



## OPT - második példa

| 5 | 4 | 3 | 2 | 4 | 5 | 1 | 7 | 4 | 5 | 4 | 3 | 6 | 7 | 3 | 4 | 5 | 4 | 3 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 5 | 5 | 5 | | | 5 | 5 | | | | | 6 | | | | 5 | | | |
| - | 4 | 4 | 4 | | | 4 | 4 | | | | | 4 | | | | 4 | | | |
| - | - | 3 | 3 | | | 3 | 3 | | | | | 3 | | | | 3 | | | |
| - | - | - | 2 | | | 1 | 7 | | | | | 7 | | | | 7 | | | |

**Laphibák száma: 4 + 4(!)**



## LRU - legrégebben használt

**Az optimális stratégia (egyik) közelítése**

**Az új lapot mindig annak a helyére hozzuk be, amelyre a legrégebben hivatkoztunk (a lokalitási elv alapján erre lesz a legkevésbé valószínűleg szükség a későbbiek során)**

**Előny:** viszonylag jól közelíti az optimálist

**Hátrány:** kiegészítő HW-t igényel

lassú



## LRU - első példa

Azt a lapot kell lecserélni, melyhez

- **a legrégebben fordult a folyamat**

| Igényelt lap | 6 | 8 | 3 | 8 | 6 | 0 | 3 | 6 | 3 | 5 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.lap | 6 | 6 | 6 | | | 6 | 6 | | | 6 | |
| 2.lap | | 8 | 8 | | | 8 | 3 | | | 3 | |
| 3.lap | | | 3 | | | 0 | 0 | | | 5 | |
| | * | * | * | | | * | * | | | * | |

**Laphibák száma: 3 + 3**



## LRU - második példa

| 5 | 4 | 3 | 2 | 4 | 5 | 1 | 7 | 4 | 5 | 4 | 3 | 6 | 7 | 3 | 4 | 5 | 4 | 3 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 5 | 5 | 5 | | | 5 | 5 | | | | | 5 | 5 | 7 | | | | 7 | |
| - | 4 | 4 | 4 | | | 4 | 4 | | | | | 4 | 4 | 4 | | | | 4 | |
| - | - | 3 | 3 | | | 1 | 1 | | | | | 3 | 3 | 3 | | | | 3 | |
| - | - | - | 2 | | | 2 | 7 | | | | | 7 | 6 | 6 | | | | 5 | |

**Laphibák száma: 4 + 6**



## Second Chance (SC) - második esély

**Minden laphoz tartozik egy ”hivatkozás” bit (H) is**
**Ha a lapra hivatkozunk, ezt “1”-be állítjuk**
**(tehát behozáskor is!)**

**Lapcsere esetén azt a lapot vizsgáljuk, ami a FIFO elején van**

- **ha H=0, ez lesz az áldozat**
- **ha H=1, nullázzuk a bitet, és a lap a sor végére áll, és újabb áldozatot keresünk**

**Hátrány:** HW-t igényel és elég bonyolult is

**Előny:** sokkal gyorsabb mint az LRU

## SC - példa

| 5 | 4 | 3 | 2 | 4 | 5 | 1 | 1 | 1 | 1 | 1 | 7 | 3 | 4 | 4 | 5 | 5 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $5_1$ | $5_1$ | $5_1$ | $5_1$ | $5_0$ | $5_0$ | $5_0$ | $5_0$ | $1_1$ | $1_1$ | $1_1$ | $1_1$ | $1_1$ | $1_0$ | $1_0$ | $1_0$ | | |
| - | $4_1$ | $4_1$ | $4_1$ | $4_1$ | $4_0$ | $4_0$ | $4_0$ | $4_0$ | $7_1$ | $7_1$ | $7_1$ | $7_1$ | $7_1$ | $7_0$ | $7_1$ | | |
| - | - | $3_1$ | $3_1$ | $3_1$ | $3_0$ | $3_0$ | $3_0$ | $3_0$ | $3_1$ | $3_0$ | $3_0$ | $3_0$ | $3_0$ | $5_1$ | | | |
| - | - | - | $2_1$ | $2_1$ | $2_1$ | $2_0$ | $2_0$ | $2_0$ | $2_0$ | $4_1$ | $4_1$ | $4_1$ | $4_1$ | | | | |
| * | * | * | * | ! | ! | ! | ! | * | * | ! | ! | * | ! | ! | * | | |

