Informatika 1 vizsgafeladatok

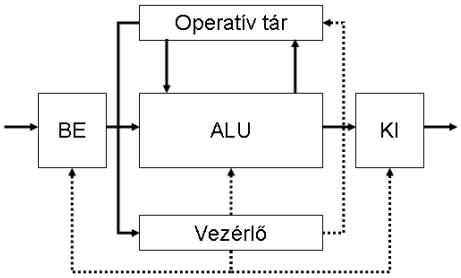
Számítógép architektúrák témakör

Számítógép architektúra:

Az elemi áramkörökből felépített funkcionális egységek alkotta hardver és az operációs rendszer között rész. Az architektúra az információfeldolgozás elméleti modelljeinek konkrét megvalósulása.

Neumann modell megvalósítása:

- belső programtárolás illetve vezérlés
- utasítás és adat azonos közegen és formában tárolva
- utasítás számláló dönti el adat vagy utasítás
- szekvenciális utasítás végrehajtás
- egydimenziós, lineáris címzésű memória
- bináris számábrázolás



CISC gép jellemzői (összetett utasításkészletű számítógép):

- sokfajta, bonyolult utasítás
- sokfajta, bonyolult címzési mód
- összetett adattípusok
- magas szintű programnyelvi egységek támogatása
- bonyolult hardver
- hardvertámogatás miatt könnyű szoftvert írni
- mikrovezérelt
- pl. INTEL 80386 mikroprocesszor

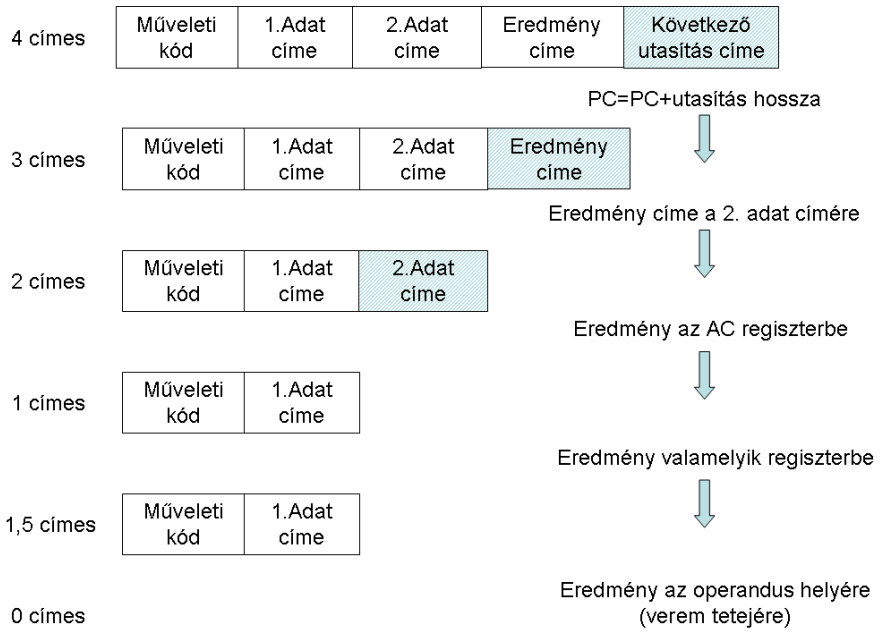
RISC gép jellemzői (csökkentett utasításkészletű számítógép):

- kevés, nagyon egyszerű, gyorsan elvégezhető, egyforma hosszú (1 órajel idő alatt végrehajtható) utasítás
- kevés, egyszerű címzési mód
- utasítások átlapolása (pipe-line) megvalósítható
- tárolóhoz csak írási-olvasási utasítás
- nagyon sok belső regiszter
- egyszerű, gyors hardver (kevesebb áramkör)
- szoftvertámogatás miatt nehéz alacsonyszintű programozás
- huzalozott vezérlő egység
- pl. IBM RISC 6000 gépcsalád

Utasításrendszer jellemzése (utasítás részei)

1. műveleti kód
2. operandusok
3. következő utasítás címe (elmaradhat)

Címzési módok:



Címzési módok:

<u>1 komponensű</u> címzés (egy egységből közvetlenül áll elő a cím)

1. *direkt címzés* (memória és regisztercímzés):
   - a címmező az operandus címét tartalmazza
   - változók támogatása
2. *indirekt címzés* (memória és regiszter címzés):
   - a címmező annak a memóriahelynek a címét tartalmazza, ahol az operandus címe található
   - pointerek támogatása
3. *immediate* címzés:
   - a címmező magának az operandusnak az értékét tartalmazza
   - konstansok támogatása

<u>2 komponensű</u> címzés (két különálló címrészből (aritmetikával) keletkezik effektív cím)

1. *indexelt* címzés:
   - a cím a címmező tartalma meg egy speciális regiszter (indexregiszter) tartalmának az összege
   - tömbök, rekordok kezelésének támogatása
2. *bázisrelatív* címzés:
   - a cím a címmező tartalma meg egy speciális regiszter (bázisregiszter) tartalmának az összege
   - a bázisregiszter tartalmazza a program kezdőcímét, az utasítás címmezeje pedig a program elejétől való távolságot.
   - a bázisregiszter tartalmának megváltoztatásával a program a memóriában bárhova elhelyezhető
   - áthelyezhető (relokálható) programok támogatása
   - virtuális tárkezelés (lapozás) illetve a szegmentálás támogatása

<u>több komponensű</u> címzés (több különböző részből keletkezik az effektív cím)

stack pointeren keresztüli címzés

- az utasítás nem tartalmaz címmezőt, a címet egy speciális regiszter (stack pointer) tartalma jelenti
- függvényhívások támogatása
- függvények paraméterátadásának támogatása

Magasszintű nyelvek támogatása:

1. speciális adattípusok támogatása
    - BCD, lebegőpontos számok
    - sztringek, karakterláncok
2. Összetett adatszerkezetek kezelése
    - tömbök
    - rekordok
    - listák, láncolt listák

Funkciók támogatása:

- rendszerhívások támogatása: a rendszernek bizonyos szolgáltatásai ezeken a rendszerhívásokon keresztül lesznek elérhetők
- védelmi rendszer támogatása: meg kell védeni a rendszert magát illetve multitasking esetén egyes felhasználókat önmaguktól és egymástól is
- párhuzamos futtatás támogatása

Programszerkezetek támogatása:

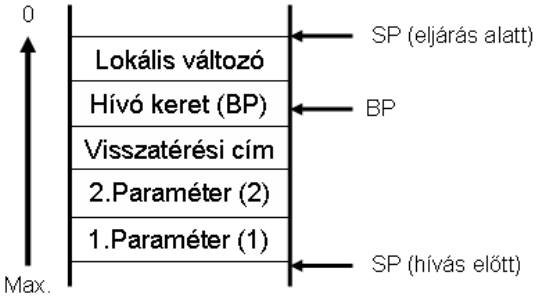
- eljárások, függvények
- szubrutinhívás
- verem használat
- programmodulok kezelése

Eljáráshívások futási modellje:

1. globális változók
    - hívás előtt és után is értékkel bírnak
    - lehetnek fix tár címen
2. paraméterek
    - csak eljárásokon belül bírnak értékkel
    - nem helyezhetőek el fix címen
    - a stacken keresztüli paraméterátadás
3. lokális változók
4. visszatérési érték

Eljáráshívás Stack Frame-en (verem keret) keresztül: (kisebb memóriacímek felé nő)

```
Eljárás:
function f(i,j:integer):integer;
   var l:integer;
   begin … end;
Hívás:
   f(1,2);
```



Hívás

1.	1. paraméter berakása a verembe	MOV AX,1
		PUSH AX
1.	2. paraméter berakása a verembe	MOV AX,2
		PUSH AX
1.	Szubrutinhívás (visszatérési érték elmentése)	CALL f

Eljárás

4.	Hívó kerete címének mentése a verembe	PUSH BP
5.	Hívott verem keretének beállítása	MOV BP,SP
6.	Helyfoglalás a 2 byte hosszú lokális változónak	SUB SP,2
7.	Eljáráson belüli keret-relatív hivatkozás a lokális paraméterre	MOV AX, [BP-2]
8.	Visszatérési érték beállítása	MOV AX

Visszatérés

9.	Lokális változó helyének felszabadítása	MOV SP,BP
10.	Hívó keret visszaállítása	POP BP
11.	Visszatérés a hívás helyére, paraméterek lebontása (4 byte)	RET 4

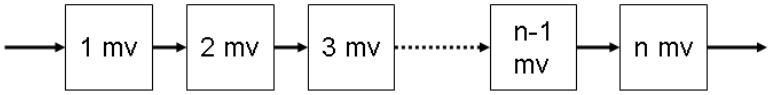
Utasítás végrehajtás fázisai:

1. utasítás felhozatal: külső gépi ciklus, memóriaolvasás
2. értelmezés: utasítás dekódolása, belső művelet
3. paraméter felhozatal: külső gépi ciklus, memóriaolvasás
4. művelet: művelet elvégzés, belső művelet
5. eredmény kivitele: külső memória, I/O, belső regiszterbe írás

Utasítás végrehajtás gyorsítása:

- utasítás alapfázisainak rövidítése (működési frekvencia növelése)
- szükséges gépi ciklusok számának csökkentése: regiszterek alkalmazása, adatszélesség növelése (4,8,16,32 bit)
- gépi ciklushossz rövidítése: egy gépi ciklus végrehajtásához szükséges idő, memóriához fordulás idejének csökkentése, hierarchikus memória felépítés (CACHE memória)
- műveletvégzés gyorsítása: huzalozott vagy mikroprogramozott logika
- memóriához fordulás és belső munka párhuzamosítása: több utasítás, vagy egyes utasítások részműveleteinek végrehajtása egyidőben: PIPE-LINE

PIPE-LINE alkalmazása:



elv: az utasítást szekvenciálisan végrehajtandó műveletekre bontjuk. Minden egyes utasítás a feldolgozás más-más részfázisában van.

Elindul az algoritmus végrehajtása az első műveletvégzőbe behozzuk az első utasítás kódját (utasítás felhozás). A második ütemben az első műveletvégző kimenetéről az első utasítás felhozott kódját beteszem a második műveletvégzőbe, miközben az első elemi műveletvégző felveheti a következő utasítást és így tovább…

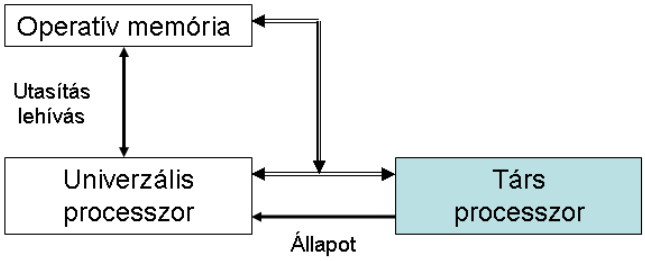
- utasítás lehívás
- utasítás dekódolás
- utasítás végrehajtás

4 mv				1 utasítás	2 utasítás	3 utasítás
3 mv			1 utasítás	2 utasítás	3 utasítás	4 utasítás
2 mv		1 utasítás	2 utasítás	3 utasítás	4 utasítás	5 utasítás
1 mv	1 utasítás	2 utasítás	3 utasítás	4 utasítás	5 utasítás	6 utasítás

Egymásra hatások:

- feldolgozási egymásra hatás: két egymást követő utasítás ugyanazt az erőforrást igényli
- adat egymásrahatás: utasítás egyik operandusa az előző, utasítás végrehajtásának eredménye
- procedurális egymásrahatás: a második utasítás függhet az előző által meghatározott úttól

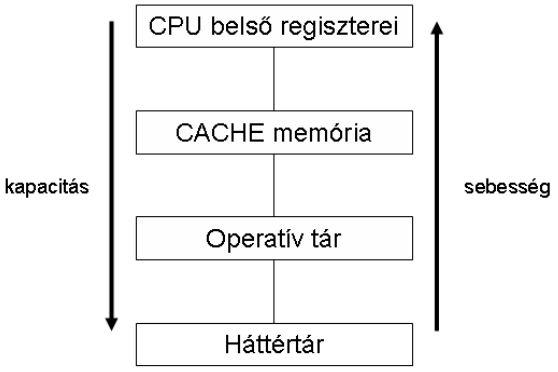
Processzor tehermentesítése:



- bizonyos feladatokra specializált processzorok
- a lehívott utasítást mindkét processzor érzékeli, az hajtja végre, amelyik sajátjának ismeri fel
- vagy az univerzális processzor, vagy a társprocesszor dolgozik
- a specializált társprocesszor a központi egység sínjére csatlakozik, így a számára szükséges adatokhoz közvetlenül hozzáfér
- ha az univerzális processzor egy olyan műveleti kódot érzékel, melyet a társprocesszor fog végrehajtani, akkor felfüggeszti a működését és várakozik az eredményre, majd annak elkészültét a társprocesszor a kész jelzésén keresztül jelenti
- a társprocesszor működése alatt az univerzális processzor áll

Memória hierarchikus szervezése:

- a kommunikáció a szintek között blokkos
- blokk: két memória között mozgatott legkisebb információs egység
- tranziens állapottól eltekintve a felső szint részhalmaza az alsó szintnek
- sikeres hozzáférés a felsőbb szinthez (HIT ratio=sikeres hozzáférés/összes hozzáférés)



Típus	Kapacitás	Hozzáférési idő
Belső regiszterek	10-100*16-64 bit	Ps
CACHE memória	n*100 kByte	10 ns
Operatív tár	1 Mbyte - 1 GByte	40-100 ns
Háttértár	100 Mbyte - 100GByte	ms

Lokalitási elvek:

- időbeli lokalitás: egy hivatkozott címet a folyamat várhatóan a közeljövőben újra használni fog.
- térbeli lokalitás: az időben egymáshoz közel álló hivatkozások nagy valószínűséggel egymáshoz közeli címekre történnek.

CACHE memória:

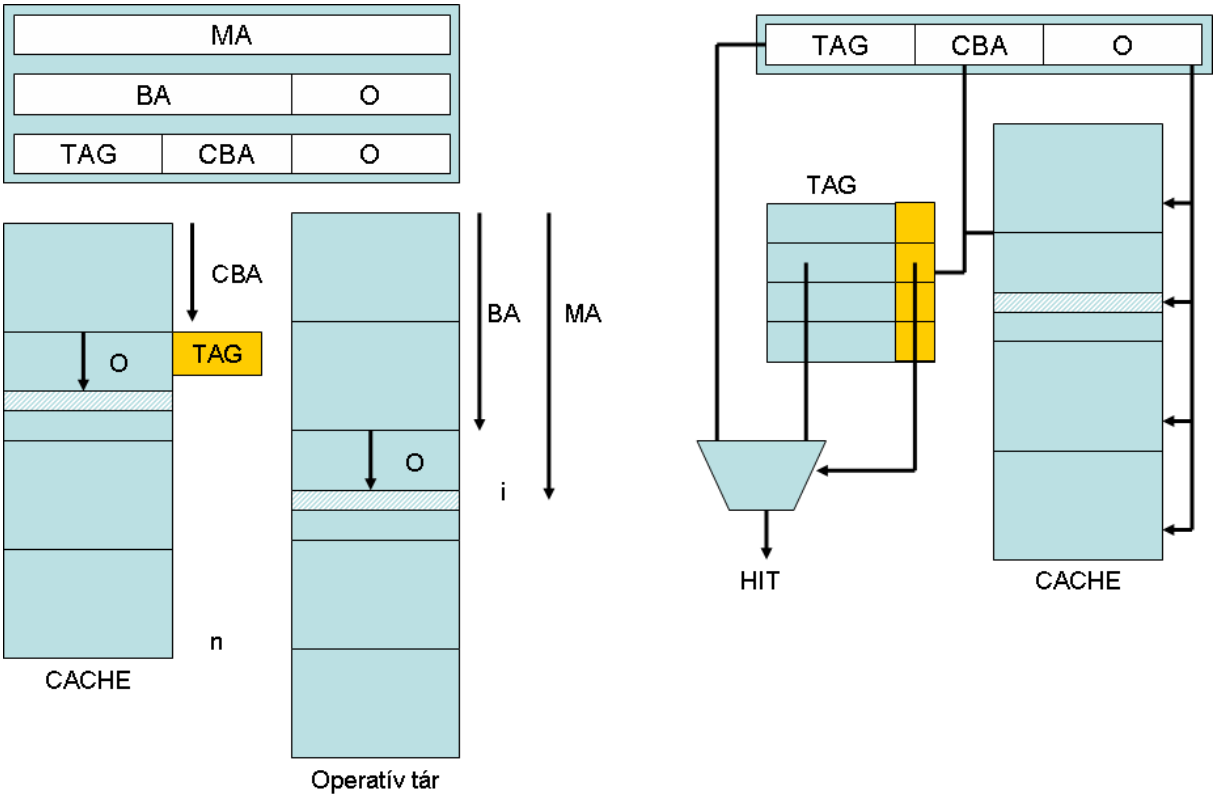
- a virtuális tárkezelés egy gyors, de viszonylag kis kapacitású operatív tároló és egy lassú, de nagy kapacitású háttértároló megfelelő együttműködtetésével a felhasználó számára (majdnem) átlátszó módon egy nagy kapacitású és gyors tároló érzetét biztosítja
- minden időpillanatban a cache tároló az operatív tároló bizonyos rekeszeinek másolatát tárolja, a másolatok meg vannak jelölve a nekik megfelelő operatív tárolóbeli címekkel
- ha a processzor egy operatív tárolóbeli rekeszhez kíván fordulni, akkor annak címét keresi a cache tároló jelölő mezejében. Ha megtalálja, akkor a kívánt átvitel a processzor és a cache között azonnal megtörténik, ha nem találja meg, akkor a kívánt operatív tárolóbeli rekesszel veszi fel a processzor a kapcsolatot, és egyben a cache-ben is elhelyezi annak másolatát

Leképzési módszerek: operatív memória meghatározott blokkja, hogyan kerül a CACHE-be

- Közvetlen (direkt) leképzés
- Részben asszociatív (n-utas direkt) leképzés
- Teljesen asszociatív leképzés

Közvetlen (direkt) leképzés:

- merev, egyszerűen realizálható szabály a leképzéshez
- ha a cache-ben van n db modul, akkor azt, hogy valamelyik i. blokk hová kerülhet a cacheben, hogy az i-nek veszem a modulo n-es osztáselemét
- O: offset: a blokk elejétől való eltolást adja meg
- minden cache-blokkhoz hozzá van rendelve a TAG-rész
- bekapcsoláskor minden rekesz érvénytelen TAG-et fog kapni
- a TAG-regiszter bitjeinek a számától függ a cache memória mérete



- előny: egyszerű hardvert igényel, gyors működés
- hátrány: rossz a kihasználtsága, nem teljes körű felhasználhatóság

Részben asszociatív (n-utas direkt) leképzés:

- az eredeti cache-t osszuk fel n db részre
- 1 részt kezeljünk direkt cache módjára
- a TAG-résznek kell tartalmaznia, hogy melyik cache-részben van benne a keresett információ
- egy blokk több helyre kerül

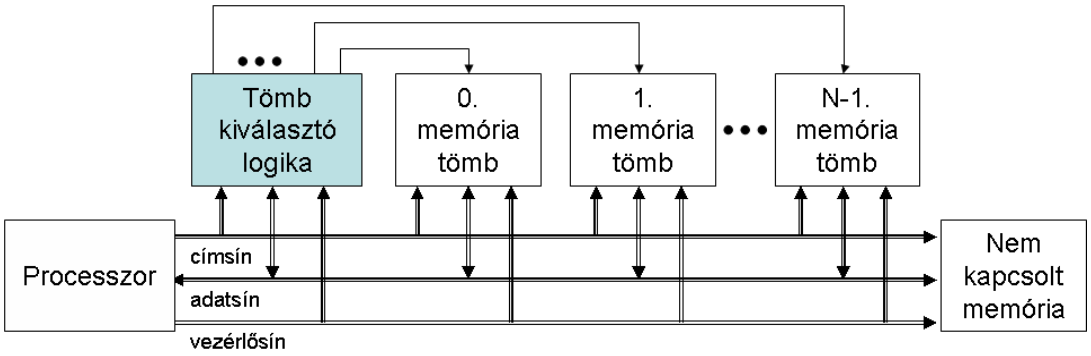
Teljesen asszociatív leképzés:

- bármelyik memóriabeli blokkot a cache bármelyik blokkjába betölthetem (nincs CBA)
- ha a memóriához fordulok egy adott címmel a cache-ben lévő összes blokk TAG-jét meg kell vizsgálnom
- minden TAG-ot egyszerre komparálunk

CACHE kezelési stratégiák:

1. Behozatali (fetch policy) stratégiák:
   - közvetlen igény szerinti behozatal: ha olvassuk és nincs ott, akkor behozzuk
   - előrelátó stratégia: úgy fog betölteni egy blokkot, ha az i.-re hivatkozom, akkor automatikusan be fogja tölteni az i+1.-et is
   - szelektív behozatali stratégia:
2. Írási (write policy) stratégiák:
   - write-back: egy írásnál adott esetben ha benne van a cache-ben, akkor csak a cache-ben írja át. új blokk behozatalánál nem lehet simán felülírni a tartalmat, akkor kell egy adminisztrációs bit minden tartalomhoz, amiben jelzi, hogy írt-e bele, ha ír bebillenti, akkor a hardver ha másik blokkot akar behozni a helyébe, akkor előtte ezt ki kell menteni.
   - write-trough: ha írni akarok, és a hivatkozott információ bent van (találatot jelez), akkor beírom a cache-ben lévő rekeszbe is, és ezzel egy ütemben (de nem egy időben), beírom a memóriába is (keresztülírás). ha írni akarok, de nincs találat, akkor az áthelyezés (az allocation) azt fogja jelenteni, hogy beírja a főmemóriába, és utána behozza a cache-be.
3. Blokk behozatali (block replacement policy):
   - direkt leképzés esetén minden blokknak megvan a meghatározott helye: nincs behozatali stratégia
   - legrégebben használt (LRU)
   - legrégebben betöltött (FIFO)
   - legritkábban használt (LFU)
   - véletlenszerű (RANDOM)

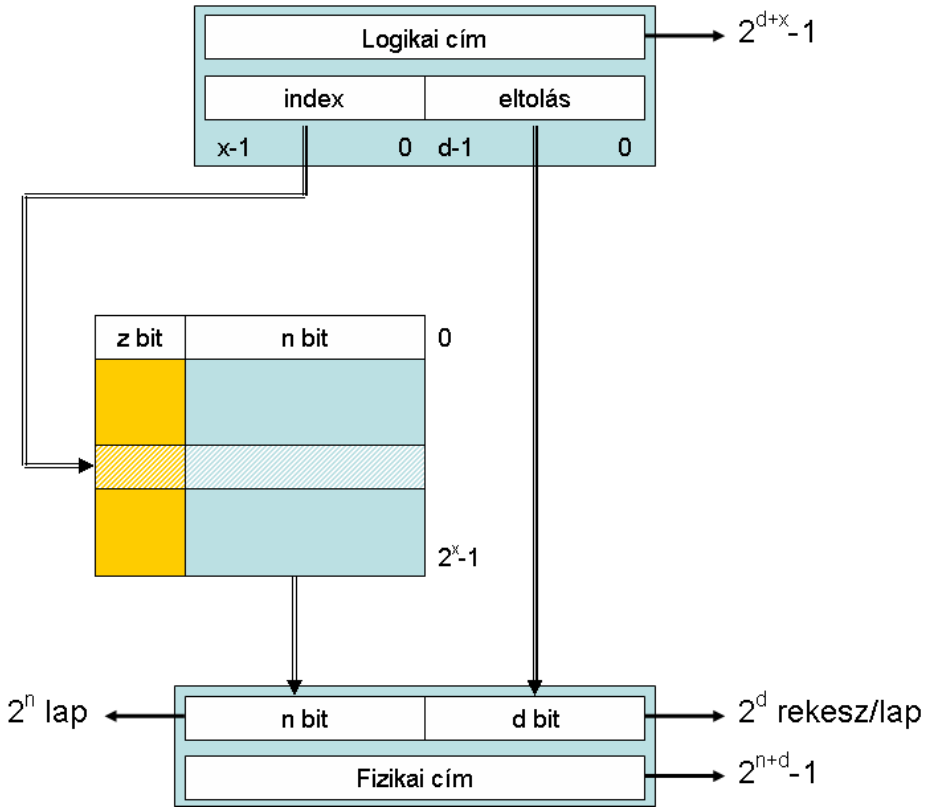
Tömbkapcsolás: túl kicsi a processzor által fizikailag megcímezhető tárterület



