Az operációs rendszerek belső működése

Memóriakezelés

Mészáros Tamás
http://www.mit.bme.hu/~meszaros/

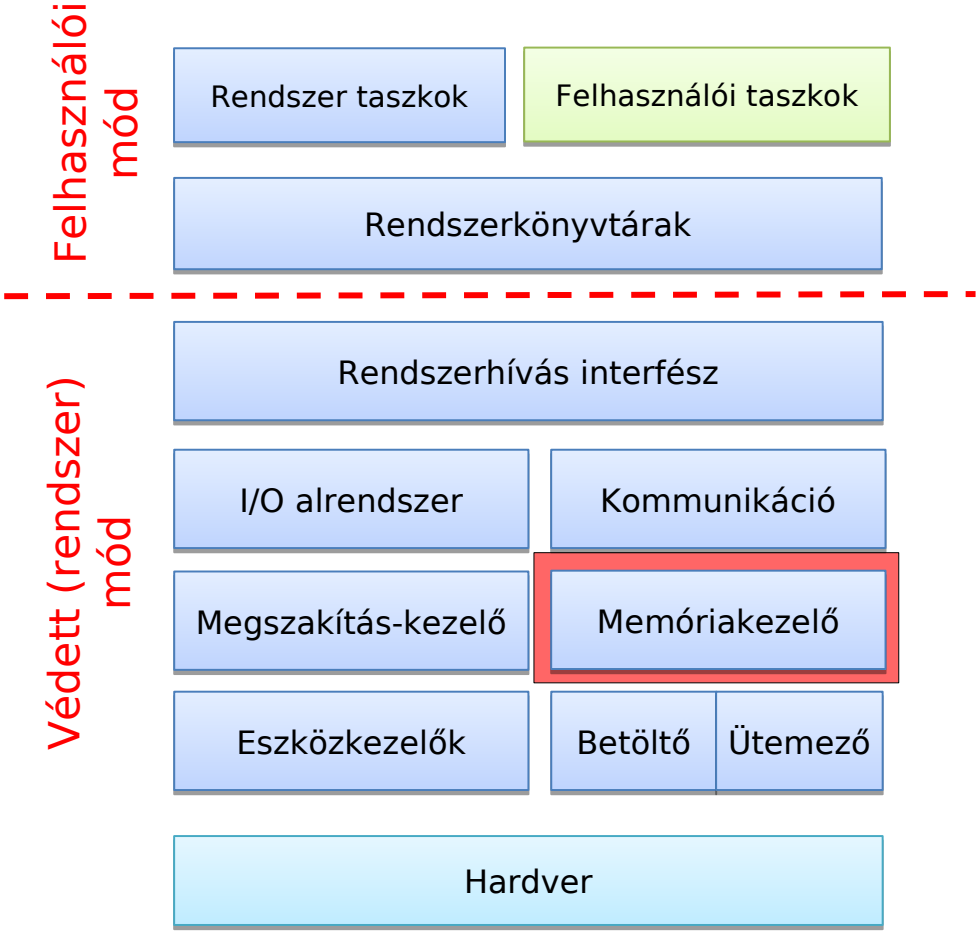
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Az eddigiekben történt...

- Az operációs rendszer
  - feladatok végrehajtása
  - vezérlőprogram
  - **erőforrás-allokátor**
- Erőforrások
  - absztrakt virtuális gép (CPU, mem)
  - sok taszk osztozik az erőforrásokon

multiprogramozott rendszer

Az OS felépítése



A taszkok adatai (ismétlés)

- Saját
  - programkód
  - statikusan allokált adatok
  - verem, átmeneti adattár pl. függvényhívások számára
  - halom, a futásidőben, dinamikusan allokált adattár

- Adminisztratív (kernel)

  taszk- (folyamat-, szál-) leíró
  - egyedi azonosító (PID, TID)
  - állapot (l. később)
  - a taszk kontextusa
    - CPU regiszterek (pl. PC)
    - ütemezési információk
    - **memóriakezelési adatok**
  - tulajdonos és jogosultságok
  - I/O állapotinformációk
  - ...