Laphibák száma: 4 + 4



## Not Used Recently (NUR) - Mostanában nem használt

**Az LRU közelítése**

**Az operációs rendszer azok közül a lapok közül választja ki a lecserélendőt, amelyek**

***az előző lapcsere óta***

**NEM MÓDOSULTAK**
**és/vagy**
**NEM HIVATKOZTAK RÁJUK**



## A címszámítás gyorsítása

**A virtuális tárkezelésnél a címszámítás sebessége több mint a felével csökken**

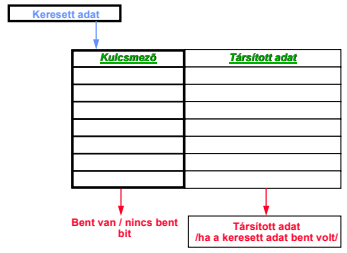
**Gyorsítás igénye**

**Az utoljára használt néhány lap címét egy speciális gyorstárban (TLB) tárolják (asszociatív memória)**

**A laptáblán keresztüli címszámítással párhuzamosan keresünk a TLB-ben is, ha ott megtaláljuk, leállítjuk a laptáblán keresztüli keresést**

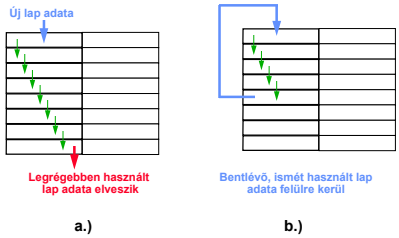


## Asszociatív memória



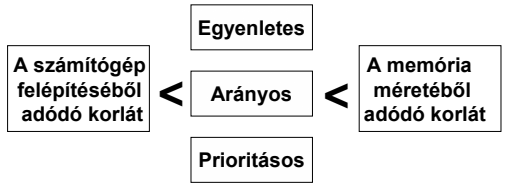



## A TLB működése






## Keretek számának meghatározása




**Lokális stratégia:** A lapok száma futás alatt állandó
**Globális stratégia:** A lapok száma igény szerint változik

## Vergődés

**Egy folyamat több időt tölt lapozással, mint hasznos tevékenységgel (kevés lapja van)**

**Elkerülése:**

**Lokális lapkiosztási algoritmus: Minden folyamatnak rögzített darabszámú lapja van. Így, ha egy folyamat vergődik, a többi (majdnem) szabadon futhat**

**Munkahalmaz: viszonylag lassan változik; az op. r. olyan lapozási stratégiát követ, hogy minden folyamatnak igyekszik bent tartani a munkahalmazát; bonyolult SW és HW igény**

**Laphiba-gyakoriság figyelése: kevés laphibánál a folyamatnak túl sok kerete van, ezért elveszünk tőle egyet, ill. sok laphibánál kevés kerete van, tehát adunk neki még egyet**



## Védelem

**Szegmentálás: bontsuk a programjainkat LOGIKAI egységekre (lapozás: fizikai egységekre tördelés)**

**Ezzel megoldható**

1. **Egy folyamaton belül a logikai egységek védelme egymástól**
2. **A folyamatok védelme egymástól (DE biztosítva az esetleges együttműködés lehetőségét)**
3. **Az operációs rendszer védelme a felhasználóktól**