- feladata a processzor címtartományának kibővítése
- egy tömbkiválasztó logika, mely a processzor egyik perifériájaként működik, több tárolótömb közül egyet kiválaszt (engedélyez), az összes többit pedig letiltja
- a perifériába kell beírni azt, hogy melyik memóriát szeretnénk használni a fixen bent tartott memória mellett. A memória modulokat egyébként azonos címtartományra rakjuk, de csak azt tudjuk kezelni, amit a periféria éppen kiválaszt.
- a nem kapcsolt memória az alapprogramot tartalmazza

- előny:nagyon egyszerű, könnyen bővíthető és alig okoz futásiidő-növekedést
- hátrány: durva és merev tárolófelosztás, Minden pillanatban csak egyetlen memóriát választunk ki, egy tömbnek mindig elérhetőnek kell lennie, ami az operációs rendszer közös részeit tartalmazza: nem kapcsolható memória

Indexelt leképzés: túl kicsi a processzor által fizikailag megcímezhető tárterület



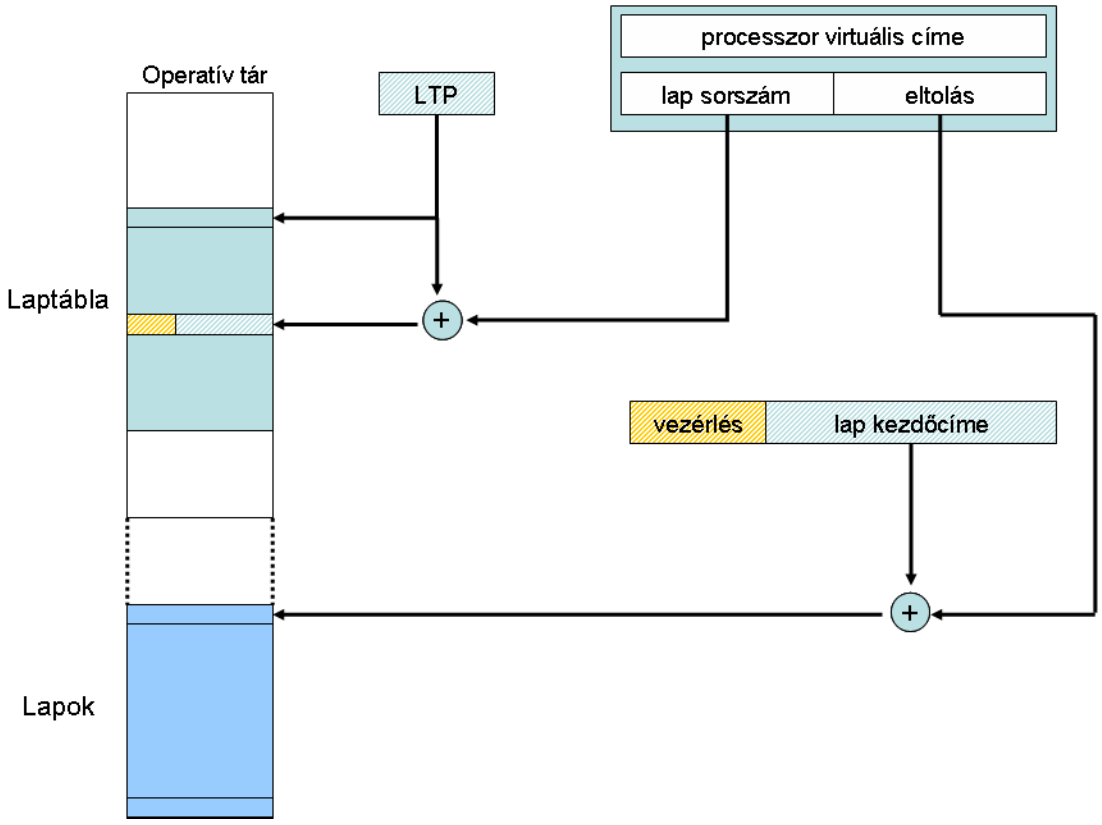
- az indexelt leképzés a processzor által elérhető címtartományt úgy terjeszti ki, hogy a tárolót rögzített kapacitású lapokra bontja. A processzor logikai címét két részre bontjuk: egy index- és egy eltolásmezőre.
- választott indexregiszter címkiterjesztés bitjei egy lap kezdőcímét adják meg az operatív tárolóban (a tároló 2^{n} lapból állhat).
- az eltolásmező (d bit) a kívánt tárolórekesz lapkezdethez viszonyított helyét adja meg (2^{d} rekesz van egy lapban).
- a vezérlésbitek a kiválasztott lapra vonatkozó privilégiumokat tartalmazzák
- operatív tároló kapacitása nagy lehet, a processzor ebből egyidőben csak egy kisebb tartományt (2^{x+d} rekeszt) tud megcímezni.

- előny: egyszerű, rugalmas, logikai/fizikai címátalakítás gyors, feladatokhoz dinamikusan hozzárendelhető, programok a tárolóban szabadon mozoghatnak
- hátrány: egy lapot mindig bent kell tartani, az operációs rendszer közös részeihez hozzá kell férni

Virtuális tárkezelés: processzor által megcímezhető tárterület nagyobb a megvalósíthatónál

- a logikai címtartományt háttértárolón tároljuk állandó méretű lapok vagy változó méretű szegmensek formájában.
- az op. rendszer kapcsolatot teremt az operatív tár és a virtuális címtartomány között
- az MMU (Memory Management Unit) a felhasználó számára átlátszó módon hajtja végre a háttértároló - operatív tároló átviteleket
- címtranszformáció, nyilvántartás, átvitel kezdeményezése

Lapszervezésű virtuális tárkezlés:



- az egy felhasználóhoz tartozó teljes tárolóterületet a háttértárolón tároljuk. A tárolóterületet egyforma hosszúságú lapokra osztjuk fel
- létrehozunk egy laptáblázatot, melyet az operatív tárolóban helyezünk el
- a processzor egy virtuális címet ad a tárkezelő egységek számára, mely két részből áll: a kívánt lapsorszámból és a kívánt rekesz lapon belüli sorszámából (eltolás)
- a laptáblamutató (LTP) az operatív tárolóban levő laptábla elejére mutat, így a lapsorszám a laptáblában lévő tétel sorszámát adja meg
- ezzel az indexelt címzéssel kiolvassuk a kívánt lapra vonatkozó információt a laptáblából
- ez az információ két részből áll: vezérlőbitekből és a lap operatív tárbeli kezdőcíméből
- így a kívánt rekesz kiválasztása a lapkezdőcím és az eltolás egyesítéséből származó fizikai cím alapján történik
- a vezérlőbitek az illető lapra vonatkozó hozzáférési jogokat, az illető lap bent van / nincs bent az operatív tárolóban, ha nincs bent, akkor a háttértárolón hol található, ha bent van, akkor módosítás volt-e, milyen régen hoztuk be információkat tartalmazzák.
- ha a kívánt lap nincs bent az operatív tárolóban, akkor a tárkezelő egység a vezérlőbitek alapján megkeresi a háttértárolóban és behozza az operatív tárolóba, kitöltve egyben a lapkezdőcím és a vezérlőbitek részt
- ha a processzor által kiadott virtuális címnek megfelelő rekesz nincs bent az operatív tárolóban, akkor a processzort le kell állítani, a kívánt lapot vagy szegmenst a háttértárolóból be kell hozni az operatív tárolóba, majd a feldolgozást folytani kell.

- előny: tetszőleges méretű virtuális címtér rendelhető tetszőleges méretű fizikai címtérhez
- hátrány: lassú

Szegmens szervezésű virtuális tárkezelés:

- a felhasználó rendelkezésére álló tárolóterületet logikailag tetszőleges számú, változó hosszúságú szegmensekre osztjuk fel
- a szegmensindextábla tartalmazza a szegmensleírók címeit. A szegmensleíró a kívánt szegmens kezdőcímét, valamint hosszát és védettségi/használati állapotát (vezérlés) tartalmazza
- ha a keresett szegmens nincs az operatív tárolóban, akkor az MMU kezdeményezi betöltését a háttértárolóról, és a betöltést követően módosítja a megfelelő leírókat is
- az új szegmens betöltése előtt egy megfelelő méretű helyet kell számára találni az operatív tárolóban
- ha a processzor által kiadott virtuális címnek megfelelő rekesz nincs bent az operatív tárolóban, akkor a processzort le kell állítani, a kívánt lapot vagy szegmenst a háttértárolóból be kell hozni az operatív tárolóba, majd a feldolgozást folytani kell
- a processzor utasítás-végrehajtásának ilyen felfüggesztésekor a processzor állapotát el kell menteni, hogy az áttöltés után a felfüggesztett utasítás végrehajtható legyen.
- további problémát okoz, ha egy task a jelenlegi szegmenséből kilépő címhez fordul

- előny: jobban illeszkedik a programozói szemlélethez, hatásosabb védelem
- hátrány: az eltérő méretek miatt memória fregmentáció jöhet létre, bonyolultabb algoritmusok

Multitasking elve:

- task-nak nevezzük, azt a tevékenységet, mely a számítógépben elvben párhuzamosan folyhat, ezt általában processznek nevezik
- minden task egy utasításokból és egy kiinduló adatokból álló programot futtat
- a program egy algoritmust leíró szöveg és a task ezen algoritmus egy végrehajtása
- tipikus taskok: file szerkesztés, forrás file fordítás
- ugyanazt a programot több task is futtathatja

Virtuális processzor:

- a rendszernek az egyes taskok számára átlátszónak kell lennie
- minden tasknak (felhasználónak) úgy kell érezni, mintha saját külön processzora lenne
- elméletileg egy task teljesen párhuzamosan fut a többi taskkal
- valóságban az egyes taskok ugyanazon processzoron (erőforrásokon) osztoznak
- az operációs rendszer valamilyen időtartamra a fizikai processzort az egyik kiválasztott task számára rendelkezésre bocsájtja
- az operációs rendszer mindenegyes időpillanatban a fizikai processzort hozzárendeli az egyik virtuális processzorhoz
- kontextus váltás: egy task váltás tehát a felfüggeszteni kívánt task jellemzőit elmenti a taskhoz rendelt speciális adatszerkezetbe és a futtatni kívánt taskhoz rendelt adatszerkezetből feltölti a hardver regisztereket és a megfelelő operáció rendszer táblákat

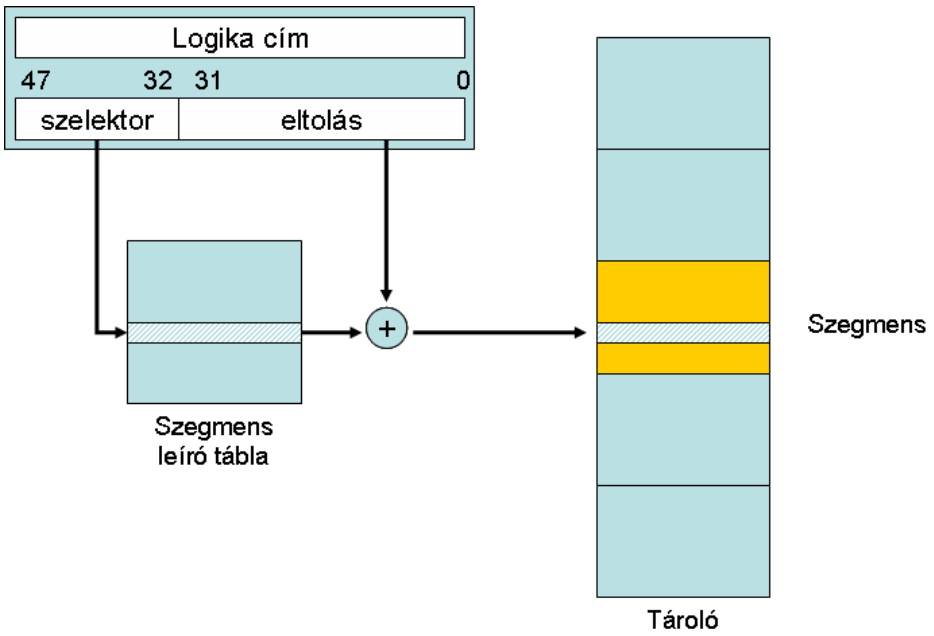
I80386 mikroprocesszor:

- 32 bites mikroprocesszor
- 4 GB fizikai címtartomány
- 64 TB logika címtartomány
- beépített lapszervezésű virtuális tárkezelés

I80386 címzési rendszere:

- felhasználói programok logikai címeket használnak
- tárkezelő egység (MMU) a logikai címeket fizikai címekké alakítja át

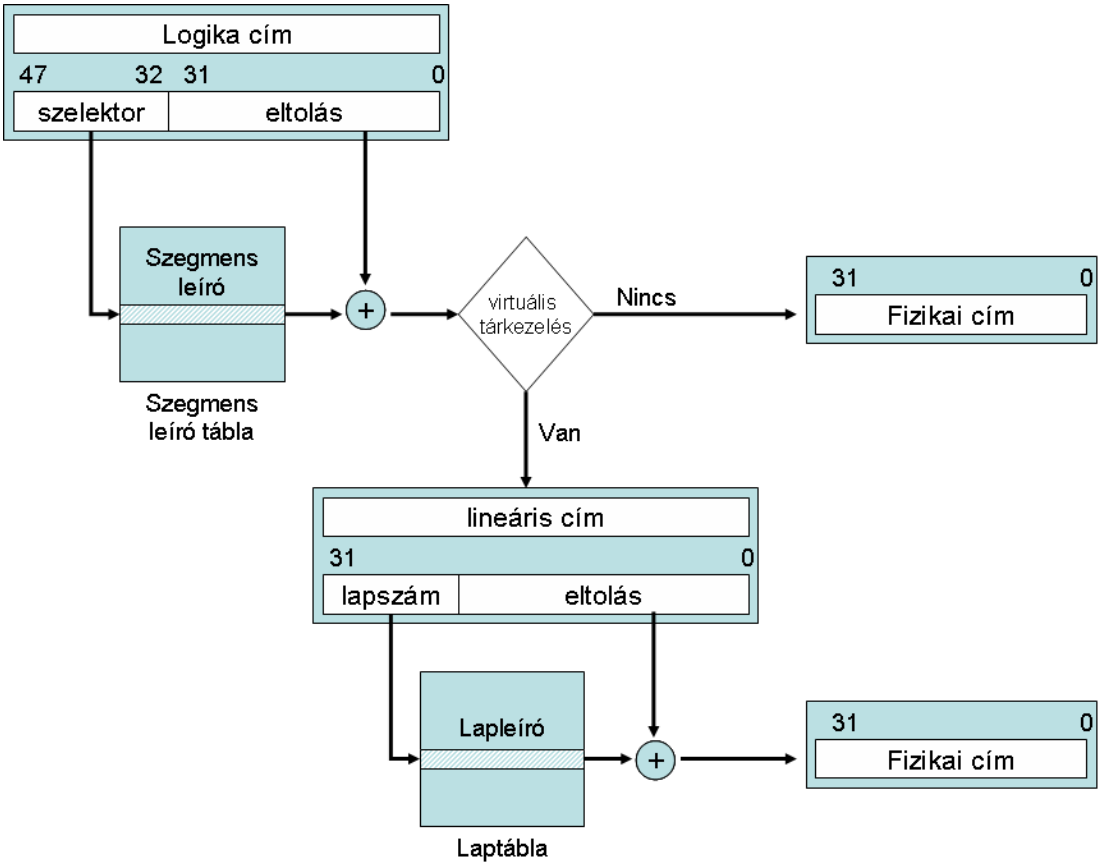
1. *lineáris címtartomány:*
   - maximum 4GB méretű egybefüggő strukturálatlan tárterület
2. *szegmensek:*
   - szegmensek logikai egységet jelentenek, melyek hossza tetszőleges (1 byte – 4GB)
   - a szegmens hosszát a program struktúrának megfelelően választjuk



   - mikroprocesszor két részből álló logikai címet ad ki (szegmens szelektor és szegmensen belüli eltolás)
   - szegmens leíró tartalmát az operációs rendszer kezeli
   - a leíró tartalmazza a szegmens operatív memóriabeli kezdőcímét (32 bit), a szegmens hosszát és a védőbiteket
   - a szegmensleírókat tartalmazó tábla az operatív memóriában helyezkednek el
   - annyi tételt (8 byte hosszú) tartalmaz, ahány szegmenset definiáltunk
   - lassú működés elkerülése érdekében hat belső szegmensregiszterek

- ha a processzor egy logikai címhez kíván fordulni, akkor az utasítás típusa kijelöli milyen típusú szegmensez kell hozzáférni
- regiszterek tartalmához felhasználói programból nem férünk hozzá

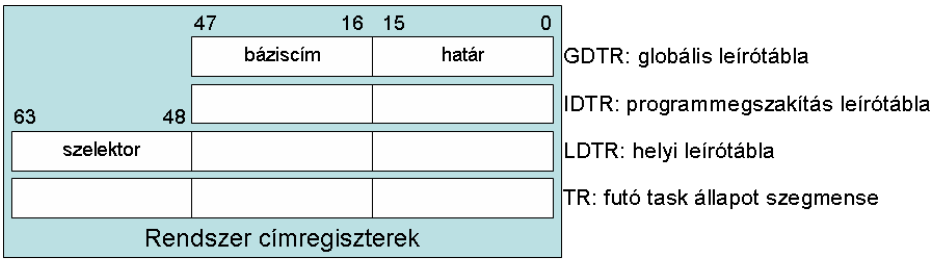
3. *lapok:*
   - egy lap mérete 4 kByte
   - strukturálatlan (lineáris) címzés, szegmentált címzés, strukturáltan virtuális tárkezelés, szegmentált virtuális tárkezelés
   - alkalmazott operációs rendszer határozza meg melyiket kell használni
   - operatív memória által megvalósított fizikai címtartomány és a felhasználó által látott logikai címtartomány
   - a tárkezelő egység (MMU) feladata a programokban szereplő logikai címtartomány leképzése a fizikai címtartományra



- virtuális tárkezelést alkalmazó szegmens címzés esetén a logikai cím egy szegmens számból és a szegmensen belüli eltolásból áll
- ezt az MMU egy lapszámmá és a lapon belüli eltolássá számítja át
- a lapszám alapján a laptáblából kiolvasható a keresett lap kezdőcíme az operatív memóriából vagy a háttértárolón

Rendszerregiszterek:

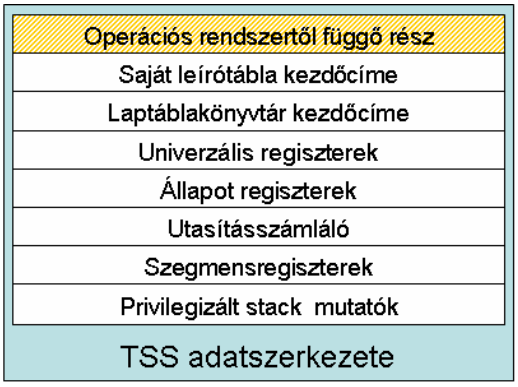
- olyan regiszterek, amelyeket nem a felhasználói program utasításai, hanem az operációs rendszer használ



- ezeket a regisztereket az operációs rendszer inicializálja
- kivételes állapot esetén az operációs rendszer felhasználja ezen regiszterek tartalmát
- a rendszerregisztereket csak privilegizált utasítások kezelhetik

- regiszterek hardvertámogatást nyújtanak egy multitasking operációs rendszer létrehozásához
- lehetnek olyan program- adatrészek, melyeket több task is fel kíván használni, illetve olyanok, melyekhez csak egyetlen task kíván hozzáférni
- minden task rendelkezhet egy globális és egy saját logikai címtartománnyal
- kell egy (globális) GDT egész rendszerre kiterjedő és egy (lokális) LDT leírótábla
- ezekre a GDTR és az LDTR rendszer címregiszterek tartalma mutat rá
- az éppen végrehajtandó utasítás szelektorának egy bitje választ a két leíró között

- task váltáskor a mikroprocesszor hardver úton új értéket tölt be az LDTR rendszer címregiszterbe az új taskhoz tartozó TSS (task állappot szegmens) területéről
- a TSS egy task virtuális processzorának állapotát tartalmazza
- a mikroprocesszor egy külön utasítást (JUMP TSS) is tartalmaz a task váltás támogatására



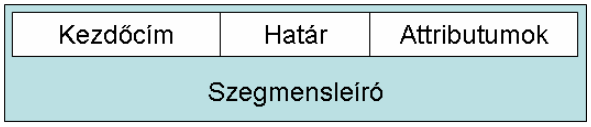
- mindegyik taskhoz tartozik egy TSS, ami két részből áll
- első az operációs rendszertől függő rész, ami lehetővé tezsi, hogy eltérő operációs rendszereket alkalmazhassunk
- a TSS alsó része a regiszter értékek elmentésére és visszaállításához szükséges
- egy új task létrehozásakor az operációs rendszer létrehoz egy TSS-t és annak tételeit beállítja azokra az értékekre, ami a task elindításához szükségesek
- TSS alsó részét hardveres felső részét szoftveres úton kezelik
- a processzor a jelenleg futó task szelektorát, leíróját és határát a TR rendszer címregiszterben tárolja
- egy új task futtatására való áttéréskor az operációs rendszer egy JUMP TSS utasítást ad ki, melynek operandusa az új task szelektora

    1. regiszterek értékeinek elmentése a jelenleg futó taskhoz tartozó TSS megfelelő rekeszeiben
    2. az új task szelektorának beírása a TR regiszterbe
    3. az új TSS címe alapján a processzor feltölti regisztereit az új TSS megfelelő rekeszeinek tartalmával, így az utasítások feldolgozása az új task utasításszámlálója által kijelölt utasítással folytatódik

I80386 védelmi rendszere:

- különböző taskok illetve egy taskon belül a kód adat és stack területek szétválasztása, szegmensekre bontása
- a különböző szegmensekre és lapokra védelmi hierarchia alakítható ki
- különféle hibák hatásának a task belsejére történő korlátozása

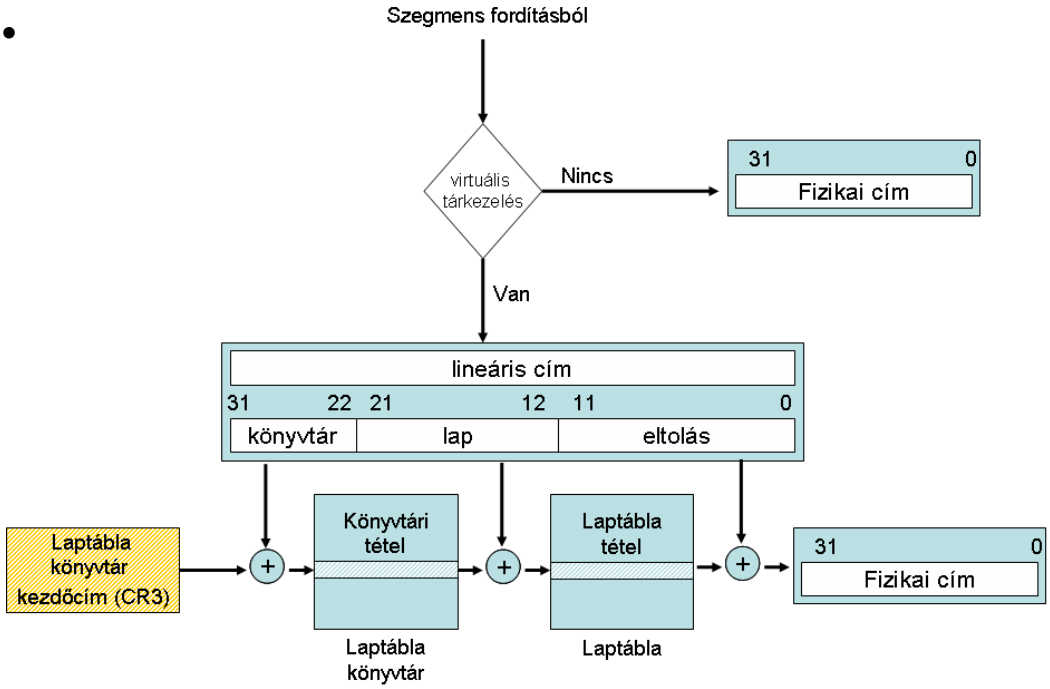
Szegmensek:



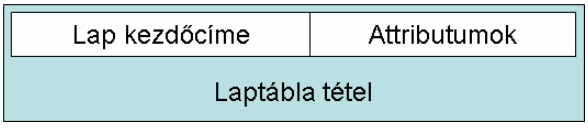
- Kezdőcím: a szegmens kezdőcíme a virtuális címtartományban
- Határ: a szegmens hossza
- Attributomok: vezérlési és védelmi célokat szolgáló mezők
- *ACCESSED*: 1-et tartalmaz, ha a szegmensből információt olvastunk ki, vagy írtunk be. *PRIVILEGE, RIGHTS, TYPE, PRESENT*: 1-et tartalmaz, ha az illető szegmens az operatív memóriában van.

