# OPERÁCIÓS RENDSZEREK

## *ELLENŐRZŐ KÉRDÉSEK*

## Tartalomjegyzék

## FOLYAMAT KOMMUNIKÁCIÓ

## HOLTPONT

## ÜTEMEZÉS

## TÁRKEZELÉS

## VIRTUÁLIS TÁRKEZELÉS

## HÁTTÉRTÁR KEZELÉS

## Állományok kezelése

## ELOSZTOTT RENDSZEREK

# BEVEZETÉS

## 1. MI AZ OPERÁCIÓS RENDSZEREK CÉLJA?

**Felhasználó kényelme**:

- egyszerű, kényelmes, biztonságos használat
- elsősorban „kis" gépeknél elsődleges
- fontossága növekszik

**Hatékony gépkihasználás**:

- az erőforrások minél jobb kihasználtsága: adott idő alatt minél több program végrehajtása
- nagy gépeknél elsődleges
- fontossága csökken

## 2. ADJA MEG AZ OPERÁCIÓS RENDSZER SZŰKEBB ÉS TÁGABB DEFINÍCIÓJÁT!

**Szűkebb definíció:**

A számítógépen közvetlenül, állandóan futó program (OS magja (kernel)), amely közvetlenül vezérli a gép működését (minden egyéb alkalmazói program).

**Tágabb definíció:**

Az összes program, ami a szállítótól „operációs rendszer"-ként érkezik (gyakorlati feladatokat ellátó programok, minden, ami a gép „általános" felhasználásához szükséges, pl. grafikus felület, editor, számológép stb.).

## 3. ADJA MEG AZ OPERÁCIÓS RENDSZER 3 FŐ FELADATÁT! ADJON RÖVID MAGYARÁZATOT IS!

**Végrehajtási környezet:**

- olyan környezet, ahol a felhasználók és programjaik hasznos munkát végezhetnek
- a számítógép hardver szolgáltatásainak bővítése
- elrejti a „piszkos" részleteket, könnyű felhasználhatóságot biztosít

**Erőforrás kiosztás**

- kezeli a rendszer erőforrásait (CPU, központi tár, merevlemez stb.)
- tulajdonságai: hatékony, biztonságos, igazságos felhasználás

**Vezérlő program**

- vezérli a felhasználói programok működését
- a felhasználói programok helyett vezérli a perifériák működését

## 4. DEFINIÁLJA, HOGY MI AZ A VÉGREHAJTÁSI KÖRNYEZET!

- olyan környezet, ahol a felhasználók és programjaik hasznos munkát végezhetnek
- a számítógép hardver szolgáltatásainak bővítése
- elrejti a „piszkos" részleteket, könnyű felhasználhatóságot biztosít

## 5. MILYEN TULAJDONSÁGOKAT KELL BIZTOSÍTANI AZ OPERÁCIÓS RENDSZERNEK AZ ERŐFORRÁS KIOSZTÁS SORÁN?

- hatékony, biztonságos és igazságos felhasználás

## 6. (MI JELLEMEZTE AZ ELSŐ GENERÁCIÓS RENDSZEREKET?)

- 1945-1955
- ugyanazok az emberek tervezték, készítették és használták a gépeket
- nincs operációs rendszer
- hardver: nagy, drága, megbízhatatlan, plugboard vezérelt CPU, egyszerű, papír alapú perifériák
- használat: programozó = operátor, kézi vezérlés, gépidő foglalás
- mindent mindenki maga csinál ⇨ sok hiba ⇨ hibakeresések miatt rossz kihasználtság
- programozás gépi nyelven
- programok: egyszerű matematikai és számtani problémák (szinusz, koszinusz, logaritmus táblázatok, trajektóriák, adószámítások)
- lyukkártya (plugboard leváltása)

## 7. (MI JELLEMEZTE A MÁSODIK GENERÁCIÓS RENDSZEREKET?)

- 1955-1965
- nincs operációs rendszer
- hardver: kisebb, olcsóbb, megbízhatóbb, egyszerű perifériák, mágnesszalag
- használat: programozó és operátor szétválik
- megkülönböztetünk: tervezőket, építőket, operátorokat, programozókat, karbantartókat
- képzett operátor: gyorsabb gépkezelés, de hibakeresés még nincs
- programozó: papír, lyukkártya, job, papír
- job: program vagy programok halmaza (program, azonosítók, végrehajtási utasítások, fordító)
- programozási nyelvek fejlődése: első fordítóprogramok, programkönyvtárak (pl. I/O műveletek elvégzésére), programozás egyszerűsödik, de a futtatás elkészítésének ideje még mindig nagy

**Hátrány:** a program befejeződésének vagy hiba miatti leállásának vizsgálata az operátor feladata ⇨ lassú reakcióidő

**Kötegelt feldolgozás:**

**Probléma:** sok idő megy el a job összeállításával ⇨ áll a drága CPU

- nincs közvetlen hibakeresés
- offline I/O műveletek
- a programozó futtatáshoz szükséges adatokat, utasításokat rendel a programhoz
- a szakképzett operátor kötegekbe (batch) csoportosítja a munkákat (jobs), ill. azok egyes fázisait (kötegelt feldolgozás)
- egyszerre több job bevitele a gépbe
- a gép sorosan feldolgozza a job-okat

**Mágnesszalag megjelenése:**

- bemenet szalagról (lényegesen gyorsabb, mint a kártya), a másolást kártyáról a szalagra egy másik gép végzi, kimenet hasonló (+ 2 gép)
- I/O műveletek felgyorsulnak, CPU kihasználtság nő
- 3 számítógép párhuzamosan működik
- periféria-független programok, szabványos felületek

**Programok:** tudományos és mérnöki számítások (parciális differenciál egyenletek, könyvelés)

**Merevlemez megjelenése:**

**Egyszerű monitor:** ős operációs rendszer

- a gép vezérlését egy állandóan a memóriában lévő program végzi (operátor perifériákat kezel)
- a munkához rendelt vezérlő információkat a monitor értelmezi, végrehajtja, akár sorrendet is módosíthat a véletlen elérésnek köszönhetően
- egy tevékenység befejezése ⇨ monitor kapja meg a vezérlést és beolvassa a következő munkát (nem kell az operátorra várni)

## 8. MUTASSA BE, HOGY MI AZ A KÖTEGELT FELDOLGOZÁS!

**Probléma:** sok idő megy el a job összeállításával ⇨ áll a drága CPU

- nincs közvetlen hibakeresés
- offline I/O műveletek
- a programozó futtatáshoz szükséges adatokat, utasításokat rendel a programhoz
- a szakképzett operátor kötegekbe (batch) csoportosítja a munkákat (jobs), ill. azok egyes fázisait (kötegelt feldolgozás)
- egyszerre több job bevitele a gépbe
- a gép sorosan feldolgozza a job-okat

## 9. MUTASSA BE, HOGY MI A SPOOLING!

- a lassú perifériák adatait a feldolgozásig a kisméretű puffer helyett mágneslemezen tárolják
- a partícióba a mágneslemezről kerül be
- különböző munkák perifériás és feldolgozási műveletei átlapolódhatnak ⇨ nagyobb az esély a kiegyenlítődésre
- a lemez véletlen hozzáférése lehetővé teszi a munkák sorrendjének megválasztását ⇨ kisebb teljesítményingadozás

## 10. MIK A MULTIPROGRAMOZÁS LÉPÉSEI?

- a rendszer nyilvántartja és tárolja a futtatandó munkákat
- a kiválasztott munka addig fut, amíg várakozni nem kényszerül
- az OS feljegyzi a várakozás okát, majd kiválaszt egy másik futni képes munkát és azt elindítja
- ha a félbehagyott munka várakozási feltételei teljesülnek, akkor azt elindítja, ha szabad a CPU

## 11. MUTASSA BE AZ IDŐOSZTÁSOS RENDSZER FŐBB TULAJDONSÁGAIT!

- több felhasználó (⇨ speciális védelmi mechanizmusok) interaktív kezelése
- közvetlen interaktív kommunikációt biztosít a felhasználó és a programja, ill. az OS között
- adatok tárolása közvetlenül (on-line) az állományrendszerben
- a felhasználó interakciója nagyon lassú, közben az OS más tevékenységet tud végrehajtani
- gyors reakció a parancsokra, válaszidő (response time) kicsi, sűrűn kell a programok között kapcsolgatni
- felhasználók függetlenül használják a gépet, mintha mindenki egy saját gépen dolgozna

## 12. MUTASSA BE, HOGY MI AZ ELOSZTOTT OPERÁCIÓS RENDSZER!

- a számításokat több központi egység között osztják meg egyenletesen (multiprocessing)
- felhasználó ezt nem veszi észre (transzparens)

**Szervezési elvek:**

**Azonosak-e a processzorok:** homogén ↔ inhomogén

**Van-e kitüntetett processzor:** szimmetrikus ↔ aszimmetrikus

**Csatolás:**

- szorosan csatolt: tár egy részét közösen használják
- nem szorosan csatolt: CPU-kat kommunikációs csatorna köti össze (elosztott rendszerek általában ilyenek)

**Kommunikáció:**

- hálózati infrastruktúra kell: LAN/WAN
- architektúra: kliens-szerver/peer-to-peer

## 13. MIK AZ ELOSZTOTT OPERÁCIÓS RENDSZEREK ELŐNYEI?

- **erőforrások megosztása:** felhasználó a másik gép erőforrásait (spec. hardver, perifériák, állományok adatbázisok) használhatja
- **terhelés megosztás:** párhuzamosan végrehajtható részeket szét lehet osztani, megnövelt teljesítmény
- **megbízhatóság:** egy elem kiesése esetén a többi elem átveheti a szerepét (homogén esetén könnyebb, inhomogén esetében nehezebb, „graceful degradation"), redundancia szükséges
- **kommunikáció:** programok, felhasználók közötti információ csere

## 14. MUTASSA BE AZ OPERÁCIÓS RENDSZEREK SZERKEZETÉT!

- OS: nagyméretű, komplex program, amelynek belső szerkezete részekből áll

**Monolitikus szerkezet:**

- nincs struktúra
- az OS szolgáltató eljárások gyűjteménye
- felhasználói programból rendszerhívással: CPU felhasználói módból kernel módba vált (trap)
- másik szolgáltató rutinból hívással: nincs szükség módváltásra

**Rétegszerkezet:**

- nagy program, komplex felépítés
- tovább bontott rétegek, pl.: felhasználói programok, periféria meghajtók, virtuális tár kezelése, I/O csatornák, CPU ütemezése, hardver (tendencia a kevés réteg)
- **tiszta rétegszerkezetű rendszerek:** csak az alatta és felette lévő réteggel kommunikál, a rétegek feladatainak egymásra épülését megnehezíti
- modulok, a rétegen belüli strukturálás: egy-egy jól meghatározott feladatcsoportra egy modul
- a modulok egymás között kommunikálnak, egységes mechanizmus

**Rétegek tulajdonságai:**

- egymásra épülnek
- az alattuk elhelyezkedő réteggel jól definiált felülettel érintkeznek
- egyre bővülő funkcionalitás
- logikus részekre bontás ⇨ legalább 3 réteg: hardver, operációs rendszer, felhasználói programok

**Virtuális gépek:**

- „Virtual Machine Monitor" fut a hardveren, amely több virtuális gépet biztosít a felette lévő réteg számára
- minden virtuális gép egyforma, bit szinten megegyezik a rendszerrel
- szeparáltan valósítja meg a multiprogramozást
- valódi operációs rendszer is kell a virtuális gépre

**Kliens-szerver modell:**

- a kernel méretének csökkentése érdekében sok funkció magasabb rétegekbe kerül
- a folyamatok üzenetküldéssel igényelnek szolgáltatásokat a szolgáltató folyamatoktól
- a szolgáltató folyamatok önállóan linkelhetőek
- ebből az ötletből fejlődött ki az elosztott rendszerek elve

## 15. ADJA MEG A RENDSZERHÍVÁS DEFINÍCIÓJÁT!

A rendszer szolgáltatásainak igénybevétele speciális gépi utasítás segítségével (trap), mely a vezérlést az OS programjának adja (assembly-ből közvetlenül kiadhatók (pl. DOS int010 megszakítás), magasszintű nyelveknél könyvtári eljárások)

## 16. MIK A RENDSZERHÍVÁS LÉPÉSEI?

1. **Paraméter átadás:** az OS által megadott helyre kell elhelyezni
2. **Hívás, hardver működési mód váltás:**
   - felhasználói mód (korlátozott, bizonyos utasítások nem) ⇨ rendszer mód (módváltás)
   - a rendszerhívó gépi utasítás végzi el
3. **Paraméterek kiolvasása, elágazás a hívott szolgáltatáshoz:** a rendszerhívás célját megvalósító kód végrehajtása
4. **Visszaadott paraméterek másolása**
5. **Visszatérés a hívóhoz (módváltás)**
6. **Hívó tovább fut, eredmények felhasználása**

## 17. ADJA MEG A FONTOSABB RENDSZERMODULOKAT!

- folyamatok kezelése
- központi tár kezelése
- perifériák kezelése
- állományok kezelése
- védelmi mechanizmusok
- hálózatok kezelése
- kezelői felület

## 18. ADJON MEG LEGALÁBB 3 OPERÁCIÓS RENDSZER SZOLGÁLTATÁST!

- folyamatok vezérlése
- állományok kezelése
- perifériás eszközök kezelése
- rendszerinformációk kezelése
- kommunikáció

## 19. DEFINIÁLJA, HOGY MIK AZOK A RENDSZERPROGRAMOK, ADJON RÁ NÉHÁNY PÉLDÁT IS!

- az OS-hez adott, annak általános célú felhasználását támogató programok
- pl.: parancsértelmező, állományrendszer kezelő, szövegszerkesztő, fordító program, linker, loader, könyvtár kezelő, kommunikációs programok (pl. levelezés)

# FOLYAMATOK

## 20. ADJA MEG A FOLYAMAT DEFINÍCIÓJÁT!

A multiprogramozott OS rendszerek alapfogalma: végrehajtás alatt álló program.

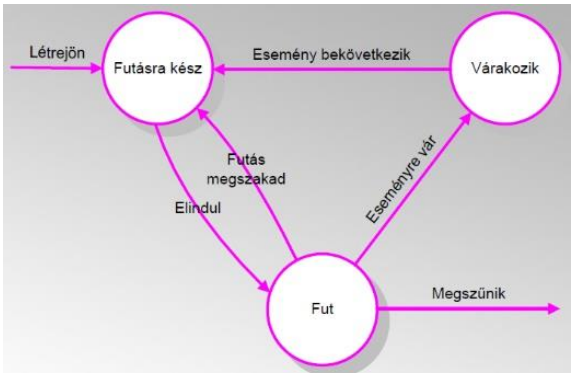
## 21. SOROLJA FEL A MULTI PROGRAMOZOTT RENDSZERBEN A FOLYAMATOK LEHETSÉGES ÁLLAPOTAIT, RÖVIDEN ÍRJA IS KÖRÜL ŐKET!

- **fut:** a CPU a folyamathoz tartozó utasításokat hajtja végre (CPU-nként egyetlen ilyen folyamat lehet)
- **várakozik, blokkolt:** a folyamat várakozni kényszerül, működését csak valamilyen esemény bekövetkezésekor tudja folytatni (több ilyen is lehet a rendszerben)
- **futásra kész:** minden feltétel adott, de a CPU éppen foglalt (több ilyen is lehet a rendszerben)

## 22. SOROLJA FEL A MULTI PROGRAMOZOTT RENDSZERBEN A FOLYAMATOK LEHETSÉGES ÁLLAPOTÁTMENETEIT!

- folyamat létrehozása
- folyamat befejeződése
- eseményre vár
- esemény bekövetkezik
- folyamat elindul
- futás megszakad

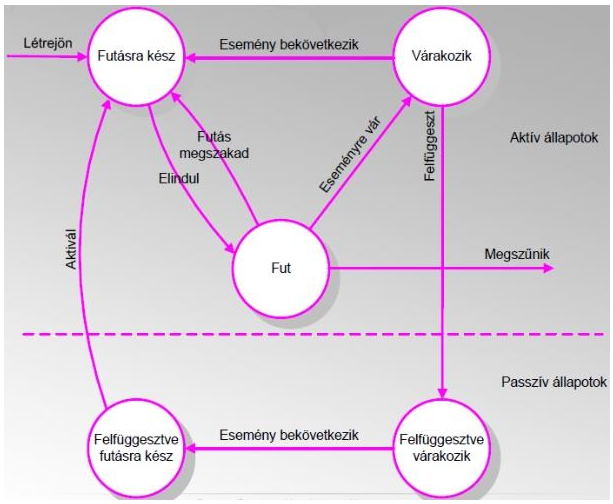
## 23. RAJZOLJA FEL A FOLYAMATOK ÁLLAPOT ÁTMENETI GRÁFJÁT!




## 24. RAJZOLJA FEL A FOLYAMATOK BŐVÍTETT ÁLLAPOT ÁTMENETI GRÁFJÁT ÉS ÍRJA KÖRÜL A BŐVÍTÉSHEZ KÖTŐDŐ PLUSZ ÁLLAPOTOKAT ÉS ÁLLAPOT ÁTMENETEKET! MI A CÉLJA A BŐVÍTÉSNEK?