## Logikai egységek védelme

**Különítsük el a folyamat LOGIKAI egységeit!**
- kód-, adat-, veremszegmens

**Minden utasításnak legyen alapértelmezett szegmense!**
- JUMP kódszegmens;
- MOVE adatszegmens;
- PUSH stack szegmens,

**Az utasítás végrehajtásakor a HW ellenőrizze, hogy tényleg adott típusú szegmensben van-e az operandus.**

**Tárolni kell a szegmens típusát, kezdőcímét és HOSSZÁT**
(Az univerzalitás ellen hat: “prefix utasítások”)



## Folyamatok védelme egymástól

**Lokális leíró tábla (LDT)**

**Legyen minden egyes folyamatnak SAJÁT szegmens leíró táblája, amelyben CSAK az általa elérhető szegmensek vannak feltüntetve, így a memória MÁS területeit egyáltalán nem használhatja az adott folyamat!**

**Globális/Interrupt leíró tábla (GDT, IDT)**

**A közösen használt szegmensek leíróit helyezzük el az összes együttműködni szándékozó folyamat leíró táblájában, DE esetleg különböző jogokkal!**



## Az operációs rendszer védelme

**Prioritási szintek bevezetése:**

**Egy adott szegmenst csak az a folyamat használhat, melynek prioritása minimum megegyezik a szegmens prioritásával**
- szegmensek prioritási szintje: a vezérlés mezőben
- folyamat prioritási szintje: a KÓDSZEGMENSÉHEZ rendelt prioritási érték

**(Természetesen az operációs rendszer folyamatai kerülnek a magas prioritású szintekre, míg a felhasználói folyamatok az alacsonyakra)**



## Tárvédelem

| | |
|---|---|
| **Folyamatok logikai egységeinek védelme egymástól** | **Szegmensleíró tábla, szegmenshossz mező** |
| **Folyamatok védelme egymástól** | **Szegmensleíró tábla létezése** |
| **Operációs rendszer védelme** | **Szegmensleíró tábla védelmi (prioritás) mezői** |

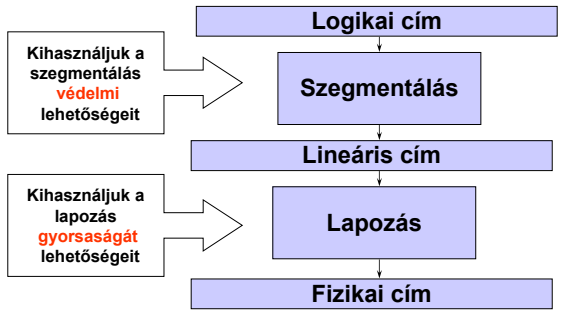
## Lapozás vs. Szegmentálás

**Lapozás:**
- Fizikai darabolás eredménye
- Egyforma méretű lapok
- **Gyors, hardver által végezhető lapcsere**
- **A védelemnek nem sok értelme van**

**Szegmentálás**
- Logikai darabolás eredménye
- Különböző méretű szegmensek
- **Lassabb betöltés/mentés**
- **Védelemhez jól használható**



## Lapozott szegmentálás






## Lapozott szegmentálás - példa

**pl. JMP *valahova* (szegmens: CS, offset: *valahova***
- **A folyamat által adott cím a LOGIKAI cím**
- **A szegmensleíró tábla CS-hez tartozó sorában ellenőrizzük, hogy a folyamat jogosult-e használni a szegmenst, illetve, hogy a valahova cím benne van-e a tartományban.**
- **Kiszámítjuk a szegmens:offset alapján a LINEÁRIS címet (pl. szegmens kezdőcím+offset)**
- **A lineáris cím a lapozó egység bemenete, felosztódik laptábla címre (pl. 6 bit -> 4kB) és lapon belüli offset-re, majd a laptábla által tartalmazott fizikai lapcímet összeillesztve az eltolással, megkapjuk a FIZIKAI címet.**