Lapok:

- a processzor lapokat kezel, azaz lapszervezésű virtuális tárolót használ, ha a CR0 rendszer vezérlőregiszter PG bitje 1-be van állítva.
- ezt a bitet csak privilegizált utasítással lehet beállítani
- a lapozásnál kétszintű hierarchiát valósítanak meg
-
-



- mindenegyes task saját laptábla könyvtárral rendelkezik
- a laptábla könyvtár sorolja fel az illető task által hozzáférhető laptáblákat
- a CR3 vezérlőregiszter tartalmazza az éppen futó task laptábla könyvtárának kezdőcímét
- a laptábla könyvtár 1024 könyvtári tételt tartalmazhat (ezek hossza 4 byte)
- maximálisan 1024 laptábla tartozhat egy taskhoz
- minden egyes laptábla 1024 laptábla tételt tartalmazhat (ezek hossza 4 byte)



- attributumok: vezérlési és védelmi célokat szolgáló mezők
- *PRESENT*: az operációs rendszer 1-be állítja, ha a lap az operatív memóriában van, *RIGHTS, PRIVILEGE, ACCESSED*: a processzor hardver 1-be állítja, ha a laphoz hozzáfértünk, *DIRTY*: a processzor hardver 1-be állítja, ha a lapra írtunk, *USER-DEFINED*: a felhasználó tetszése szerint felhasználhatja
- ha logikai/fizikai címfordítás során keletkező lineáris cím egy olyan lapra hivatkozik, mely nincs bent az operatív memóriában (PRESENT=0), akkor a processzor laphibát (PAGE FAULT) észlel.
- ekkor a CR2 rendszer vezérlőregiszterbe betöltődik a hiányzó lap lineáris címe, és meghívódik az operációs rendszer laphiba kezelő eljárása
- az operációs rendszer kiszámítja a hiányzó lapnak megfelelő laptábla tétel helyét,

- hiányzó lap esetén a laptábla tétel a lap mágneslemez tárolón elfoglalt helyét (címét) tartalmazza
- ennek alapján az operációs rendszer betöltheti a mágneslemez tárolóról a megfelelő lapot az operatív memóriába és beállítja a laptábla tételben a lap kezdőcímét, és a PRESENT bitet
- ezután a processzor megismétli azt a műveletet, melynél a laphiba fellépett
- a két többlet memória ciklus miatt a rendszer működési sebessége lecsökken, ezért a processzor egy TLB (fordítást kikerülő) regisztertömböt is tartalmaz
- a TLB32 olyan regiszter, melyben a legutoljára használt 32 lap fordítási információja helyezkedik el
- egy címfordítás során a processzor a keresést egyszerre elindítja a táblákon keresztül és a TLB regisztereiben
- ha a TLB-ben megtalálja a keresett lapra vonatkozó információt, akkor az operatív memóriában lévő táblákban abbahagyja a keresést, ha nem találja meg folytatja a keresést, és miután talált, kicseréli a TLB egyik tételét az új lapéval

Privilégiumok:

- a taskok védelmi szempontból négy privilégium szint valamelyikén lehetnek
- a 0. szint a legfontosabb, a 3. szint a legkevésbé fontos
- a task privilégiumszintje az általa végrehajtott kódszegmens privilégium szintjével egyezik meg
- mindenegyes szegmensleíróban van egy PRIVILEGE mező melynek tartalma az illető szegmens privilégium szintje
- egy 1. privilégium szintű task csak azokat a szegmenseket használhatja, amelyeknek a privilégiumszintje ≥ 1
- ha egy task egy számára meg nem engedett szintű szegmenshez fordulna, ajkkor a hardver hibát jelez
- *0. szint:* operációs rendszer magja (kernel), *1. szint:* operációs rendszer kevésbé fontos részei, *2. szint:* utility programok, *3. szint:* felhasználói programok
- a privilegizált utasításokat csak 0. privilégium szintű taskok hajthatják végre

Szegmensek védelme:

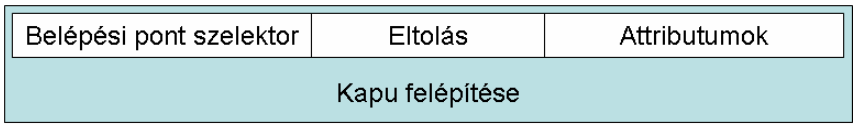
- egy szegmens védettségi módját a szegmensleíró attributum mezejének tartalma határozza meg
- PRIVILEGE: a szegmens privilégium szintjét tartalmazza
- RIGHTS: a szegmensleíró ezen mezejének előírható, hogy milyenfajta műveleteket engedünk meg ezen szegmensen végrehajtani
- TYPE: a szegmens típusa, a felhasználói task kód típusú és adat típusú szegmenseket használ. így a processzor hibát jelezhet, ha a kód típusú szegmensbe írni akarunk

Lapok védelme:

- az I80386 processzornál lap szervezésű virtuális tárkezelést alkalmazhatunk, tehát a szegmenseket is lapokra bonthatjuk
- a lapnak két privilégium szintje lehet: supervisor és user
- a user szint a 3. privilégium szintnek felel meg, míg a supervisor szintű laphoz, csak 0., 1., 2. privilégium szintű taskok férhetnek hozzá
- egy konkrét fizikai című tárlórekeszhez való hozzáféréskor először ellenőrzi a szegmens védelmet, majd a lap védelmet

Rendszerhívások:

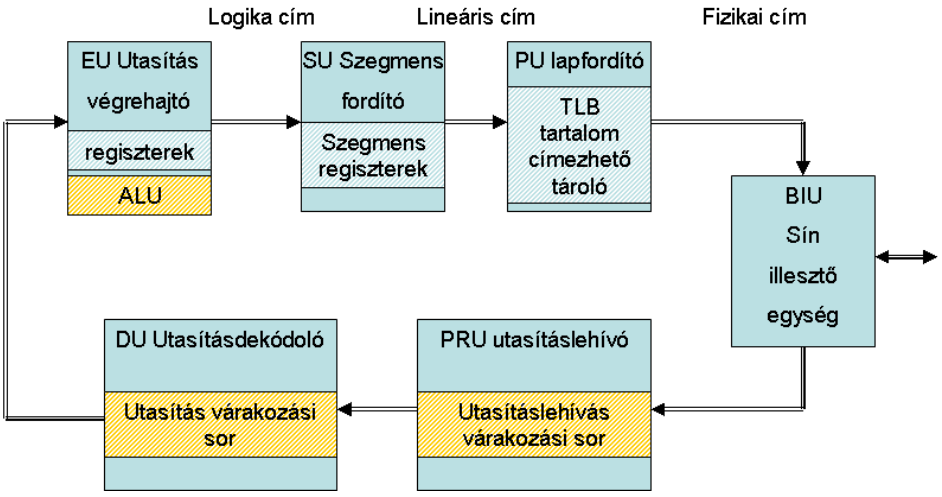
- egy szolgáltatást egy eljárás valósít meg, mely eljárás meghívásával érhetjük el a kívánt operációs rendszer szolgáltatást
- egy gate (kapu) egy szigorúan specifikált belépési pont az operációs rendszerbe, melyen keresztül egy magasabb privilégium szintű kódszegmensből elérhető az operációs rendszer valamilyen szolgáltatása: *CALL GATE, TRAP GATE*
- ezeket a kapukat az operációs rendszernek kell kialakítania és azokat a GTD-ben vagy az LDT-ben elhelyeznie



- a belépési pont szelektor és az eltolás a kívánt belépési pont logikai címét adja meg.
- Attributumok: vezérlési és védelmi célokat szolgáló mezők
- *DWORD COUNT*: az átadandó paraméter hossza (csak CALL GATE), *TYPE, PRIVILEGE:* az a privilégiumszint, mellyel egyenlő vagy kisebb privilégium szintű kódszegmensből át lehet jutni a kapun, *PRESENT*
- egy task a CALL GATE-en egy közönséges CALL utasítás végrehajtásával, a TRAP GATE-en INTERRUPT utasítás végrehajtásával juthat keresztül

Az I80386 hardver struktúrája:

- egy utasítás feldolgozását hat részre bontja fel, azaz hat funkcionális egysége működik átlapoltan
- processzor szempontjából külső sínről általában nem lehet periodikusan beolvasni az egymást követő utasításokat
- PREFETCH QUEUE: az utasításokat a BIU egy várakozási sorba (FIFO) tesz be és az utasításokat a pipeline következő fokozata (DU) onnan veszi ki a további feldolgozáshoz.
- az utasítások végrehajtásának ideje nem egyforma
- INSTRUCTION QUEUE: a dekódolt és végrehajtásra váró utasításokat az EU egy várakozási sorból veszi ki, ha az előző utasítás végrehajtásával végzett.



I/O kezelés, perifériák:

- a perifériák létesítenek kapcsolatot a számítógép és a külvilág között
- ember-gép kapcsolat megvalósítása
- adattárolók-számítógép kapcsolata
- számítógépek közötti kapcsolat megvalósítása
- sebesség, vezérlési lehetőségek: szinkron, aszinkron működés, karakteres, blokkos átvitel, vezérlési felületek egységesedése

Eszközszintű kezelés:

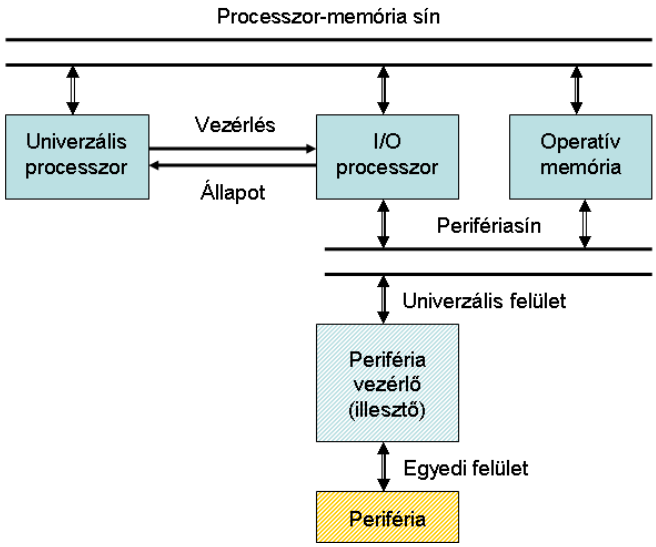
- a perifériális eszköz fizikai sajátosságainak megfelelő illesztési felületet vagy utasításkészletet biztosítunk
- feltétel nélküli bevitel/kivitel: periféria mindig adatátvitelre kész állapotban van, nincs szinkronizáció a processzor és a periféria között, pl.:kapcsolóállás beolvasás, világító kijelzők működtetése
- feltételes bevitel/kivitel: periféria nincs adatátvitelre kész állapotban, a perifériák és a processzor működését szinkronizálni kell
- jelzőbitre alapozott feltételes bevitel/kivitel: átvitel során a processzor és a periféria működésének szinkronizálásáért a processzor felelős, a processzor mindig adatátvitelre kész állapotban van, a periféria állapotát jelzőbit segítségével közli a processzorral
- jelzőbit valamilyen időközönkénti vizsgálata: programozott lekezelés
- periféria kész jelzés egy programmegszakítás kérést jelent a processzor számára: megszakításos kérés
- szemaforra alapozott bevitel/kivitel: változtatható sebességű perifériák esetén, a processzor és a periféria kölcsönösen szinkronizálják egymást (handshaking)

Logikai kezelés:

- a számítógépek beviteli/kiviteli rendszerében általánosított beviteli/kiviteli eljárásokat és illesztési felületeket biztosítanak
- processzor és perifériák nagy sebességkülönbsége miatt nem használnak közvetlen processzor irányítást
- háromféle utasítástípus: *vezérlő*, *perifériállapot lekérdező*, *információátviteli*
- a beviteli/kiviteli hardver részleteit az operációs rendszer takarja el
- a felhasználó programok függetlenné tehetők az alkalmazott periféria típusától

Perifériakezelés

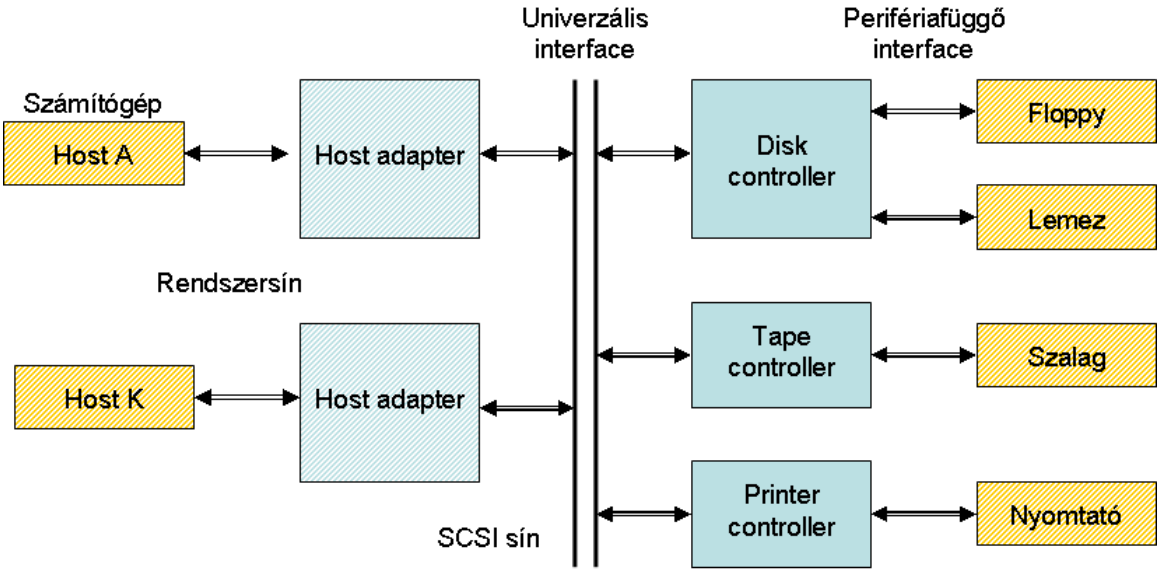
- a csatornára és I/O processzorra alapozott perifériakezelést rögzített feladatú modulok hajtják végre, így felszabadítva a processzort
- statikus feladat-hozzárendelésű elosztott feldolgozást valósítanak meg
- I/O processzor egy olyan hardver eszköz, melynek utasításkészletét a beviteli/kiviteli műveletekre optimalizálták
- processzor felé egy szabványos illesztési felületet biztosít
- a végrehajtandó programot az univerzális processzor helyezi el az operatív memóriában és az I/O processzor onnan hajtja végre
- a perifériasínre az adatokat az operatív memóriából veszi ki, illetve a perifériasínről behozott adatokat az operatív memóriába teszi be
- a csatornák a beviteli/kiviteli műveletek autonóm irányítására szolgálnak
- a csatorna egyszerűsített I/O processzor



- Szelektor csatorna: a csatornaprogram végrehajtásának egész időtartama alatt a csatornát a kiválasztott perifériális eszköz lefoglalja, nagy adatátviteli sebességű perifériák
- Multiplexer csatorna: a csatornát megosztva több perifériális eszköz használhatja és a csatorna az egyes eszközök byte-onként átvitt adataiból a memória felé adatblokkokat állít elő

SCSI sín:

- a sínt egy vagy több (max. 8) számítógép vezérelheti prioritásos alapon
- max. 8 perifériavezérlő egység kapcsolható rá és bármelyik számítógép max. 8 perifériát érhet el mindenegyes perifériavezérlőn keresztül
- 8 bites adatátvitelt támogat szinkron vagy aszinkron módon
- az SCSI sín soros hozzáférés vezérlésű
- közvetlen adatátvitel két periféria között
- átlapolt periféria működés

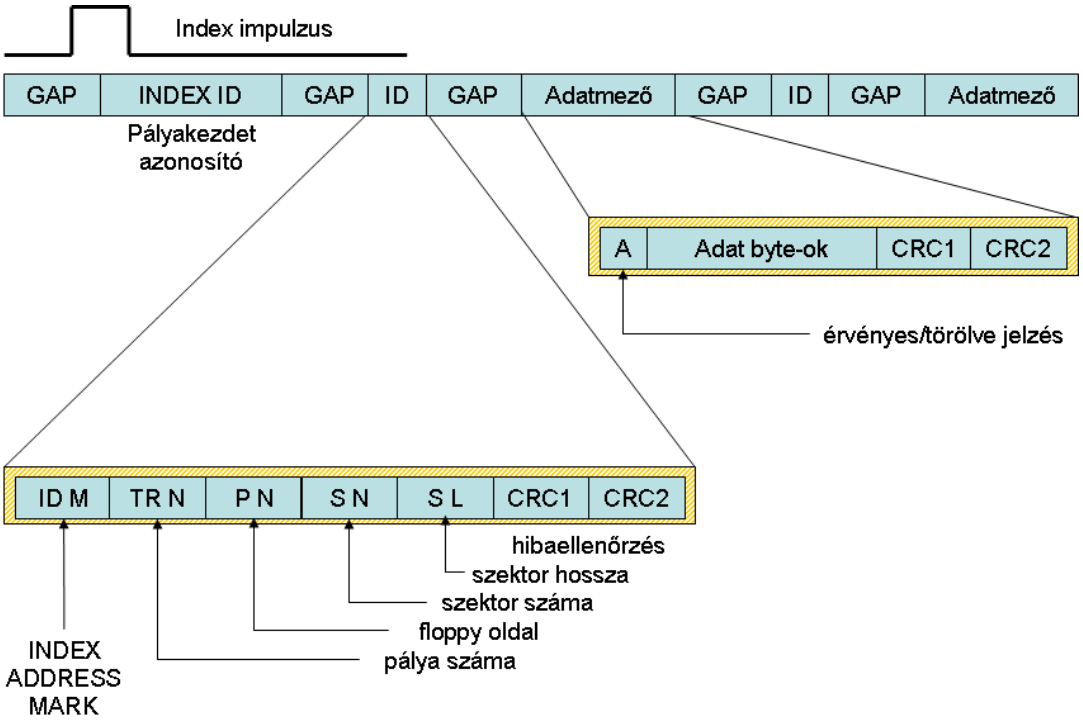


Háttértárolók:

- nagykapacitású, nagy hozzáférési idejű tárolók alakíthatók ki q mágneses tárolási elv segítségével
- egy mágneslemez tárolón a mágneses réteget egy köralakú hordozóra visszük fel
- ez lehet hajlékony (floppy) vagy merev (hard disk)
- a lemezen koncentrikus kör alakú pályák (track) helyezkednek el és a pályán belül szektorokat alakítunk ki az adatok blokkos tárolására
- a lemezoldal felett általában egy író/olvasó fej van, melyet mechanikus mozgatással kell a kívánt pálya fölé beállítani
- az olvasófej a mágnesezésváltást érzékeli
- a kiolvasott jel nem ideális, alakja a mágneses jellemzőktől, a geometriai méretektől és a sebességtől függ
- nagy írássűrűségnél az állapolások miatt a kiolvasott jel helye és nagysága is változik
- az adatok rögzítésére a háttértároló típusától, kapacításától és a gyártási technológiától függő kódolást alkalmaznak

Szektorszervezés elve:

- a lemez kezdetét egy index lyuk jelzi
- minden pálya több szektorra van osztva, melyek kezdetét lyuk jelzi (hard sector), vagy mágnesesen felírt információ (soft sector)
- a szektorok között üres helyeket (GAP) hagynak a toleranciák miatt, ide csupa 0-át írnak a szinkronizálás érdekében
- a floppyt használatbavétel előtt formattálni kell, melynek során az azonosítókat a vezérlőegység felírja a lemezre
- ha egyszer megtaláltuk a keresett szektor elejét, akkor az adatok nagy sebességgel érkeznek, így ha több szektornyi adatot kíván feldolgozni, akkor a soronkövetkező szektor már esetleg túl hamar érkezne: *sector interleave*



Multiprocesszoros rendszerek:

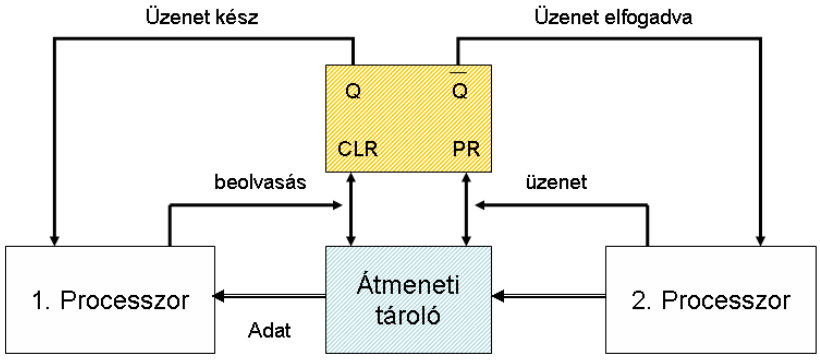
- olyan struktúra, amelyben ugyanazon rendszeralgoritmus egymástól független feladatainak konkurens végrehajtása folyik
- két esemény konkurens, ha egyik sem tudja kauzálisan befolyásolni a másikat
- osztályozhatjuk a feladat-hozzárendelés módja és a processzorközi kapcsolat jellege szerint
- bizonyos funkciókat (szolgáltatásokat) kell biztosítani a felhasználó számára, és minél több felhasználó kiszolgálását lehetővé kell tenni
- a rendszer egyes processzorainak együtt kell működnie, ezt a kapcsolatot mind hardveresen, mind szoftveresen létre kell hozni