Az absztrakt virtuális gép koncepció (ismétlés)

- Ideális esetben minden taszk teljesen önállóan fut
- A valóságban osztoznak az erőforrásokon
  - processzor, memória stb.
- Az OS elszeparálja egymástól a taszkokat

absztrakt virtuális gép

virtuális CPU (ezt volt) + **virtuális memória (ez jön)**

A memóriakezelés felhasználói szemmel

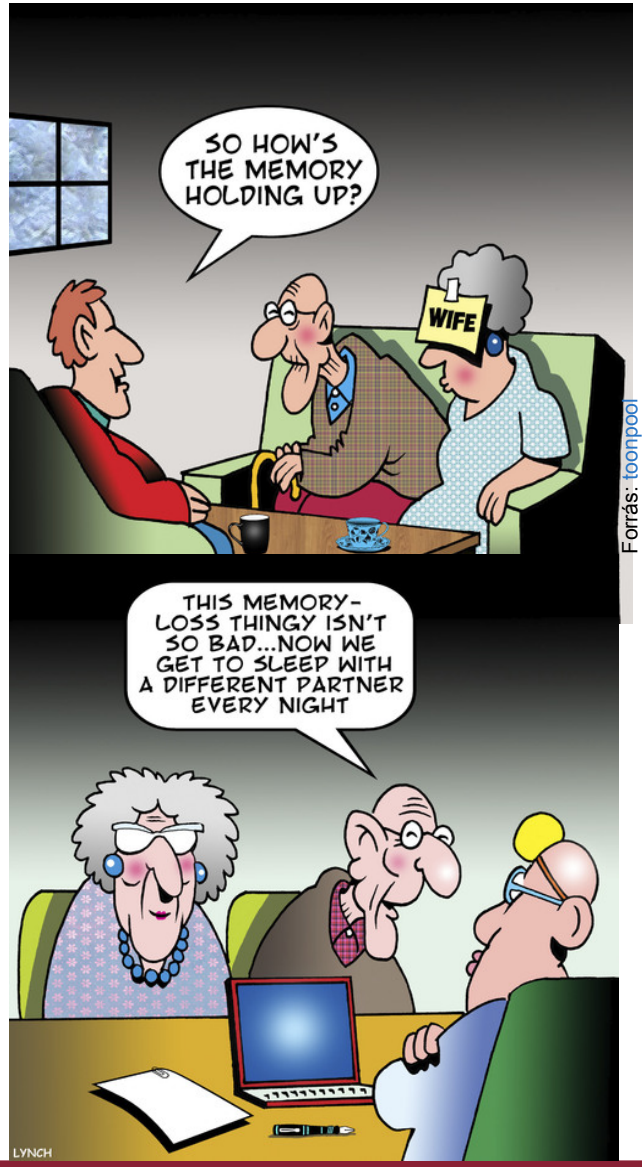
- Felhasználó („end user”)
  - alapvetően nem érdekli
  - van-e elég egy feladat megoldásához (program futtatásához)?
- Adminisztrátor
  - mennyi foglalt, mennyi szabad, hogyan növelhető
  - `top`, `free`, `mkswap`, Erőforrás-figyelő
- Programozó
  - hogyan használható (foglalható, olvasható, írható)
    Hozzáférek a programomból a RAM-hoz?
  - mennyi használható
    4GB RAM van a gépben, használhatok 5GB memóriát?
  - milyen részei vannak a memóriának, és azokban mennyi hely foglalható le
    Mindegy, hogy lokális vagy globális változókat használok? (demo)
  - milyen hibák fordulhatnak elő a használata során
    Mi történik, ha elfogy? … ha hibás címet használok?

```
char a[30000000];
int main () {
}
```

```
int main () {
  char a[30000000];
}
```

Mivel foglalkozik a memóriakezelés?