**Bővítés:** az OS felfüggeszthet folyamatokat (középtávú CPU ütemezés)

- a rendszer túl van terhelve (sok program vetélkedik a futás jogáért, vagy a tár túlzottan megtelt stb.)
- vészhelyzet esetén
- felhasználó kezdeményezésére
- a felfüggesztett folyamatok erőforrásaikat elvesztik (de a rendszer számon tartja őket és később folytatódhatnak)




**Két új állapot:**

- felfüggesztve vár
- felfüggesztve futásra kész

**Új állapotátmenetek:**

**Felfüggeszt:**

- OS felfüggeszti (futásra kész vagy várakozik állapotból)
- a felfüggesztett folyamattól a fontosabb (és elvehető) erőforrásokat elveszi (pl. memória), de néhányat megtarthat (pl. nyomtató)

**Aktivál:** erőforrásokat visszaad

**Felfüggesztve várakozik ⇨ felfüggesztve futásra kész:** esemény bekövetkezik, de CPU-t nem kaphat

Felfüggesztve vár ⇨ várakozik: az átmenetnek nincs értelme (tovább várakozna, de lekötné az erőforrásokat)

## 25. MI A RÖVIDTÁVÚ CPU ÜTEMEZÉS CÉLJA? HOGYAN MŰKÖDIK?

**Cél:** annak meghatározása, hogy mely futásra kész folyamat kapja meg a CPU-t

**Működés:**

- gyakran fut, ezért gyorsnak kell lennie, különben a rendszer túl sok időt töltene az ütemezéssel, elvéve a CPU-t a folyamatoktól
- az ütemező mindig a tárban van, része az OS magjának

## 26. MI A KÖZÉPTÁVÚ CPU ÜTEMEZÉS CÉLJA? HOGYAN MŰKÖDIK?

**Cél:** a rendszer időszakos terhelésingadozásainak kiegyenlítése az egyes folyamatok felfüggesztésével ill. újraaktiválásával

**Működés:**

- felfüggesztés esetén a folyamat a háttértáron tárolódik, megfosztják elvehető erőforrásaitól
- az ilyen folyamatok nem versengenek tovább az erőforrásokért

## 27. MI A HOSSZÚ TÁVÚ CPU ÜTEMEZÉS CÉLJA? HOGYAN MŰKÖDIK?

**Cél:** annak meghatározása, hogy a háttértáron várakozó, még el nem kezdett munkák közül melyek kezdjenek futni

**Működés:**

- ritkán kell futnia ⇨ nem kell, hogy gyors legyen
- egy munka befejeződésekor választunk ki egy új elindítandót

**Követelmény:**

- olyan munka-halmaz (job-mix) előállítása, mely a rendszer erőforrásait kiegyensúlyozottan használja
- CPU-korlátozott és periféria-korlátozott munkák egyenletesen forduljanak elő

## 28. ADJA MEG, HOGY MI AZ A KÖRNYEZETVÁLTÁS, ÍRJA LE, HOGY MIT KELL AZ OPERÁCIÓS RENDSZERNEK CSINÁLNI A KÖRNYEZETVÁLTÁS SORÁN!

**Környezetváltás (context switch):** a futó folyamat elhagyja a futó állapotot, egy futásra kész pedig elindul

**OS teendői:** állapotleírókat meg kell őrizni, hogy az átkapcsolás zökkenőmentes legyen: folyamat és végrehajtó gép állapota

- multiprogramozásnál fontos a gyors átkapcsolás (hatékonyság)

**Folyamat állapota:**

**Állapotváltozók:**

- programkód, változók aktuális értéke, verem tartalma
- programszámláló: a végrehajtó gép tartalmazza

- tárban több folyamat van, így a folyamatok állapotleírói megmaradnak a tártartalom megőrzésével ⇨ csak a végrehajtó gép állapotát kell menteni

**Végrehajtó gép állapota:**

- a végrehajtó gép a folyamat környezete, a gép állapotleíróinak összességét kontextusnak nevezzük
- a rendszer rétegszerkezete miatt több szintű (legegyszerűbb eset, amikor a folyamat kódja csak gépi utasításokat tartalmaz ⇨ hardver-szoftver határfelület)
- legtöbbször az OS alapszolgáltatásait is tartalmazó virtuális gép határfelülete
  - CPU regiszterek
  - OS változók: rendszertáblák, memóriakezelési információk, periféria hozzárendelések stb.

## 29. Mutassa be mi a folyamat leíró blokk (PCB), adja meg, hogy mit tartalmaz!

- speciális adatszerkezet, amely az OS-nek a folyamatok kezeléséhez szükséges adatait tárolja

**Tartalma:**

- folyamat azonosítója
- szülők, gyerekek azonosítója
- folyamat állapota
- folyamathoz tartozó összes tárterület leírása (mutatók, virtuális tárkezeléshez tartozó adatok, cím transzformáció)
- a folyamat által használt egyéb erőforrások leírása (pl. nyitott állományok)
- regiszterek tartalma
- várakozó folyamatoknál: várt esemény leírása
- ütemezéshez információk (prioritás, várakozási idő)
- statisztikák

## 30. Mutassa be mi az I/O leíró blokk (IOCB), adja meg, hogy mit tartalmaz!

- speciális adatszerkezet, amely az OS-nek az I/O egységek kezeléséhez szükséges adatait tárolja

**Tartalma:**

- művelet kijelölése (írás, olvasás, …)
- tárterület címe (ahonnan, ahova a művelet végrehajtandó)
- I/O készülék egyéb adatai (mágneslemez szektorcíme stb.)
- átviendő adatmennyiség
- állapotjelző

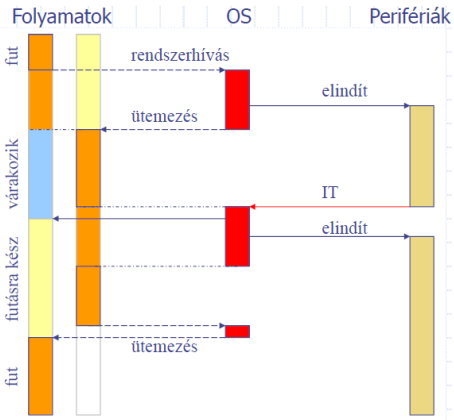
## 31. Mik az I/O műveletek végrehajtásának lépései (mind a folyamat, mind a periféria szemszögéből)?

**Folyamat:**

- kitölti az IOCB-t
- rendszerhívás, paraméter az IOCB

**OS**

- láncolja az IOCB-t a PCB-hez
- PCB-t befűzi a periféria várakozási sorába
- ha a sor üres, indítja a perifériát az IOCB paramétereivel
- folyamatot várakozó állapotba teszi
- újraütemez és visszatér (másik folyamatra)




**Periféria**

- feladat végeztével megszakítást okoz

**OS**

- IOCB-be a végrehajtás eredményére utaló jelzést ír (helyes/helytelen)
- PCB-t a futásra kész állapotba helyezi
- ha van még várakozó a perifériára, akkor a perifériát indítja
- újraütemez
- visszatér

## 32. ADJA MEG A SZÁL DEFINÍCIÓJÁT, MUTASSA BE A SZÁL ÉS A FOLYAMAT KÖZÖTTI KÜLÖNBSÉGET!

**Szál (thread)/pehelysúlyú folyamat (lightweight process):**

- folyamatokhoz hasonló fogalom
- kevesebb adat kell a kezeléséhez: nincs saját memória, saját erőforrás
  - csak a regiszterek és a verem sajátja (+állapot)
  - minden más közös a folyamatával/más szálakkal
  - futási környezete egy folyamat, amely tulajdonosa az erőforrásoknak, amit közösen használ a többi szállal
- állapot-átmeneti gráfja megegyezik a folyamatoknál tárgyalttal

**Előny:**

- gyors információcserét és környezetváltást tesz lehetővé
- osztott erőforrások (memória!) lehetősége

**Szál ↔ folyamat:**

- **minden folyamatnak van:** címtér, globális változók, megnyitott fájlok, gyermek folyamatok, várt események listája stb.
- **minden szálnak van:** programszámláló, regiszterek, stack, állapot

## 33. MI A MEGSZAKÍTÁS? ADJA MEG A MEGSZAKÍTÁS KEZELÉSÉNEK LÉPÉSEIT!

- olyan esemény, jelzés, mely a folyamatok működését befolyásolja
- nagy erőforrás-igényű pollozás-t váltja ki
- pollozás: egy adott jellemző változásának figyelése az adatok folyamatos újrakérésével és az előző állapottal történő összehasonlításával
- prioritása van
- nagyobb prioritású megszakítás megszakíthatja egy kisebb prioritású kiszolgálását
- a megszakításokat a rendszer letilthatja, de csak rövid időre, mert ezek fontos eseményeket jeleznek (pl. egyes perifériákat adott idő alatt ki kell szolgálni, különben elvész az információ)

**Osztályok:**

- periféria megszakítások (átvitel befejezése, periféria állapotának megváltozása stb.)
- belső hardver megszakítások (belső óramegszakítás stb.)
- utasítás végrehajtási hibák (nullával való osztás, memória hiba, virtuális tárkezelésben laphiba stb.)
- hardver hiba (tápfeszültség kimaradás stb.)
- szoftver megszakítások (speciális gépi utasítások, pl. rendszerhívások stb.)

**Kezelés:**

1. **Nincs környezet váltás:** megszakítás hatására OS rutin indul el, folyamat felfüggesztve, csak a kiszolgáló rutin által használt regisztereket mentjük el (ezek általában rövid kis programok)
2. **Van környezet váltás** (ritkábban, lassabb): a megszakítás elindít egy arra váró folyamatot, amely várakozóból rögtön futni fog (elkerüli a futásra kész állapotot)

**Kezelés lépései:**

- futó folyamat megszakad, a vezérlést az OS kapja meg
- megszakított folyamat állapotmentése (pl. regiszterek mentése (hardveres támogatás is van))
- vezérlést a kiszolgáló rutin kapja meg
- befejezés után állapot visszaállítás
- a megszakított (vagy egy másik) folyamat folytatja a működést

# FOLYAMAT KOMMUNIKÁCIÓ

## 34. FOLYAMAT KOMMUNIKÁCIÓHOZ KAPCSOLÓDÓAN DEFINIÁLJA, HOGY MIK A FÜGGŐ ÉS A FÜGGETLEN FOLYAMATOK!

**Függő folyamatok:**

- logikailag független folyamatok, de megosztott erőforrás használat (pl. több user azonos gépen dolgozik)
- logikailag is függő folyamatok: közösen oldanak meg valamely feladatot (együttműködnek, kommunikálnak, közös változók stb.)

**Független folyamatok:**

- egymás működését nem befolyásolják,
- aszinkron működés: egymással párhuzamosan is végrehajtódhatnak, nincs időbeli függőség

## 35. MI INDOKOLJA, HOGY EGYÜTTMŰKÖDŐ FOLYAMATOKAT HASZNÁLJUNK?

- **erőforrások megosztása:** átlapolt működés, jobb kihasználtság
- **számítások felgyorsulása (több processzor):** számítások párhuzamosítása, végrehajtási sebesség nő
- **felhasználók kényelme:** egy időben több feladat megoldása
- **modularitás:** egy adott folyamat kisebb részekre való bontása
  - jobb áttekinthetőség
  - bizonyos feladatoknál (párhuzamos részek) kézenfekvő modell
  - pl. vezérlés, folyamatirányítás
  - a függőséget valahogy biztosítani kell

## 36. ADJA MEG A SZINKRONIZÁCIÓ DEFINÍCIÓJÁT! MIK AZ SZINKRONIZÁCIÓ ALAPTÍPUSAI?

**Szinkronizáció:** a folyamat végrehajtásának olyan időbeli korlátozása, ahol ez egy másik folyamat futásától, ill. egy külső esemény bekövetkezésétől függ

**Alaptípusai:**

- precedencia (előidejűség)
- egyidejűség
- kölcsönös kizárás (versenyhelyzet, kritikus szakasz)

## 37. MI A PRECEDENCIA? ADJON RÁ PÉLDÁT!

- meghatározott sorrend biztosítása
- egy $P_k$ folyamat $S_k$ utasítása csak akkor mehet végbe, ha a $P_i$ folyamat $S_i$ utasítása már befejeződött
- pl: szakács-kávét főz ⇦ kukta-cukrot vesz (különben kihűl a kávé)

## 38. MI AZ EGYIDEJŰSÉG? ADJON RÁ PÉLDÁT!

- két vagy több folyamat bizonyos utasításait ($S_k$; $S_j$) egyszerre kell elkezdeni
- két folyamat találkozása (randevú)
- két folyamat bevárja egymást mielőtt további működését elkezdené
- pl. szakács-kávét főz || kukta-habot ver (különben kihűl a kávé vagy összeesik a hab)

## 39. MI A KÖLCSÖNÖS KIZÁRÁS? ADJON RÁ PÉLDÁT!

- a résztvevő folyamatok utasításainak sorrendjére nincs korlátozás, de egy időben csak egyik futhat
- pl. szakács-habot ver X kukta-habverőt mosogat

## 40. ADJA MEG A VERSENYHELYZET DEFINÍCIÓJÁT! ADJON RÁ PÉLDÁT!

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

## 41. DEFINIÁLJA, HOGY MI A KRITIKUS SZAKASZ, ADJON RÁ PÉLDÁT!

- a program olyan (általában osztott változókat használó) utasításszekvenciái, amelyeknek egyidejű (párhuzamos) végrehajtása nem megengedett
- versenyhelyzet elkerülésére a kritikus szakaszok kölcsönös kizárását biztosítani kell (ha az egyik folyamat már a kritikus szakaszában van, akkor más folyamat nem léphet be a (természetesen saját) kritikus szakaszába)
- pl. közös memória használata

## 42. ADJA MEG, HOGY MIK A KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁNAK KRITÉRIUMAI!

- **biztosítsa a kölcsönös kizárást:** egy időben csak egyetlen folyamat hajthatja végre a kritikus szakaszban lévő utasításokat
- **haladjon:** ha nincs folyamat a kritikus szakaszban, de van több belépő, akkor az algoritmus ezek közül véges idő alatt kiválaszt egyet és beengedi a kritikus szakaszba
- **folyamat várakozása korlátozott legyen (ne éheztessen):** csak véges számú esetben előzhetik meg (nem veszik annyira szigorúan (nem minden algoritmus teljesíti)

## 43. SOROLJON FEL KÜLÖNBÖZŐ MEGOLDÁSOKAT KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁRA!

- **interrupt tiltása**
- **busy waiting megoldások:**
  - nincs HW támogatás, **tiszta SW megoldás**: naiv megközelítés (ROSSZ), strict alternation (JOBB). Peterson algoritmusa (HELYES)
  - **HW támogatással:** TestAndSet, Swap
- **szemafor**
- **magas szintű módszerek**

## 44. MUTASSA BE A KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁT INTERRUPT TILTÁSÁVAL, ADJA MEG AZ ELŐNYÖKET ÉS A HÁTRÁNYOKAT!

```
disable_interrupt();
critical_region();
enable_interrupt();
```

- egyszerű megoldás: nincs IT ⇨ nem lehet a végrehajtást megszakítani (ütemező sem tud futni)
- probléma: a folyamatnak joga van letiltani az IT-t, amely az egész rendszert lefagyaszthatja
- hasznos az OS szintjén, nem célszerű a felhasználói szinten

## 45. MUTASSA BE A KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁT SZOFTVERES MÓDSZERREL, ADJA MEG AZ ELŐNYÖKET ÉS A HÁTRÁNYOKAT!

**Naiv megközelítés:**

```
//Process 0                          //Process 1
while (TRUE){                        while (TRUE){
   while (flag!=0) /*loop*/;            while (flag!=0) /*loop*/;
   flag = 1;                            flag = 1;
   critical_region();                   critical_region();
   flag = 0;                            flag = 0;
   non_critical_region();               non_critical_region();
}                                    }
```

- közösen használt változókon (flag) alapul
- egy foglaltsági bitet használ, amit a belépni kívánó folyamat tesztel
- gond: a változót többen is olvashatják, mielőtt az első foglaltra állítja

**Strict alternation (szigorú váltás):**

```
//Process 0                          //Process 1
while (TRUE){                        while (TRUE){
   while (turn!=0) /*loop*/;            while (turn!=1) /*loop*/;
   critical_region();                   critical_region();
   turn = 1;                            turn = 0;
   non_critical_region();               non_critical_region();
}                                    }
```

- közös változó: `turn` (jelentése: ki van soron)
- csak felváltva lehet belépni.
- gond: ha jelentős a sebességkülönbség, akkor az egyik feleslegesen sokat vár ⇨ nem halad

**Peterson algoritmusa:**

```
define FALSE 0
define TRUE 1
define N 2 /* a folyamatok száma */
int turn; /* ki van soron? */
int interested[N]; /* kezdetben csupa 0 */

void enter_region(int process); /* a folyamat belép a kritikus szakaszba*/
{
   int other;
   other = 1 - process; /* a másik folyamat */
   interested[process] = TRUE; /* az érdeklődés jelzése */
   turn = process; /* flag beállítása */
   while (turn == process && interested[other] == TRUE) /* nop */;
}

void leave_region(int process); /* a folyamat elhagyja a kritikus szakaszt */
{
   interested[process] = FALSE;
}
```

- ha csak egy folyamat akar belépni, az `enter_region()` visszatér és a folyamat a kritikus szakaszba léphet
- a másik folyamat addig nem léphet be, amíg a `leave_region()` az `interested` változót nem törli
- ha két folyamat egyszerre akar belépni, a `turn` változót utoljára állító folyamat várakozik, míg a másik végre nem hajtja a saját `leave_region()` eljárását.
- megjegyzés: a `turn` változó írásának megszakíthatatlanak kell lenni (ez könnyen teljesíthető)

## 46. MUTASSA BE A KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁT HARDVER TÁMOGATÁSSAL, ADJA MEG AZ ELŐNYÖKET ÉS A HÁTRÁNYOKAT!