Feladat-hozzárendelés módjai:

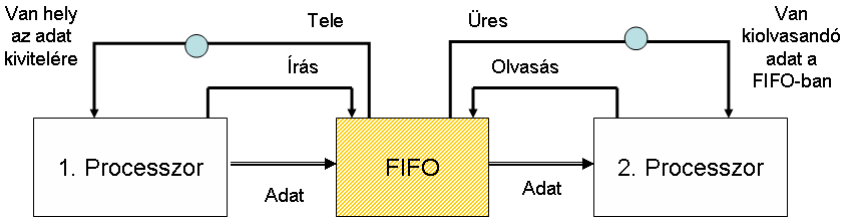
1. Dinamikus feladat-hozzárendelés: egy processzor a rendszerkapacitás egy részére minden funkciót(szolgáltatást) elláthat, azaz a pillanatnyi terhelési viszonyok alapján egy feladat bármelyik processzornak odaadható
   - előny: egy processzor meghibásodása a rendszer egy részére terjed ki, processzorok teljesítőképessége jól kihasználható
   - hátrány: a processzor teljesítőképessége korlátozza a megvalósítható funkciókat (szolgáltatásokat), bonyolult szoftver
2. Statikus feladat-hozzárendelés: egy processzor egy meghatározott funkciót (szolgáltatást) valósít meg az egész rendszer számára
   - előny: egyszerű szoftver, processzor felépítése a funkcióhoz optimalizálható
   - hátrány: processzorok terhelése jelentősen eltérhet, érzékenyebb a meghibásodásokra

Processzorközi kapcsolat módjai:

1. Lazán csatolt rendszer: a processzorközi kapcsolat üzenetorientált, az egyes processzorokon különálló operációs rendszerek vannak
   - a processzorokat kommunikációs alrendszerekből megvalósított csatornák kötik össze
   - logikai, fizikai közeg függ a rendszer nagyságától, térbeli elhelyezkedésétől
   - az üzenet átvitelének idejére a forrás processzor a fogadót perifériájának tekinti
   - kis rendszer esetén, kevés információ átvitele esetén használatos a *szemaforra alapozott pont-pont összeköttetés*
   - előny: egyszerű,
   - hátrány: lassú, merev szerkezet, operációs rendszerek együtt működése nehézkes

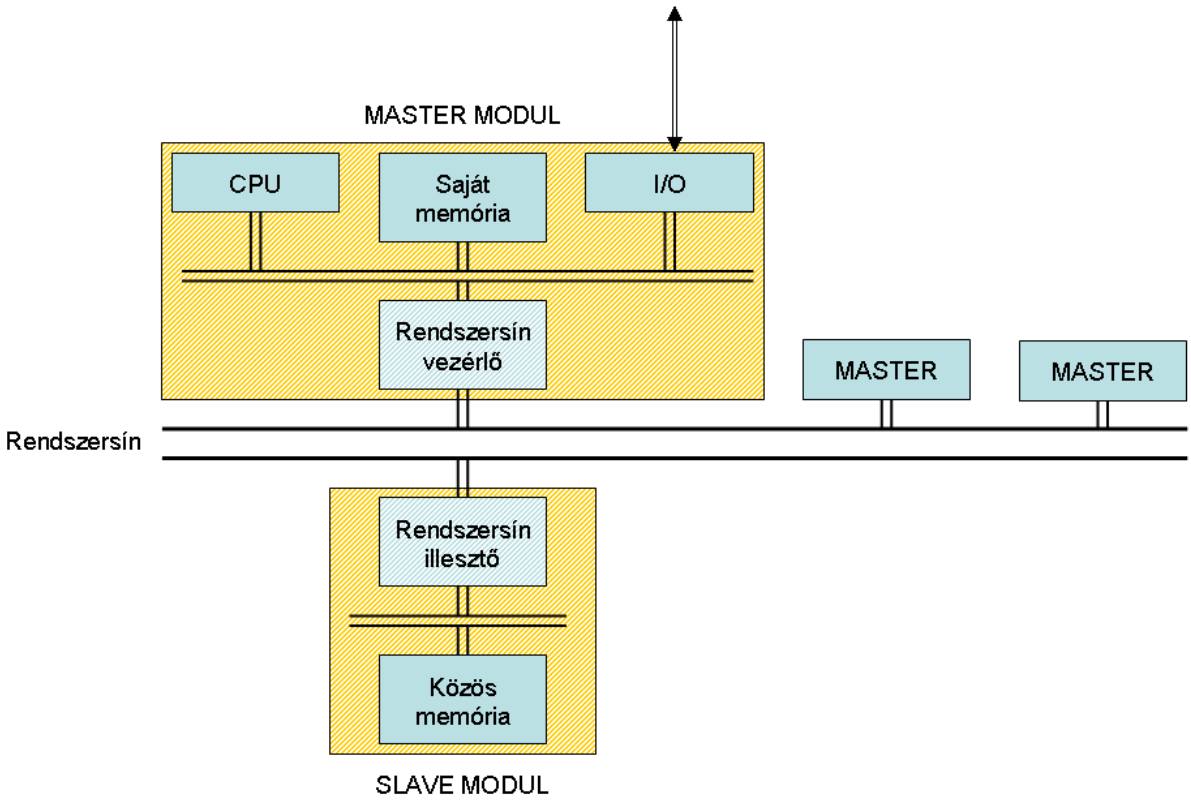


- előfordul, hogy a forrás és a nyelő processzorok közötti adatátvitel sebessége ingadozik a közösen használt szemafor miatt
- mindkét oldalt külön szemaforral látjuk el és az átmeneti tároló helyére egy FIFO tárolót teszünk



2. Szorosan csatolt rendszer: a processzorközi kapcsolat közös erőforráson keresztül van megvalósítva, közös az operációs rendszer
   - közös erőforráson keresztüli kapcsolat
   - legrugalmasabb megoldás a *crossbar kapcsolás rendszer*
   - kapcsoló hálózat megfelelő vezérlés esetén bármelyik processzort bármelyik memóriával összeköthető
   - hátrány: nagy rendszereknél nem lehet, bonyolult, nehéz a kapcsolók vezérlése

   - nagy rendszereknél *rendszersínre alapozott struktúrát* alkalmaznak

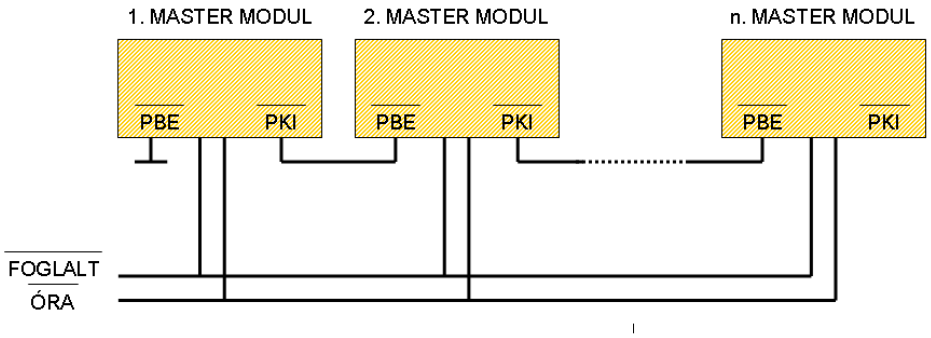


   - a processzorközi kommunikációnak két szintje van
   - *felső szinten*: a folyamatok közötti adat- és vezérléskommunikáció
   - *alsó szinten*: a master moduloknak a rendszersínhez való hozzáférése

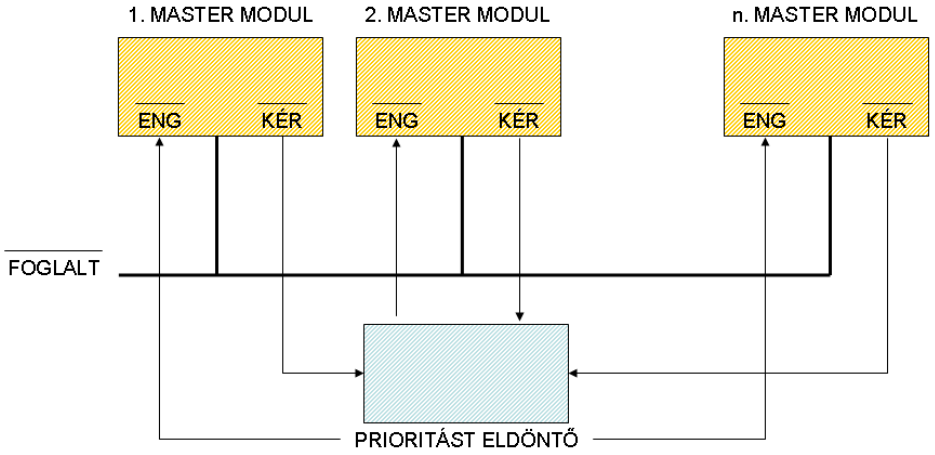
- a master modul processzora utasításában szereplő memória vagy perifériacím alapján dönti el a rendszersínvezérlő, hogy saját memóriához vagy perifériához, illetve közös memóriához vagy perifériához kell fordulni
- a rendszersínhez való hozzáférés két módon történhet

Rendszersínhez való hozzáférés:

1. Soros hozzáférés vezérlés:
   - a legjobboldalibb master modulnak meg kel várnia a hozzáférési igények végighaladását a láncon
   - a rendszersín foglaltságát a FOGLALT jelzi, a rendszersínt éppen használó master modul alacsony szintre húzza le
   - a master modulok hozzáférési igényeiket egymás számára a PBE és PKI jelek közlik
   - ha az i. master modul PBE bemenetére alacsony szint kerül, akkor ez azt jelenti, hogy jelenleg egyetlen tőle balra elhelyezkedő master modul sem kíván a rendszersínhez hozzáférni
   -



   - előny: egyszerű
   - hátrány: master modulok száma a terjedési késleltetések miatt korlátozott, nehézkes a hibakezelés
2. Párhuzamos hozzáférés vezérlés:
   - a kérést a master modul csak akkor adhatja ki, ha a rendszersín éppen szabad
   - a prioritást eldöntő a legmagasabb prioritású kérést kiadó master modul ENG bemenetére ad engedélyező jelet, így az a FOGLALT lehúzásával magához ragadhatja a rendszersín vezérlését
   - a prioritás lehet előre rögzített vagy körbenforgó

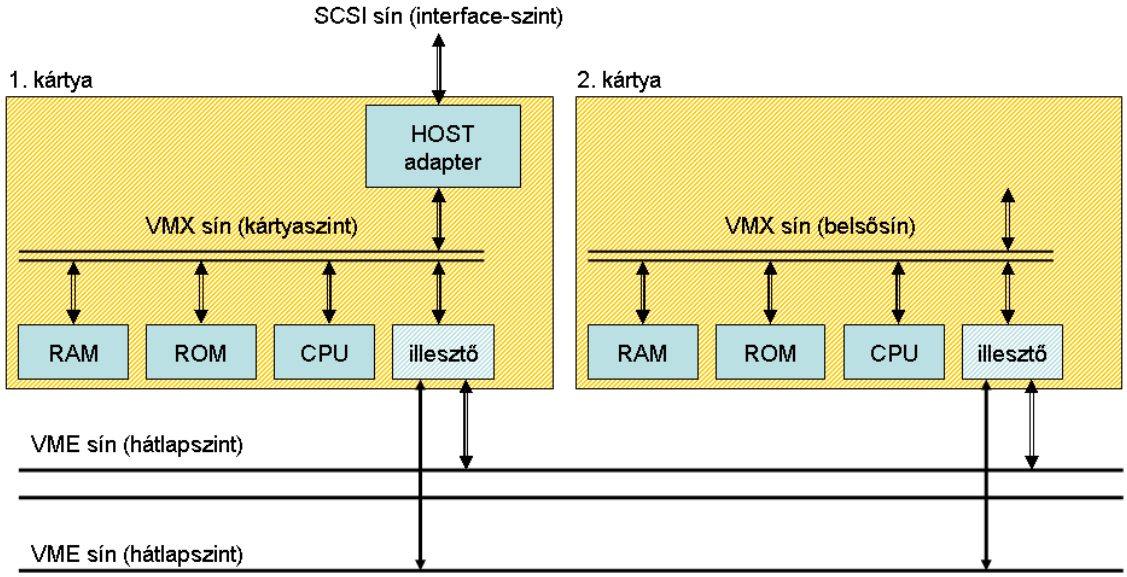


Postafiók elv:

- adat- és vezérléskommunikáció normál formája
- küldő master modul a rendszersínen keresztül adatokat visz át a közös slave tároló feladathoz rendelt területére
- a fogadó master modul a rendszersínen keresztül adatokat vesz ki a közös slave tároló feladathoz rendelt területéről
- az adatok betöltését és kivételét megfelelő szemaforok vizsgálatával és beállításával szervezzük
- amikor az i. master modul a postafiókhoz fordul, akkor ezalatt ki kell zárni a k. master modult ugyanennek az erőforrásnak a használatából
- előny: a fogadó modul szempontjából érdektelen ki rakta be azokat a postafiókba, adatot küldő modul szempontjából is érdektelen, hogy melyik modul fogja az adatokat fogadni, a többi modul a rendszersín művelet alatt is dolgozhat a saját erőforrásaival
- hátrány: rendszersín szűk keresztmetszetté válhat

Sínrendszerek:

- vezetékek egy csoportja, melyek használatán digitális rendszerelemek osztoznak, és amelyeken keresztül a rendszer elemei egymással kommunikálnak
- egy rendszerben többféle sínt használnak hierarchikus rendben



- *kártyaszint*: egy vagy több nyomtatott áramköri kártyán elhelyezett elemet köt össze az összeköttetések számának csökkentésével
- *hátlapszint:* kártyákból felépített modulok között kommunikációs utat biztosít
- *interface*-szint: biztosítják a közös kommunikációs utat az I/O eszközök és a rendszer között

Rendszersínek:

1. Adatátvitel:
   - sínre csatlakozó modulok között információcsere lebonyolítása a sín felügyeletet ellátó modul irányítása alatt
2. Programmegszakítás:
   - a sínre csatlakozó modulok kérhetik valamelyik modul normál működésének megszakítását
3. Arbitrálás:
   - olyan rendszerekben, ahol több modul is képes at adatátvitel vezérlésére biztosítani kell egy algoritmust, mellyel a versengő igények közül egy adott időben csak egy nyeri el a sín vezérlési jogát
4. Szolgáltatások:
   - tápfeszültség ellátás, hibajelzés, rendszer-órajel, reset

Rendszersínre csatlakozó modulok:

1. Rendszervezérlő:
   - a sín vezérlési jogát biztosító jeleket, a rendszer-órajelet, az inicializáló jelet és a vészjelzéseket adja a sínre
2. Master:
   - képes magához ragadni a sín vezérlési jogát a rendszervezérlő segítségével, azaz adatátvitelt kezdeményezni a sín cím-és vezérlőjeleinek segítségével
3. Slave:
   - nem képes magához ragadni a sín vezérlési jogát, ha a sínen megjelenő cím-és vezérlőjelek kijelölik, akkor reagál a master által elindított adatátviteli folyamatra
4. Megszakításkérő:
   - képes megszakításkérő jel kiadásával kiszolgálást kérni valamelyik master modultól
5. Megszakításkezelő:
   - képes a megszakításkérések észlelésére és a kiszolgálási folyamat kezdeményezésére

Sínműveletek:

1. kezdeményezés (sínkérés)
2. arbitráció (melyik master modul használhatja a sínt)
3. címzés (kivel kíván a nyertes master modul kapcsolatba lépni)
4. adatátvitel
5. hibaészlelés és hibajelzés

Sín címzési módok:

- nyertes master modul kétféleképpen bonyolít le kommunikációt
  1. egy vagy több slave modul kiválasztása
  2. minden slave modul kiválasztása
- a master modul két részből álló címet ad ki a sín címvezetékeire
  1. slave kártya címe (felső címbitek)
  2. slave modulon belüli rekesz címe (alsó címbitek)
- a slave kártyát kétféleképpen lehet megcímezni
  1. logika cím
  2. helyzeti cím
- legtöbb adatátvitelnél egyetlen slave modult választunk ki
- van olyan este, hogy az átvitelben több, vagy az összes slave modul részt vesz
- broadcast: a master modul valamennyi slave modulra kiírást végez
- broadcall: a master modul valamennyi slave modulról beolvasást végez

Logikai címzés:

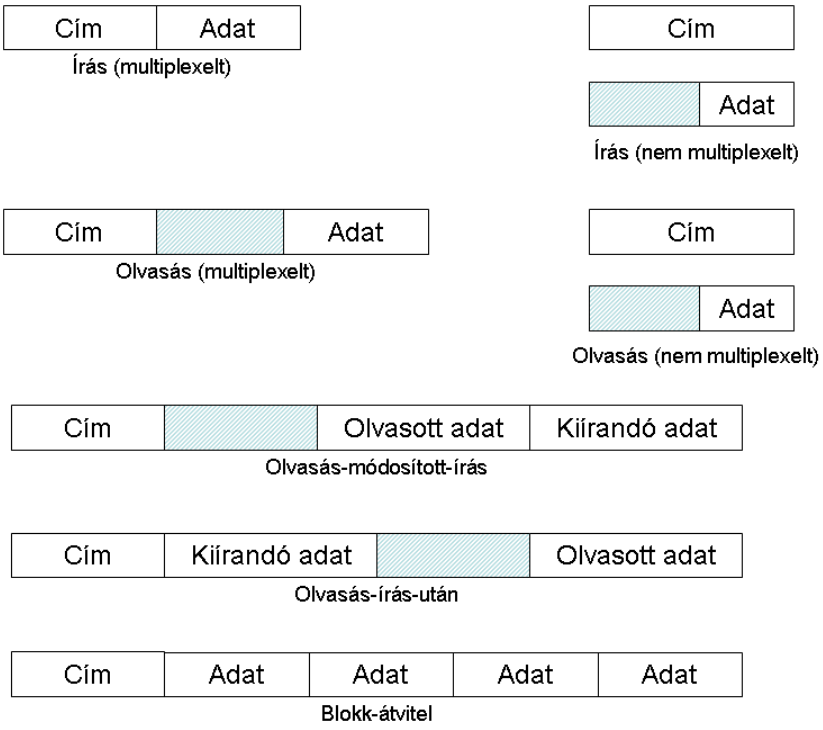
- helytől független címzés
- minden egyes kártyának egy egyedi azonosítója (logikai címe) van, melyet a kártyán elhelyezett kapcsolósorral vagy EPROM-mal lehet beállítani
- a kártyát a rendszer bármely helyére be lehet tenni
- pl.: VME sín

Helyzeti címzés:

- a kártyát fizikai helye, azaz melyik helyre dugaszoltuk be a hátlapon azonosítja
- ez az azonosító a hátlapra van behuzalozva
- kártyán elhelyezhető egy címregiszter és az operációs rendszer a rendszer inicializáláskor a helyzeti címzés felhasználásával a kiválasztott kártyának átküldhet egy tetszőleges értéket ebbe a címregiszterbe és ezután ezt az értéket használhatjuk a kártya logikai címeként
- pl.: MULTIBUS II

Sín átviteli üzemmódok:

1. áramkörkapcsolt üzemmód:
   - átvitel teljes lebonyolítása alatt a forrás-nyelő kapcsolat fennmarad
2. üzenetkapcsolt üzemmód:
   - master kéri az átvitelt a megfelelő cím és üzenethossz kiadásával
   - amikor a slave képes a kért átvitelt végrehajtani, akkor master-ként lépve fel kezdeményezi a tényleges adatátvitelt, az eredeti master most slave modulként viselkedik

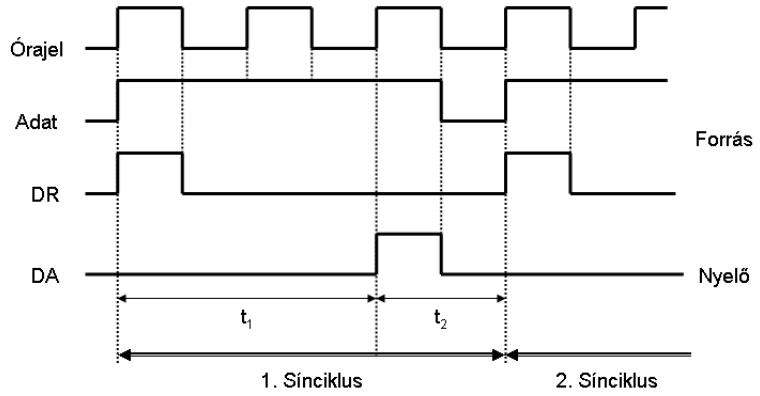


Sín időzítési protokoll:

Forrás	***Nyelő***
1. információ ráadása a sínvezetékekre	
2. a vett információ stabilizálódik	
3. Jelzésadás , hogy az információ stabil	
	4. felismeri, hogy az információ stabil,
	5. információ elvétele
	6. az információ elvételének nyugtázása
7. felismeri, hogy az információt elvették	
8. információ levétele a sínvezetékekről	
9. jelzésadás, hogy az információ eltávolította	
	10. felismeri, az információ levették a sínről
	11. az átvitel befejeződésének nyugtázása
12. új tevékenység elkezdése	

Szinkron átvitel:

- valamennyi elemi tevékenység előre meghatározott időpillanatokban történik

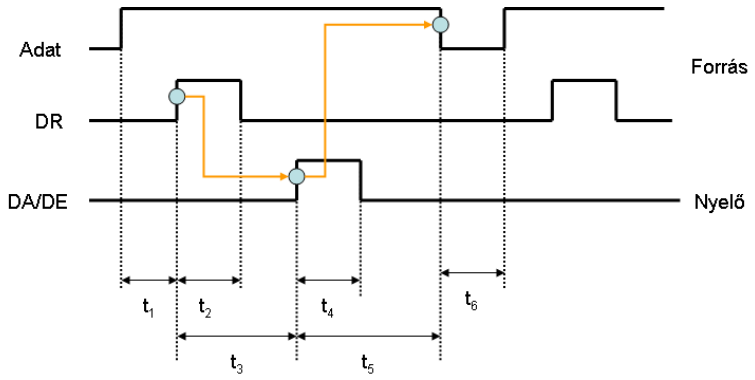


- az adatátvitel egy közös órajellel vezéreljük, ezt általában a rendszervezérlő modul állítja elő a sínre kapcsolt valamennyi modul számára
- az adatot a forrás meghatározott időpillanatban adja rá az adatvezetékekre a DR jel vezérlésével
- a nyelő az adatot a DA jel vezérlésével fogadja el
- előny: gyors
- hátrány: nincs ellenőrzési lehetőség, nincs visszajelzés

Aszinkron átvitel:

- az elemi tevékenységek tetszőleges időpillanatokban történhetnek
- az egyes események fellépése nem meghatározott időpillanathoz kötött, így fellépésükről a partnert tájékoztatni kell