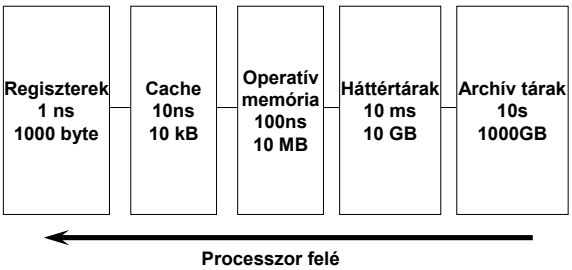


## Gyorsítótár (cache) - elv

**A leggyakrabban használt adatok legyenek a leghamarabb elérhető helyen**



## Tároló hierarchia






## Összefoglalás

**Valóságos tárkezelés**
- Overlay, swapping, particionálás, lapozás

**Virtuális tárkezelés**
- Laphiba, lapkezelés, címszámítás
- Lapcsere stratégiák: FIFO, OPT, LRU, SC, NUR
- A címszámítás gyorsítása

**Tárvédelem**
- A logikai egységek védelme
- A folyamatok védelme egymástól
- Az operációs rendszer védelme

## Párhuzamos programozás

- Párhuzamosság, szinkronizálás, konkurencia
- Precedenciagráf, leíró szerkezetek
  - Fork-join
  - Parbegin-parend szerkezet
  - Szemaforok



## Párhuzamosság, szinkronizálás, konkurencia

```
u1:  x:=12;
u2:  y:=x/4;
u3:  a:=x+y;
u4:  b:=x-y;
u5:  c:=a*b;
u6:  d:=x+1;
u7:  e:=c+d;
```



## Párhuzamosság, szinkronizálás, konkurencia

```
u1:  x:=12;
u2:  y:=x/4;
u3:  a:=x+y;
u4:  b:=x-y;
u5:  c:=a*b;
u6:  d:=x+1;
u7:  e:=c+d;
```

Bizonyos utasítások csak egymás után (szekvenciálisan) hajthatók végre, pl. u1 és u2, hiszen u2 használja u1 eredményét



## Párhuzamosság, szinkronizálás, konkurencia

```
u1:  x:=12;
u2:  y:=x/4;
u3:  a:=x+y;
u4:  b:=x-y;
u5:  c:=a*b;
u6:  d:=x+1;
u7:  e:=c+d;
```

Bizonyos utasítások csak egymás után (szekvenciálisan) hajthatók végre, pl. u1 és u2, hiszen u2 használja u1 eredményét

DE: pl. u3 és u4 egyszerre, egymással párhuzamosan is végrehajtható, hiszen nem függenek egymástól



## Precedenciagráf

Csomópontok: UTASÍTÁSOK

Élek: Abban a csomópontban lévő utasítás, ahonnan egy él indul, MEG KELL, HOGY ELŐZZE azt az utasítást, amely abban a csomópontban található, ahová az él mutat



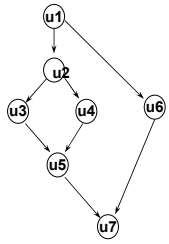
## Szerkesszük meg a precedenciagráfot!

```
u1:  x:=12;
u2:  y:=x/4;
u3:  a:=x+y;
u4:  b:=x-y;
u5:  c:=a*b;
u6:  d:=x+1;
u7:  e:=c+d;
```

# Precedenciagráf

**ha több él találkozik egy csomópontban: szinkronizálás!**

**u1: x:=12;**
**u2: y:=x/4;**
**u3: a:=x+y;**
**u4: b:=x-y;**
**u5: c:=a*b;**
**u6: d:=x+1;**
**u7: e:=c+d;**






**A precedenciagráfok szemléletesen tükrözik a párhuzamosítási lehetőségeket, de programozásra közvetlenül nem használhatók**

1. **megoldás: fork / join utasításpár**
2. **megoldás: parbegin/parend utasításpár**



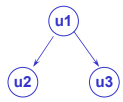
## *Fork / join utasításpár*

**fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található**



## *Fork / join utasításpár*

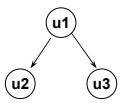
**fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található**