- speciális megszakíthatatlan gépi utasítások
- megkönnyíti a kritikus szakaszok megvalósítását
- nem garantálják az éhezés kiküszöbölését

**TestAndSet:** kiolvassa és visszaadja a bit értéket, majd azonnal 1-be állítja

```
int TestAndSet(int *flag){
   tmp = *flag;
   *flag = 1;
   return(tmp)
}

//init
flag = 0;
...

//Process i
while (TRUE){
   while (TestAndSet(&flag)!=0) /*empty loop*/;
   critical_region();
   flag = 0; /*exit critical section*/
   non_critical_region();
}
```

**Swap:** két változó értékét cseréli fel

```
void swap(int *a, int *b){
   tmp = *a;
   *a = *b;
   *b = tmp;
}

//init
flag = 0;
...

//Process i
...
while (TRUE){
   myflag = 1;
   while (myflag!=0) swap(&myflag, &flag);
   critical_region();
   flag = 0; /*exit critical section*/
   non_critical_region();
}
```

## 47. DEFINIÁLJA, HOGY MI A SZEMAFOR, MUTASSA BE RÖVIDEN A FÜGGVÉNYEKET!

- egy speciális adattípus, amely nullát, ill. pozitív egész számokat tartalmazhat és két elemi, megszakíthatatlan művelet van értelmezve rajta

**P(s):** vizsgál/belép művelet

```
while s < 1 do üres utasítás;
s := s - 1
```

**V(s)**: kilép művelet

```
s := s + 1
```

## 48. MUTASSA BE A SZINKRONIZÁCIÓ KÜLÖNBÖZŐ TÍPUSAINAK MEGVALÓSÍTÁSÁT SZEMAFOR SEGÍTSÉGÉVEL!

**Előidejűség** ($U_1$ előbb, mint $U_2$):

```
s := 0
P1 : ... U1; V(s); ...
P2 : ... P(s); U2; ...
```

**Randevú** ($U_1$ és $U_2$ egyszerre):

```
s1 := 0; s2 := 0
P1 : ... V(s1); P(s2); U1; ...
P2 : ... V(s2); P(s1); U2; ...
```

**Kölcsönös kizárás:**

```
s := 1
P1 : ... P(s); U1; V(s); ...
P2 : ... P(s); U2; V(s); ...
```

- az *s* kezdő értékétől függ, hogy hány folyamat lehet egyszerre a kritikus szakaszban
- legtöbbször bináris szemafort (mutex) használunk

## 49. MUTASSA BE A SZEMAFOR MŰKÖDÉSÉT MULTIPROGRAMOZOTT RENDSZERBEN!

- a P(s) az üres utasítás helyén a folyamattól elveszi a CPU-t és várakozó állapotba teszi (feljegyzi a szemaforhoz tartozó valamilyen adatszerkezetekbe)
- a V(s)-nél nem csak a számláló nő, hanem egy folyamatot (ha van) futásra kész állapotba tesz (többféle stratégia lehet)

## 50. ADJA MEG A SZEMAFOR NEM BUSY WAITING MEGVALÓSÍTÁSÁNAK PSEUDO KÓDJÁT!

```
type semaphore = record
                    value: integer
                    list : list of process
                 end; s
```

**P(s):**

```
s:value := s:value - 1
if s:value < 0 then
begin
   a folyamat felfűzése a s.list-re
   a folyamat felfüggesztése, újra ütemezés
end
```

**V(s):**

```
s:value := s:value + 1
if s:value <= 0 then
begin
   leveszünk egy folyamatot s:list -ről
   felébresztjük ezt a folyamatot
end
```

## 51. MUTASSA BE A KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁT SZEMAFOR SEGÍTSÉGÉVEL, ADJA MEG AZ ELŐNYÖKET ÉS A HÁTRÁNYOKAT!

- aktív várakozás: erőforrások pazarlása
- passzív várakozás: korrekt megoldás, a szemafort nem lehet a felébresztett folyamattól ellopni

**Hátrányok:**

- túl alacsonyszintű eszköz
- nehéz megbízhatóan programozni (sok hibalehetőség, nehéz a hibakeresés)

## 52. MUTASSA BE A KRITIKUS SZAKASZ MEGVALÓSÍTÁSÁNAK MAGAS SZINTŰ MÓDSZEREIT!

- léteznek magas szintű programozási nyelvekben olyan programszerkezetek, amelyek a kritikus szakasz automatikus, megbízható megvalósulását támogatják
- magas szintű programozási nyelvet használva a fordítóprogram ellenőrzi a programot, kiszűrheti a szinkronizációs hibák jelentős részét

**(Feltételes) kritikus szakasz:**

- egy `T` típusú `v` változóhoz kijelölhetünk összetett utasításokat

```
var v: shared T;
region v do s;
```

  - `s` utasítás végrehajtása közben más folyamat nem férhet hozzá `v` változóhoz
- logikai feltételt is rendelhetünk hozzá

```
region v when b do s;
```

  - folyamat csak akkor léphet be kritikus szakaszba (hajthatja végre az `s` utasítást), ha `b` feltétel igaz

**Monitor:**

- egy magas szintű szinkronizációs primitív
- egy programszerkezet, amelybe a programozó programrészleteket zárhat
- eljárások, változók, adatszerkezetek speciális gyűjteménye
- eljárásai szabadon hívhatóak, de a változókhoz nem lehet kívülről közvetlenül hozzáférni
- egyszerre csak egy eljárás lehet aktív a monitoron belül, a többi a monitorba lépésnél várakozik

**Akadály (barrier):**

- több folyamat szinkronizálására szolgál
- az akadályt elérő folyamatok blokkolódnak
- az összes folyamat megérkezésekor az akadály ledől, az összes folyamat egyszerre folytatja futását

## 53. MILYEN MÓDON TÖRTÉNHET INFORMÁCIÓCSERE EGYÜTTMŰKÖDŐ FOLYAMATOK KÖZÖTT?

- közös tárterületen keresztül (szorosan csatolt) (kölcsönös kizárás témaköre)
- kommunikációs csatornán keresztül (lazán csatolt)

## 54. MI A FOLYAMATOK MEGNEVEZÉSE? MIK AZ ALAPVETŐ TÍPUSAI?

- az a módszer, amellyel a kommunikációban résztvevő folyamatok egymásra hivatkoznak

**Alapvető típusok:**

**Közvetlen kommunikáció:**

- folyamatok: P, Q
- a kommunikációban résztvevő mindkét folyamat megnevezi a másikat
- üzenet küldés: send (Q; message);
- üzenet fogadás: message=receive (P);
- csak egy csatorna létezik, és más folyamatok nem használhatják

**Közvetett kommunikáció:**

- közbülső adatszerkezeten (pl. postaládán (mailbox)) keresztül valósul meg
- küldés: send (mailbox, message);
- fogadás: message=receive (mailbox);
- két folyamat között lehet több postaláda
- ugyanazt a postaládát több folyamat is használhatja egyidejűleg
- a postaláda megosztása a vevők között:
  - csak két folyamat használhatja
  - egy időben csak egy vevő lehet
  - a rendszer választ, hogy melyik vevőnek küldi az üzenetet és erről értesíti az adót

**Aszimmetrikus megnevezés:**

- adó vagy vevő megnevezi, hogy melyik folyamattal akar kommunikálni
- a másik fél egy kaput (port) használ, ezen keresztül több folyamathoz, is kapcsolódhat
- ha minden folyamathoz csak egy kapu tartozik, akkor ennek megnevezése szükségtelen
- az adó információt kap arról, hogy a kommunikáció másik végén melyik folyamat található
- tipikus eset: a vevőhöz tartozik a kapu, az adóknak kell a vevő folyamatot és annak a kapuját megnevezni (pl. szerver, szolgáltató folyamat)
- küldés: send (Q:port ,message);
- fogadás: message=receive ();

**Üzenetszórás:**

- a közeg több folyamatot köt össze (pl. lokális hálózatok)
- az üzenetet több folyamat, esetleg mindegyik veheti
- küldés: send (message, to_all);
- fogadás: message=receive ();

## 55. MI AZ IMPLICIT SZINKRONIZÁCIÓ?

- folyamatok között információcsere implicit szinkronizációval jár, ennek típusa a puffertől függ
- **tárolás nélküli átvitel:** nincs tárolás, így az adás és vétel egy időben zajlik (randevú)
- **véges kapacitású tároló:**
  - egy időben csak az egyik folyamat használhatja (kölcsönös kizárás)
  - vevő várakozik, amíg legalább egy elem nincs elküldve
  - adó várakozik, ha a puffer megtelt
  - a két folyamat utasításai között adott sorrendiségnek kell teljesülni (precedencia)
- **"végtelen" puffer:**
  - az adó folyamat nem várakozhat (pl. külső folyamat, amelyre az OS-nek nincs hatása)
  - a valóságban természetesen a puffer véges, így megtelhet
  - ilyenkor információvesztés: új adat figyelmen kívül hagyása vagy régi adat felülírása a pufferben

# HOLTPONT

## 56. ADJA MEG A HOLTPONT DEFINÍCIÓJÁT!

Folyamatok egy halmazának minden folyamata olyan eseményre (többnyire egy erőforrás felszabadulására) vár, amelyet csak egy másik, ugyancsak halmazbeli (várakozó) folyamat tudna előidézni.

## 57. ADJA MEG A HOLTPONT VIZSGÁLATÁNÁL HASZNÁLT RENDSZERMODELLBEN AZ ERŐFORRÁS HASZNÁLAT LÉPÉSEIT ÉS AZONOSÍTSA AZOKAT, AMELYEK RENDSZERHÍVÁSOKAT HASZNÁLNAK!

1. **Igénylés:** ha az igény nem teljesíthető (erőforrás foglalt), akkor a folyamat várakozik
2. **Felhasználás:** a folyamat az erőforrást kizárólagosan használja
3. **Felszabadítás:** a folyamat elengedi az erőforrást és ha más folyamatok várakoznak rá, akkor valamelyik várakozó folyamat továbbléphet

- 1. és 3. lépések általában rendszerhívások: request - release (device), open - close (file), allocate - free (memory)

## 58. ADJA MEG A HOLTPONT KIALAKULÁSÁNAK SZÜKSÉGES FELTÉTELEIT!

**Kölcsönös kizárás:** egy erőforrást egyszerre legfeljebb egy folyamat használhat

**Foglalva várakozás:** van olyan folyamat, amely lefoglalva tart erőforrásokat, miközben más erőforrásokat kér (és azokra várakozik)

**Nem elvehető erőforrások vannak a rendszerben:** erőforrást a folyamat csak önszántából szabadíthat fel

**Körkörös várakozás:** létezik egy olyan $\{P_0, P_1, \ldots, P_n\}$ sorozat, amelyben $P_0$ egy $P_1$ által lefoglalva tartott erőforrásra vár, $P_i$ egy $P_{i+1}$ által foglaltra, végül $P_n$ pedig egy $P_0$ által foglalt erőforrásra vár

## 59. ADJA MEG A HOLTPONTHOZ KAPCSOLÓDÓAN A KÖRKÖRÖS VÁRAKOZÁS DEFINÍCIÓJÁT!

Létezik egy olyan $\{P_0, P_1, \ldots, P_n\}$ sorozat, amelyben $P_0$ egy $P_1$ által lefoglalva tartott erőforrásra vár, $P_i$ egy $P_{i+1}$ által foglaltra, végül $P_n$ pedig egy $P_0$ által foglalt erőforrásra vár.

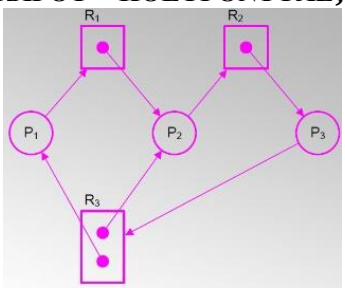
## 60. MI AZ ERŐFORRÁS HASZNÁLATI GRÁF, MIK AZ ELEMEI?

- a rendszer pillanatnyi állapotát leíró gráf

**Elemei:**

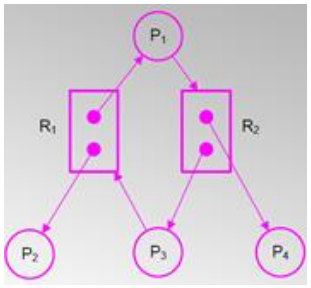
- $P = \{P_1, P_2, \ldots, P_n\}$: a rendszer folyamatainak halmaza
- $R = \{R_1, R_2, \ldots, R_n\}$: az erőforrás-osztályok halmaza
- $P_i \Rightarrow R_i$: erőforrás igénylés
- $R_i \Rightarrow P_i$: erőforrás használat
- ha az igény kielégíthető, akkor az erőforrás igénylés használattá vált át

## 61. RAJZOLJON FEL EGY ERŐFORRÁS-HASZNÁLATI GRÁFOT HOLTPONTTAL, INDOKOLJA A HOLTPONT MEGLÉTÉT!




- egyik folyamat sem tud futni, mert egymásra várakoznak:
  - $P_1$ vár $R_1$-re, de $R_1$-et $P_2$ használja
  - $P_2$ vár $R_2$-re, de $R_2$-t $P_3$ használja
  - $P_3$ vár $R_3$-ra, de $R_3$-t $P_1$ és $P_2$ használja
- van kör és minden erőforrás osztályban egy erőforrás van

## 62. Rajzoljon fel egy erőforrás-használati gráfot körrel, de holtpont nélkül, indokolja, hogy miért nincs holtpont!




- van kör a gráfban, de ha $P_2$ felszabadítja $R_1$-et, akkor $P_3$ megkaphatja, ill., ha $P_4$ felszabadítja $R_2$-t, akkor $P_1$ megkaphatja
- a kör szükséges és elégséges feltétel, ha minden erőforrásosztályba csak egy erőforrás tartozik
- szükséges, de nem elégséges, ha egy erőforrás-osztály több egyedet is tartalmaz

## 63. Adja meg erőforrás használathoz kapcsolódóan az erőforrás-használati gráfban található esetleges kör és a holtpont kapcsolatát!

- ha nincs kör, nincs holtpont (a kör szükséges feltétel)
- ha van kör és minden erőforrás-osztályba csak egy erőforrás tartozik, akkor holtpont van (Egyszeres erőforrásoknál a kör elégséges feltétel is)
- ha van kör és egy erőforrás-osztály több egyedet is tartalmaz, akkor holtpont kialakulhat, de nem szükségszerűen (a kör léte többszörös erőforrásoknál nem elégséges feltétel)

## 64. Sorolja fel és röviden írja körül a holtpont kezelésére használható módszereket!

**Strucc algoritmus:** nem veszünk tudomást a problémáról és nem teszünk semmit

- bizonyos típusú rendszereknél (nagy kockázat) nem engedhető meg
- megengedhető kisebb kockázatú rendszereknél, ahol tolerálható a „kiszállunk-beszállunk elv”
- mérlegelni kell a probléma súlyát és a megoldás árát

**Védekezés holtpont kialakulása ellen:** az erőforrások használatánál bevezetünk szabályokat, ami biztosítja a holtpont elkerülését

- holtpont megelőzés (deadlock prevention)
- holtpont elkerülés (deadlock avoidance)

**Detektálás/feloldás:** a holtpont kialakulása után avatkozunk bele

- holtpont felismerés (deadlock recognition)
- holtpont megszüntetés (deadlock recovery)

## 65. Sorolja fel a holtpont megelőzésére használható módszereket!

- foglalva várakozás kizárása
- erőforrások elvétele
- körkörös várakozás elkerülése

## 66. MUTASSA BE, HOGY A HOLTPONT HOGYAN ELŐZHETŐ MEG A FOGLALVA VÁRAKOZÁS KIZÁRÁSÁVAL! MIK A LEHETSÉGES PROBLÉMÁK?

**Stratégiák:**

1. A folyamat elindulásakor egyszerre igényli az összes szükséges erőforrást. Csak akkor mehet tovább, ha mindegyiket megkapta.
2. Folyamat csak akkor igényelhet, ha más erőforrást nem foglal.

**Problémák:**

- rossz erőforrás kihasználtság, szükségesnél tovább birtokolják azokat
- fennáll a **kiéheztetés** veszélye: ha egy folyamat több "népszerű" erőforrást használ, nagy az esélye, hogy egyszerre az összes erőforrást soha nem kapja meg

- **pl.:** Számítás szalagról nyomtatóra. Erőforrások: szalagos tároló – diszk – nyomtató

## 67. MUTASSA BE, HOGY A HOLTPONT HOGYAN ELŐZHETŐ MEG AZ ERŐFORRÁSOK ELVÉTELÉVEL! MIK A LEHETSÉGES PROBLÉMÁK?

**Stratégiák:**

1. Ha egy folyamat valamely erőforrásigénye nem elégíthető ki, akkor az összes többit is elvesszük tőle. Ezekre a továbbiakban várakozik. Akkor futhat tovább, ha az összes erőforrásigényét egyszerre ki lehet elégíteni.
2. Ha egy P folyamatnak olyan erőforrásigénye van, amelyeket más várakozó $\{Q_i\}$ folyamatok foglalnak, akkor az erőforrásokat $Q_i$ folyamatoktól elveszik és $P$ futhat tovább (de csak ha $P$ összes igénye egyszerre kielégíthető), különben pedig $P$ is várakozik.

**Problémák:**

- az erőforrások egy része csak úgy vehető el, ha közben a futási eredmények is elvesznek
- fennáll a kiéheztetés veszélye

## 68. MUTASSA BE, HOGY A HOLTPONT HOGYAN ELŐZHETŐ MEG A KÖRKÖRÖS VÁRAKOZÁS KIZÁRÁSÁVAL! MIK A LEHETSÉGES PROBLÉMÁK?

- rendeljük a rendszer összes erőforrásához egy növekvő számsorozat egy-egy elemét

**Stratégiák:**

1. A folyamatok csak növekvő sorrendben igényelhetik az erőforrásokat.
2. A folyamat csak akkor igényelhet egy erőforrást, ha nem használ az igényeltnél magasabb sorszámút.