- Kiosztja az erőforrást
  - erőforrás: fizikai memória
  - igénylők: taszkok és kernel
- Elhelyezi a taszkok adatait
  - programkód + statikus adatok
  - dinamikusan allokált
- Elhelyezi a kernel adatait
  - programkód
  - adminisztratív adatok
- Biztosítja a védelmet
  - szeparáció
  - hibák
- Támogatja a kommunikációt
  - adatcsere taszkok között



Forrás: toonpool

A memóriakezelés kihívásai

- Nem elég az erőforrás
  - sok taszk → sok memória
  - memória-intenzív taszkok
- Hatékonyság
  - minden CPU művelet érint
- Biztonság
  - sok incidens forrása

Forrás: toonpool

Hogyan valósítsuk meg?

Megfigyelések a taszkok memóriahasználatáról

- Neumann-architektúra (lásd szga)
- Induláskor nincs szükségük a teljes programra és adatkészletre
- Működésük során dinamikusan foglalnak memóriát
  - a rendelkezésre álló fizikai memória méretével nem törődnek
  - az allokált memória „szellős”
- Megfigyelhetők lokalitási jellemzők (szga)
  - időbeli, térbeli és algoritmikus
- Vannak sosem használt memóriarészeik
  - sokféle lehetséges lefutásból csak egy következik be
  - hibakezelés, ritkán használt funkciók
- Vannak közösen használt memóriaterületek
  - pl. dinamikus rendszerkönyvtárak, folyamatklónozás (`fork()`)

A virtuális tárkezelés

Összefüggő virtuális memóriatartomány a taszkok számára.
Részekre bontjuk, és csak a használatban levő részeit tároljuk.

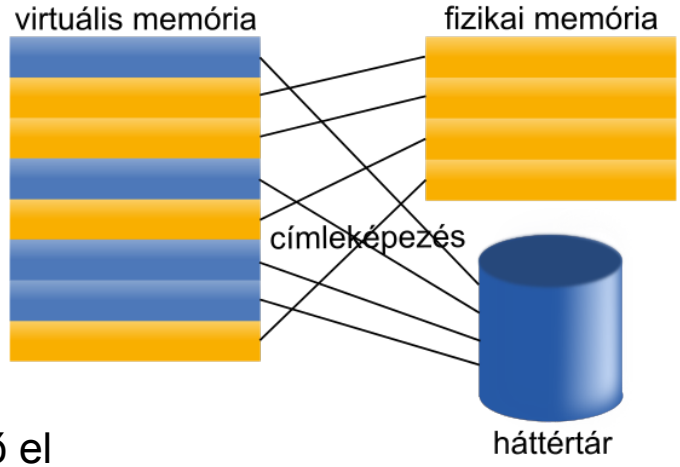
- A feladat
  - a virtuális és fizikai címek megfeleltetése
    → **címleképezés (address translation)**
  - a taszkok memóriatartományának részekre bontása
    → **lapozás (paging)**
  - a (gyors) fizikai memória kapacitásának kiterjesztése
    → **cserehely (swap)**

hardvertámogatással

- Elvárásaink
  - minél több taszk működjön párhuzamosan
  - feleslegesen ne foglaljon erőforrást
  - a fizikai memóriát meghaladó igények kiszolgálása
  - szeparáció és együttműködés
  - **alacsony rezsiköltség**

Címleképezés és lapszervezés (szga)

- MMU (Memory Management Unit)
  - virtuális és fizikai címek összerendelése



- Virtuális és fizikai címek
  - a taszkok a CPU teljes címtartományát látják
    - ez a **virtuális címtartomány**

      pl. x86-64 esetében 2^{48} byte = 256 terabyte
  - a fizikai memória a **fizikai címtartománnyal** érhető el

    jellemzően a ... gigabyte tartományban (néhány száz megabyte / gigabyte)
  - ami a fizikai memóriában nem fér el, azt a háttértáron tároljuk