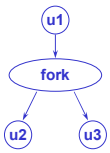





## *Fork / join utasításpár*

**fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található**

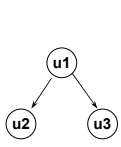


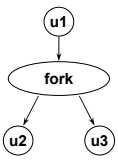






## *Fork / join utasításpár*

**fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található**




```
     u1;
     fork   L;
     u2;
     ...
L:   u3;
```

## Fork / join utasításpár

**join utasítás: két vagy több ág egyesítése;**

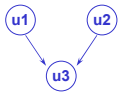
- az ágak "bevárják egymást" (szinkronizáció) és csak a legutoljára "érkező" folytatódik, a többi "meghal"
- az egyesítendő ágak számát egy számláló mutatja



## Fork / join utasításpár

**join utasítás: két vagy több ág egyesítése;**

- az ágak "bevárják egymást" (szinkronizáció) és csak a legutoljára "érkező" folytatódik, a többi "meghal"
- az egyesítendő ágak számát egy számláló mutatja

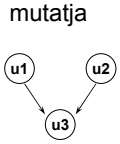





## Fork / join utasításpár

**join utasítás: két vagy több ág egyesítése;**

- az ágak "bevárják egymást" (szinkronizáció) és csak a legutoljára "érkező" folytatódik, a többi "meghal"
- az egyesítendő ágak számát egy számláló mutatja



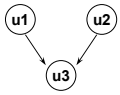


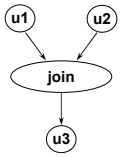


## Fork / join utasításpár

**join utasítás: két vagy több ág egyesítése;**

- az ágak "bevárják egymást" (szinkronizáció) és csak a legutoljára "érkező" folytatódik, a többi "meghal"
- az egyesítendő ágak számát egy számláló mutatja

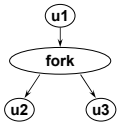


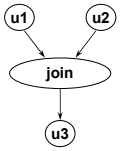

```
    száml:=2;
    u1;
    goto L;
    u2;
L: join száml;
    u3;
```



## Fork / join utasításpár



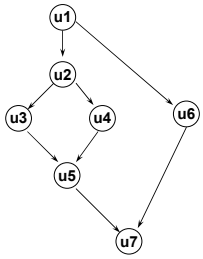

```
    u1;
    fork   L;
    u2;
    ...
L:  u3;
```

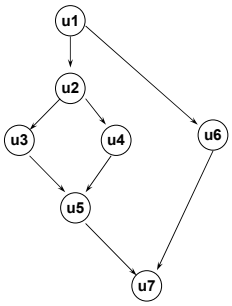



```
    száml:=2;
    u1;
    goto L;
    u2;
L: join száml;
    u3;
```



## Példa a fork/join utasítás használatára






```
        száml2 := 2;
        száml1 := 2;
        u1;
        fork    L1;
        u2;
        fork    L2;
        u3;
        goto    L3;
L2: u4;
L3: join száml1;
        u5;
        goto    L4;
L1: u6;
L4: join száml2;
        u7;
```



## Parbegin / parend utasításpár

**A fork / join utasítások használatával áttekinthetetlen programstruktúra jön létre, a sok "goto" utasítás miatt**

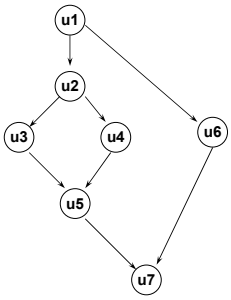
**Megoldás: parbegin / parend utasítások**

**parbegin u1; u2; .... parend;**

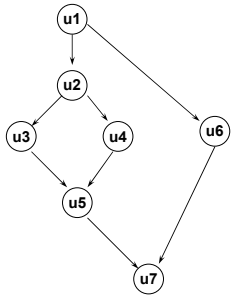
**Jelentése: u1, u2, .... egymással párhuzamosan végrehajtható utasítások**