**Problémák:**

- nehéz az erőforrásokat olyan módon beszámozni, hogy az tükrözze az erőforrás szokásos sorrendjét
- interaktív rendszereknél nem jó (nem lehet megjósolni a folyamatok erőforrás használatát)

- logikailag függő folyamatokra alkalmazható

## 69. DEFINIÁLJA A HOLTPONTHOZ KAPCSOLÓDÓAN A BIZTONSÁGOS ÁLLAPOT ÉS A BIZTONSÁGOS SOROZAT FOGALMÁT!

**Biztonságos állapot:** létezik az összes folyamatot tartalmazó biztonságos sorozat

**Biztonságos sorozat:** folyamatok olyan $\{P_0, P_1, \ldots, P_n\}$ sorozata, ahol bármelyik $P_k$ folyamat erőforrásigénye kielégíthető a rendelkezésre álló, valamint a többi $P_i$ ($i < k$) folyamat által használt (és majdan felszabadított) erőforrással

## 70. MUTASSA BE, HOGY MI KÖZE VAN EGYMÁSHOZ A BIZTONSÁGOS ÁLLAPOTNAK, A NEM BIZTONSÁGOS ÁLLAPOTNAK ÉS A HOLTPONTNAK!

- a rendszer biztonságos állapotban van ⇨ nincs holtpont
- a rendszer nem biztonságos állapotban van ⇨ holtpont lehetséges
- holtpont elkerülése: biztosítani kell, hogy a rendszer soha ne kerüljön nem biztonságos állapotba

## 71. MUTASSA BE A HOLTPONT ELKERÜLÉSÉRE HASZNÁLT BANKÁR ALGORITMUS MŰKÖDÉSÉT SZÖVEGESEN!

**Előfeltételek:**

1. az erőforrás-osztályok több egyedből állhatnak
2. minden folyamat előzetesen megadja maximális igényét
3. egy igénylő folyamat várakozni kényszerülhet
4. ha egy folyamat megkapja az igényelt erőforrásait, véges időn belül visszaadja

**Adatszerkezet:**

- $n$ = folyamatok száma
- $m$ = erőforrás-osztályok száma
- *SZABAD*: m elemű vektor. Ha *SZABAD[j] = k*, akkor az $R_j$ típusú erőforrásból *k* példány elérhető.
- *MAX*: n x m méretű mátrix. A *MAX[i] m* elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból $P_i$ folyamat maximum hány példányt használhat (2. előfeltétel alapján).
- *FOGLAL*: n x m méretű mátrix. A *FOGLAL[i] m* elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból $P_i$ folyamat jelenleg hány példányt használ.
- *MÉG*: n x m méretű mátrix. A *MÉG[i] m* elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból $P_i$ folyamatnak feladata befejezéséhez még hány példányra lehet szüksége.
- *KÉR*: n x m méretű mátrix. A *KÉR[i] m* elemű sorvektor jelzi $P_i$ folyamat kérését az egyes erőforrásosztályokra.
- nyilvánvalóan: *MÉG[i,j] = MAX[i,j] – FOGLAL[i,j]*

**Kérés feldolgozása:**

**Alapötlet:**

- ha a kérés egyébként teljesíthető, akkor tegyünk úgy, mintha már teljesítettük volna
- vizsgáljuk meg, hogy ez az állapot biztonságos-e
- ha igen, valóban teljesíthetjük a kérést

**$P_i$ folyamat kéri a *KÉR[i]* erőforrásokat:**

A kérés ellenőrzése:

```
if KÉR[i] > MÉG[i] then STOP; (HIBA: túllépte a maximális igényt)
if KÉR[i] > SZABAD then VÉGE; (Most nincs elég szabad erőforrás)
```

**1.** A nyilvántartás átállítása az új állapotra:

```
SZABAD := SZABAD - KÉR[i];
FOGLAL[i] := FOGLAL[i] + KÉR[i];
MÉG[i] := MÉG[i] - KÉR[i];
```

**2.** Vizsgálat: a létrejött állapot biztonságos-e? (lásd később)

**3.**

```
if BIZTONSÁGOS then
   KÉRÉS TELJESÍTÉSE;
else állapot visszaállítása a (2) pont előttire:
   SZABAD := SZABAD + KÉR[i];
   FOGLAL[i] := FOGLAL[i]-KÉR[i];
   MÉG[i] := MÉG[i] - KÉR[i];
 KÉRÉS ELUTASÍTÁSA: A FOLYAMATNAK VÁRNIA KELL
```

**Biztonságos állapot vizsgálata:** biztonságos sorozat keresése

- keressünk olyan folyamatot, ami le tud futni a most rendelkezésre álló szabad erőforráskészlettel (ha nincs ilyen, de van várakozó folyamat, akkor holtpont van)
- a gondolatban lefuttatott folyamat által birtokolt erőforrásokat visszaadjuk, így most már több erőforrással próbálkozhatunk újra

**Újabb változók:**

- *SZABAD_MOST*: mint *SZABAD*. Munkaváltozó.
- *LEFUT*: *n* elemű vektor. Ha *LEFUT[j] = igaz,* akkor $P_j$ folyamat mindenképpen le tud futni

**Algoritmus:**

B1. Kezdőérték beállítása:

```
SZABAD_MOST := SZABAD
LEFUT[i] := hamis minden i-re (i=1,2,...,N)
```

B2. Továbblépésre esélyes folyamatok keresése:

```
Keress i-t amelyre (LEFUT[i] = HAMIS AND MÉG[i] <= SZABAD_MOST);
if van ilyen i, then
   SZABAD_MOST := SZABAD_MOST + FOGLAL[i];
   LEFUT[i] := igaz;
   ismételd a B2. lépést
else folytasd a B3. lépéssel
```

B3. Kiértékelés:

```
if LEFUT[i] = igaz minden i-re (i=1,2,...,N), then
   BIZTONSÁGOS
else
   NEM BIZTONSÁGOS (Pi folyamatok, amelyekre LEFUT[i] = hamis holtpontra
   juthatnak)
```

## 72. MIK A BANKÁR ALGORITMUS PROBLÉMÁI?

- időigényes
- az alapfeltételek (ismert folyamatszám, maximális igények, folyamat biztos befejeződése) nem biztosíthatók
- túlzott óvatosság, feleslegesen várakoztat folyamatokat (az erőforrások kihasználtsága rosszabb, mint holtpont elkerülés nélkül)

## 73. MUTASSA BE A HOLTPONT FELISMERÉSÉRE HASZNÁLT COFFMAN ALGORITMUST ANNAK PSEUDO KÓDJÁVAL!

**Adatszerkezet:**

- *N* = folyamatok száma
- *M* = erőforrás-osztályok száma
- *SZABAD*: m elemű vektor. Ha *SZABAD[j] = k*, akkor az $R_j$ típusú erőforrásból *k* példány elérhető.
- *SZABAD_MOST*: mint *SZABAD*. Munkaváltozó.
- *FOGLAL*: n x m méretű mátrix. A *FOGLAL[i] m* elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból $P_i$ folyamat jelenleg hány példányt használ.
- *KÉR*: n x m méretű mátrix. A *KÉR[i] m* elemű sorvektor jelzi $P_i$ folyamat kérését az egyes erőforrásosztályokra.
- *LEFUT*: n elemű vektor. Ha *LEFUT[j] = igaz,* akkor a $P_j$ folyamat a mostani erőforrásigények szerint le tud futni
- nyilvánvalóan: *MÉG[i,j] = MAX[i,j] – FOGLAL[i,j]*

**Algoritmus:**

C1. Kezdőérték beállítása:

```
SZABAD_MOST := SZABAD
LEFUT[i] := hamis minden i-re (i=1,2,...,N)
```

C2. Továbblépésre esélyes folyamatok keresése:

```
Keress i-t amelyre (LEFUT[i] = HAMIS AND KÉR[i] <= SZABAD_MOST);
if van ilyen i, then
   SZABAD_MOST := SZABAD_MOST + FOGLAL[i];
   LEFUT[i] := igaz;
   ismételd a C2. lépést
else folytasd a C3. lépéssel
```

C3. Kiértékelés:

```
if LEFUT[i] = igaz minden i-re (i=1,2,...,N), then
   NINCS HOLTPONT
else
   HOLTPONT: Azon Pi folyamatok, amelyekre LEFUT[i] = hamis
```

## 74. MUTASSA BE A HOLTPONT FELSZÁMOLÁSÁRA HASZNÁLHATÓ MÓDSZEREKET!

**Folyamatok terminálása:**

- minden holtpontban résztvevő folyamatot megszüntetünk (radikális): biztos, de költséges (megszűnt folyamatok eredményei elvesznek)
- egyesével szüntetjük meg folyamatokat, amíg a holtpont meg nem szűnik:
  - minden terminálás után újabb detektálás

  Szempontok a folyamat kiválasztásához:
  - hány holtpont körben szerepel
  - mekkora a prioritása
  - mennyi ideje futott, mennyit futna még (ha ismerjük)
  - mennyi erőforrást tart lefoglalva
  - mennyi további erőforrásra lenne szüksége
  - interaktív vagy batch program

**Erőforrások elvétele:**

- a holtpontra jutott folyamatoktól egyesével elvesszük az erőforrásokat

Megoldandó problémák:

- Kitől vegyük el és melyik erőforrást?
- Kiszemelt folyamatot vissza kell léptetni egy olyan állapotba, ahonnan a futását folytatni tudja (leggyakrabban újra kell kezdenie a futását). Egyes OS-ek a folyamatokhoz ellenőrzési pontokat rendelnek (checkpoint); a futást a legutóbbi ellenőrzési ponttól kell folytatni.
- el kell kerülni a folyamatok kiéheztetését (ne mindig ugyanattól a folyamattól vegyünk el erőforrásokat)

## 75. ADJA MEG, HOGY MI AZ A KOMMUNIKÁCIÓS HOLTPONT?

- a folyamatok tetszőleges, olyan együttműködése, amely a folyamatok körkörös várakozásához vezet
- pl.: kliens-szerver architektúrájú rendszer, ahol az ügyfelek és a szolgáltatók is folyamatok és az ügyfél-szolgáltató lánc záródik
- a gráfos reprezentáció itt is használható: várakozási gráf (wait-for graph)
  - csomópontjai a folyamatok
  - irányított élei a várakozást jelzik (várakozóból a várakoztatóhoz vezet)

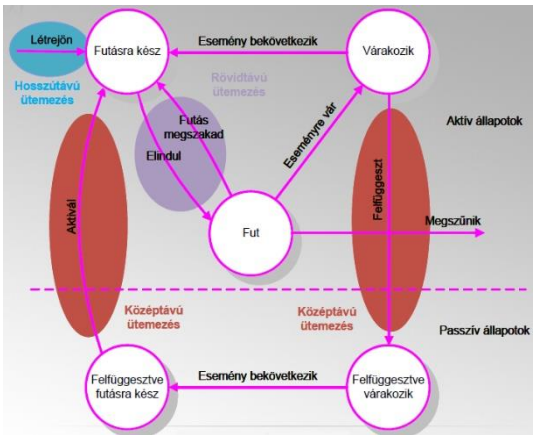
# ÜTEMEZÉS

## 76. ADJA MEG AZ ÜTEMEZÉS DEFINÍCIÓJÁT!

Az a tevékenység, amely eredményeként eldől, hogy az adott erőforrást a következő pillanatban mely folyamat használhatja.

## 77. MUTASSA BE, HOGY IDŐTÁV SZEMPONTJÁBÓL MILYEN CPU ÜTEMEZÉSI MEGOLDÁSOK VANNAK! *(25-27. kérdés)*

## 78. A FOLYAMATOK ÁLLAPOT ÁTMENETI GRÁFJÁN MUTASSA BE, HOGY HOL MILYEN IDŐTÁVÚ CPU ÜTEMEZÉS TÖRTÉNHET!




## 79. MUTASSA BE A HOSSZÚ TÁVÚ CPU ÜTEMEZÉST! *(27. kérdés)*

## 80. MUTASSA BE A KÖZÉPTÁVÚ CPU ÜTEMEZÉST! *(26. kérdés)*

## 81. MUTASSA BE A RÖVIDTÁVÚ CPU ÜTEMEZÉST! (*25. kérdés)*

## 82. DEFINIÁLJA, HOGY MI A CPU LÖKET, ADJA MEG, HOGY MIKOR CPU KORLÁTOS EGY FOLYAMAT!

**CPU löket:** Egy folyamat futása során végzett tevékenység, amelynek ideje alatt (átlagos hossza folyamatonként változik) a folyamatnak csak CPU-ra és az operatív tárra van szüksége

- a folyamat CPU korlátos, ha a CPU löketek hossza nagy a köztük levő I/O löketek hosszához képest

## 83. DEFINIÁLJA, HOGY MI A PERIFÉRIA LÖKET, ADJA MEG, HOGY MIKOR PERIFÉRIA KORLÁTOS EGY FOLYAMAT!

Egy folyamat futása során végzett tevékenység, amely alatt a folyamat perifériás átvitelt hajt végre, annak lezajlására várakozik, s eközben nincs szüksége a CPU-ra.

- a folyamat periféria korlátos, ha a CPU löketek hossza kicsi a köztük lévő I/O löketek hosszához képest

## 84. ADJA MEG, HOGY MELY ÜTEMEZÉSEKNÉL – AZ ÁLLAPOT ÁTMENETI GRÁFON MELY ÁLLAPOTÁTMENETNÉL VAN MINDIG KÖRNYEZETVÁLTÁS!

Ha a futó folyamat várakozni kényszerül, ill. befejeződik, akkor környezetváltás történik, hiszen a futó folyamat nem folytatja a működését.

## 85. ADJA MEG, HOGY MELY ÜTEMEZÉSEKNÉL – AZ ÁLLAPOT ÁTMENETI GRÁFON MELY ÁLLAPOTÁTMENETNÉL NINCS MINDIG KÖRNYEZETVÁLTÁS!

Ha a futó folyamat lemond a CPU-ról vagy elveszik tőle, ill. felébred, futásra késszé válik.

## 86. MUTASSA BE A NEM PREEMPTÍV CPU ÜTEMEZÉST!

- ha egy folyamattól, miután megkapta a CPU-t, nem lehet azt elvenni
- a folyamat csak az általa kiadott utasítások hatására vált állapotot:
  - erőforrásra, eseményre várakozás
  - befejeződés
  - a CPU-ról önként lemondás

## 87. MUTASSA BE A PREEMPTÍV CPU ÜTEMEZÉST!

- az OS elveheti a futás jogát egy folyamattól
- futásra kész állapotba teszi a futó folyamatot és egy másik (futásra kész) folyamatot indít el
- pl. időosztásos, valósidejű OS-ek

## 88. ADJA MEG A CPU KIHASZNÁLTSÁG DEFINÍCIÓJÁT!

Az ütemezési algoritmusok teljesítményének mérésére használt paraméter, amely megmutatja, hogy a CPU az idejének hány százalékát használja a folyamatok utasításainak végrehajtására. A kihasználtságot csökkenti: a CPU henyél (idle), rendszer-adminisztrációra fordított idő (rezsi) sok.

## 89. ADJA MEG AZ ÁTBOCSÁJTÓ KÉPESSÉG (CPU ÜTEMEZÉSHEZ KAPCSOLÓDÓ) DEFINÍCIÓJÁT!

Az ütemezési algoritmusok teljesítményének mérésére használt paraméter, amely megmutatja, hogy az OS időegységenként hány munkát futtat le.

## 90. ADJA MEG AZ KÖRÜLFORDULÁSI IDŐ (CPU ÜTEMEZÉSHEZ KAPCSOLÓDÓ) DEFINÍCIÓJÁT!

Az ütemezési algoritmusok teljesítményének mérésére használt paraméter, amely megmutatja, hogy egy munka a rendszerbe helyezéstől számítva mennyi idő alatt fejeződik be.

## 91. ADJA MEG A VÁRAKOZÁSI IDŐ (CPU ÜTEMEZÉSHEZ KAPCSOLÓDÓ) DEFINÍCIÓJÁT!

Az ütemezési algoritmusok teljesítményének mérésére használt paraméter, amely megmutatja, hogy egy munka (vagy folyamat) mennyi időt tölt várakozással (futásra kész állapot, várakozó állapot, felfüggesztett állapotok, (long-term) előzetes várakozás).

## 92. ADJA MEG A VÁLASZIDŐ (CPU ÜTEMEZÉSHEZ KAPCSOLÓDÓ) DEFINÍCIÓJÁT!

Az ütemezési algoritmusok teljesítményének mérésére használt paraméter, amely megmutatja, hogy mennyi az OS reakció ideje, azaz a kezelői parancs után a rendszer első látható reakciójáig eltelt idő (időosztásos (interaktív) rendszereknél fontos).

## 93. SOROLJA FEL A CPU ÜTEMEZÉSSEL KAPCSOLATOS KÖVETELMÉNYEKET!

- valamely (előbbi paraméterekből képzett) célfüggvény szerint legyen optimális
- legyen korrekt: kezeljen minden folyamatot (vagy bizonyos típusú folyamatokat) azonos módon (igazságos)
- biztosítson egyes folyamatoknál prioritást
- kerülje el a folyamatok kiéheztetését
- legyen megjósolható viselkedésű: meg lehessen becsülni a várható maximális körülfordulási időt
- minimalizálja a rezsi időt: gyakran kis többlet adminisztrációval jobb általános rendszerteljesítmény érhető el
- részesítse előnyben a kihasználatlan erőforrásokat igénylő, ill. a fontos erőforrásokat foglaló folyamatokat
- növekvő terhelés esetén a rendszer teljesítménye "elegánsan", fokozatosan romoljon le (graceful degradation), ne omoljon hirtelen össze

## 94. MUTASSA BE A LEGRÉGEBBEN VÁRAKOZÓ (FCFS) ALGORITMUST! ELŐNYÖKET ÉS HÁTRÁNYOKAT IS ADJA MEG!

A futásra kész folyamatok a várakozási sor végére kerülnek, az ütemező a sor elején álló folyamatot kezdi futtatni.

- nem preemptív
- egyszerűen megvalósítható (FIFO)
- a folyamatok átlagos várakozási ideje nagy ⇨ konvoj hatás (egy hosszú CPU löketű folyamat feltartja a mögötte várakozókat)

## 95. MI AZ A KONVOJ HATÁS?

Egy hosszú CPU löketű folyamat feltartja a mögötte várakozókat.