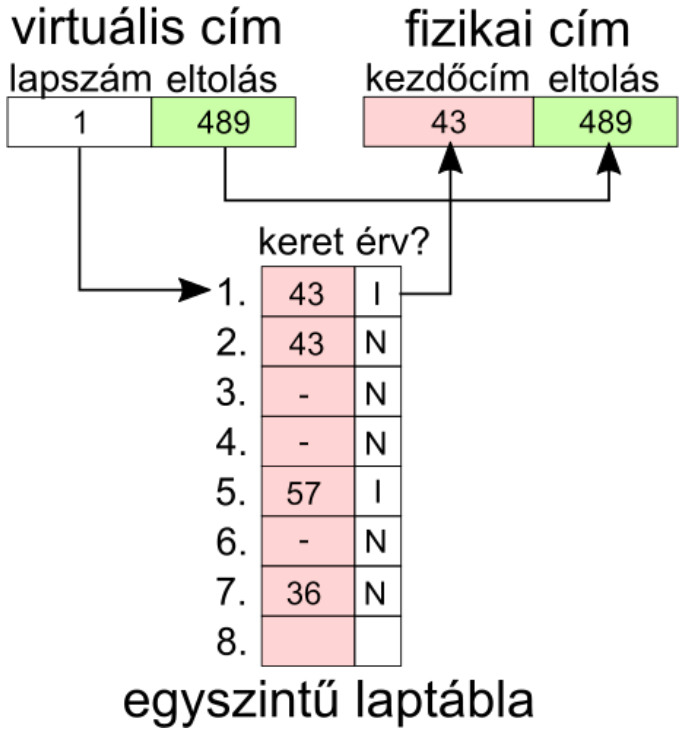
- Lapszervezésű virtuális memória-kezelés
  - virtuális címtartomány ← **lapok** (page)
  - fizikai memória ← **keretek** (frame)
  - háttértár ← **blokkok**
  - **laptábla**: lapok ←→ keretek
  - Translation Lookaside Buffer (TLB): címfordító gyorsítótár

    (van szegmens+lapszervezésű is, pl. x86 valós mód)

A címleképezés és a laptábla

- A címleképezés lépései
  - a virtuális cím kettébontása
    - lapszám index
    - eltolás
  - index → fizikai keret
  - fizikai cím előállítása
    - fizikai keret kezdőcím
    - eltolás

  lásd szga jegyzet és x86 példa



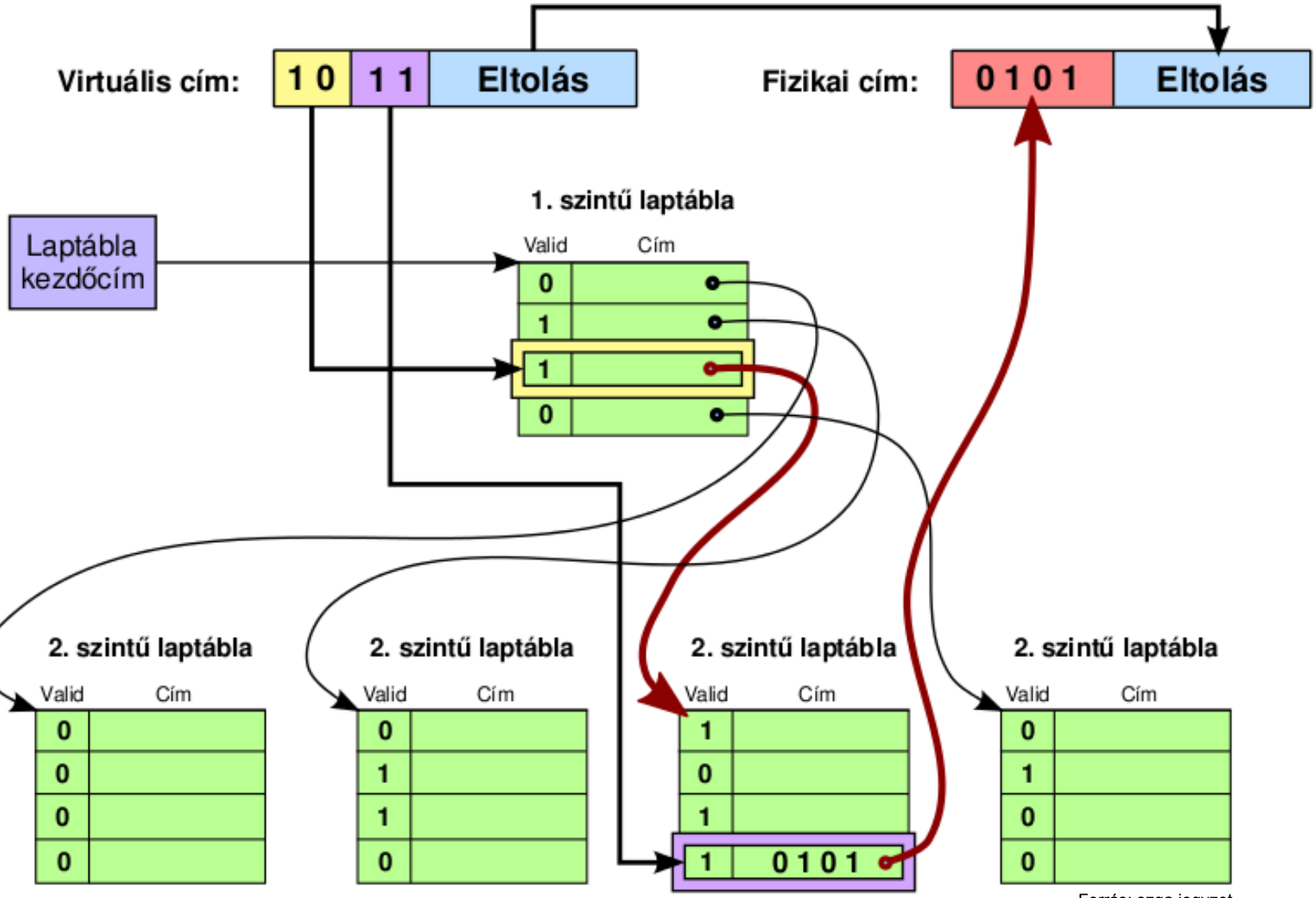
- Címtér-elkülönítés (szeparáció)
  - taszkonkénti laptábla