## Példa a parbegin/parend utasítás használatára



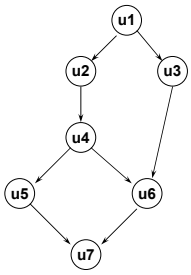



```
u1;
parbegin
    u6;
    begin
        u2;
        parbegin
            u3;
            u4;
        parend;
        u5;
    end;
parend;
u7;
```



## Parbegin / parend utasításpár

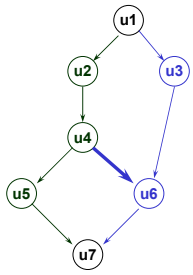



**A parbegin/parend utasításpár nem univerzális**

**Ez a precedenciagráf nem írható le segítségükkel**



## Parbegin / parend utasításpár




**A parbegin/parend utasításpár nem univerzális**

**Ez a precedenciagráf nem írható le segítségükkel**

## A precedenciagráf, a fork/join és a parbegin/parend összehasonlítása

- A legszemléletesebb a precedenciagráf, de programozásra nem alkalmas
- A fork/join segítségével minden leírható, ami precedenciagráffal, de áttekinthetetlen programot eredményez
- A parbegin/parend áttekinthető programot ad, de "kevesebbet tud" mint a precedenciagráf és a fork/join. Ahhoz, hogy egyenrangú legyen velük, még további támogatás kell (szemaforok bevezetése)



- A nagy adminisztrációs igény miatt a párhuzamosítás általában nem utasításszintű, hanem folyamat szintű
- Folyamatok dinamikus létrehozása
  - szülő / gyerek folyamatok
- Végrehajtás:
  - A szülő a gyerekkel párhuzamosan fut
  - A szülő felfüggesztődik és megvárja minden gyerekének befejeződését, csak utána fut tovább
- Erőforrás használat:
  - A közös változókat osztottan használják
  - A gyerek a szülő változóinak csak egy részét kapja (UNIX: változókat nem, csak az állomány hozzáférési jogokat; kommunikáció speciális egységen (pipe))

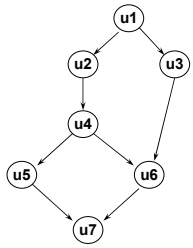


## Folyamatok törlése

- Folyamatok törlése: "KILL foly.azonosító" utasítással
  - folyamat létrehozásakor egyedi azonosítót kell rendelnünk hozzá
- Általában csak a szülő törölheti a gyerekét
- Általában egy folyamat törlése maga után vonja összes gyerekének törlését is



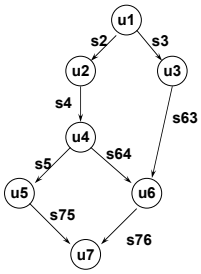
## A parbegin / parend általánosítása szemaforokkal




- Tisztán parbegin/parend utasításokkal nem írható le
- Rendeljünk minden élhez egy-egy átmenetet engedélyező szemafort
- E szemaforok kezdeti értéke "foglalt" kell legyen



## A parbegin / parend általánosítása szemaforokkal




**parbegin**
- begin u1; V(s2); V(s3); end;
- begin P(s2); u2; V(s4); end;
- begin P(s3); u3; V(s63); end;
- begin P(s4); u4; V(s5); V(s64); end;
- begin P(s5); u5; V(s75); end;
- begin P(s64); P(s63); u6; V(s76); end;
- begin P(s75); P(s76); u7; end;

**parend;**



## A parbegin / parend általánosítása szemaforokkal

A szemaforokkal kiegészített megoldás lassabb, mint a tisztán parbegin/parend utasításokat tartalmazó megoldás, ezért csak akkor használjuk, ha feltétlen szükséges!

## *Összefoglalás*

- Párhuzamos folyamatok szinkronizálása
- Precedenciagráf, leíró szerkezetek
- A fork-join utasításpár - előnyök, hátrányok
- A parbegin-parend szerkezet - előnyök, hátrányok
- A parbegin-parend szerkezet általánosítása