## 96. MUTASSA BE A KÖRFORGÓ (RR) ALGORITMUST! ELŐNYÖKET ÉS HÁTRÁNYOKAT IS ADJA MEG!

- az FCFS preemptív változata
- az időosztásos rendszerek valamennyi ütemezési algoritmusának az alapja
- folyamatok időszeletet kapnak (time slice):
  - ha a CPU löket nagyobb, mint az időszelet, akkor az időszelet végén az ütemező elveszi a CPU-t, a folyamat futásra kész lesz és beáll a várakozó sor végére
  - ha a CPU löket rövidebb, akkor a löket végén a folyamatokat újraütemezzük
- nincs konvoj hatás

**Időszelet**

- méretének meghatározása nehéz:
  - nagy időszelet: nem szakít meg műveletet, azaz FCFS algoritmushoz hasonló lesz
  - kis időszelet: sok folyamatot szakít meg, folyamatok a CPU-t egyenlő mértékben használják, viszont a sok környezetváltás a teljesítményt rontja
- ökölszabály: a CPU löketek kb. 80%-a legyen rövidebb az időszeletnél
- általában 10-100 ms

## 97. HOGYAN MŰKÖDNEK ÁLTALÁNOSAN A PRIORITÁSOS CPU ÜTEMEZŐ ALGORITMUSOK? MI A STATIKUS ÉS A DINAMIKUS PRIORITÁS?

**Működés:**

- a futásra kész folyamatokhoz egy prioritást (rendszerint egy egész számot) rendelünk
- a legnagyobb prioritású folyamat lesz a következő futtatandó folyamat

**Prioritás:**

**Meghatározása:**

- belső: OS határozza meg
- külső: az OS-en kívüli tényező (operátor, a folyamat saját kérése stb.) határozza meg

- futás során: statikus (végig azonos) vagy dinamikus (OS változtathatja)
- kiéheztetés veszélye

## 98. MUTASSA BE A STATIKUS PRIORITÁSOS ALGORITMUST! ELŐNYÖKET ÉS HÁTRÁNYOKAT IS ADJA MEG!

- prioritást sokszor a löketidő alapján határozzák meg
- löketidő szükséglet meghatározása:
  - a folyamat (felhasználó) bevallása alapján (a „hazugságot" az OS később bünteti)
  - előző viselkedés alapján (a korábbi löketidők alapján becslés)
- kiéheztetés: a folyamat sokáig (esetleg soha) nem jut processzorhoz
- öregedés: a kiéheztetés kivédéséhez a régóta várakozó folyamatok prioritását növeljük

## 99. DEFINIÁLJA, HOGY MI A KIÉHEZTETÉS ÉS MI AZ ÖREGEDÉS (AGEING)!

**Kiéheztetés:** a folyamat sokáig (esetleg soha) nem jut processzorhoz

**Öregedés:** a kiéheztetés kivédéséhez a régóta várakozó folyamatok prioritását növeljük

## 100. MUTASSA BE A LEGRÖVIDEBB LÖKETIDEJŰ (SJF) ALGORITMUST! ELŐNYÖKET ÉS HÁTRÁNYOKAT IS ADJA MEG!

- nem preemptív algoritmus
- a futásra kész folyamatok közül a legrövidebb löketidejűt indítja
- nincs konvoj hatás, optimális körülfordulási idő, optimális várakozási idő
- kiéheztetés veszélye
- alkalmazása: hosszú távú és rövid távú (RT rendszerek) ütemezésnél

## 101. MUTASSA BE A LEGRÖVIDEBB HÁTRALEVŐ IDEJŰ (SRTF) ALGORITMUST! ELŐNYÖKET ÉS HÁTRÁNYOKAT IS ADJA MEG!

- az SJF preemptív változata
- ha egy új folyamat válik futásra késszé, akkor az ütemező újra megvizsgálja a futásra kész folyamatok, ill. az éppen futó folyamatok hátralévő löketidejét és a legrövidebbet indítja tovább
- a folyamat megszakítása és egy másik elindítása környezetváltozást igényel, így ezt az időt is figyelembe kell vennünk
- kiéheztetés lehetséges

## 102. MUTASSA BE A LEGJOBB VÁLASZARÁNY (HRR) ALGORITMUST!

- az SJF algoritmus változata, ahol a várakozó folyamatok öregednek ⇨ nincs éhezés
- a kiválasztás (a löketidő helyett) a $\frac{löketidő + k \cdot várakozási\_idő}{löketidő}$ képlet szerint történik, ahol $k$ egy jól megválasztott konstans

## 103. MUTASSA BE A STATIKUS TÖBBSZINTŰ SOROK (SMQ) ALGORITMUST!

- folyamatot elindulásakor valamilyen kritérium alapján besorolunk egy várakozó sorba
- a folyamat élete során végig ugyanabban a sorban marad
- egy lehetséges példa a prioritások besorolására:
  - rendszer folyamatok (magas prioritás, közvetlen hatással vannak a rendszer működésére)
  - interaktív folyamatok (biztosítani kell a felhasználó számára az elfogadható válaszidőt)
  - interaktív szövegszerkesztők (kevésbé kritikusak)
  - kötegelt feldolgozás (általában akkor futnak, ha "van idő")
  - rendszerstatisztikákat gyűjtő folyamatok (alacsony prioritás, nincsenek közvetlen hatással a rendszer működésére)
- nagy a veszélye az alsó sorokban lévő folyamatok kiéheztetésének

## 104. MUTASSA BE A VISSZACSATOLT TÖBBSZINTŰ SOROK (MFQ) ALGORITMUST!

- a sorokhoz egy időszelet tartozik: minél nagyobb a prioritás, annál kisebb az időszelet
- a folyamatok futásuk során átkerülhetnek másik sorokba:

  **Kisebb prioritású sorba:**
  - sorokhoz különböző időszelet tartozik
  - a folyamat mindig a legnagyobb prioritású sorban indul
  - ha a folyamatnak nem elég az adott időszelet, akkor egy kisebb prioritású sorban kerül
  - merev algoritmus: ha a folyamat egyszer is túllépte valamelyik sor löketidejét, akkor az nem kerülhet vissza

  **Nagyobb prioritású sorba:**
  - érdemes időnként a folyamatot nagyobb prioritású sorba átsorolni (pl. löketidő mérése alapján)
  - a régóta várakozó folyamatot öregedésük miatt a rendszer nagyobb prioritású sorba helyezi
- a folyamatok sor- és prioritás-váltása

## 105. MUTASSA BE A CPU ÜTEMEZÉST HETEROGÉN TÖBBPROCESSZOROS RENDSZEREK ESETÉN!

- egy folyamat csak bizonyos processzorokon (amelynek gépi utasításait tartalmazza) futhat

## 106. MUTASSA BE A CPU ÜTEMEZÉST HOMOGÉN TÖBBPROCESSZOROS RENDSZEREK ESETÉN!

- a futásra kész folyamatok közös sorokban tárolódnak és a rendszer bármilyen szabad processzorán elindulhat

**Szimmetrikus multiprocesszoros rendszer:**
- minden CPU saját ütemezőt futtat, amely a közös sorokból választ
- a sorok osztott használatához a kölcsönös kizárást biztosítani kell

**Aszimmetrikus multiprocesszoros rendszer**
- az ütemező egy dedikált CPU-n fut, amely szétosztja a folyamatokat a szabad CPU-k között

# TÁRKEZELÉS

## 107. ÍRJA LE, HOGY MI A LOGIKAI CÍMTARTOMÁNY, A FIZIKAI CÍMTARTOMÁNY ÉS A MAPPING!

**Logikai címtartomány:** folytonos címtartomány, amely 0-tól kezdődik és lineárisan nő a maximális értékig

**Fizikai címtartomány:** a gyakorlatban

- nem 0 fizikai címtől kezdve történik a programok végrehajtása
- sokszor nem folytonos memóriaterület áll rendelkezésre

**Mapping** (leképezés): a logikai és fizikai címtartomány közötti megfeleltetés

## 108. ADJA MEG A TÁRKEZELÉSNÉL A CÍMEK KÖTÉSÉNEK LEHETŐSÉGEIT!

**Statikus:**

**Fordítás közben (compile time):**

- a fordítóprogram a program és az adatterület elemeihez abszolút címet rendel
- merev technika, elsősorban ROM-ban lévő programok esetében alkalmazzák

**Szerkesztés közben (link time):**

- a program több modulból áll, amelyek hivatkoznak egymásra (logikai címek)
- a függetlenül lefordított modulok saját logikai címtartományt használnak
- a linker feladata, hogy az összes modult - egymás mögé - elhelyezze a fizikai tárba, valamint feloldja a modulok kereszthivatkozásait

**Betöltés közben (load time):**

- a fordító áthelyezhető kódot generál, ennek a címhivatkozásait a betöltő program az aktuális címkiosztás szerint módosítja

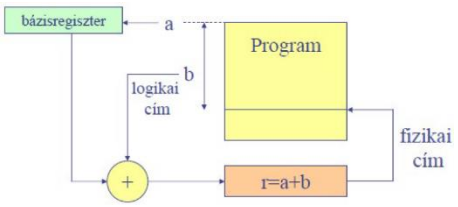
**Dinamikus:**

**Futás közben (run time):**

- a program memóriaképe csak logikai címeket tartalmaz, ezért a konkrét fizikai címet speciális hardver elemek határozzák meg az utasítás végrehajtásakor

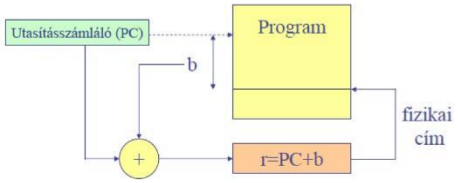
## 109. MUTASSA BE A BÁZIS RELATÍV CÍMZÉS MŰKÖDÉSÉT!

- a program tetszőleges helyre betölthető
- a bázisregisztert a betöltési kezdőcímre állítva a program végrehajtható




## 110. MUTASSA BE AZ UTASÍTÁSSZÁMLÁLÓ RELATÍV CÍMZÉS MŰKÖDÉSÉT!

- pozíció-független kód: csak pozíció-független virtuális címeket tartalmaz



## 111. Adja meg, hogy mi a tárkezelésnél használt dinamikus betöltés (dynamic loading)!

- a programhoz tartozó egyes eljárások a háttértáron vannak, ha valamelyikre szükség van, akkor egy speciális programrészlet ezt betölti ⇨ lényegesen kisebb tárterületet foglal el
- a ritkán használt eljárások nem foglalják a tárat (pl. biztonsági mentés)
- programozó szervezi meg, az OS nem nyújt támogatást
- pl.: ritkán használt részek, hibakezelés stb.

**Lépései:**

1. Hívás
2. Ellenőrzés: memóriában van?
3. Speciális programrész betöltése a memóriába, vezérlést átadja

## 112. Adja meg, hogy mi a tárkezelésnél használt dinamikus könyvtár (dynamic linking) használat!

- a dinamikus betöltés változata, az OS támogatásával
- a programban használt rendszerkönyvtárak eljárásai helyett csak egy csonk (stub) kerül a programba
- a csonk tartalmaz egy hivatkozást a könyvtárra és az azon belüli eljárásra
- az csonk első meghívásakor az OS az eljárást betölti a tárba
- a következő hívások már az eljárást hívják meg időveszteség nélkül

**Előny:**

- csökken a tárfelhasználás
- hibás eljárás javításakor nem kell az összes programot újrafordítani

**Hátrány:** verzió-kontrol nehéz: „dll pokol”

- pl.: Windows dll, Unix shared library

## 113. Adja meg, hogy mi a tárkezelésnél használt átfedő programrészek (overlay) módszer!

- a program részekre bontása:
  - közös adat- és programrészek (nem változik)
  - olyan átfedő részek, amelyek közül egy időben csak egyre van szükség
- az átfedő részeket egyesével töltjük be a memóriába
- a megoldás a programozó feladata, nincs szükség OS támogatásra
- az átfedés számára fenntartott tárterület a legnagyobb átfedő programrész hosszával egyenlő

## 114. Mutassa be az egypartíciós rendszert!

**Memóriaszervezés:**

**Védett területek:** OS, speciális tárterületek (megszakítás vektorok, periféria címtartományok)

**Felhasználói terület:**

- a nem védett területen felüli folytonos cím-tartományt csak egy folyamat használja
- a program az első szabad címre töltődik
- ha az OS-nek szüksége van memóriára, akkor vagy áthelyezi a programot (hardver támogatás nélkül nehézkes) vagy a nem használt területről allokál

**Védelem:** felhasználói és rendszer mód

- OS területének védelmére elég egy regiszter, amely a program legkisebb címét tartalmazza
- **felhasználói mód:** futás közben egy hardver figyeli, hogy minden hivatkozás a tárolt cím felett legyen
- **rendszer mód:** rendszerhíváskor a védelem kikapcsol, az OS az egész címtartományt eléri

## 115. ÍRJA LE, HOGY MI A FIX PARTÍCIÓS RENDSZER, ADJA MEG, HOGY MI A BELSŐ TÖRDELŐDÉS!

**Fix partíciós rendszer:**

- az OS feletti tárterületet partíciókra bontják
- a határok nem változnak
- rossz hatékonyság: belső tördelődés
- védelem: határ-regiszterek

**Belső tördelődés:** a folyamatok nem használják ki a rendelkezésére bocsátott partíciót

## 116. ÍRJA LE, HOGY MI A VÁLTOZÓ PARTÍCIÓ MÉRETŰ RENDSZER, ADJA MEG, HOGY MI A KÜLSŐ TÖRDELŐDÉS!

**Változó partíció méretű rendszer:**

- a partíció a program igényeinek megfelelő méretű
- belső tördelődés nincs, hiszen csak a szükséges memóriát kapják meg a folyamatok

**Problémák:** szabad terület tördelődése

- egy folyamat lefutásakor a használt memória felszabadul
- az OS nyilvántartja ezeket a területeket, az egymás melletti szabad területeket automatikusan összevonja
- sokszor ezen területek nem szomszédosak, így a szabad memória kis részekre oszlik (külső tördelődés)

**Külső tördelődés:** a szabad memória kis, egymással nem szomszédos részekre oszlik

## 117. ADJA MEG, HOGY MI A SZABAD TERÜLETEK TÖMÖRÍTÉSE ÉS MIÉRT VAN RÁ SZÜKSÉG!

- a külső tördelődés bizonyos fok után lehetetlenné teszi újabb folyamat elindítását (elég a szabad terület, de a leghosszabb összefüggő szabad terület nem elég a folyamatnak)

**Megoldás:** a szabad helyek tömörítése

**Tömörítő algoritmusok:** szabad helyeket a tár egyik végére rendezik

- időigényes (nem biztos hogy megéri futtatni, esetleg jobban járunk, ha megvárjuk, míg néhány folyamat befejeződik)
- HW támogatást igényel
- váratlanul lehet szükség rá (ezért pl. egy interaktív rendszer válaszideje hirtelen megnőhet)
- a tárterületek mozgatását körültekintően kell végrehajtani (az áthelyezési információk megőrzése, stb.)

## 118. MUTASSA BE A MEMÓRIATERÜLET FOGLALÁSNÁL HASZNÁLT LEGINKÁBB MEGFELELŐ (BEST FIT) MÓDSZERT!

- legkisebb még elegendő méretű terület lefoglalása

## 119. MUTASSA BE A MEMÓRIATERÜLET FOGLALÁSNÁL HASZNÁLT ELSŐ MEGFELELŐ (FIRST FIT) MÓDSZERT!

- a kereséssel a tár elejétől indulunk, az első megfelelő méretűt lefoglaljuk

## 120. Mutassa be a memóriaterület foglalásnál használt következő megfelelő (next fit) módszert!

- a kereséssel az utoljára lefoglalt tartomány végéről indulunk, az első megfelelő méretűt lefoglaljuk
- igen gyors algoritmus, a memória 30%-a marad kihasználatlan
- (50%-os szabály, mivel a folyamatok által foglalt memória fele nincs kihasználva)

## 121. Mutassa be a memóriaterület foglalásnál használt legrosszabbul illeszkedő (worst fit) módszert!

- a legnagyobb szabad területből foglaljuk le, abban bízva, hogy a nagy darabból fennmaradó terület más folyamat számára még elegendő lesz
- ez a legrosszabb hatékonyságú

## 122. Definiálja, hogy mi az a tárcsere (swap), adja meg, hogy mikor van rá szükség, és hogy az operációs rendszernek milyen problémákat kell megoldani ennek során!

- a tárcsere során az OS egy folyamat (teljes) tárterületét a háttértárra másolja, így szabadítva fel területet más folyamatok számára
- a perifériás átvitel miatt időigényes (sokkal több idő, mint egy környezetváltás, így CPU ütemezéskor ezt figyelembe kell venni)

**Problémák:**

Melyik folyamatot tegyük ki háttértárra?

- figyelembe kell venni a folyamat állapotát, prioritását, futási, várakozási idejét, vagy a lefoglalt partíció nagyságát

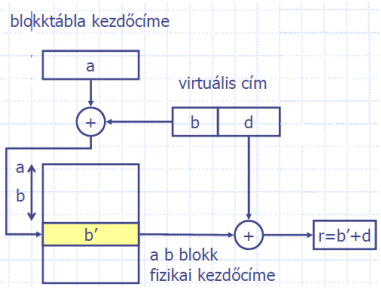
Melyik folyamatot mikor hozzunk be a háttértárról?

- az előző szempontok itt is érvényesek
- ügyelni kell a kiéheztetés veszélyére

Kerülendő a folyamatok felesleges pakolgatása

## 123. Adja meg, hogy mi az a futás közbeni címleképezés!

- virtuális cím: **<b, d>**
  - **b**: blokkcím,
  - **d**: eltolás (displacement)
- a transzformáció a blokktábla segítségével megy végbe
- minden folyamatnak saját blokktáblája van
- a folyamatok virtuális címtartománya fedi egymást, de a fizikai címtartomány természetesen nem




## 124. Mutassa be a szegmens szervezés elvét!

- a logikai címtartományban a program memóriája nem egybefüggő terület, hanem önmagukban folytonos blokkok (szegmensek) halmaza
- a folyamat memóriája különböző méretű blokkokból (szegmens) áll
- fellép a külső tördelődés: szegmensek közötti kihasználatlan memóriaterületek
- a szegmens logikai egység, pl.: főprogram; eljárások, függvények, módszerek, objektumok; lokális változók, globális változók; stack, szimbólumtábla stb.
- a címtranszformáció az általános modellnek megfelelő:
  - blokktábla ⇨ szegmenstábla
  - cím: <szegmenscím, eltolás>

## 125. MUTASSA BE A SZEGMENS SZERVEZÉS VÉDELMÉNEK ELVÉT!

- a folyamat nem címezhet ki a saját szegmenséből:
  - szegmensen belüli cím ≤ szegmens mérete (hardver figyeli), különben megszakítás (segment overflow fault)
- a szegmenstábla tárolja a szegmens méretét (limit)

**Hozzáférés ellenőrzés:**

- a folyamatoknak az egyes szegmensekhez különböző hozzáférési módokat engedélyezhetünk
  - olvasási jog: szegmens területét olvashatja
  - írási jog: szegmens területére írhat, az ott lévő értékeket módosíthatja
  - végrehajtási jog: a szegmensben gépi utasítások vannak, melyeket a folyamat végrehajthat
- a jogosultság megsértése megszakítást generál (segment protection fault)

## 126. MUTASSA BE AZ OSZTOTT SZEGMENS HASZNÁLAT ELVÉT!

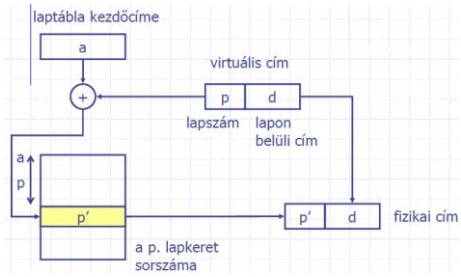
- több folyamat ugyanazt a szegmenst használja:
  - közös utasítások használata:
    - több folyamat azonos programot futtat
    - kevesebb memóriahasználat
    - lehet teljes program is, de rendszerkönyvtár is
  - közös adatterület
    - folyamatok közötti kommunikáció
    - gyerek folyamat használhatja a szülő adatait, eredményeit

**Megvalósítás:**

- a folyamatok szegmenstáblájában valamelyik szegmensnél azonos fizikai cím van
  - a jogosultságok természetesen lehetnek különbözőek
  - biztosítani kell a kölcsönös kizárást

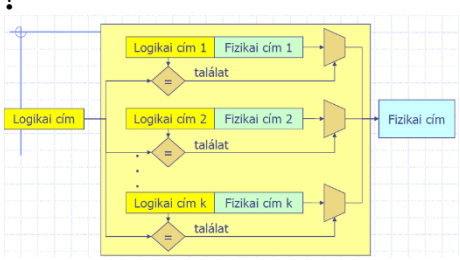
## 127. MUTASSA BE AZ EGYSZINTŰ LAPSZERVEZÉS ELVÉT!