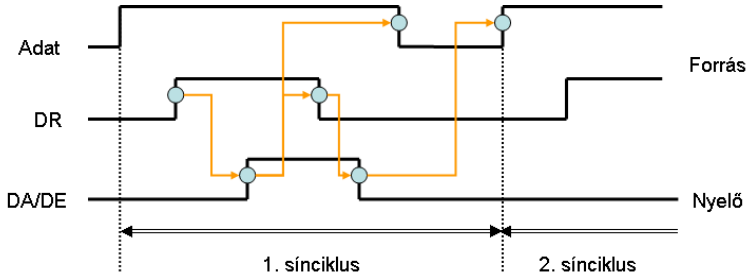
1. <u>Reteszeletlen protokoll</u>:



- a forrás ráteszi az adatot a sín adatvezetékeire, majd t_1 idő múlva a DR jellel jelzi a nyelő felé, hogy az adatot rátette a sínre
- a DR jel időtartama rögzített
- a nyelő a DR jel vételekor kiolvassa az adatot a sínről és t_3 idő múlva jelzi ezt a forrás felé a DA jel kiadásával
- miután a forrás t_5 késleltetéssel megkapta ezt a jelzést leveszi az adatot a sínről és a következő tevékenységet a sín stabilizálódása után, t_6 elteltével indítja
- a nyelő paritáshiba esetén a DE jel visszaküldésével értesítheti a forrást
- a forrás és a nyelő sebessége tetszőleges lehet, mivel a t_3 időtartam kizárólag a nyelőtől a t_5 pedig kizárólag a forrástól függ

2. Félig-reteszelt protokoll:
   - a t_2 és t_4 időtartamok rögzítettek (nem függ a nyelőtől és a forrástól)
   - gyors forrás esetén (kis t_5) elindulhat a következő sínciklus, még DA/DE befejeződése előtt
   - a forrás DR jelét a nyelő DA/DE jelének megjelenése szünteti meg

3. Reteszelt protokoll:



   - minden vezérlőjel változás az együttműködő partner vezérlőjel változásának eredménye
   - DA felfutó éle jelzi, hogy a nyelő elfogadja az adatot, a DR felfutó élének megérkezése után
   - DR lefutú éle jelzi, hogy a forrás leveszi az adatot a sínről a következő sínciklus előkészítése érdekében
   - DA lefutó éle jelzi, hogy a nyelő befejezte a sínről az adat levételét és az új sínciklus számára rendelkezésre áll
   - előny: együttműködő modulok sebessége tetszőleges lehet, az átviteli sebesség igazodik a pillanatnyilag aktív modulok sebességéhez
   - hátrány: minden eseménynek végig kell haladnia a sínen, hibákra érzékeny

Részben szinkron átvitel:

- az elemi tevékenységek órajelekhez rögzített időpillanatokban történhetnek
- asz aszinkron átvitel zajérzékenységét csökkenti
- a vezérlőjelek minden átmenete csak egy órajel idejében történhet
- átmenetek közötti idő, változó számú órajel periódusnyi lehet
- a modulok sebességéhez való alkalmazkodás durva

Statikus busz arbitráció:

- előre rögzített módon az egyes master modulok számára időszeleteket jelölünk ki, melyben jogosultak a rendszersín használatára
- valamennyi master egy előre meghatározott módon adja át a vezérlést
- előny: egyszerű hardver, garantált busz áteresztő képesség mindegyik master számára
- hátrány: túl merev, sín sebessége az egyes masterek sebességének összege kell hogy legyen, kicsi a rendszersín kihasználása, mivel nem minden master fog minden sínciklusban hozzáférést igényelni

Dinamikus sín arbitráció:

- a rendszersín vezérlésének jogát az igények felmerülése szerint osztjuk ki

Busz megszerzési stratégiák:

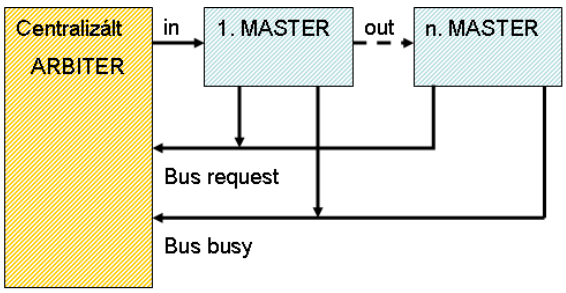
1. Prioritásos:
   - az egyes master modulok között fontossági sorrendet állapítunk meg és a rendszersín használati jogát az igénylők közül a legnagyobb prioritású master kapja meg
2. Igazságos:
   - az egyes modulok fontossága azonos
   - egy igénylő biztosan megkapja a rendszersín használatának jogát mielőtt az éppen azt használók közül bármelyik másodszor is hozzáférhetne
3. Kombinált:
   - a két előző módszer kombinációja
   - ezt használja a VME sín

Busz elengedési stratégiák:

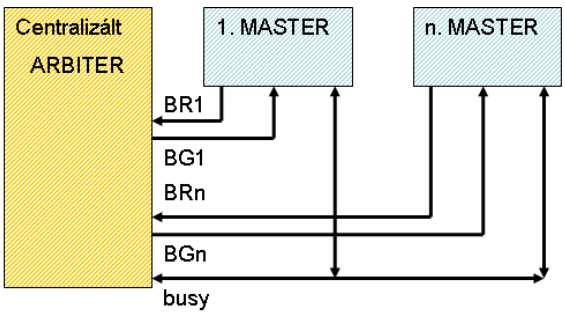
1. Elengedés kérésre:
   - a rendszersínt az éppen használó modul magánál tartja a rendszersín használati jogát a tényleges használat után is mindaddig, amíg valamelyik másik master modul nem kéri a rendszersín használati jogát
2. Elengedés a sínhasználat befejeztekor:
   - a rendszersín minden használathoz külön kérni kell
3. Preemptív elengedés:
   - egy nagyobb prioritású master modul kérése esetén a rendszersínt éppen használó master modul lemond a rendszersín használatáról

Busz arbitáció hardver mechanizmusai:

1. Centralizált: arbiter hardver egy helyen, a masterek kérik a buszvezérlés jogát, az arbiter dönt és nyugtáz/visszajelez a jog a odaítéléséről

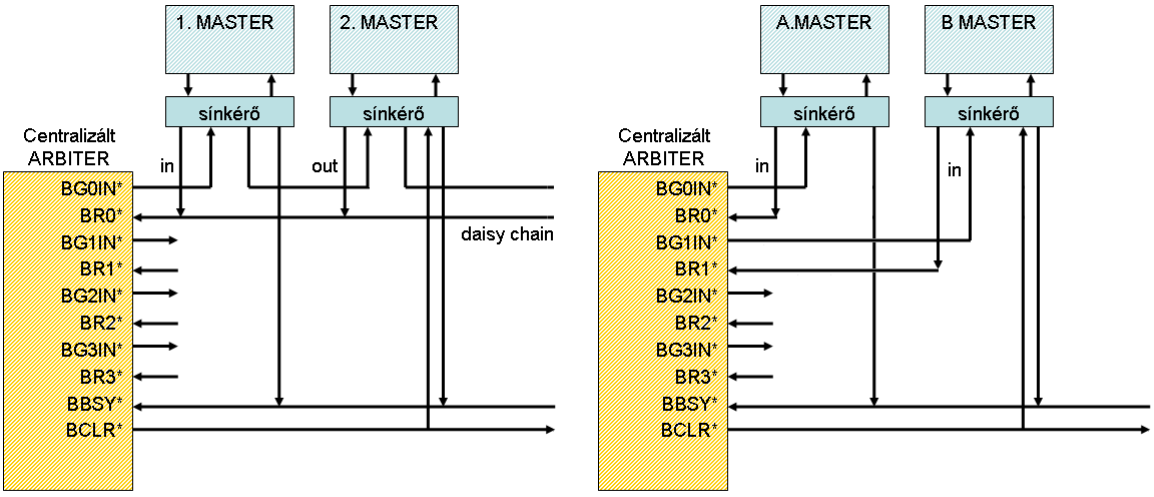


Közös kérés - felfűzött válasz



Független kérés - független válasz

Kombinált mechanicmuz:



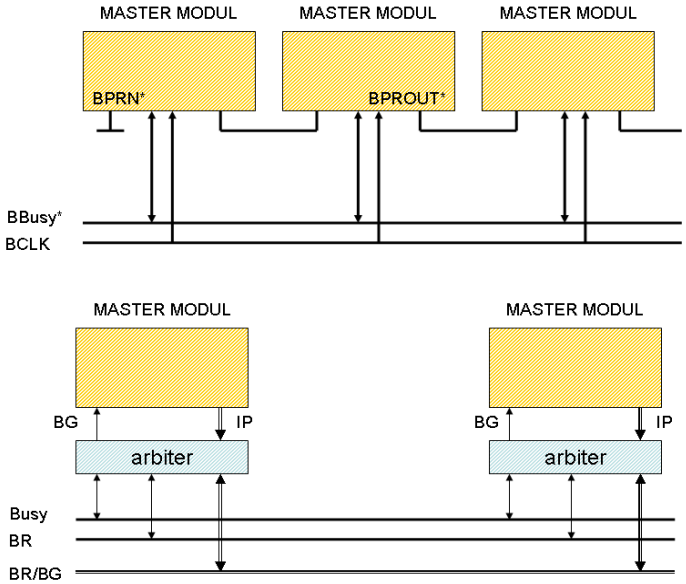
Két egyforma fontossági szintű kérés

Két különböző fontossági szintű kérés

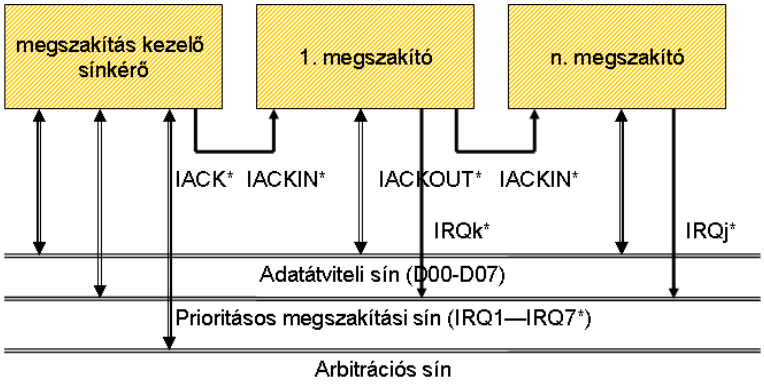
1. *Két különböző fontossági szintű sínkérés*:
   - az A és B masterek sínhozzáférés igényeiket a hozzájuk tartozó A és B sínkérőkkel közlik
   - a két sínkérő a BR0* és a BR1* sínkérő vonalakra csatlakozik
   - két sínkérő igényét a BR0* és BR1* lehúzásával jelzi
   - arbiter észlelve a két sínkérést a fontosabb kérő felé a BG1IN* kiadásával jelzi a kérés elfogadását ha a sín szabad (BBSY*=1)
   - amikor a B sínkérő modul észleli ezt az engedélyező jelet, akkor lehúzza a BBSY* vezetéket, jelezve a sín használatának megkezdését
   - a BR1* vezeték elengedésével jelzi az arbiternek, hogy nincs újabb kérése
   - a sínkérő jelzi a B master felé, hogy használhatja a rendszersínt
   - arbiter észlelve a sín lefoglalását megszünteti a BG1IN* jelet
   - ha a B master befejezte a rendszersín használatát, akkor sínkérő modulja elengedi a BBSY* vezetéket
   - az arbiter a BBSY* elengedését észlelve új arbitrációt indít el

2. *Két egyforma fontossági szintű kérés*:
   - a két master igényét az 1. és a 2. sínkérőn keresztül a BR0* lehúzásával közli az arbiterrel
   - az arbiter ezt a jelet érzékelve a BG0IN* lehúzásával kezdi el az arbitrálást, ha a sín szabad (BBSY*=1)
   - az 1. sínkérő megkapja ezt a jelet, de nem adja tovább, hanem a BBSY* lehúzásával jelzi az arbiternek a sín használatának elkezdését, egyben megszünteti kérését is
   - BR0* lehúzva marad, hiszen a 2. master kérését még nem szolgáltuk ki
   - az arbiter megszünteti a BG0IN* jelet
   - ha az 1. master modul befejezte a sín használatát, akkor azt az 1. sínkérőn keresztül a BBSY* elengedésével jelzi
   - a BBSY*=1 –et észlelve az arbiter egy új arbitrálási ciklust kezd a fennálló igények alapján , az arbiter ismét kiad egy BG0IN* jelet
   - mivel az 1.master modul már nem kívánja a rendszersínt használni, az 1.sínkérő továbbadja a (BG0OUT* jelként) a 2. sínkérőnek

2. Decentralizált:



Programmegszakítás kezelése:



- prioritásos alsín hét dedikált programmegszakítást kérő vezetéket tartalmaz
- maximum hét programmegszakítás kezelő modul lehet, melyek mindegyike egy vagy több megszakítást kérő vezetéket szolgálhat ki
- a hét megszakítást kérő vezeték mindegyikére egy vagy több megszakító modul kapcsolódhat, a megszakítás kéréseket nyugtázó jelet soros láncolással kezeljük
- ha a megszakítás kezelő egy megszakítási kérést kap, akkor először megpróbálja megszerezni a rendszersín használati jogát az arbitrációs sínen keresztül
- ha megkapta a rendszersín használati jogát, akkor a programmegszakítás kérést az IACK* jel kiküldésével nyugtázza
- ez a jel a lánc első megszakítást kérő moduljának IACKIN* bemenetére jut
- ha egy lehetséges megszakító nem kért programmegszakítást, akkor IACKOUT* kimenetén továbbadja az IACK* jelet a következő megszakítónak, így a programmegszakítást kérő modul megkapja a megszakítási kérést nyugtázó jelet
- a megszakítás kezelő modul a nyugtázó ciklusban ráteszi az adatátviteli alsínre az elfogadott programmegszakítási szintet is
- a megszakítást kérő modul a megszakítási kérést nyugtázó jel vételekor (prioritási szintje megegyezik az adatátviteli alsínen vett elfogadott prioritási szinttel) ráteszi az adatátviteli alsín alsó 8 bitjére a saját azonosítóját, és annak érvényességét a DTACK*jel kiadásával jelzi

Operációs rendszerek témakör

Operációs rendszer feladata:

- a számítógépen állandóan futó program, az a program ami közvetlenül vezérli a gép működését
- végrehajtási környezet, olyan környezet, ahol a felhasználók és programjaik hasznos munkát végezhetnek
- erőforrás kiosztó, kezeli a rendszer erőforrásait (CPU, központi tár, perifériák), biztosítja ezek igénylőkhöz rendelését, hatékony, biztonságos és igazságos felhasználásukat
- vezérli a felhasználó programok működését, meggátolja a számítógép hibás felhasználását, a felhasználói programok helyett vezérli a géphez kapcsolt perifériák működését

Operációs rendszerek hatékonysági mértékei:

- CPU kihasználás [%]
- Áteresztőképesség [job/óra]

Operációs rendszerek fejlődése:

1. *Nincs operációs rendszer*: CPU kihasználás: 7%, áteresztőképesség 4 job/óra
   - nagy, drága hardver, egyszerű (papírt használó) perifériák
   - hibakeresés ideje kiszámíthatatlan, kevés a lefoglalt idő, vagy a gép kihasználatlanul állt
   - OPEN SHOP: gépidőfoglalás, off-line adatrögzítés, programozó egyben operátor is, kézi vezérlés, online debug
   - CLOSED SHOP: kiképzett, betanított operátor kezelje a gépet, a programozó beadja az adathordozót (forráskód, feldolgozandó adatok), valamint egy írásos operátori utasítást az elvégzendő feladatokról, az operátor futtat és eredményeket ad vissza, Rendellenesség esetén az operátor kinyomtatja a memóriaképet és a regiszterképet, a programozó ennek alapján off-line debugol
   - Kötegelt feldolgozás (batch): tapasztalati alapon az operátorok csoportosították a munkákat illetve egyes fázisait, lyukkártya-csomagokból a kötegek alakultak ki, kötegek végrehajtása közben a program befejeződésének észlelése és újraindítása az operátor feladata, operátor későn veszi észre, hogy a futó program befejeződött, vagy hiba miatt leállt, reakcióideje, kézi beavatkozásai lassúak

2. *Egyszerű monitor*: CPU kihasználás: 55%, áteresztőképesség: 33 job/óra
   - a gép vezérlését egy állandóan a memóriában lévő programra, a monitorra bízzák, az operátor csak a perifériákat kezeli.
   - a munkához mellékelt vezérlő információkat a monitor értelmezi és hajtja végre, elindítva a megfelelő programrészletet
   - vezérlő információk vezérlő kártya formájában jelennek meg
   - egy tevékenység befejeződése után ismét a monitor kapja meg a vezérlést, amely beolvassa a következő vezérlő kártyát
   - a vezérlő programot mindig a memóriában kellett tartani, és meg kellett védeni az esetleges hibás felhasználói programoktól (memóriavédelem)
   - meg kell tiltani, hogy a felhasználói program maga olvashassa az adatfolyamot, mert hiba esetén túlolvashat saját adatain, és elronthatja a következő job végrehajtását (I/O védelem).

3. *Offline-műveletek:* CPU kihasználás: 90%, áteresztőképesség: 55 job/óra
   - perifériák sebességének a processzorhoz viszonyított lassúsága
   - a bemeneti adatfolyamot olvassa a CPU gyors mágnesszalagról, a kimeneti adatfolyamat pedig írja mágnesszalagra
   - egyszerű másoló segédprocesszorok, amelyek egyike a papír perifériáról mágnesszalagra másolja az adatfolyamot, a másik pedig mágnesszalagról nyomtatóra
   - miután a rendszer feltöltődött a három gép párhuzamosan működhet
   - a bemeneti szatelit másolja a következő (n+1). adagot szalagra, miközben a CPU feldolgozza az aktuális n. adagot, és eközben a kimeneti szatelit nyomtatja az előző (n-1). adagot
   - ha valamelyik szatelit szűk keresztmetszetté válik, megkettőzhető
   - egységes perifériakezelési felület kialakítása: periféria független programozás
   - pufferelés megjelenése: feldolgozás és a perifériás műveletek egymáshoz viszonyított sebesség ingadozásának kiegyenlítése

4. SPOOLING rendszer
   - mágnesszalag helyett a diszk egy-egy területe szolgál az adatfolyam gyors elérhetőségű, FIFO elvű átmeneti tárolására
   - a lemez bemeneti adatsorát egy megszakításvezérelt kártyaolvasó program tölti lyukkártyáról, a kimeneti adatsort pedig egy megszakításvezérelt nyomtató program üríti.
   - megszakításvezérelt programok a szatelit processzorok megfelelői
   - működésük során csak annyi időre kötik le a CPU-t, amennyi a perifériák pufferregisztere és a memória közötti elektonikus sebességű adatátvitelhez szükséges
   - különböző munkák perifériás és feldolgozási műveletei lapolódhatnak át
   - diszk véletlen hozzáférése lehetővé teszi a munkák futtatási sorrendjének a pillanatnyi teljesítményviszonyokhoz igazítását.

5. Multiprogramozás:
   - a rendszer nyilvántartja és tárolja a futtatandó munkákat
   - a kiválasztott munka addig fut, amíg várakozni nem kényszerül
   - az operációs rendszer feljegyzi a munkához a várakozás okát, kiválaszt egy másik, futni képes munkát és elindítja
   - ha a félbehagyott munka várakozási feltétele teljesült, az operációs rendszer a következő alkalomkor, amikor a futó folyamat várakozni kényszerül, elindítja

   - az átkapcsolásokhoz több programot kell egyszerre a tárban tartani: ***tárgazdálkodás***
   - egy időben több futásra kész program lehet: ***CPU ütemezés***
   - a gépi erőforrások felhasználását koordinálni kell: ***erőforrás allokáció, holtpont kezelés***
   - programok ne zavarják egymás, illetve az operációs rendszer működését: ***védelmi mechanizmusok***

Időosztásos rendszerek:

- közvetlen, interaktív kommunikációt biztosítanak a felhasználó és programja, valamint az operációs rendszer között
- minden felhasználónak külön beviteli eszköze (terminál, konzol) van, melyen keresztül on-line módon adhat parancsokat és kaphat a rendszertől válaszokat
- CPU egyszerre több párhuzamosan dolgozó felhasználó között meg van osztva számítógéphez csatlakozó hálózati rendszert szerettek volna létrehozni, melyben mindenki számára lehetővé válik (otthonról) a nagy, drága, közös erőforrásokhoz való hozzáférés, vagyis az egy hatalmas közös számítógéphez (MULTICS)

Valósidejű rendszerek:

- érzékelők segítségével észlelt, a környezetben (külvilágban) történt változásokra adott időn belül válaszolniuk kell, vagyis a rendszernek garantálnia kell valamilyen előírt időkorláton belüli válaszidőt
- kemény valósidejű (hard real-time) rendszerek: kritikus munkák befejeződnek időben
- lágy valósidejű (soft real-time) rendszerek: kritikus munkák prioritással futnak

Operációs rendszer és környezete:

- operációs rendszer fő környezeti kapcsolatai
  1. a kezelők
  2. az alkalmazói programok
  3. a számítógéphardver
- operációs rendszernek működése során ezek felé kell csatlakozási felületet nyújtania
- kényelmesen használható virtuális gép megvalósítása a kezel#k és az alkalmazások felé
- a számítógép-hardver hatékony és biztonságos működtetése
- hardver részleteit el kell fedni, a felhasználók számára áttekinthető modelleket kell kínálni, és azokat meg kell valósítani a fizikai rendszeren
- az alkalmazások futtatását úgy kell koordinálni, hogy a fizikai eszközök kihasználtsága minél jobb legyen, és a felhasználó által előírt prioritások teljesüljenek

Kezelői (operátori) felület:

- a kezelői felület ember-gép kapcsolat
- arra szolgál, hogy az operációs rendszer ezen keresztül működtethető legyen, illetve működéséről a felhasználó tájékoztatást kapjon
- egyszerű felhasználók: programokat (alkalmazásokat) futtatnak, amelyek segítségével szokásos napi feladataikat látják el. Az egyszerű felhasználók jelentős része elsősorban az alkalmazásokat és azok kezelői felületét használja, az operációs rendszerrel közvetlenül alig kerül kapcsolatba. Az operációs rendszer olyan gép, amelyik egy felhasználói körnek lehetőséget ad adat- és programfájlok védett és rendezett tárolására, valamint alkalmazások futtatására.
- alkalmazásfejlesztők: részletes ismeretekkel rendelkeznek az operációs rendszer alkalmazói programok számára nyújtott szolgáltatásairól és bizonyos mélységig ismerik az operációs rendszer belső működését. Ezeket az ismereteiket a programkészítésben használják fel. Az operációs rendszer olyan gép, amelyik a programok számára meghívható eljárásokat biztosít.
- rendszermenedzserek: feladata az operációs rendszer üzemeltetése, valamennyi ezzel kapcsolatos probléma megoldása, részletes és alapos ismeretekkel rendelkezik. Rendszert a rendelkezésre álló hardverhez és az ellátandó feladatokhoz illesztett

kiépítésben kell telepíteni. Rendszer felhasználóinak, a rendszerhez kapcsolódó alkalmazásoknak a nyilvántartását, jogosultságaik kiosztását, a rendszer üzemeltetési szabályainak, biztonsági előírásainak meghatározását, azok betartásának felügyeletét. Hangolási feladatokat, ami a hardver lehetőségeit, a tipikus alkalmazásokat és a rendszer statisztikáit figyelembe véve azoknak a rendszerparaméterek beállítását jelenti, amelyek az üzemeltetés hatékonyságát befolyásolják. Az operációs rendszer olyan gép, amelyik a hardver adott célú hatékony alkalmazását segíti, és amelynek a hardverhez és a feladatokhoz illeszkedő, megfelelő telepítése, behangolása, használatának adminisztrációs és általános üzemeltetési feladatai rá hárulnak.