    a kontextus része
  - futó taszk esetén

    az MMU támaszkodik a tartalmára

Előfordulhat-e a rendszerben két egyforma virtuális cím különböző tartalommal?
Létezhet-e két egyforma fizikai cím különböző tartalommal?
Mi a helyzet a túl nagy méretű laptáblákkal?

Többszintű laptáblák (szga)



Forrás: szga jegyzet

Cserehely (swap) avagy lapozófájl (page file)

- Ami nem fér a központi memóriába...
  vagy felesleges ott tárolni...

- Nagy kapacitású, de LASSÚ tároló
  - részekre (blokkokra) bontott
  - a lapokat a blokkokban helyezzük el
  - **a CPU közvetlen módon nem fér hozzá**

- A cserehely használata
  - amikor a cserehelyen tárolt adatokra van szükség
    - be kell tölteni azokat a fizikai memóriába
  - amikor a fizikai memóriában szabad helyre van szükség
    - gondoskodni kell a memóriában tárolt adatok cserehelyre mentéséről
  - mindezek a memóriakezelés feladatkörébe tartoznak

- Demo: cserehely létrehozása

Tárcsere (swapping)

- Taszkok teljes memóriatartományának háttértárra írása
  - a lapozás előtti időkben fejlesztették ki
- A memória és a cserehely egyre nagyobb **töredezettségét** okozta
  - változó méretű „lyukak” jelennek meg
  - nehéz jól kitölteni a szabad helyeket
- A töredezettség csökkentése
  - kezelés:
    taszkok áthelyezése (töredezettség-mentesítés)
  - megelőzés:
    ügyesebb elhelyezési algoritmusok
- A lapozás is megoldja a töredezettség problémáját.
- A teljes tárcsere a lapozás mellett is működhet
  - túlterhelés esetén a felfüggesztett taszkok összes lapját kiírhatjuk a cserehelyre