- a folyamathoz tartozó minden lap fizikai címe egy táblában (laptérkép) van
- laptábla mérete nagy lehet, mivel a lapok száma sok
- nehéz gyors elérésű tárban tartani
- pl.: 32 bites címtér ($2^{32}$), 4 Kbites lapok ($2^{12}$) ⇨ $2^{20}$ ($\approx 10^6$) db bejegyzés
- megoldás: laptábla tördelése ⇨ többszintű laptábla
- felfogható a laptábla lapozásának is




## 128. MUTASSA BE AZ ASSZOCIATÍV TÁR MŰKÖDÉSÉT!

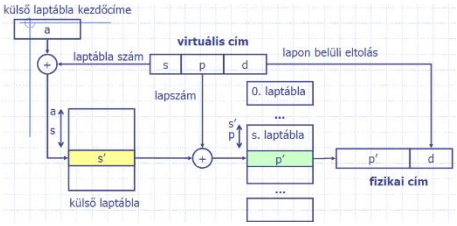
- speciális gyors elérésű tár (asszociatív tár) segíti a címzést (translation look-aside buffer (TLB))
- a laptábla gyorsító tárban a várhatóan gyakran használt lapok címét tároljuk
- a tár mérete itt sem elég nagy
- keresés: párhuzamosan az összes tárolt logikai cím alapján
- találat esetén a találatnak megfelelő fizikai cím kerül a kimenetre
- drága ⇨ kis kapacitású



## 129. MUTASSA BE AZ OSZTOTT LAP HASZNÁLAT ELVÉT!

- hasonló az osztott szegmenshasználathoz, több folyamat laptérkép táblája azonos fizikai címekre hivatkozhat
- mivel a lap mérete nem tükrözi a folyamatok logikai tárfelosztását, így a lapokon, nem osztottan használható területek is lehetnek (kerülendő)
- pl.: közösen használt kód ⇨ OS gondoskodik a védelemről:
    - szövegszerkesztő: 3 lap kód, 1 lap adat
        - ha 20 példány fut, akkor a memóriaigény
            - $4 \cdot 20 = 80$ lap (nem osztott laphasználat)
            - $1 \cdot 20 + 3 = 23$ lap (osztott laphasználat)

## 130. MUTASSA BE A TÖBBSZINTŰ LAPSZERVEZÉS ELVÉT!




## 131. HOGYAN VALÓSUL MEG A LAPSZERVEZÉSNÉL A TÚLCÍMZÉS ELLENI VÉDELEM?

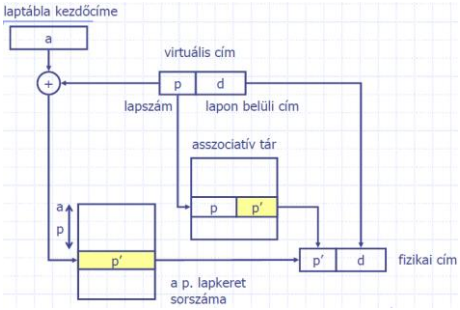
- lapon belüli túlcímzés ellen nem kell védeni, hiszen minden lapon belül kiadható cím megfelelő (kivétel: utolsó lap, mert az nincs teljesen tele)
- a lapok érvényességét egy bit jelzi
- címtranszformáció a háttértáron lévő lap esetén:
    - az érvényesség bit jelzi, hogy a memóriában van-e a lap
    - ha nincs (invalid bit), akkor lehet, hogy a háttértáron van
    - utóbbi esetben a tábla a háttértáron való elhelyezkedés információját tárolja

## 132. MUTASSA BE A KOMBINÁLT SZEGMENS- ÉS LAP SZERVEZÉS ELVÉT!

- egyesíti a két technika előnyeit:
    - lap szervezés: nincs külső tördelődés, nem kell a teljes szegmensnek a tárban lennie, csak az éppen szükséges lapjainak
    - szegmens szervezés: tükrözi a folyamat logikai társzerkezetét, osztott használat egyszerűbb, hozzáférési jogosultság megoldható
- címtranszformáció: lényegében egy kétszintű táblakezelés
    - első szint: laptábla címeket tartalmazó szegmenstábla
    - második szint: szegmensenként egy-egy fizikai lapcímeket tartalmazó laptábla
- a logikai cím három részre tagozódik (szegmenscím (a szegmenshez tartozó laptábla kezdőcímének kijelölése), lapcím, lapon belüli eltolás)
- a fizikai címet a laptábla alapján a lapszervezésnek megfelelően számítjuk ki
- hozzáférési jogok ellenőrzése a szegmens szervezésének megfelelően történik
- osztott tárhasználat a szegmens szervezésének megfelelően működik

## 133. Mutassa be a kombinált asszociatív tár és lap szervezés elvét!

- a fizikai lapcím keresése egyszerre kezdődik mind az asszociatív tárban, mind a direkt laptáblában
- ha az asszociatív tárban van találat, akkor a direkt keresés leáll
- az asszociatív tárban lévő lapokat frissíteni kell, környezetváltás esetén az asszociatív laptáblát is cserélni kell
- egy adott időszak alatt csak a teljes címtartomány kis része van kihasználva, így a találati arány elég magas lehet (80-99%)




## 134. Adott egy kombinált asszociatív tár és lap szervezésű tárkezelés. A memória hozzáférési ideje X ns. Az asszociatív tár elérési ideje Y ns. A találati arány Z%. Adja meg, hogy mekkora az átlagos elérési idő?

Fizikai cím elérése:

- asszociatív memóriával: Y ns + X ns
- asszociatív memória nélkül: 2 · X ns

Átlagos elérési idő: $\frac{Z}{100} \cdot (Y + X)\ \text{ns} + \left(100 - \frac{Z}{100}\right) \cdot 2 \cdot X\ \text{ns}$

## 135. Adott egy lapszervezésű tár. A rendszerben az átlagos folyamat méret s=X MB. Egy laptábla bejegyzés mérete e= Y byte. Adja meg, hogy mekkora a lapok ideális mérete (p)?

$p = \sqrt{2se} = \sqrt{2 \cdot X \cdot Y}$

# Virtuális tárkezelés

## 136. Definiálja, hogy mi a virtuális tárkezelés!

- olyan szervezési elvek, az OS algoritmusainak összessége, amelyek biztosítják, hogy a folyamatok logikai címtartományának csak egy - a folyamat futásához szükséges - része legyen a központi tárban, de bármelyik folyamat a virtuális tartományán belül tetszőleges címre hivatkozhat
- korábbi „helytakarékos módszerek”: késleltetett betöltés, tárcserék stb. (ezek nem voltak általános megoldások)

## 137. Sorolja fel, hogy mi motiválja a virtuális tárkezelés használatát!

- a programok nem használják ki a teljes címtartományukat
  - tartalmaznak ritkán használt kódrészleteket (pl. hibakezelő rutinok)
  - a statikus adatszerkezetek általában túlméretezettek (statikus vektorok, táblák stb.)
  - a program futásához egy időben nem kell minden részlet (overlay)
  - időben egymáshoz közeli utasítások és adatok általában a térben is egymáshoz közel helyezkednek el (lokalitás)

- nem célszerű az egész programot a tárban tartani:
  - programok méretét nem korlátozza a tár nagysága, a program nagyobb lehet, mint a ténylegesen meglévő memória mérete
  - a memóriában tartott folyamatok száma növelhető (nő a multiprogramozás foka, javítható a CPU kihasználtság és az átbocsátó képesség)
  - a programok betöltéséhez, háttértárba mentéséhez kevesebb I/O művelet kell
  - memóriaárak csökkenése ⇨ a fizikai memória nagymértékű bővítése, ezzel párhuzamosan a programok átlagos mérete is növekedett
  - Parkinson törvénye alátámasztani látszik ezt a jelenséget: „A programok kitöltik a rendelkezésükre álló memóriát”
  - hiába bővítjük a központi memóriát, az egy idő után mindig kevéssé válik, akadnak majd olyan programok, amelyek nem férnek el benne:
    - grafikus felhasználói felületek
    - az objektumorientált nyelveken írt alkalmazások memóriaigénye is nagyobb

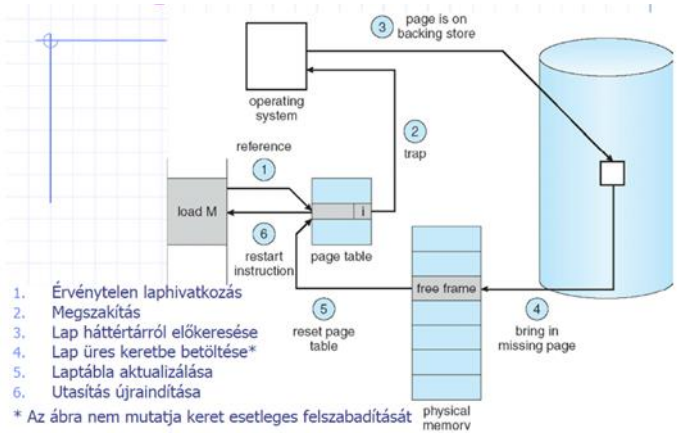
## 138. MUTASSA BE A VIRTUÁLIS TÁRKEZELÉS ELVI MEGVALÓSÍTÁSÁNAK LÉPÉSEIT (ÁBRA NEM KELL)!

- amikor a folyamat érvénytelen, a valós memóriában nem található címre hivatkozik, hardver megszakítást okoz (laphiba), ezt az OS kezeli, és behozza a kívánt blokkot

**Lépései:**

- az OS megszakítást kezelő programrésze kapja meg a vezérlést (1,2)
  - elmenti a folyamat környezetét (környezetváltás)
  - elágazik a megfelelő kiszolgáló rutinra
  - eldönti, hogy a megszakítás nem programhiba-e (pl. kicímzés)
- a kívánt blokk beolvasása a tárba (4,5)
  - az OS a blokknak helyet keres a tárban
  - ha nincs szabad terület, akkor fel kell szabadítani egy megfelelő méretű címtartományt, ennek tartalmát esetleg a háttértárra mentve
  - beolvassa a kívánt blokkot
  - megjegyzés: a perifériás műveletek sok időt vesznek igénybe, ki kell várni az eszköz felszabadulását, az előkészítő műveletet (fejmozgás) és az átvitelt. A rendszer jobb kihasználtsága érdekében a megszakított folyamatot az OS várakozó állapotba helyezi, és egyéb folyamatot indít el. Amikor a kívánt blokk bekerül a tárba, a folyamat futás kész lesz.
- a folyamat újra végrehajtja a megszakított utasítást (6)

## 139. SZEMLÉLTESSE ÁBRÁN A VIRTUÁLIS TÁRKEZELÉS ELVI MEGVALÓSÍTÁSÁT!



## 140. VIRTUÁLIS TÁRKEZELÉS ESETÉN A LAPHIBA VALÓSZÍNŰSÉGE P. A MEMÓRIA HOZZÁFÉRÉSI IDŐ X NS, EGY LAP HÁTTÉRTÁRRA ÍRÁSÁNAK / HÁTTÉRTÁRRÓL BEOLVASÁSÁNAK IDEJE Y MS. A RENDSZERBEN A HELYETTESÍTENDŐ LAPOKNAK ÁTLAGOSAN Z %-A MÓDOSUL VISSZAÍRÁS ELŐTT. SZÁMOLJA KI AZ EFFEKTÍV HOZZÁFÉRÉSI IDŐT (EAT)!

$$\mathrm{EAT} = (1-\mathrm{p}) \cdot \text{memória hozzáférési idő} + \mathrm{p} \cdot \text{laphiba idő} =$$
$$= (1-\mathrm{p}) \cdot \mathrm{X}\,[\mathrm{ns}] + \mathrm{p} \cdot \left(\mathrm{Y} + \frac{\mathrm{Y} \cdot \mathrm{Z}}{100}\right) \cdot 10^6\,[\mathrm{ns}] \sim \left(\mathrm{Y} + \frac{\mathrm{Y} \cdot \mathrm{Z}}{100}\right) \cdot 10^6 \cdot \mathrm{p}\,[\mathrm{ns}]$$

## 141. DEFINIÁLJA, HOGY MIT JELENT AZ IGÉNY SZERINTI LAPOZÁS (DEMAND PAGING), ADJA MEG AZ ELŐNYEIT ÉS A HÁTRÁNYAIT!

- csak laphiba esetén hozunk be lapot

**Előnyei:** egyszerű a lapot kiválasztani, a tárba csak a biztosan szükséges lapok kerülnek be

**Hátrányai:** új lapokra való hivatkozás mindig laphibát okoz

## 142. DEFINIÁLJA, HOGY MIT JELENT AZ ELŐRE TEKINTŐ LAPOZÁS (ANTICIPATORY PAGING), ADJA MEG AZ ELŐNYEIT ÉS A HÁTRÁNYAIT!

- az OS megpróbálja kitalálni, hogy a folyamatnak a jövőben melyik lapokra lesz szüksége és azokat "szabad idejében" betölti
  - ha a jóslás gyors és pontos, akkor a futási sebesség jelentősen felgyorsul
  - ha a döntés hibás, a felesleges lapok foglalják a tárat
- a memória ára jelentősen csökken, így a mérete nő, a hibás döntés ára (a felesleges tárfoglalás) egyre kisebb
- egyre népszerűbb

## 143. DEFINIÁLJA, HOGY MI AZ A LAPCSERE, ADJA MEG, HOGY MI A CÉLJA!

- az a folyamat, melynek során kiválasztjuk azt a lapot, amelyiket feláldozzuk az új lap betöltése érdekében
- általános esetben 2 lépésből áll: lecserélendő lap kimentése és az új lap betöltése

**Cél:** optimális eset közelítése, azaz annak a lapnak a kiválasztása, amelyre a folyamatnak legtovább nem lesz szüksége a jövőben

## 144. ADJA MEG, HOGY MI LENNE AZ OPTIMÁLIS LAPCSERE STRATÉGIA, MIÉRT NEM LEHET ILYET MEGVALÓSÍTANI?

- azt a lapot választjuk ki cserére, amelyre a folyamatnak a legtovább nem lesz szüksége a jövőben
- nem lehet előre látni ⇨ gyakorlatban nem megvalósítható

## 145. ADJA MEG, HOGY MI AZ A LAPKERET!

- megadja, hogy folyamatonként hány lap lehet egyszerre a memóriában

## 146. MUTASSA BE A VÉLETLEN KIVÁLASZTÁS LAPCSERE STRATÉGIÁT!

- egyszerű, buta algoritmus, csak elvi jelentőségű, nem használják

## 147. MUTASSA BE A LEGRÉGEBBI LAP (FIFO) LAPCSERE STRATÉGIÁT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- a tárban lévő legrégebbi lapot cseréli le
- megvalósítása: egyszerű FIFO listával (könnyen implementálható, gyors)

**Hibája:**

- olyan lapot is kitesz, amelyet gyakran használnak
- felléphet egy érdekes jelenség, a Bélády-anomália

## 148. ADJA MEG, HOGY MI AZ A BÉLÁDY-ANOMÁLIA (PÉLDA NEM KELL)!

Ha növeljük a folyamatokhoz tartozó lapok számát, a laphibák száma esetenként nem csökken, hanem nő.

## 149. MUTASSA BE AZ ÚJABB ESÉLY (SECOND CHANCE) LAPCSERE STRATÉGIÁT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- a FIFO egy változata, a sor elején lévő lapot csak akkor cseréli le, ha nem hivatkoztak rá
- ha hivatkoztak a lapra, akkor a hivatkozás bitet töröljük, és a lapot visszatesszük a FIFO végére
- minden lap tartalmaz egy *R* hivatkozás bitet, ami kezdetben törölve van
- a lapra hivatkozáskor *R =1*
- lapcsere esetén:
  - ha a sor elején levő lapon *R = 0,* akkor csere
  - ha a sor elején levő lapon *R = 1,* akkor az *R* bitet töröljük és a lapot a FIFO végére rakjuk, majd a következő (most már első) lappal próbálkozunk tovább
- kiküszöböli a FIFO fő hibáját ⇨ a gyakran használt lapokat nem cseréli le

## 150. MUTASSA BE AZ ÓRA ALGORITMUS (CLOCK) LAPCSERE STRATÉGIÁT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- az újabb esély algoritmus másik implementációja
- a lapok körkörös láncban vannak felfűzve a betöltés sorrendjében
- lapcsere előtt az algoritmus megvizsgálja az *R* bitet: ha egynek találja, akkor nem veszi ki a lapot, törli az *R*-t és a mutatót továbblépteti

## 151. MUTASSA BE A LEGRÉGEBBEN HASZNÁLT LAP (LRU) LAPCSERE STRATÉGIÁT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT (IMPLEMENTÁCIÓ NEM KELL)!

- azt a lapot választjuk, amelyre a leghosszabb ideje nem hivatkoztak
- a múltbeli információk alapján próbál előrelátni, az optimális algoritmust közelíteni
- szimulációs eredmények alapján jó teljesítmény, de nehéz implementálni (csak hardvertámogatással lehetne jól megvalósítani)

## 152. MUTASSA BE A LEGRÉGEBBEN HASZNÁLT LAP (LRU) LAPCSERE STRATÉGIA SZÁMLÁLÓ ALAPÚ IMPLEMENTÁCIÓJÁT!

- a lapra történő hivatkozáskor feljegyezzük annak idejét
- a helyettesítendő lap kiválasztáskor a tárolt időpontok közül keressük a legrégebbit

## 153. MUTASSA BE A LEGRÉGEBBEN HASZNÁLT LAP (LRU) LAPCSERE STRATÉGIA LÁNCOLT LISTA ALAPÚ IMPLEMENTÁCIÓJÁT!

- a lapok egy láncolt listában vannak
- a frissen behozott lap a lista elejére kerül
- hivatkozáskor a lista elejére kerül a lap, a végén van a legrégebben nem használt lap
- nem kell hosszadalmas keresés

## 154. MUTASSA BE A LEGRITKÁBBAN HASZNÁLT LAP (LFU) LAPCSERE STRATÉGIÁT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT (IMPLEMENTÁCIÓ NEM KELL)!

- a leggyakrabban használt lapok a memóriában

**Hátrány:**

- a valamikor sokat használt lapok a memóriában maradnak (öregítéssel lehet segíteni ezen)
- a frissen betöltött lapok könnyen kiesnek (frissen behozott lapok befagyasztása, page locking)

## 155. MUTASSA BE A MOSTANÁBAN NEM HASZNÁLT LAPOK (NRU) LAPCSERE STRATÉGIÁT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT (IMPLEMENTÁCIÓ NEM KELL)!