Alkalmazási (programozói) felület

- A futtatható program tehát nem zárt abban a tekintetben, hogy saját maga tartalmazza valamennyi, a futása során végrehajtódó gépi utasítását, hanem rendszerhívó utasításokat is tartalmaz, amelyek hatására az operációs rendszer lép működésbe, az operációs rendszer részét képez# program kezd futni
- Maga a rendszerhívás a legtöbb operációs rendszer és a legtöbb processzor esetén egy programozott megszakítás el#idézésével történik meg.
- Az alkalmazások számára az operációs rendszer egy olyan gép, amelyik kiterjeszti a processzor utasításkészletét.
- A programozó számára a programozási nyelv magasabb szintű műveleteket enged meg, mint a processzorok gépi nyelve.
- Az operációs rendszert tehát a nyelvi szint elfedi a programozó elől.
- Az alkalmazási felület nyelvi szint/ megragadása abból a szempontból is előnyös, hogy a felület ezzel processzorfüggetlenné válik.

Hardver felület:

- Az operációs rendszer maga is program, ami az adott számítógéprendszeren fut, tehát a processzor és az architektúra lehetőségei azok, amiket az operációs rendszer felhasználhat.
- Az operációs rendszer kezeli a hardver eszközöket és egyben gazdálkodási feladatokat is ellát. Az alkalmazások számára tárat, processzorhasználatot, lemezterületet biztosít, végrehajtja az alkalmazások által kezdeményezett be/kiviteli műveleteket.

Folyamat:

- A folyamat (process) tehát a multiprogramozott operációs rendszerek alapfogalma.
- Folyamaton általában műveletek meghatározott sorrendben történő végrehajtását értjük.
- A folyamat tehát elkezdődik és befejeződik, közben pedig minden részművelet végrehajtása csak akkor kezdődhet el, ha az előző részművelet végrehajtása már befejeződött. A folyamat tehát önmagában szekvenciális.
- Az operációs rendszerek körében tehát a folyamat egy program végrehajtása.
- A folyamat egy végrehajtás alatt álló program.

- Egy program egy végrehajtása egy utasítássorozat végrehajtását jelenti.
- Egy végrehajtás jellemezhető egy vezérlési szállal, amelyik az irányított gráfon a program belépési pontjától egy befejeződési pontig vezető útnak felel meg.

Operációs rendszer logikai modellje:

- Minden folyamathoz tartozik egy logikai processzor és egy logikai memória. A memória tárolja a programkódot, a konstansokat és a változókat, a programot a processzor hajtja végre.
- Egy utasítás végrehajtását oszthatatlannak tekintjük, azaz a folyamat állapotát csak olyan időpontokban vizsgáljuk, amikor egy utasítás már befejeződött, a következő pedig még nem kezdődött meg.
- A programvégrehajtás egy vezérlési szál mentén, szekvenciálisan történik, alapvetően az utasítások elhelyezkedésének sorrendjében.

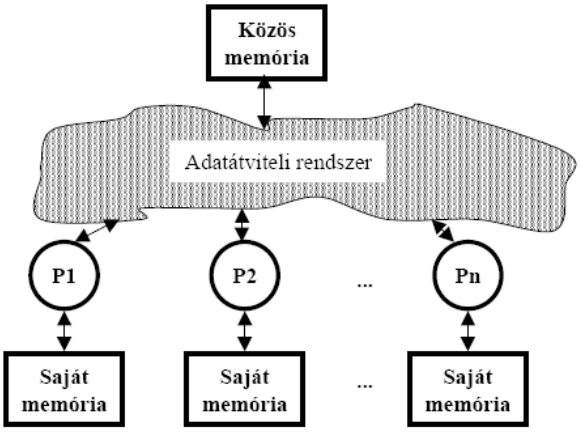
- A memória a RAM-modell szerint működik, azaz:
  - tárolórekeszekből áll
  - egy dimenzióban, rekeszenként címezhető
  - csak írás és olvasás műveletekkel érhető el
  - az írás a teljes rekesztartalmat felülírja az előző tartalomtól független új értékkel
  - az olvasás nem változtatja meg a rekesz tartalmát, tehát tetszőleges számú, egymást követő olvasás az olvasásokat megelőzően utoljára beírt értéket adja vissza

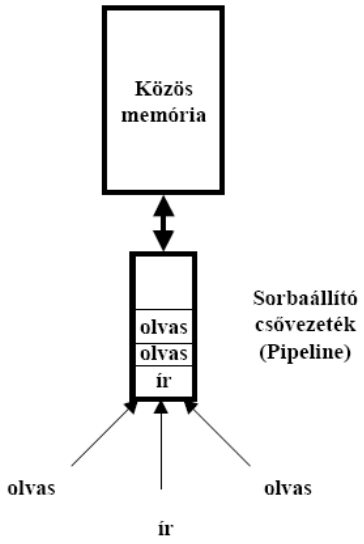
- A folyamatot egy adott pillanatban leíró információk: a memória tartalma, azaz a végrehajtandó programkód, valamint a változók pillanatnyi értéke valamint a végrehajtó processzor állapota (programszámláló állása, további regiszterek, jelzőbitek értéke).
- Az operációs rendszer feladata, hogy a fizikai eszközökön (fizikai processzor, fizikai memória) egymástól elkülönítetten (védetten) létrehozza és működtesse a folyamatoknak megfelelő logikai processzorokat és memóriákat.
- A folyamat logikai modellje mind multiprogramozott, mind multiprocesszoros rendszerek esetén használható.
- A multiprogramozott rendszerek jellemzője, hogy egyetlen fizikai processzoron, valamint a hozzá tartozó memórián és perifériákon kell egyidejűleg több folyamatot végrehajtani, azaz több logikai processzort és memóriát működtetni.
- Multiprocesszoros rendszerek esetén az egyes folyamatoknak megfelel_ logikai processzorokat több fizikai processzorra lehet szétosztani.

Független, versengő és együttműködő folyamatok:

- Az operációs rendszer saját, belső folyamatait rendszerfolyamatoknak, a többit felhasználói folyamatoknak nevezzük.
- független folyamatok: egymás működését semmilyen módon nem befolyásolják
- versengő folyamatok: nem ismerik egymást, de közös erőforrásokon kell osztozniuk
- együttműködő folyamatok: ismerik egymást, együtt dolgoznak egy feladat megoldásán, információt cserélnek

Folyamatok együttműködése közös memórián:





- Közös memórián keresztül történő adatcsere esetén az együttműködő folyamatok mindegyike saját címtartományában lát egy közös memóriát.
- A folyamatok párhuzamos futása miatt a közös memóriát egyidejűleg több folyamat is írhatja, illetve olvashatja.
- Közös memóriát a RAM modell kiterjesztésével kapott PRAM (Pipelined Random Access Memory) modell szerint kell kialakítani.

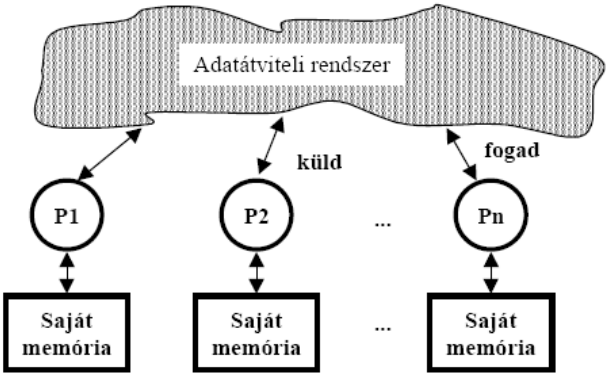
PRAM modell:

- több processzor írhatja és olvashatja egyidejűleg.
- az olvas és ír műveletek egyidejű végrehajtására

- olvasás – olvasás: ütközésekor mindkét olvasás ugyanazt az eredményt adja, és ez megegyezik a rekesz tartalmával
- olvasás – írás: ütközésekor a rekesz tartalma felülíródik a beírni szándékozott adattal, az olvasás eredménye vagy a rekesz régi, vagy az új tartalma lesz, más érték nem lehet
- írás – írás: ütközésekor valamelyik művelet hatása érvényesül, a két beírni szándékozott érték valamelyike írja felül a rekesz tartalmát, harmadik érték nem alakulhat ki.

- egyidejű műveletek nem interferálhatnak, azaz nem lehet közöttük zavaró kölcsönhatás.
- hatásuk olyan lesz, mintha valamilyen előre nem meghatározható sorrendben hajtódnának végre (ezt tükrözi a pipelined elnevezés, arra utalva, hogy a memóriához egy sorosítást végző csővezetéken jutnak el a parancsok)
- ezek a műveletek a modell szintjén oszthatatlanok

Együttműködés üzenetváltással:



- üzenetváltásos adatcsere esetén a folyamatoknak nincs közös memóriája
- az adatátviteli rendszer most a logikai processzorokat kapcsolja össze
- rajta keresztül a folyamatok üzeneteket tudnak küldeni, illetve fogadni
- az üzenetküldésre, illetve fogadásra a folyamatok logikai processzorainak utasításkészletében megfelelő utasítások állnak rendelkezésre, legyenek ezek a Küld (Send) és a Fogad (Receive) műveletek

- Küld (<cím>,<folyamat>) művelet végrehajtásakor a műveletet végrehajtó folyamat elküldi a saját memóriájának megadott címén tárolt adatot a megadott folyamatnak
- Fogad (<cím>,<folyamat>) művelet végrehajtásakor a megadott folyamattól érkező üzenetet a saját memória megadott címén tárolja

Folyamatok szinkronizációja:

1. Kölcsönös kizárás:
   - több folyamatban lehetnek olyan utasítássorozatok (kritikus szakaszok), amelyek egyidejű (konkurens) végrehajtása nem megengedett
   - kölcsönös kizárás szükséges jellegzetesen a megosztottan használt erőforrásokon végzett oszthatatlan műveletsorozatok esetén (nyomtatás)
   - ezeknek a műveletsorozatoknak az idejére az adott erőforrás használatára kizárólagosságot kell biztosítani a folyamat számára

2. Egyidejűség (randevú):
   - különböző folyamatokban elhelyezett műveletekre előírhatjuk, hogy azok várják be egymást és "egyidejűleg" hajtódjanak végre
   - egyik folyamat sem léphet túl a benne egyidejű végrehajtásra kijelölt végrehajtási szakaszon mindaddig, amíg valamennyi többi folyamat meg nem kezdte a saját kijelölt szakaszának végrehajtását
   - az egyidejűség tipikus alkalmazása az átmeneti tárolót nem tartalmazó adatátvitel Küld és Fogad műveleteinek végrehajtása

3. Előírt végrehajtási sorrend (precedencia):
   - különböző folyamatok kijelölt műveletei között végrehajtási sorrendet írhatunk elő
   - a precedencia előírása jellegzetesen annak biztosítására használatos, hogy egy folyamat csak akkor kezdje felhasználni a másik folyamat által előállított adatokat, amikor azok már rendelkezésre állnak
   - egy P_i folyamat S_i és egy P_j folyamat S_j utasításainál a precedencia akkor áll fenn, ha S_j végrehajtása csak akkor kezdődhet el, ha S_i már befejeződött

Kölcsönös kizárás:

- egy erőforrás használatát valamelyik folyamatnak be kell fejeznie, mielőtt egy másik folyamat elkezdené
- azon programrészletet, amelynek működése közben a folyamat tevékenységét nem lehet megszakítani, *kritikus szakasznak* nevezzük
- a kritikus szakasz védelmét a kritikus szakasz elején elhelyezkedő belépő (entry) és a végén található kilépő (exit) utasítással valósíthatjuk meg

- Tisztán programozott megoldás:

```
            var közös_jelző:{foglalt,szabad}:=szabad

Valamennyi folyamat:
                  ...
    belépés:  OlvasÉsÍr (közös_jelző)
              if foglalt then goto belépés

              <kritikus szakasz>

    kilépés:  ír (közös_jelző,szabad)
                  ...
```

- Hardver támogatás:

```
    belépés:while testandset(foglal)do üres_utasítás;

    kilépés:foglal:=false;
```

Peterson algoritmus:

```
              var  jelző:array[1..2]of{foglal,szabad}:=szabad
                   következő: {1,2}

P1 folyamat:
                   ...
                   ír (jelző[1],foglal)
                   ír (következő,2)
     belépés1: olvas (jelző[2])
                   if szabad then goto belép1
                   olvas (következő)
                   if 2 then goto belépés1
     belép1:
               <kritikus szakasz>

     kilép1:       ír (jelző[1],szabad)
                   ...
     -----------------

P2 folyamat:
                   ...
                   ír (jelző[2],foglal)
                   ír (következő,1)
     belépés2: olvas (jelző[1])
                   if szabad then goto belép2
                   olvas (következő)
                   if 1 then goto belépés2
     belép2:
               <kritikus szakasz>

     kilép2:       ír (jelző[2],szabad)
                   ...
     -----------------
```

Szemafor:

- a szemafor egy speciális változó, amelyet csak a hozzá tartozó két, oszthatatlan művelettel lehet kezelni
- az általános szemafor esetén a szemafor-változó egész (integer) típusú
- a két művelet elnevezése többféle lehet: Belép és Kilép, vagy Vár (Wait) és Jelez (Signal)
- a P(S) művelet *definíciós programja:*

```
while s<1 do üres_utasítás;
s:=s-1;
```

- a V(S) művelet *definíciós programja:*

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

- mindkét művelet oszthatatlan, valamint azt is, hogy a szemaforváltozó más utasításokkal (írás, olvasás) nem érhető el
- precedencia és kölcsönös kizárás megvalósítására alkalmas a bináris szemafor, amelynek szemaforváltozója csak két értéket vehet fel (0 és 1, vagy foglalt és szabad)
- kölcsönös kizárásra 1 kezdőértékű bináris szemafor használható, amelyre a kritikus szakaszba belépni kívánó folyamat P műveletet, a kritikus szakaszból kilépő pedig V műveletet hajt végre
- precedencia megvalósításához 0 kezdőértékű bináris szemafor használható

Sorrendiség (precedencia): P_i folyamat S_m utasítás csoportja megelőzi P_j folyamat S_n csoportját

s=0

P1	P2
S1	P(s)
V(s)	S2

Kölcsönös kizárás: P_i folyamat S_m és P_j folyamat S_n utasításcsoportjai nem hathatnak egyszerre

s=1

P1	P2
P(s)	P(s)
K1	K2
V(s)	V(s)

Egyidejüség (randevú): P_i és P_j folyamatokban S_m és S_n utasításcsoportok egy időben kezdődnek és bevárják egymást

s_1=s_2=0

P1	P2
V(s_1)	V(s_2)
P(s_2)	P(s_1)
S1	S2

A szemafor megvalósításának pszeudo kódja:

```
type semaphore = record
                   érték:integer:=1;
                   várósor: list of folyamat;
                 end;

P(s):
     s.érték:=s.érték-1;
     if s.érték < 0 then begin
                         Felfűz(s.várósor);
                         Elalszik;
                         end;

V(s):
     s.érték:=s.érték+1;
     if s.érték < 1 then begin
                         Lefűz(p,s.várósor);
                         Felébreszt(p);
                         end;
```

Erőforrás:

- szinkronizációs eszköz egy logikai objektum, amelyet egy folyamat lefoglalhat és felszabadíthat
- lefoglalás és a felszabadítás között a folyamat kizárólagosan használhatja az erőforrást, azaz erre a szakaszára és más folyamatok ugyanezen erőforrásra vonatkozó foglalási szakaszaira kölcsönös kizárás valósul meg
- Lefoglal(<erőforrás>) és Felszabadít(<erőforrás>) műveletek egyenértékűek egy s bináris szemaforra kiadott P(s) és V(s) műveletekkel

Esemény:

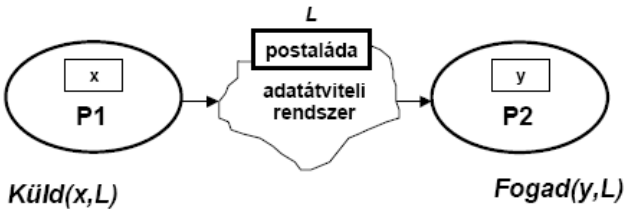
- egy pillanatszerű történés a rendszerben, amelyre folyamatok várakozhatnak
- az esemény bekövetkezése valamennyi rá várakozó folyamatot továbbindítja
- összetett precedencia valósítható meg, ahol több, különböző folyamatokban elhelyezett műveletre ugyanazt az előzményt írjuk elő.
- két folyamat esetén egy esemény jelzése és az arra való várakozás egyenértékű egy szemaforra kiadott V, illetve P műveletekkel
- több folyamat esetén azonban lényeges különbség, hogy a szemaforra kiadott V művelet hatására csak egyetlen várakozó folyamat indulhat tovább, míg az esemény bekövetkezése valamennyi várakozó folyamatot továbbindítja

Folyamatok kommunikációja:

1. Közvetett (direkt) megnevezés:

- kommunikáció két folyamat között zajlik, mind a Küld, mind a Fogad művelet megnevezi a partner folyamatot
- P1 elküldi a saját címtartományában tárolt x változó értékét P2-nek, aki azt saját y változójába teszi el

2. Közvetlen (indirekt) megnevezés:

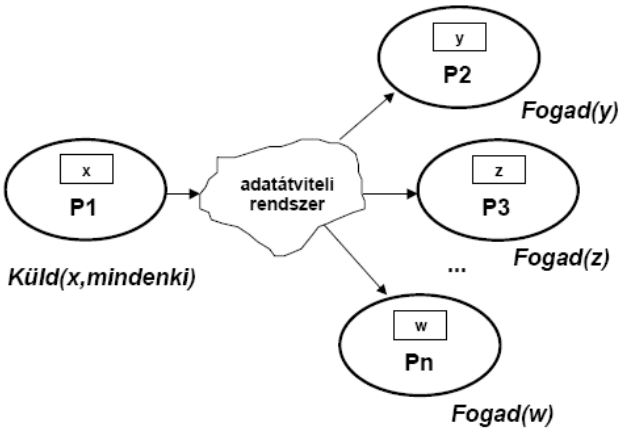


- a partnerek nem egymást nevezik meg, hanem egy közvetítő objektumot (postaládát, vagy csatornát)
- postaláda egy általában véges, de elméletileg esetleg korlátlan befogadóképességű, az üzenetek sorrendjét megtartó (FIFO) tároló, amely a Küld - Fogad, (betesz - kivesz) műveletpárral kezelhető
- a Küld(<cím>,<postaláda>) művelet a saját címtartományban elhelyezkedő üzenetet a postaláda következő szabad tárolóhelyére másolja
- ha a postaláda tele van, ürülésig várakozik
- a Fogad(<cím>,<postaláda>) művelet a postaládában legrégebben várakozó üzenetet kimásolja a megadott, saját címtartománybeli címre, helyét pedig felszabadítja
- a csatorna olyan kommunikációs objektum, amelyik két folyamatot kapcsol össze
- egyirányú (szimplex), osztottan kétirányú, azaz egyidejűleg egyirányú, de az irány változtatható (félduplex), vagy kétirányú (duplex)
- egy véges tárolókapacitású duplex csatorna egyenértékű két véges befogadóképességű postaládával, amelyeket csak két folyamat használ, egyiket egyik irányú, másikat az ellenkező irányú adatcserére

3. Aszimmetrikus megnevezés:

- az egyik folyamat, az adó vagy vevő, megnevezi, hogy melyik folyamattal akar kommunikálni, a másik partner viszont egy saját be/kimeneti kaput (port) használ, amelyiket nem is kell megneveznie
- ha az üzenet vevője használ bemeneti kaput, akkor a műveletek alakja Küld(<cím>,<folyamat>), Fogad(<cím>)

4. Csopotkommunikáció:



- az üzenet küldője folyamatok (esetleg kommunikációs objektumok) egy csoportját nevezheti meg vevőként
- egyetlen üzenetküldő művelet végrehajtása azt eredményezi, hogy az üzenetet a csoport valamennyi tagja megkapja
- az üzenetszórás (broadcasting) logikailag a csoportkommunikáció azon esete, amikor az egyetlen művelettel elküldött üzenet a rendszer valamennyi folyamatához eljut

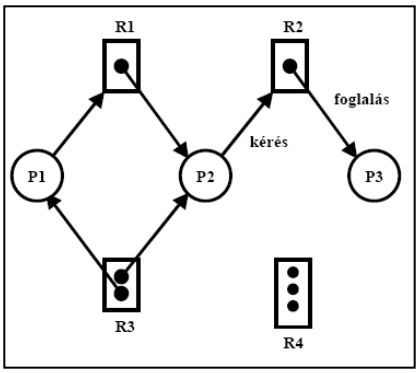
Holtpont:

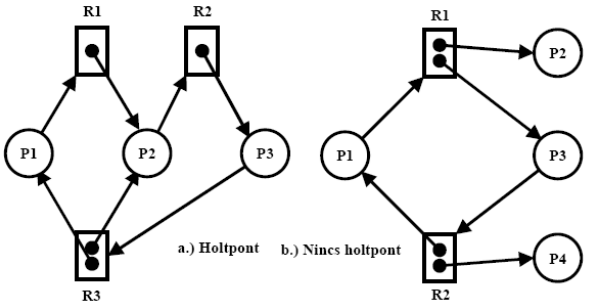
- egy rendszer folyamatainak egy H részhalmaza holtponton van, ha a H halmazba tartozó valamennyi folyamat olyan eseményre vár, amelyet csak egy másik, H halmazbeli folyamat tudna előidézni

Holtpont kialakulásának feltételei:

- *Kölcsönös kizárás:* legyenek olyan erőforrások a rendszerben, amelyeket a folyamatok csak kizárólagosan használhatnak
- *Foglalva várakozás:* legyen olyan folyamat, amelyik lefoglalva tart erőforrásokat, miközben más erőforrásokra várakozik
- *Nincs erőszakos erőforrás-elvétel*: minden folyamat addig birtokolja a megkapott erőforrásokat, amíg saját jószántából fel nem szabadítja azokat
- *Körkörös várakozás:* rendszerben lévő folyamatok közül létezik egy olyan $\{P_0, P_1, ...P_n\}$ sorozat, amelyben P_0 egy P_1 által lefoglalva tartott erőforrásra vár, P_i egy P_{i+1}-re, végül P_n pedig P_0-ra vár

Erőforrás foglalási gráf:

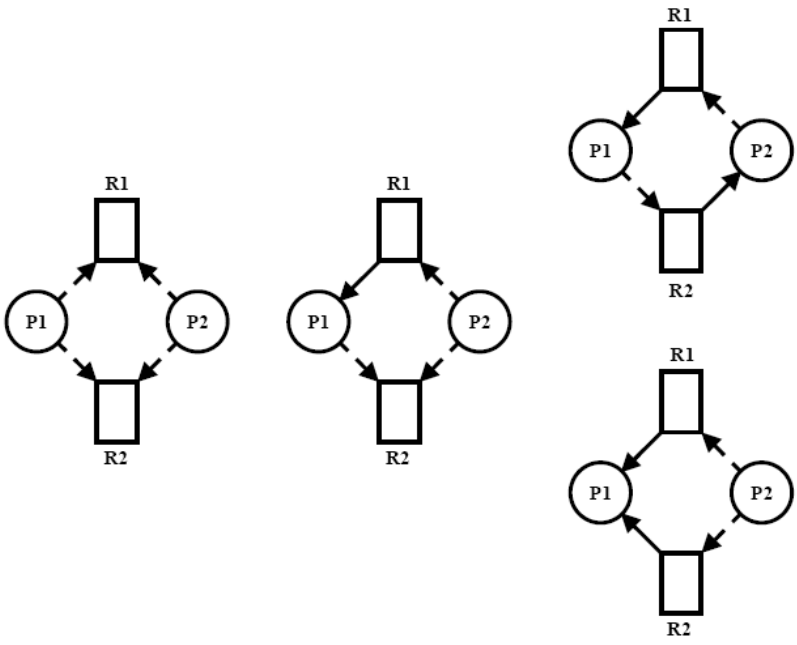




- a gráfnak kétféle csomópontja van: erőforrás (R_i, téglalap) és folyamat (P_i, kör)
- Irányított él vezet P_i folyamattól R_j erőforráshoz (kérés él), ha P_i már kérte R_j-t, de még nem kapta meg
- Irányított él vezet R_i-t_l P_j-hez (foglalás él), ha P_j birtokolja (megkapta és még nem szabadította fel) R_i-t
- többpéldányos erőforrások esetén a példányokat megfelelő számú ponttal jelezzük az erőforrástípust jelképező téglalapon belül
- gráf a rendszer működése során folyamatosan változik, ahogyan új kérések és foglalások történnek, következtethetünk holtponthelyzetek fennállására
- körkörös várakozás esetén a gráfon is irányított kör van
- ha minden erőforrás egypéldányos, a gráfon kimutatható kör egyben elégséges feltétel is a holtpont fennállására
- többpéldányos esetben azonban kör jelenléte nem jelenti feltétlen azt, hogy a rendszerben holtpont van

Holtpont kezelési stratégiák:

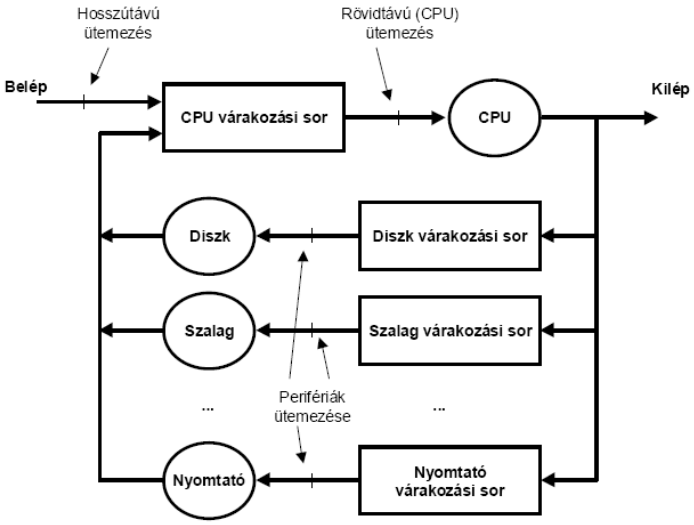
1. Strucc-algoritmus: nem veszünk tudomást a problémáról és nem teszünk semmit
2. Detektálás és feloldás: észrevesszük, ha holtpont alakult ki, és megpróbáljuk feloldani
   - holtpontdetektálást végző program felderíti, hogy vannak-e a rendszerben holtponton lévő folyamatok
   - detektálás indítása történhet az operációs rendszer által automatikusan, vagy kezelői parancsra
   - irányított gráfok kördetektáló algoritmusai: egypéldányos erőforrások
   - Coffman algoritmus: többpéldányos erőforrások
   - ahhoz, hogy a holtponton lévő folyamatok továbbléphessenek, illetve az általuk foglalt erőforrások felszabaduljanak, valamilyen erőszakos megoldáshoz kell folyamodni
   - erőforrások oldaláról: erőforrás elvétele egy vagy több folyamattól
   - folyamatok oldaláról: holtponton lévő folyamatok abortálás
3. Megelőzés: amikor is strukturálisan holtpontmentes rendszert tervezünk, amelyben a szükséges feltételek valamelyikének kizárása miatt eleve nem alakulhat ki holtpont
4. Elkerülés: a rendszer futás közben csak olyan erőforrás-kéréseket elégít ki, amelyek nem vezetnek holtpontveszélyhez
   - rendszer minden erőforrás-igény kielégítése előtt mérlegeli, hogy nem vezet-e holtpontveszélyre a kérés teljesítése
   - Bankár-algoritmus: többpéldányos eset
   - lehetséges kérés élet bevezetése: egypéldányos eset



CPU ütemezés:

- egy multiprogramozott operációs rendszer minden időpillanatban folyamatok egy csoportjának végrehajtásával foglalkozik, amelyeket egyetlen fizikai processzoron futtat
- a multiprogramozás foka a rendszerben egy adott pillanatban jelenlévő (megkezdett, de még be nem fejezett) folyamatok száma
- az operációs rendszer egyik alapvető feladata az erőforrás-gazdálkodás és-kiosztás
- két alapvető erőforrás a processzor (CPU), és a memória
- a processzorért csak azok a folyamatok versenyezhetnek, amelyek következő végrehajtandó kódrésze a memóriában van
- a processzor felszabadulásakor az operációs rendszer választja ki valamilyen algoritmussal a sorban várakozók közül azt a folyamatot, amelyik a következő időszakra megkapja a processzort
- egy processzor által végrehajtott utasítássorozatot (processzor-löket (*CPU-burst*)) egy be/kiviteli művelet (be/kiviteli löket (*I/O-burst*)) követ, és ez a két fázis ismétlődik ciklikusan

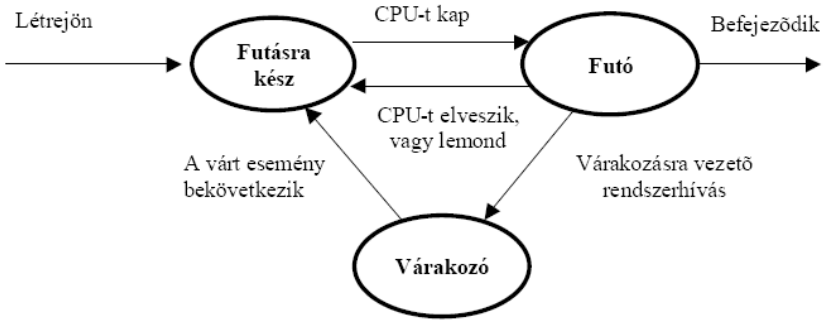
Sorállási modell:



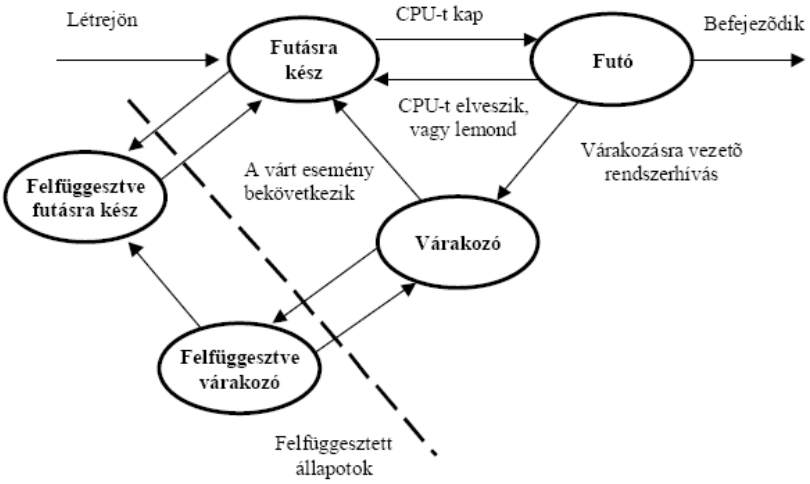
- korlátos erőforráskészletért versengő folyamatok rendszerének elemzésére alkalmas
- diagram a processzor-löket, I/O-löket ciklikusságot szemlélteti azzal a feltételezéssel, hogy a folyamatoknak közben folyamatosan elegendő memória áll rendelkezésükre
- az erőforrásokat körök, a várakozási sorokat téglalapok jelölik
- a rendszer működését a folyamatok irányított élek mentén történő vándorlása szemlélteti
- az új, induló folyamatok processzor-lökettel kezdődnek, ezért a CPU várakozási sorába lépnek be
- normál befejeződés esetén ugyancsak processzor-löketből lépnek ki. Közben a be/kiviteli műveleteiktől függően kerülnek át a különböző perifériák várakozási soraiba, illetve lesznek a perifériák használói.
- egy be/kiviteli löket után ismét a CPU várakozási sorába kerülnek vissza
- a hosszútávú ütemező általában akkor fut le, amikor egy job végrehajtása befejeződik
- a rövidtávú ütemező pedig biztosan lefut, amikor egy processzorlöket befejeződik

- középtávú ütemező – észlelve a memória-szűkét – egyes folyamatokat felfüggeszt, memóriaterületüket háttértárra menti és felszabadítja, átmenetileg kivonja őket az erőforrásokért folytatott versengésből
- a felfüggesztett folyamatok memóriaterületét a többi folyamat használhatja
- később, ha a terhelés csökken, a felfüggesztett folyamatok visszatölthetők és folytathatják működésüket
- középtávú ütemező akkor is felfüggeszthet folyamatot, ha az várhatóan igen hosszú várakozásra kényszerül
- a középtávú ütemező helye nem mutatható meg a diagramon, mert a memória használata és a memóriaigény növekedése más erőforrások (CPU) használata közben lép fel, az erőforrás-használat nem szekvenciális

Állapotmodell:



- egy folyamat végrehajtásának dinamikája a multiprogramozott rendszerben ezzel írható le
- futásra kész (ready) állapotban vannak azok a folyamatok, amelyeknek következő műveletét a CPU bármikor végrehajthatná
- futó (running) állapotban egy multiprogramozott rendszerben egyidejűleg egyetlen folyamat lehet, amelyiknek az aktuális műveletét a CPU éppen végrehajtja
- várakozó (waiting, blocked) állapotban vannak azok a folyamatok, amelyek nem tudják használni a CPU-t, mert valamilyen feltétel teljesülésére várakoznak



- rendszer várakozó, vagy legfeljebb futásra kész folyamatokat függeszt fel, és a felfüggesztett folyamatokat is áthelyezi felfüggesztve futásra kész állapotba a várt esemény ekövetkezésekor

Processzor ütemezés:

1. Preemptiv ütemezés: az operációs rendszer elveheti a futás jogát az éppen futó folyamattól, "futásra kész" állapotúvá teheti, és a CPU-t egy másik folyamatnak adhatja, azaz egy másik folyamatot indíthat el
2. Nem preemptiv ütemezés: az operációs rendszer nem veheti el a futás jogát a folyamattól, azaz a futó folyamat addig birtokolja a CPU-t, amíg egy általa kiadott utasítás hatására állapotot nem vált, azaz csak maga a folyamat válthatja ki az új ütemezést

Központi egység kihasználtság:

- azt mutatja, hogy a CPU idő hány százaléka fordítódik ténylegesen a folyamatok utasításainak végrehajtására
- a kihasználtság tipikus értékei 40-90 % közé esnek

$$\frac{\sum CPUidő - \sum (henyélés + admin isztrációsidő)}{\sum CPUidő}$$

Átbocsátó képesség:

- az egy időegység alatt elvégzett munkák számát mutatja

$$\frac{elvégzettmunkákszáma}{idő}$$

Körülfordulási idő:

- egy-egy munkára vonatkozóan a rendszerbe helyezéstől a munka befejeződéséig eltelt időt mutatja

$$\sum végrehajtásiidő + várakozásidő$$

Várakozási idő:

- értéke azt mutatja, hogy egy-egy munka összességében mennyi időt tölt várakozással

$$\sum várakozásidő + futásrakészidő + felfüggesztettidő + hosszútávúütemezésigelteltidő$$

Válasz idő:

- az az idő, amely az operációs rendszer kezelői felületének adott kezelői parancs után a rendszer első látható reakciójáig eltelik

Ütemezési algoritmusok:

1. Legrégebben várakozó FCFS:
   - nem preemptív algoritmus
   - legrégebben várakozó folyamatot választja ki futásra
   - nagy lehet az átlagos várakozási idő, mivel egy-egy hosszú CPU löketű folyamat feltartja a mögötte várakozókat
2. Körbeforgó RR:
   - preemptív algoritmus
   - minden folyamat, amikor futni kezd, kap egy időszeletet
   - ha a CPU lökete nagyobb ennél, akkor az időszelet végén az ütemező elveszi a folyamattól a processzort, és a futásra kész állapotúvá tett folyamatot a futásra kész sor végére állítja
   - ha a CPU löket rövidebb az időszeletnél, akkor a löket végén a rendszer folyamatait újraütemezzük, és a futó folyamat időszelete újraindul
3. Prioritásos:
   - futásra kész folyamatok mindegyikéhez egy, a futás kívánatos sorrendjére utaló számot
   - algoritmusok pedig a futásra kész folyamatok közül a „legnagyobb” prioritásút választják ki következő futtatandónak
4. Legrövidebb (löket)idejű SJF:
   - nem preemptív algoritmus
   - a futásra kész folyamatok közül a legrövidebb becsült löketidővel rendelkező folyamatot választja ki következőnek futásra
5. Legrövidebb hátralevő idejű SRTF:
   - preemptív algoritmus
   - ha egy új folyamat válik futásra késszé, akkor az ütemező megvizsgálja, hogy az éppen futó folyamat hátralevő löketideje, vagy az új folyamat löketideje kisebb, és a rövidebbet indítja el

Elhelyezési algoritmusok:

1. első megfelelő (first fit):
   - a tár elejéről indulva az első elegendően nagy területet foglalja le a folyamat számára
2. következő megfelelő (next fit):
   - keresést nem a tár elején, hanem az utoljára lefoglalt tartomány végétől kezdi
3. legjobban megfelelő (best fit):
   - a legkisebb, még elegendő méretű területet foglalja le
4. legrosszabban illeszkedő (worst fit):
   - a legnagyobb szabad területből foglal

Virtuális tárkezelés:

- olyan szervezési elvek és az operációs rendszer olyan algoritmusainak összességét takarja, mely megengedi és biztosítja, hogy a rendszer folyamataihoz tartozó logikai címtartományoknak csak egy - a folyamat futásához éppen szükséges - része legyen a központi tárban, de ennek ellenére bármelyik folyamat szabadon hivatkozhasson bármilyen, tartományában szereplő logikai címre
- előnyös, ha a folyamatok tárterületének csak egy részét tartjuk egyidejűleg a memóriában
- amikor egy folyamat érvénytelen - azaz nem a valós memóriában levő - címre hivatkozik, a címképző hardver hibamegszakítást okoz, amelyet az operációs rendszer kezel, és behozza a háttértárról a szükséges blokkot

Virtuális tárkezelés lépései:

1. az operációs rendszer megszakítást kiszolgáló programrésze kapja meg a vezérlést
2. elmenti a folyamat környezetét
3. elágazik a megfelelő kiszolgáló rutinra
4. eldönti, hogy a megszakítást programhiba (pl. kicímzés) vagy logikailag érvényes, de nem betöltött blokkra való hivatkozás okozta
5. az operációs rendszer behozza a kívánt blokkot a központi tárba
6. a blokknak helyet keres a tárban; ha nincs szabad terület, fel kell szabadítania egy megfelelő méretű címtartományt, ennek tartalmát esetleg a háttértárba mentve
7. beolvassa kívánt blokkot
8. az operációs rendszer átadja a vezérlést a folyamatnak, amely ismételten végrehajtja, vagy folytatja a megszakított utasítást

- a folyamatok futásának sebességét pedig döntően a tárhoz férés effektív ideje határozza meg
- ez virtuális tárkezelés esetén a megfelelő előfordulási valószínűséggel súlyozott memória hozzáférés, illetve (laphiba esetén) lapbehozatali időből számítható
- ha p jelöli a laphiba előfordulás valószínűségét, akkor

$$effektívhozzáférésiidő = (1-p)\cdot memóriahozzáférésiidő + p\cdot laphibaidő$$

Tárhierarchia:

- egy számítógépes rendszerben a tárak hierarchikus rendbe szervezettek
- a hierarchia legalsó szintjén a fizikai processzor regiszterei helyezkednek el
- fölötte sorrendben az operatív tár (memória)
- a háttértárak, vagy másodlagos tárolók
- valamint a külső tárak, vagy harmadlagos tárolók
- minél magasabb hierarchia-szinten van egy tár, általában annál nagyobb mennyiség, adat befogadására alkalmas viszont annál lassabb működésű, és annál nagyobb egységekben címezhető
- a processzor-regiszterekben tárolt adatok élettartama néhány utasítás
- a memóriatartalomé jó esetben a program futási ideje
- a programokat túlélő, maradandó adatokat fájlok tárolják,
- egy aktualitási időszakban a másodlagos tárakon, majd archiválva, harmadlagos tárolókon

Szintek közötti kommunikáció:

- a szintek közötti adatmozgatás hagyományos megoldása, hogy a processzor utasításonként esetleg többször is a memóriához fordul
- programokat végrehajtás előtt a háttértárról a memóriába kell tölteni, a háttértáron tárolt adatokat csak fájlműveletekkel lehet elérni,
- a harmadlagos külső tárakból pedig a személyzet közreműködésével kell kikeresni és a megfelelő perifériába helyezni a megfelelő adathordozókat
- az egyes hierarchiaszintekhez más-más elérési, hivatkozási módok tartoznak
- operatív tár címeire az utasítások meghatározott címzési módokkal és numerikus címértékekkel hivatkoznak
- a háttértárakon tárolt fájlokra ezzel szemben nevekkel, amelyekhez csak futási időben kötődnek konkrét blokkcímek
- a memória bájtonként, szavanként címezhetően érhető el, a fájlok ezzel szemben gyakran szekvenciális elérésűek
- az alacsonyabb szint elérési módjait terjesztik ki a magasabb szint, tárra (virtualizálás)
- a magasabb szint, tár elérési módját használva valójában egy alacsonyabb szint, tárat kezelnek (gyorsítótár, cache)

Gyorsítótárak:

- a processzorokba épített hardver gyorsítótárak (utasítás- és adatcache)
- a fájlok egy részének tárolására a memóriában kialakított átmeneti tárterületek (buffer-cache)
- a memóriában kialakított, több fájl tárolására alkalmas virtuális diszkek (RAM-diszk, elektronikus diszk)
- a harmadlagos tárak fájlrendszereit tároló mágneslemez területek

Lapcsere stratégiák:

- Lapcsere esetén ki kell választani azt a lapot, amelyiket feláldozzuk az új lap betöltése érdekében
- az áldozat kimentése és az új lap betöltése
- minden fizikai laphoz tartozik egy jelzőbit
- ezt a bitet a lap betöltésekor az operációs rendszer törli, a tárkezelő hardver pedig minden, a lapra író memóriaművelet végrehajtásakor beállítja
- hivatkozott bitet a címképző hardver állítja be minden esetben, amikor az adott lapon belüli címre történik hivatkozás

1. Optimális algoritmus OPT:
   - az algoritmus előrenéz és a lapok következő használatának idejét veszi figyelembe
   - legkevesebb a laphibák száma
   - jövőbeni laphivatkozásokra vonatkozó információt igényel
   - gyakorlatilag megoldhatatlan
2. Legrégebbi lap algoritmus FIFO:
   - hátranéz és a behozatal idejét figyeli
   - azt a lapot cseréli le, amelyik legrégebb óta a tárban van

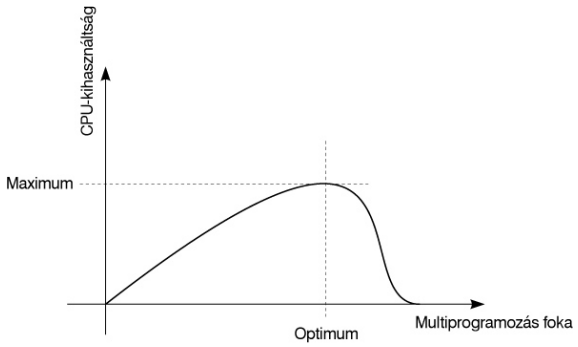
Bélady anomália:

- bizonyos esetekben, ha növeljük a folyamathoz tartozó fizikai memória keretek számát, akkor nem csökken, hanem éppen növekszik a laphiba gyakoriság

3. Újabb esély algoritmus SC:
   - FIFO olyan változatát valósítja meg, amely a sor elején levő lapot csak akkor cseréli le, ha nem hivatkoztak rá
   - hátranéz és a használat tényét figyeli
   - ha hivatkoztak a lapra, akkor a hivatkozott bitet törli és a lapot visszateszi a FIFO lista végére, vagyis a lap kap egy újabb esélyt
4. Legrégebben nem használt algoritmus LRU:
   - azt a lapot választja áldozatul, amelyre a folyamatok leghosszabb ideje nem hivatkoztak
   - ez az algoritmus közelíti legjobban az optimálist, mivel ugyan hátrafele néz, viszont a használat idejét veszi figyelembe
5. Legkevésbé használt algoritmus LFU:
   - a közelmúltban gyakran használt lapokat a folyamatok a közeljövőben is használni fogják még, és ugyanígy, a közelmúltban ritkán, vagy nem használt lapokra a közeljövőben nem lesz szükség
   - a legkisebb számláló értékkel rendelkező, vagyis a legritkábban használt lapot választja ki kivitelre

Tárgazdálkodás:

- hány lapot adjunk az egyes folyamatoknak
- minél több lap van a tárban, hiszen nagy valószínűséggel annál kevesebb laphibát okoznak
- túl kevés lap esetén állandóan laphiba lép fel
- ha a rendszerben átlagosan nem tud befejeződni egy laphiba kiszolgálása, mielőtt egy újabb laphiba fellép, a folyamatok felgyűlnek a mágneslemez várakozási sorában, és a CPU-nak nem lesz futtatható folyamata, CPU-tétlenség alakul ki
- gyakori laphibák által okozott teljesítménycsökkenést vergődésnek nevezzük
- ha egy folyamatnak sok lapot adunk, azt jelenti, hogy kevesebb folyamatot tudunk a tárban tartani, így alacsonyabb lesz a multiprogramozás foka és ezzel várhatóan a CPU kihasználtság is



- a görbe kezdeti szakaszán kevés folyamat van a rendszerben
- minél kevesebben vannak, annál nagyobb annak a valószínűsége, hogy mindegyikük várakozik, a CPU pedig tétlen
- a folyamatok számának a növekedésével a CPU-kihasználás aszimptotikusan közelít az adminisztrációs (pl környezetváltási) veszteségekkel csökkentett 100%-os maximumhoz
- a folyamatok számának további növekedésekor azonban - mivel az egy folyamatra jutó tárterület csökken - belép a tárkezelés hatása és kialakul a vergődés

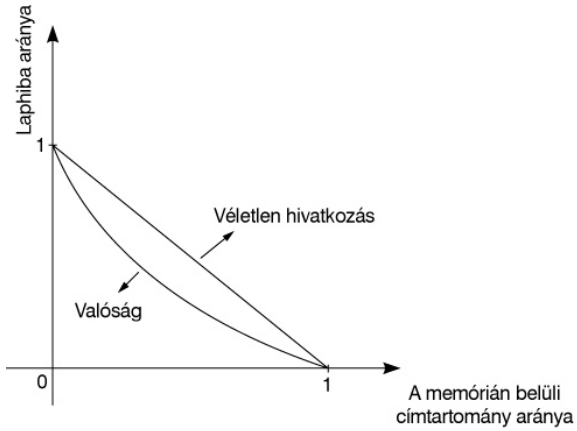
- a CPU-kihasználás pedig a folyamatok számának további növelésével meredeken leesik.
- laphiba gyakoriság (page fault frequency)
- egy folyamatnak annyi lapot adni, amennyi szükséges az egyensúlyhoz, azaz ahány lapra hivatkozik a laphiba kiszolgálás átlagos ideje alatt
- ezt az értéket a munkahalmaz méretének nevezzük