- hivatkozott (R) és módosított (M) bitek használata
- OS időközönként törli az R bitet, M bitet viszont őrizni kell (lap törlése információ veszteséghez vezetne)

**Négy prioritás**

1. nem hivatkozott, nem módosult
2. nem hivatkozott, módosított
3. hivatkozott, nem módosult
4. hivatkozott, módosult

- a módszer a legkisebb prioritásból választ véletlenszerűen

## 156. DEFINIÁLJA, HOGY MI A LAPOK ALLOKÁCIÓJA, ADJA MEG, HOGY MI A CÉLJA!

- lapok lefoglalás a tárban a folyamatok számára

**Cél:** a folyamat szempontjából az, hogy minél több lapja legyen bent a tárban

## 157. ADJA MEG, HOGY MI A VERGŐDÉS (TRASHING)!

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

**Oka:**

- folyamat: kevés lap van a tárban, gyakran hivatkozik a háttértáron lévőkre
- rendszer: túl sok a folyamat, ezek egymás elől lopkodják el a lapokat, mindegyik folyamat vergődni kezd

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

## 158. ADJA MEG, HOGY MI A LOKALITÁS! MIND A TÉRBELI, MIND AZ IDŐBELI KELL!

- statisztikai tulajdonság: a folyamatok egy időintervallumban csak címtartományuk szűk részét használják
- **időbeli:** a hivatkozott címre a közeljövőben nagy valószínűséggel újra hivatkozni fog (ciklusok, eljárások, verem változók stb.)
- **térbeli:** hivatkozott címek egymás melletti címre történnek (soros kód, tömbkezelés)

## 159. ADJA MEG, HOGY MI A MUNKAHALMAZ!

- az elmúlt Δ időben (munkahalmaz ablak) hivatkozott lapok halmaza
- a lokalitás miatt a folyamatnak nagy valószínűséggel ezekre a lapokra lesz szüksége a közeljövőben
- munkahalmaz mérete folyamatonként és időben is változik
- OS célja minden aktív folyamat számára biztosítani a munkahalmazt
- pontos méréséhez bonyolult hardver kellene, ezért fix időintervallumonkénti mintavételezéssel oldják meg

Mekkora legyen Δ?

- ha túl kicsi: nem tartalmazza az egész lokalitást
- ha túl nagy: több lokalitást is tartalmaz (feleslegesen)

## 160. ÍRJA LE, HOGY MI A KAPCSOLAT A LOKALITÁS ÉS A MUNKAHALMAZ KÖZÖTT!

- vergődés akkor lép fel, ha D > m (a lokalitások összes mérete nagyobb, mint a rendelkezésre álló memória mérete)

## 161. ÍRJA LE, HOGY MI A KAPCSOLAT A MUNKAHALMAZ ÉS A VERGŐDÉS KÖZÖTT!

- $WSS_i$: az i-ik folyamat munkahalmazának mérete
- a rendszer teljes lapigénye: $D = \sum WSS_i$
- a rendszerbeli lapkeretek száma: m
- vergődés akkor lép fel, ha D > m (a lokalitások összes mérete nagyobb, mint a rendelkezésre álló memória mérete)

## 162. DEFINIÁLJA, HOGY MI AZ ELŐRE LAPOZÁS!

- folyamat indításakor, illetve aktiválásakor az összes lapja a háttértáron van
- érdemes a várható munkahalmazt behozni a tárba
- akkor hasznos, ha az előre lapozás találati aránya nagy, különben pazarlás
- pl.: munkahalmaz modell használata esetén a felfüggesztett folyamat teljes munkahalmazát aktiválás esetén előre be lehet lapozni

## 163. ADJA MEG, HOGY MI A LAPOK TÁRBA FAGYASZTÁSA, ÍRJA LE, HOGY MIKOR JELENT SEGÍTSÉGET!

- az elindított perifériás műveleteknél a kijelölt címtartományt az átvitel idejére érdemes befagyasztani
- a beemelt lapokat az első hivatkozásig érdemes befagyasztani
- tipikus példák: OS magjának befagyasztása, I/O lapok befagyasztása
- megoldás módja: minden laphoz „lock” bit

I/O alatt: háttértárról beolvasás pufferbe (segédprocesszor)

- ha kész, a CPU megszakítást kap
- művelet közben a lapnak a memóriában kell lenni!

## 164. Sorolja fel, hogy az operációs rendszer mikor foglalkozik lapkezeléssel!

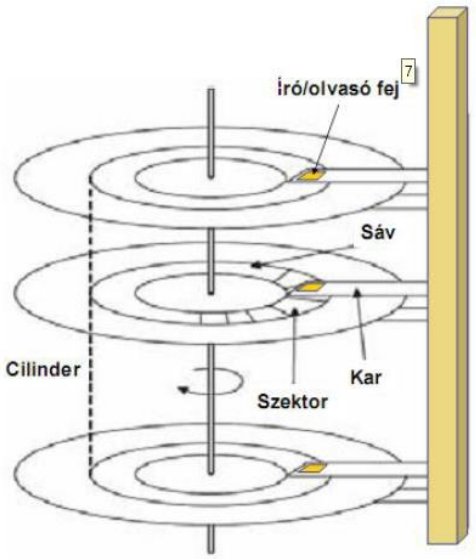
1. Folyamat létrehozása
   - programméret meghatározása
   - laptábla létrehozása
2. Folyamat végrehajtása
   - új folyamat esetén MMU reset
   - TLB kiürítése
3. Laphiba
   - a hibát okozó virtuális cím meghatározása
   - lap kivitele (felszabadítás), szükséges lap behozatala
4. Folyamat terminálása: a laptábla és a lapok felszabadítása

# Háttértár kezelés

## 165. Mondja meg, hogy miért van szükség háttértárra!

- központi tár drága és kicsi a tárolókapacitása
- a kikapcsolással az információk elvesznek a központi tárban

## 166. Rajzolja le a merevlemez fizikai szervezését!




## 167. Egy merevlemezben 3 db kétoldalas lemez van, lemezenként 512 sáv, sávonként 2048 szektor. Számolja ki, hogy az operációs rendszerben milyen logikai címmel rendelkezik a 2. lemez felső oldalának 86. sávjában található 122. szektor!

$b = s \cdot (i \cdot t + j) + k$; ahol:

- $b$: lineáris szektorcím
- $s$: egy sávon lévő szektorok száma (konstans)
- $t$: egy cilinderen lévő sávok száma (konstans)
- $i$: kijelölt cilinder sorszáma

- $j$: fej (lemez felület) száma
- $k$: sávon belüli szektorszám

$$b = 2048 \cdot (86 \cdot 3 \cdot 2 + 3) + 122 = 1063034$$

## 168. Egy merevlemezben 3 db kétoldalas lemez van, lemezenként 512 sáv, sávonként 2048 szektor. Adja meg, hogy az operációs rendszer 4168422 című logikai szektora a merevlemez hányadik lemezének, hányadik fejének, hányadik sávjának, hányadik szektorát jelenti!

$j = 512 \cdot \frac{2048}{4168422} \approx 3{,}975 \rightarrow$ fejszám = 4, lemezszám = 2

## 169. Adja meg, hogy a merevlemeznél mi a fejmozgási idő (seek time)!

- az az idő, amely alatt a fej a kívánt sávra (cilinderre) áll

## 170. Adja meg, hogy a merevlemeznél mi az elfordulási idő (latency time)!

- az az idő, amely alatt a kívánt szektor a fej alá fordul

## 171. Adja meg, hogy a merevlemeznél mi az átviteli idő (transfer time)!

- az információ átviteli ideje (transfer time): lineárisan függ az átviteli adatmennyiségtől

## 172. Rendezze nagyság szerinti sorrendbe (merevlemeznél): elfordulási idő, átviteli idő, fejmozgási idő!

- átviteli idő, elfordulási idő, fejmozgási idő

## 173. Mi a célja a lemezműveletek ütemezésének?

- multiprogramozott rendszerben egyszerre több folyamat verseng a háttértár perifériáért, így egyszerre több kérés várakozhat kiszolgálásra
- az ütemezési algoritmusok a kérések megfelelő sorrendbe állításával próbálják a rendszer teljesítményét növelni

**Cél:**

- az átlagos seek és a latency idő csökkentése
- természetesen így egyes folyamatok rosszabbul járnak, de a cél a globális teljesítmény növelése

## 174. Adja meg, hogy merevlemeznél mi az átbocsátó képesség!

- időegység alatt lebonyolított átvitelek száma

## 175. Adja meg, hogy merevlemeznél mi az átlagos válaszidő!

- a kéréstől a végrehajtásig eltelt idő

## 176. ADJA MEG, HOGY MEREVLEMEZNÉL MI A VÁLASZIDŐ SZÓRÁSA!

- folyamatok előre látható sebességgel fussanak, futásuk rajtuk kívül álló okok miatt ne ingadozzon nagyon (megjósolható viselkedés)

## 177. MUTASSA BE A MEREVLEMEZNÉL A SORRENDI KISZOLGÁLÁS (FCFS) ALGORITMUS MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- kiszolgálás a kérések érkezési sorrendjében

**Előnye:** viszonylag kicsi szórás

**Hátrányai:**

- kicsi átbocsátó képesség
- nagy az átlagos válaszidő

## 178. MUTASSA BE A MEREVLEMEZNÉL A LEGRÖVIDEBB FEJMOZGÁSI IDŐ (SSTF) ALGORITMUS MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- az aktuálishoz legközelebbi cilinderhez tartozó kérés kiszolgálása (ennek eléréséhez szükséges a legrövidebb idő)

**Előnyei:**

- teljesítménye jobb, mint az FCFS-é
- kis átlagos válaszidő

**Hátrányai:**

- nagy szórás
- fennáll a kiéheztetés veszélye: távoli cilinderre vonatkozó kérést a folyamatosan érkező közeli kérések mindig megelőzik
- közepes átbocsátás

## 179. MUTASSA BE A MEREVLEMEZNÉL A PÁSZTÁZÓ (SCAN) ALGORITMUS MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- az aktuális mozgási iránynak megfelelő kérések kiszolgálása
- ha nincs több ilyen kérés, akkor irányváltás történik

**Előnyei:**

- jobb, mint az SSTF, szórása is kisebb
- nagy átbocsátás
- kis szórás

**Hátránya:** közepes válaszidő

- sajátossága, hogy a középső cilindereket többször látogatja meg

## 180. MUTASSA BE A MEREVLEMEZNÉL AZ N-LÉPÉSES PÁSZTÁZÓ (N-SCAN) ALGORITMUS MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- egy irányba mozogva N kérést (amelyek a pásztázás elején már megérkeztek) szolgál ki
- a közben érkező kérésekre a következő irányváltás után kerül sor

**Előnyei:**

- nagy átbocsátás
- kis válaszidő (akkor sem nő meg, ha az aktuális cilinderre sok kérés érkezik)
- kis szórás (kisebb, mint a SCAN)

## 181. MUTASSA BE A MEREVLEMEZNÉL A KÖRBEFORGÓ-PÁSZTÁZÓ (C-SCAN) ALGORITMUS MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- csak az egyik irányú fejmozgás során történik a kérések kiszolgálása (a másik irányba a fej a legtávolabbi kérés cilinderére ugrik)
- az algoritmus elkerüli a középső sávoknak a külsőkhöz viszonyított magasabb fokú látogatottságát
- ez is lehet N lépéses

**Előnyei:**

- nagy átbocsátás
- kis válaszidő
- kis szórás

## 182. MI A RAID ÉS MI AZ ALAPÖTLET MÖGÖTTE?

- egy nagy kapacitású lemezegység helyett több kicsit használunk, amelyekre az információt hibajavító kódolással megtoldva szétterítik
- egy egység kiesése esetén az információ még visszaállítható

**Cél:**

- adatátvitel sebességét növelni
- adattárolás biztonságát növelni

**Alapötletek:**

- lemezegységek kétszerezése (disc shadowing, mirroring): az írásokat mindkét egységen elvégezzük, hiba esetén a másik példány használható ⇨ megbízhatóság nő, sebesség nem változik
- tároljuk egy adatbájt bitjeit külön tárolókon, párhuzamos hozzáférés ⇨ megbízhatóság kissé csökken, sebesség kb. 8x

## 183. MUTASSA BE A RAID 0 MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- non-redundant stripping
- egymás utáni blokkok külön háttértárakon helyezkednek el, nincs redundancia (tükör vagy paritás)
- adatátvitel sebessége megnő
- biztonság némileg csökken (több kisebb tároló közül gyakrabban hibásodik meg egy, mint egyetlen nagyobb kapacitású tároló)

## 184. MUTASSA BE A RAID 1 MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- disk mirroring
- mindkét háttértárnak van egy tükre
- hiba esetén a tükrön lévő adat használható, a hibás egység erről helyreállítható
- sokkal nagyobb biztonság, de a sebesség nem nő

## 185. MUTASSA BE A RAID 2 MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- memory-style error correcting
- minden bitet más tároló tárol
- a memóriákhoz hasonlóan paritásbiteket használunk
- 1 bit hiba a paritásbitekből javítható
- nagy sávszélesség és nagy biztonság kisebb redundanciával, mint RAID 1 esetén

## 186. MUTASSA BE A RAID 3 MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- bit-interleaved parity
- RAID 2 továbbfejlesztése
- ötlet: ez nem memória! Hiba detektálása minden háttértáron önállóan működik ⇨ elég egy paritásbit a javításhoz
- nagy sávszélesség és nagy biztonság nagyon kis redundanciával (egyetlen extra háttértárral)

### 187. MUTASSA BE A RAID 4 MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- block-interleaved parity
- mint RAID 4, de blokkos szervezés
- az egyik háttértár paritás-blokkot tartalmaz
- biztonságos, nagy adathalmaz kezelésénél nő az átviteli sávszélesség is
- kis adatméretek esetén (pl. 1 blokk) nem nő a sávszélesség (sőt: 1 blokk írása ⇨ P olvasása ⇨ adatblokk írása, P írása)

### 188. MUTASSA BE A RAID 5 MŰKÖDÉSÉT, ADJA MEG ELŐNYEIT ÉS HÁTRÁNYAIT!

- block-interleaved distributed parity
- mint a RAID 4, de a paritás blokkok elosztva
- ok: a RAID 4 esetén a paritás tárolót aránytalanul sokat használjuk ⇨ hamarabb meghibásodik
- itt a terhelés kiegyenlítődik

## ÁLLOMÁNYOK KEZELÉSE

### 189. ADJA MEG, HOGY MI A FÁJL!

- a létrehozó által összetartozónak ítélt információk gyűjteménye
- több százezer állomány egyidejűleg, egyedi azonosító (név) különbözteti meg őket (az elérési út is beletartozik
- az állomány elrejti a tárolásának, kezelésének fizikai részleteit:
  - melyik fizikai eszközön található (logikai eszköznevek használata)
  - a perifériás illesztő tulajdonságai
  - az állományhoz tartozó információk elhelyezkedése a lemezen
  - állományban lévő információk elhelyezkedése a fizikai egységen (szektor, blokk)
  - az információ átvitelénél alkalmazott blokkosítást, pufferelést

### 190. ADJA MEG, HOGY MI A KÖNYVTÁR ÉS A KATALÓGUS!

- az állományok csoportosítása, OS és a felhasználó szerint
- a könyvtár tartalmát katalógus írja le

## 191. Adja meg az állománykezelő feladatait!

- **információátvitel** (állomány és a folyamatok között)
- **műveletek** (állományokon, könyvtárakon)
- **osztott állománykezelés:** egyszerre több folyamat használhatja ugyanazt az állományt
- **hozzáférés szabályozása**
  - más felhasználók által végezhető műveletek korlátozása (access control)
  - tárolt információk védelme az illetéktelen olvasások ellen (rejtjelezés, encryption)
- **információ védelme a sérülések ellen, mentés**

## 192. Mutassa be az állományrendszer réteges megvalósítását!

- egymásra épülő programrétegek

**1. a perifériát közvetlenül kezelő periféria meghajtó:**

- a tár és a periféria közötti átvitel megvalósítása (device driver, átvitelt kezdeményező, megszakítást kezelő eljárások)

**2. elemi átviteli műveletek rétege:** a lineáris címzésű blokkok átvitele, átmeneti tárolása

**3. állományszervezés rétege:**

- háttértár szabad blokkjainak, ill. az állományhoz tartozó blokkok szervezése

**4. logikai állományszervezés:**

- kezeli a nyilvántartások szerkezetét
- azonosító alapján megtalálja az állományt
- szabályozza az állományszintű átvitelt

## 193. Mutassa be a háttértár szabad blokkjainak nyilvántartására használt bittérkép módszert annak előnyeivel és hátrányaival együtt!

- minden blokkra egy biten jelzi, hogy szabad-e
- a bittérkép a lemez kijelölt helyén van

**Előnye:** az egymás melletti szabad blokkok kiválasztása nagyon egyszerű

**Hátránya:**

- a bitvektort a memóriában kell tárolni, különben nem elég hatékony (sok blokk ⇨ sok memóriát igényel)

## 194. Mutassa be a háttértár szabad blokkjainak nyilvántartására használt láncolt lista módszert annak előnyeivel és hátrányaival együtt!

- az első szabad blokk címének tárolása, arra felfűzve a többi
- minden szabad blokk tartalmaz egy hivatkozást a következő szabad blokkra: a blokk területéből vesszük le a cím területét

**Hátrány:** nem hatékony, lassú a sok lemezművelet miatt

## 195. Mutassa be a háttértár szabad blokkjainak nyilvántartására használt szabad helyek csoportjainak nyilvántartása módszert annak előnyeivel és hátrányaival együtt!

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

## 196. MUTASSA BE A HÁTTÉRTÁR SZABAD BLOKKJAINAK NYILVÁNTARTÁSÁRA HASZNÁLT EGYBEFÜGGŐ SZABAD HELY MÓDSZERT ANNAK ELŐNYEIVEL ÉS HÁTRÁNYAIVAL EGYÜTT!

- egy táblázatban tároljuk az összefüggő szabad blokk címét (első blokk sorszáma, az egymás után következő szabad blokkok száma)
- hatékony, ha:
  - a szabad területek átlagos hossza jóval nagyobb, mint 1
  - az allokálásnál egymás melletti szabad területeket akarunk kiválasztani
- ha a táblázat kezdő cím szerint rendezett, akkor az egymás melletti szabad területek összevonása egyszerű

## 197. MUTASSA BE A LEMEZTERÜLET BLOKKJAINAK ALLOKÁCIÓJÁNÁL HASZNÁLT FOLYTONOS TERÜLET ALLOKÁCIÓJA MÓDSZERT ANNAK ELŐNYEIVEL ÉS HÁTRÁNYAIVAL EGYÜTT!

- az összetartozó információkat egymás melletti blokkokban tároljuk (első blokk sorszáma és a blokkok száma)

**Előnyei:**

- a tárolt információ soros és közvetlen elérése is lehetséges
- jól használható tárcsere által kirakott szegmensekre (tudjuk előre a méretet, az egymás melletti blokkok egyszerre való mozgatása az átvitel sebességét gyorsíthatja)

**Hátrányai:**

- külső tördelődés fennáll
  - a megoldáshoz itt is használhatók a már megismert algoritmusok (első illeszkedő, legjobban illeszkedő, legrosszabbul illeszkedő)
  - nagyméretű tördelődés esetén tömöríteni kell: egyszerűbb az egész tárat kimásolni egy másik háttértárra és rendezetten visszamásolni
- sokszor nem tudjuk előre, hogy hány blokkra lesz szükségünk
  - lefoglaláskor becsülni kell ⇨ a rossz becslés gondot okozhat (hiba és leállás, átmásolni az eddig lefoglalt területet egy másik helyre)

## 198. MUTASSA BE A LEMEZTERÜLET BLOKKJAINAK ALLOKÁCIÓJÁNÁL HASZNÁLT LÁNCOLT TÁROLÁS MÓDSZERT ANNAK ELŐNYEIVEL ÉS HÁTRÁNYAIVAL EGYÜTT!

- blokkokat egyenként allokáljuk, minden blokkban fenntartva egy helyet a következő blokk sorszámának (a rendszer az első és az utolsó blokk számát tárolja)

**Előnyei:**

- nincs külső tördelődés
- rugalmas: a lefoglalt terület növekedhet, csak a szabad blokkok száma szab határt

**Hátrányai:**

- csak soros elérés lehet
- a blokkok sorszámaival nő az állomány mérete (a blokkok sorszámai együtt tárolódnak a hasznos információval, a blokkos másoláskor a sorszámokkal külön kell foglalkozni)
- sérülékeny: egyetlen láncszem hibája a tárolt információnak jelentős részének elvesztését jelenti

## 199. ADJA MEG, HOGY MI AZ A FAT ÉS ÍRJA LE, HOGY HOGYAN MŰKÖDIK!

- állomány allokációs tábla (FAT – File Allocation Table)
- a láncolt tárolás módosított változata
- a láncelemeket az állományoktól elkülönítve tároljuk a táblázatban
- a táblázat minden eleme a lemez egy blokkjához tartozik, itt tárolhatjuk a következő blokk címét
- csak a legelső blokk címét kell tárolni
- a szabad helyek tárolására is alkalmas
- sérülés esetén nagy a gond

## 200. MUTASSA BE A LEMEZTERÜLET BLOKKJAINAK ALLOKÁCIÓJÁNÁL HASZNÁLT INDEXELT TÁROLÁS MÓDSZERT ANNAK ELŐNYEIVEL ÉS HÁTRÁNYAIVAL EGYÜTT!

- az állományhoz tartozó blokkok címei egy indextáblában vannak
- minden állománynak saját indextáblája van
- az indextáblák az adatterületen helyezkednek el

**Előnyei:**

- sérülés esetén csak az aktuális állomány veszik el
- közvetlen hozzáférés megvalósítása egyszerű
- alkalmas "lyukas" állományok tárolására (nem minden blokk tartalmaz valós információt, az üres blokkokhoz nem kell blokkot lefoglalni)

**Hátrányai:**

- az indextábla tárolása legalább egy blokkot elfoglal (kis állományok esetében pazarló)
- az indextábla mérete nem ismert ⇨ lehetővé kell tenni, hogy az növekedhessen
    - láncolt indexblokkok
    - többszintű indextábla: egy alsó szintű tábla indexei a következő szint indextábla blokkjaira vonatkoznak
    - kombinált mód: kis állományok esetében egyszintű, nagyobb állományokhoz többszintű indextábla

## 201. ADJA MEG, HOGY AZ ÁLLÁNYOK BELSEJÉNEK ELÉRÉSÉHEZ AZ OPERÁCIÓS RENDSZER MILYEN MÓDSZEREKET HASZNÁLHAT!

**Soros (sequential):**

- a tárolt információt csak a byte-ok sorrendjében lehet olvasni
- sok feldolgozási feladathoz elegendő a soros hozzáférés

**Közvetlen (direct):**

- a tárolt elemek bármelyikét el lehet érni, ehhez meg kell adni az információ elem állományon belüli sorszámát

**Indexelt, index-szekvenciális (ISAM - Index Sequential Access Method):**

- tartalom szerint akarunk hozzáférni
- kulcs szerint rendezzük az állomány rekordjait (a kulcsok egy index fájlban vannak rendezetten tárolva)
- pl. árunyilvántartás

**Partícionált:**

- az állományt soros rész-állományok alkotják, az állomány tartalmaz egy nyilvántartást arról, hogy a partíciók hol helyezkednek el a fájlban

## 202. ADJA MEG, HOGY A KÖNYVTÁR NYILVÁNTARTÁSBAN TALÁLHATÓ BEJEGYZÉSEK MIT TARTALMAZNAK!

**Állomány neve:**

- az állományt a könyvtárban a neve alapján azonosítjuk
- könyvtáranként egyedi, homogén (lehet bármilyen karaktersorozat), részekre bontott (pl. név, típus (kiterjesztés – extension), verziószám)

**Állomány fizikai elhelyezkedését leíró információk:**

- hossz, a hozzá tartozó háttértár blokkjainak leírása, hozzáférés módja

**Állomány kezeléséhez kapcsolódó információk**

- típusa
- tulajdonosának, létrehozójának (owner) azonosítója
- időpontok (létrehozás, utolsó módosítás, utolsó hozzáférés, argumentumának utolsó módosítása, érvényessége)
- hozzáférési jogosultságok
- hivatkozás számláló, amennyiben az állományra több különböző néven és/vagy helyen hivatkoznak

## 203. ADJA MEG, HOGY ÁLTALÁBAN AZ OPERÁCIÓS RENDSZEREK AZ ÁLLOMÁNYOKON MILYEN MŰVELETEKET TESZNEK LEHETŐVÉ!

**Átvitel, írás, olvasás:**

- közvetlen átvitel esetén, információ címe szükséges
- soros hozzáférésnél az aktuális pozíciót a rendszer növeli és tárolja
- szimultán írás-olvasás esetén soros hozzáférésnél közös vagy több pozíció használata

**Hozzáadás (append):**

- az állomány végéhez új információt írunk ⇨ az állomány mérete növekszik, esetlegesen új blokk lefoglalása

**Pozicionálás:** soros hozzáférés esetén megadhatjuk az aktuális pozíciót

**Állomány megnyitása:**

- az állomány megkeresése, fizikai elhelyezkedés meghatározása
- hozzáférési jogosultságok ellenőrzése
- az elvégezendő műveletek megadása
- osztott állománykezelés szabályozása
- soros hozzáférés pozíciójának beállítása
- a műveletek hatékonyabbak, ha nem kell mindegyiknél az állományt a nyilvántartásból kikeresni
- a sikeres megnyitás után OS felépít egy adatszerkezetet, amelyre a további műveletek hivatkoznak

**Állomány lezárása:**

- az átviteli műveletek befejezése: puffer esetén a ki nem írt információ kiírása, osztott állománykezelésnél az állomány felszabadítása

**Állomány végrehajtása:**

- az OS létrehoz egy új folyamatot, a programállományt betölti a folyamat tárterületére és elindítja

**A könyvtárra is hatással levő műveletek:**

- állomány létrehozása (új bejegyzés, blokkok lefoglalása)
- állomány törlése (bejegyzés megszüntetése, blokkok felszabadítása)

## 204. ÍRJA LE, HOGY MI AZ OSZTOTT ÁLLOMÁNYKEZELÉS, MILYEN MEGOLDÁSOKAT HASZNÁLNAK ÁLTALÁBAN!

- ez is egy erőforrás, amelyet egyidejűleg több folyamat is használni akar
- csak olvasás esetén osztottan gond nélkül használható
- írás esetén kölcsönös kizárással kell védeni a folyamatot
  - egész állományra vonatkozó szabályozás (file lock):
    - az állományt először megnyitó folyamat definiálja, hogy a későbbi megnyitási kérelmekből mit engedélyezhetünk
      - kizárólagos használat: több megnyitást nem engedünk
      - többi folyamat csak olvashatja
      - több folyamat is megnyithatja írási joggal
    - írás esetén az olvasó folyamatok mikor veszik észre a változást:
      - azonnal
      - ha a folyamat lezárta az állományt: addig a régi tartalmat látják
  - állomány részeire vonatkozó kizárás:
    - az egész állományra vonatkozó kizárás néha túlzottan óvatos
    - az állományok belső szerkezete szerint a folyamatok az állománynak csak egy részét foglalják le:
      - az átvitel után a lefoglalt rekordokat felszabadítja
      - kölcsönös kizárás csak a használt részekre
      - felléphet holtpont, ill. a kiéheztetés veszélye
    - rekordonként lehet kizárásokat definiálni

# ELOSZTOTT RENDSZEREK

## 205. ADJA MEG, HOGY MIK AZ ELOSZTOTT RENDSZEREK, MI INDOKOLJA A HASZNÁLATUKAT (MOTIVÁCIÓK)!

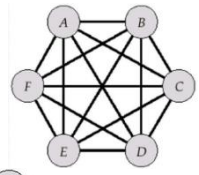
- egymással kommunikációs hálózaton kapcsolódó több-számítógépes rendszerek, amelyben a számítógépek önállóak

**Motivációk:**

- **erőforrások megosztása:**
  - speciális eszköz, nem kell mindenhova
  - kliens (erőforrás használó)
  - szerver (erőforrás szolgáltató): dedikált (csak szolgáltat)/nem dedikált (használ is)
- **megbízhatóság:**
  - ki lehet használni az erőforrások redundanciáját
  - ha egy csomópont kiesik, nem gond ⇨ a feladatokat átveszik mások.
- **terhelés megosztás:**
  - a feladatok egyenletes elosztása
  - nem eléggé terhelt csomópontokat terhelhetjük, teljesítmény nő
- **sebesség növekedés:**
  - program felosztása párhuzamosan végrehajtható részekre
  - párhuzamos feldolgozás CPU-k között
- **kommunikáció:** a meglévő alacsony szintű kommunikációra egyéb is épülhet: email, messenger stb.

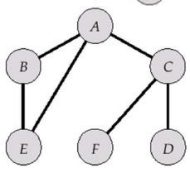
## 206. Rajzoljon fel legalább 3 kommunikációs hálózati topológiát, adja meg az előnyöket és a hátrányokat!

**Teljesen összekapcsolt:**



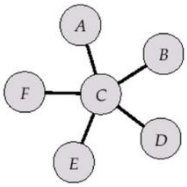

- drága (négyzetes növekedés)
- gyors
- megbízható

**Részlegesen összekapcsolt:**




- olcsóbb
- lassabb
- kevésbé megbízható

**Csillag:**

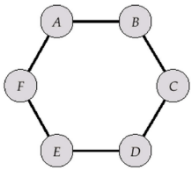



- olcsó
- gyors (de túlterhelődhet)
- megbízható
- csomópont kiesés során a részek magukra maradnak

**Kapcsolt (switched):**

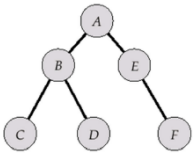
- csillag topológia
- a switch tetszőleges pont-pont kapcsolatot tud létrehozni a forgalom függvényében

**Gyűrű:**



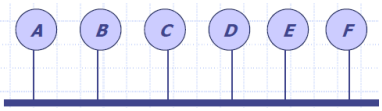

- változatok: gyűrű, kétirányú gyűrű, kettős gyűrű
- olcsó
- lassú
- megbízhatatlan

**Faszerkezet:**




- olcsó
- gyors
- egyes állomások kiesése a hálózat széttöredezését okozza

**Busz:** közösen használt csatorna




- olcsó
- gyors
- egyes állomások kiesése a többit nem érinti
- ha a busz kiesik, az katasztrofális

## 207. Adja meg, hogy kommunikációs hálózatoknál mi a forgalomirányítás (routing), mutassa be a fix útvonal módszert, adja meg az előnyöket és hátrányokat!

**Forgalomirányítás:**

- ott van rá szükség, ahol nincs teljes összekötés
- táblázat alapján határozza meg az útvonalat
  - Táblázat:
    - lehetséges útvonalak
    - egyéb információk: csatorna sebessége, költsége, terhelése
    - aktualizálni kell!

**Fix útvonal:** az üzenetváltás rögzített (legrövidebb, legkisebb költségű) útvonalon történik

## 208. Adja meg, hogy kommunikációs hálózatoknál mi a forgalomirányítás (routing), mutassa be a virtuális áramkör módszert, adja meg az előnyöket és hátrányokat!

**Forgalomirányítás:** *207. kérdés*

**Virtuális áramkör:** felépül - forgalom - leépül

## 209. Adja meg, hogy kommunikációs hálózatoknál mi a forgalomirányítás (routing), mutassa be a dinamikus útvonal módszert, adja meg az előnyöket és hátrányokat!

**Forgalomirányítás:** *207. kérdés*

**Dinamikus útvonal:** minden üzenethez keresnek útvonalat, az állomások üzenetenként döntik el, hogy azt merre küldjék tovább

## 210. Definiálja, hogy mi az osztott csatornahasználat, mutassa be a tanult módszereket!

- több állomás által közösen használt csatorna (pl. busz, vagy gyűrű szerkezet).
- egyszerre csak egy állomás használhatja
- ha többen egyszerre használják ⇨ hiba

**Kezelés:**

- CSMA/CD (Ethernet)
- Token passing
- Message slots

## 211. Mutassa be az ISO-OSI hálózati kommunikációs rétegszerkezet modellt!

**Fizikai (physical):** mechanika, elektronika

**Adatkapcsolati (link):** adatcsomagok két pont közötti átvitele, fizikai szintű hibák kezelése

**Hálózati (network):** két végpont közötti információátvitel, forgalomirányítás (pl. routerek)

**Szállítási (transport):**

- üzenetek átvitele, csomagokra bontása és összeállítása, adó-vevő információforgalom vezérlése, fizikai címek generálása

**Viszony (session):**

- kapcsolatfelvétel, szinkronizáció (pl. távoli bejelentkezés kommunikációs protokollja)

**Megjelenítési (presentation):** különböző konverziók (pl. karakter kód)

**Alkalmazási (application):** felhasználói szintű szolgáltatások (telnet, ftp, e-mail stb.)

- két végpont minden rétege között lesz kapcsolat, a szabályok a protokollok

## 212. Adja meg, hogy elosztott operációs rendszereknél mi a hálózati számítási modell!

- a felhasználó tudja, hogy melyik erőforrás melyik állomáson van
- meg kell nevezni a szolgáltatót és az erőforrást is
- pl.: távoli terminál (remote login), állományok átvitele (file transfer)

## 213. ADJA MEG, HOGY ELOSZTOTT OPERÁCIÓS RENDSZEREKNÉL MI AZ ELOSZTOTT SZÁMÍTÁS MODELL!

- az OS már több gépen fut, a felhasználó nem tudja, hogy helyi vagy távoli erőforrást lát-e
- távoli erőforráshoz szerver-kliens információcsere kell (middleware)

## 214. ADJA MEG, HOGY ELOSZTOTT RENDSZER ESETÉBEN ELOSZTOTT SZÁMÍTÁSI MODELLT HASZNÁLVA MI AZ ADATVÁNDORLÁS (DATA MIGRATION)!

- egy távoli állományt (vagy egy részét) a helyi gépre másolja, a munka végeztével törli, ha módosult akkor visszaírja a távoli gépre

## 215. ADJA MEG, HOGY ELOSZTOTT RENDSZER ESETÉBEN ELOSZTOTT SZÁMÍTÁSI MODELLT HASZNÁLVA MI A FELDOLGOZÁS VÁNDORLÁS (COMPUTATION MIGRATION)!

- ha sok az adat, akkor inkább a feldolgozó programot és az eredményt másolja

## 216. ADJA MEG, HOGY ELOSZTOTT RENDSZER ESETÉBEN ELOSZTOTT SZÁMÍTÁSI MODELLT HASZNÁLVA MI AZ ÜZENETKÜLDÉS (MESSAGE PASSING)!

- az ügyfél üzenetben megadja a kiszolgálónak műveletet (feldolgozási igényt), majd tovább megy (mehet)
- a feldolgozó üzenetküldéssel válaszol

## 217. ADJA MEG, HOGY ELOSZTOTT RENDSZER ESETÉBEN ELOSZTOTT SZÁMÍTÁSI MODELLT HASZNÁLVA MI A FOLYAMAT VÁNDORLÁS (PROCESS MIGRATION)!

- a folyamat, vagy annak egyes párhuzamos részei nem ott futnak, ahol elindították
  1. csomópontok terhelése egyenletes,
  2. végrehajtás a párhuzamos feldolgozás miatt felgyorsul,
  3. az elindított folyamat számára kedvező környezetben fut (HW, SW),
  4. adatok gyorsabb elérése adott csomóponton.
- a felhasználó ezen részleteket nem látja

## 218. ÍRJA LE, HOGY ELOSZTOTT OPERÁCIÓS RENDSZER ESETÉBEN MIT ÉRTÜNK SKÁLÁZHATÓSÁG ALATT!

- a rendszer növekedésével ne kelljen az rendszer architektúráját vagy a szoftver alkalmazásokat megváltoztatni
- növekvő terheléshez alkalmazkodjon, ne törjön le hirtelen
- további csomópontok beállításakor a teljesítmény ne romoljon, ne legyen túlterhelődés
  - hibatűréssel azonos tervezési elvek:
    - redundancia kell a rendszerbe: a kiesett elemek helyére léphetnek, ill. kiszolgálják a növekvő terhelést
- központosított szolgáltatások kerülése: kiesés végzetes lenne, növekedés túlterheléshez vezetne
- teljesen elosztott nehéz ⇨ fürtözés (cluster) (a csomópontokat csoportokba, fürtökbe szervezik, amik lazán csatoltak)