Munkahalmaz:

- folyamat azon lapjainak a halmaza, amelyre egy adott időintervallumban hivatkozik

Lokalitás:

- a folyamatok statisztikailag megfigyelhető tulajdonsága, hogy egy időintervallumban a címtartományuknak csak egy szűk részét használják
- *időbeni lokalitás* alatt azt értjük, hogy egy hivatkozott címet a folyamat a közeljöv&ben várhatóan újra használni fog,
- *térbeli lokalitás* fogalma azt takarja, hogy az id&ben egymáshoz közelálló hivatkozások nagy valószínűséggel egymáshoz közeli címekre történnek
- közelmúltra vonatkozó munkahalmaz nem fog lényegesen eltérni a közeljövőben szükséges munkahalmaztól

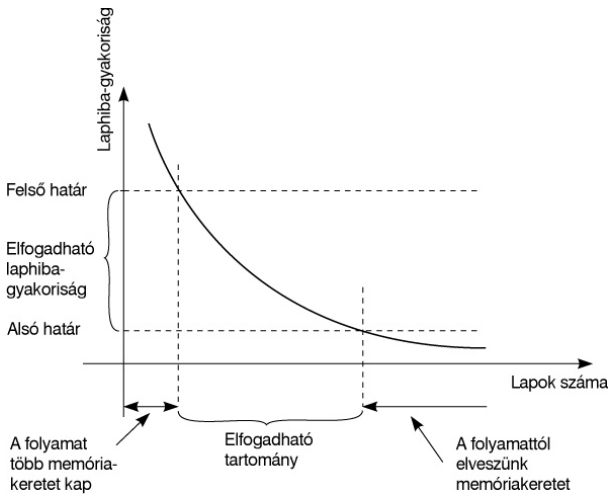


- a lokalitás hatással van arra is, hogy hogyan alakul a laphiba gyakoriság a folyamat teljes címtartományából a memóriába töltött hányad függvényében
- a laphibák aránya nem lineáris függvénye lesz a memóriába töltött címtartomány arányának, hanem ennél kisebb értékeket kapunk

Dinamikus lokális tárgazdálkodás:

- az optimális rendszerműködéshez tehát el kell kerülni a vergődést, és törekedni kell arra, hogy a folyamatoknak a lokalitás alapján meghatározott munkahalmaza futás közben a központi memóriában legyen
- a folyamatokat úgy érdemes elindítani vagy felfüggesztés után újraindítani, hogy egyszerre több lapjukat behozzuk a tárba
- az optimális (legmagasabb fokú) multiprogramozás eléréséhez a legtöbb rendszer lokális lapcsere algoritmust alkalmaz
- munkahalmaz tárban tartását a folyamatokhoz tartozó memóriaterület méretének dinamikus változtatásával biztosítja
- a globális lapgazdálkodás kedvezőtlen kölcsönhatást okozhatnának egymástól egyébként független folyamatok között

- a statikus lokális tárgazdálkodás ezt megakadályozza, de nem tud alkalmazkodni a folyamatok futás közben változó lapigényéhez
- a folyamatok aktuális lapigényének meghatározására az egyik megoldás a munkahalmaz mérése
- a folyamatok laphiba gyakoriságát, illetve az ezzel egyenértékű laphibák között eltelt időt mérik



- ha a laphiba gyakoriság meghalad egy felső határértéket, akkor a folyamathoz újabb lapot rendel, vagy pedig a multiprogramozás fokának csökkentése mellett dönt, és felfüggeszti valamelyik folyamatot
- ha pedig a laphiba gyakoriság alatta marad egy alsó határértéknek, akkor elvesz a folyamattól egy lapot
- új folyamatot pedig csak akkor lehet elindítani, ha van elegendő szabad memóriakeret a számára

File allokációs megoldások:

1. Folytonos terület lefoglalása:
   - a fájlhoz tartozó információt egymás melletti szabad blokkokban tároljuk
   - a rendszernek csupán az első blokk sorszámát, valamint a blokkok számát kell tárolni az állományt leíró adatok között
   - előny: tárolt információnak mind soros, mind közvetlen elérése egyszerű, gyors
   - hátrány: esetek nagy többségében nem tudjuk előre, hány blokkra lesz szükségünk, külső tördelődés problémája
2. Láncolt tárolás:
   - a szükséges blokkokat egyesével allokáljuk, minden blokkban fenntartva egy helyet a következő blokk sorszámának
   - előny: dinamikus adatszerkezet, az állomány mérete szükség szerint tetszőlegesen növekedhet, határt csak a szabad blokkok száma szab
   - hátrány: csak a tárolt információ soros elérést támogatja, a közvetlen eléréshez a listát az elejéről végig kell járni, lánc szervezéséhez szükséges címet a blokkok hasznos területéből kell lecsípni

FAT tábla:

- láncolt tárolás egy változata, ahol a láncokat az állományoktól elkülönülten, egy fájl allokációs táblában (FAT) tároljuk
- táblának minden eleme egy lemez blokkhoz tartozik, amelyben, ha a blokk egy állományhoz tartozik, tárolhatjuk az állományhoz tartozó következő blokk címét
- az egyes állományokhoz csak a lánc els& blokkjának sorszámát kell tárolni az állományleíróban, a következő blokkok a FAT-ból megtalálhatók
- tárolt információ közvetlen elérése is egyszerűbb, nem kell az egyes blokkokat végigjárni

3. Indexelt tárolás:
   - az állományhoz tartozó blokkokat leíró címeket külön adatterületre, az indextáblába gyűjtjük
   - az állományleíróban az indextáblát tartalmazó blokk(ok) címét kell tárolni
   - előny: közvetlen hozzáférés megvalósítása egyszerű, hiszen az indextábla segítségével minden blokk közvetlenül megtalálható, nem használt területekhez nem kell lemez blokkot lefoglalni, azok helyét az indextáblában üresen hagyjuk
   - hátrány: a blokkok címeinek tárolásához akkor is legalább egy teljes (index)blokkot kell lefoglalni, ha az állomány csak kevés blokkot tartalmaz, pazarló
4. Kombinált módszerek:
   - egyes rendszerek az állomány hozzáférési módja, vagy mérete függvényében más és más módszert alkalmazhatnak
   - kis állományokat folytonosan, míg nagyokat indexelten tárolhatunk
   - soros hozzáférés igénye esetén láncolt, közvetlen hozzáférés esetén összefüggő blokkok tárolását választhatjuk

Mágneslemez felépítése:

- sáv (track) egy-egy lemezfelület azon területe, amelyet a fej elmozdulás nélkül, a lemez egyetlen körülfordulása alatt elér
- cilindernek (cylinder) nevezik a fejtartó szerkezet egy adott pozíciójában leolvasható sávok összességét
- egy-egy sávot szektorokra (sector) osztanak
- az információtárolás a szektoron belül bitsoros
- a szektor az információátvitel legkisebb egysége, a lemezvezérl& egyszerre egy teljes szektort olvas vagy ír

Átvitel kiszolgálásának ideje:

- a fejmozgási idő (seek time) az az idő, amíg a fej a kívánt sávra áll
- az elfordulási idő (latency time) az az idő, amíg a kívánt szektor a fej alá fordul
- az információ átvitelének ideje (transfer time).

Szektorok címzése:

- a lemez szektorait az operációs rendszer lineárisan címzi, a lemezillesztő viszont több komponens címet igényel
- $b = s \cdot (i \cdot t + j) + k$
- s a sávon lévő szektorok száma, t a cilindereken lévő sávok száma, i a kijelölt cilinder, j a fej (lemez felület) száma, k pedig a sávon belüli szektorok száma
- b eredmény a szektor lineáris, 0-tól induló sorszáma.

Fejmozgás ütemezése:

1. Sorrendi kiszolgálás FCFS:
   - átviteli kéréseket érkezésük sorrendjében szolgáljuk ki
   - algoritmus nem törődik a fej mozgásával
   - kicsi az átbocsátó képessége
   - nagy az átlagos válaszideje, de ennek szórása viszonylag kicsi
2. Legrövidebb fejmozgási idő SSTF:
   - algoritmus következőnek azt a kérést szolgálja ki, amelyik az aktuálishoz legközelebb lévő cilinderre hivatkozik
   - válaszidők szórása nagy
   - fennáll a kiéheztetés veszélye
3. Pásztázó SCAN:
   - algoritmus a következő kérés kiválasztásánál csak azokat a kéréseket veszi figyelembe, amelyekhez szükséges fejmozgás az aktuális mozgási iránynak megfelelő
   - a mozgási irány akkor fordul meg, ha az aktuális irányban már nincs több kiszolgálatlan kérés
   - középső cilindereket gyakrabban látogatja, mint a szélsőket
   - válaszidő szórása is kisebb
4. N lépéses pásztázó N-SCAN:
   - egy irányba mozogva csak azokat a kéréseket szolgálunk ki, amelyek a pásztázás elején már megvoltak
   - a pásztázás közben érkező kérésekre csak a következő irányváltás után kerül sor
   - válaszidejének szórása kisebb, ami akkor sem nő meg, ha az aktuális cilinderre sok kérés érkezik.
5. Egyirányú (körforgó) pásztázó C-SCAN:
   - a kérések kiszolgálása mindig csak az egyik irányú fejmozgásnál történik, a másik irányban a fej közvetlenül a legtávolabbi kérés cilinderére ugrik
   - elkerüli a külső sávoknak a belsőkhöz viszonyított alacsonyabb fokú kiszolgálását

Fájlmodellek:

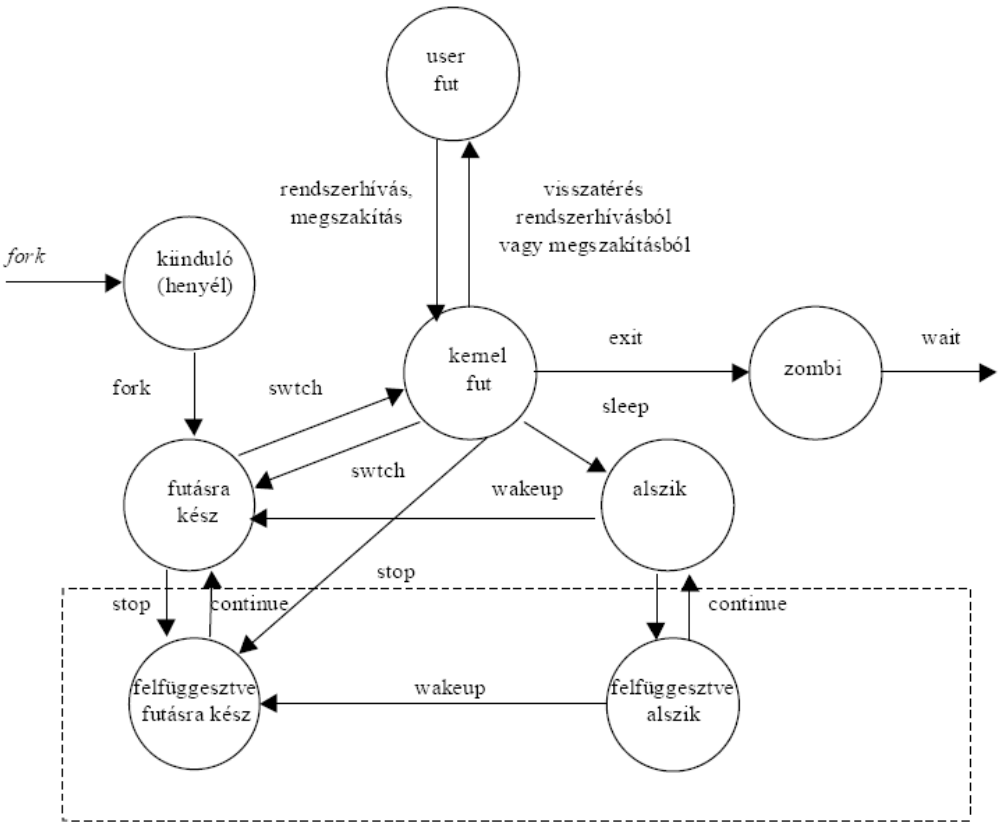
1. Soros elérésű:
   - a szalagon tárolt információt csak a tárolt byte-ok sorrendjében lehetet olvasni vagy írni
2. Közvetlen elérésű:
   - a tárolt adatelemeket közvetlenül, sorszámuk alapján el lehet érni
   - az átviteli eljárásoknak paraméterként meg kell adni az adatelem állományon belüli címét (sorszámát)
3. Indexelt, index-szekvenciális elérésű:
   - ha a fájl rekordokat tárol, akkor a rekordokat kijelölt mezőik (kulcs) tartalma alapján érhetjük el
   - a keresés megkönnyítésére egy segédfájl (index-fájl) szolgál, amelyik a fájlban el forduló kulcsértékek szerint rendezve tárol egy-egy mutatót a megfelel rekordra

Fájlműveletek:

1. Adatelérés, írás vagy olvasás:
   - közvetlen hozzáférésnél a művelethez meg kell adni az információ címét, pozícióját
   - soros hozzáférésnél az átvitel során az aktuális pozíciót a rendszer növeli és tárolja
   - read(file_id,adat)
   - write(file_id,adat)
2. Hozzáírás, bővítés:
   - az állomány végéhez új információt írunk. Az állomány mérete növekszik.
3. Pozicionálás:
   - a soros hozzáférésnél használt pozíciót a soros átvitel mindig növeli, pozicionálásnál viszont azt az állomány tetszőleges pontjára állíthatjuk
   - seek(file_id,pozíció)
4. Megnyitás (open):
   - az állomány megkeresése, fizikai elhelyezkedésének meghatározása, a további elérést segít adatszerkezetek felépítése
   - hozzáférési jogosultságok ellenőrzése
   - az állományon elvégezhető műveletek (írás, olvasás, hozzáírás) megadása
   - osztott állománykezelés szabályozása
   - fájlmutató kezdeti beállítása
   - az állomány sikeres megnyitása után az operációs rendszer visszaad egy olyan értéket amely a további műveletekben a név helyett használható az állomány azonosítására
   - open(file_név,file_azonosító)
5. Lezárás (close):
   - Jelzi a folyamat részéről az adatelérési műveletsorozat befejezését
6. Végrehajtás (execution):
   - az operációs rendszer létrehoz egy új folyamatot, a programfájlt betölti a folyamat tárterületére és elindítja
7. Létrehozás (create):
   - az állomány létrehozásakor egy új katalógus-bejegyzés is létrejön, a rendszer szükséges esetén az állományhoz szabad blokkokat foglal le
   - Create(fájl_név,(méret))
8. Törlés (delete):
   - a törlés megszünteti a katalógus-bejegyzést, felszabadítja az állományt tároló területet a másodlagos táron

UNIX folyamatok:

- a UNIX folyamatai egy jól-definiált hierarchiát alkotnak
- minden folyamatnak pontosan egy szülője (parent) van, és egy vagy több gyermek folyamata (child process) lehet
- folyamat hierarchia tetején az init folyamat helyezkedik el
- az init folyamat az első létrehozott felhasználói folyamat, a rendszer indulásakor jön létre
- minden felhasználói folyamat az init folyamat leszármazottja
- ha egy folyamat befejeződésekor még léteznek aktív gyermek folyamatai, akkor azok árvákká (orphan) válnak és azokat az init folyamat örökli



- egy új folyamatot a fork rendszerhívással lehet létrehozni, amelynek hatására a folyamat kezdeti állapotba kerül
- itt végrehajtódnak a legfontosabb inicializálások, majd a fork lefutása után a folyamat készen áll a futásra, ezt az állapotot jelöli a *futásra kész* állapot
- ütemezés hatására egy környezetváltással *kernel fut* állapotba kerül
- folyamat a rendszerhívás befejeződése után átlép *user fut* állapotba
- *user* módban futó folyamat egy rendszerhívás vagy egy megszakítás hatására léphet át *kernel fut* állapotba, ahonnan a rendszerhívásból vagy a megszakításból való visszatérés után léphet vissza a *user fut* állapotba
- amikor egy folyamat befejezi futását és végrehajtja az exit rendszerhívást, átlép *zombi* állapotba
- a *zombi* állapotban a folyamat már felszabadította a foglalt memóriát, lezárta az állományokat, minden erőforrását visszaadta a rendszernek, csak a proc struktúráját tartja fogva, amiben visszatérési és statisztikai információkat tárol a szülő számára

- a szülő wait rendszerhívásának hatására a folyamat véglegesen kilép az állapot átmenet gráfból, felszabadul a proc struktúra is, a folyamat teljesen megszűnik létezni
- amikor egy folyamatnak valamilyen erőforrásra kell várakoznia, akkor kiad egy sleep rendszerhívást és *alvó* állapotba megy
- az *alvó* állapotból a várt esemény bekövetkezése által kiváltott wakeup rendszerhívás hatására a folyamat átlép a *futásra kész* állapotba és várja, hogy az ütemező ismét futásra ütemezze
- egy alvó folyamat egy stop jelzés hatására átlép a *felfüggesztve alszik* állapotba
- ebből az állapotból a continue jelzés hatására a folyamat visszalép az *alvó* állapotba
- ha a *felfüggesztve alszik* állapotban következik be egy wakeup rendszerhívás, akkor a folyamat átlép a *felfüggesztve futásra kész* állapotba, ahonnan egy continue jelzés hatására kerül át a *futásra kész* állapotba

Folyamatok közötti kommunikáció:

1. Jelzések:
   - a folyamatok értesítése aszinkron módon bekövetkező eseményekről
   - egyszerű folyamatok közötti kommunikációt is lehetővé tesznek
2. Csővezetékek:
   - egy egyirányú, FIFO jellegű, nem strukturált, változó méretű adatokat közvetítő adatfolyam
   - a csővezeték írói adatokat helyeznek a csőbe, olvasói kiolvassák azokat
   - a kiolvasott adatok elt2nnek a csőből
3. Szemaforok:
   - általános szemafor mechanizmusát és operációit (P() és V()) valósítják meg
4. Üzenetsorok:
   - egy olyan leíró, mely üzenetek láncolt listájára mutat
   - minden egyes üzenet egy típusjelző után egy adatterületet tartalmaz
   - FIFO működésűek, azaz a kernel nyilvántartja az üzenetek érkezési sorrendjét, és a legelőször érkezett üzenetet adja vissza az első olvasási kérésre
5. Osztott memória:
   - a központi tár egy olyan régiója, melyet egyszerre több folyamat is használhat
   - a folyamatok ezen tartományt bármilyen virtuális címhez hozzárendelhetik saját címtartományukban
   - leggyorsabb kommunikációs forma azonos gépen futó folyamatok között.
6. Socket:
   - egy absztrakt objektum, ami felhasználható üzenetek küldésére és fogadására

System V állományrendszer (s5fs) adatszerkezete:

boot blokk	szuper blokk	*inode* lista	adat blokkok

- *boot blokk:* a rendszer elindulásához szükséges információkat tartalmazza
- *szuperblokk:* az állományrendszerről tartalmaz a rendszer működéséhez szükséges metaadatokat, az állományrendszer állapotát írja le
- *inode lista:* Minden állományhoz pontosan egy inode tartozik, ezek alapján lehet az állományokra hivatkozni, lista méretét a rendszergazda adja meg, meghatározva ezzel az állományrendszerben a maximális állományszámot
- *adat blokkok:* ezek szolgálnak a fizikai adattárolásra

Szuperblokk szerkezete:

- a szuperblokk az állományrendszer egészéről tartalmaz fontos információkat, ún. metaadatokat
- *szuperblokkban tárolt legfontosabb információk*: az állományrendszer mérete, a szabad blokkok száma, a szabad blokkok listája, a következő szabad blokk indexe, az inode lista mérete, a szabad inode-ok száma, inode lista (tömb), a következő szabad inode indexe, lock mezők (inode, blokk lista), módosítás jelzőbit

Inode lista:

- az inode tartalmaz egy típus mezőt, amely értéke 0 ha az inode szabad, vagyis felhasználható
- az állományrendszerben gyakori esemény, hogy új inode-ra van szükség, hisz minden megnyitott állományhoz hozzá kell rendelni egyet
- szuperblokk tartalmaz egy fix méretű tömböt (szuperblokk inode lista), amelyet gyorsítótárként (cache) használ a szabad inode-ok sorszámainak tárolására
- minden egyes elem egy szabad inode sorszámát tárolja
- ha a rendszernek szabad inode-ra van szüksége keresés helyett kiveszi a tömbben tárolt utolsó elemet
- ez megmutatja, hogy hány felhasználható elem (szabad inode sorszám) van még a tömbben, illetve éppen az utolsó felhasználható elemre mutat
- amikor kiürül a szuperblokk inode listája, a kernel elkezdi vizsgálni az (igazi) inode listát, szabad inode-okat keres, és hátulról kezdve teljesen feltölti a szuperblokkon belüli inode listát
- az utolsó megtalált szabad inode sorszám kerül a tömb első helyére
- amikor a kernel felszabadít egy inode-ot, akkor megnézi, hogy van-e szabad hely a tömbben
- amennyiben igen, egyszerűen beteszi a sorszámot a tömb első szabad helyére, és eggyel növeli az indexet
- amennyiben a tömb tele van, megnézi, hogy a megjegyzett inode sorszáma kisebb-e, mint a felszabadítandó inode-é
- amennyiben a felszabadított inode sorszáma kisebb, mint a megjegyzetté, akkor annak helyére írja

Szabad lemezblokkok listája:

- szabad lemezblokkokat adminisztrálni kell
- a lista rögzített méretű, és a szabad blokkok sorszámait tartalmazza
- az első (a felhasználás sorrendjében utolsó) elem pedig egy olyan blokk sorszáma, melyik szintén szabad blokkok sorszámait tartalmazza
- egy index segítségével a kernel mindig nyilvántartja, hogy hányadik elem tartalmazza a következő szabad blokk sorszámát
- amennyiben szabad blokkra van szükség, a kernel visszaadja az első szabad blokk sorszámot
- amennyiben felszabadul egy blokk, akkor a sorszámát a kernel beírja az első szabad helyre
- amennyiben nincs szabad hely már a listán, akkor a lista tartalmát bemásolja a felszabadított blokkba, és a visszaadandó blokk sorszámát beírja a lista utolsó helyér
- amennyiben a szuperblokkban tárolt szabad blokkok listája már csak egy elemet tartalmaz, akkor ez olyan blokkra mutat, amelyik újabb szabad blokkok sorszámait tartalmazza

Az inode:

- a fizikai állományhoz tartozó azonosító
- ***a***z állományrendszer minden egyes állományához pontosan egy inode tartozik
- az állományrendszerben az inode-ok diszken találhatók, azonban a gyorsabb működés érdekében a kernel cache-eli azokat a memóriában

Inode tárolt információi:

1. tulajdonos azonosító
2. állomány típusa
   - reguláris könyvtár
   - FIFO, karakteres vagy blokkos berendezés
3. állomány hozzáférési jogosultságok
4. időcímkék
   - az utolsó állományhozzáférés ideje
   - az utolsó állomány módosítás ideje
   - az utolsó attribútum módosítás ideje
5. linkek száma
6. címtábla
7. állomány méret

Reguláris állományok szerkezete:

-