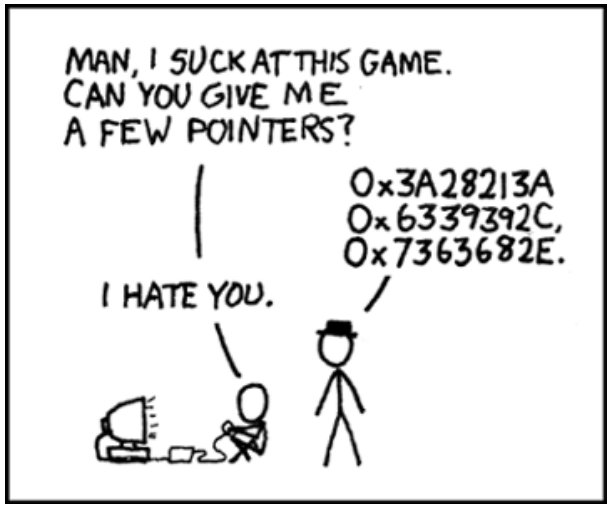
Memóriakezelés programozói szemmel (demó)

- Mit csinál? – C pointerkezelés

```
static int ns = 5;
printf("%d @ %p\n", ns, &ns);
int *pd = (int *) malloc(sizeof(int));
*pd = 10;
printf("%d @ %p\n", *pd, pd);
```

Kimenet:

```
5 @ 0x601048
10 @ 0x1430010
```



Forrás: xkcd

- Figyeljük meg (memprobe.c):
  - a memóriaszerkezetet
  - a statikus, lokális és globális változók elhelyezését
  - a dinamikusan allokációt
  - a memóriafoglalást allokáció előtt és után
  - a foglalás változását az adatok módosítása közben

- malloc() bomba

```
int Mb = 0;
while ( malloc(1<<20)) ++Mb;
```

Hogyan működik a virtuális tárkezelés?

- A taszkok memóriaterületét lapokra bontja
  - a használatban levő lapokat elhelyezi a memóriában és a háttértáron (cserehely)
  - beállítja az MMU-t a kialakított elrendezésnek megfelelően:
    - hardveres címleképezés és védelmi funkciók (taszkok szeparációja)
  - kezeli az MMU által generált megszakításokat

- A taszkok futása alatt
  - a TLB és az MMU végzi a címfordítást
  - a hardver betartatja a védelmi korlátokat
  - az MMU megszakításokat generál, amennyiben hibát észlel

- A hardver által generált megszakítások kezelése
  - **védelmi hiba**
    hibás címzés (érvénytelen cím, hozzáférési hiba)
  - **laphiba**
    a hivatkozott lap nincs a fizikai memóriában

Emlékeztető: a modern OS eseményvezérelt

Laphiba kezelése: szoftveres címleképezés

- A lap nincs a memóriában → laphiba (megszakítás)
  - a laptábla megfelelő bejegyzése nem érvényes jelzésű (valid bit = 0)

- Elindul a kernel megszakításkezelője
  - észleli a laphibát → aktiválja a memóriakezelőt (lap behozása)
  - létezik a lap a cserehelyen?
    - betölti egy szabad keretbe
  - igény szerint kitöltendő? (fill-on-demand: zero-fill, fill-from-text)
    - kitölt egy szabad keretet
  - a taszk laptáblájában beállítja az új lap-keret összerendelést (valid = 1)
  - frissíti a laptáblát az MMU számára
  - visszatér a megszakításból
  - az CPU újra végrehajtja a műveletet, ezúttal sikeresen

- Gondok?
  - Van szabad keret?
    Ha nincs, fel kell szabadítani egyet. → **lapcsere**
  - A lap betöltése a diszkről lassú
    Célszerű addig más taszkot futtatni.

A memóriakezelő további feladatai

- Szabad kereteket biztosítása
  - célszerű nem laphiba alatt foglalkozni vele
- Lapok kiírása a háttértárra
  - a nem használt lapokat célszerű a háttértárra írni, és
  - az általuk foglalt kereteket felszabadítani
- Nyilvántartás
  - taszkok lapjai → **laptábla** (page table)
  - keretek → **kerettábla** (page frame data)
  - cserehely → **diszk blokk leíró** (disk block descriptor)
    swap térkép (swap map)
- További feladatok
  - az MMU is módosíthatja a laptáblát (pl. hivatkozásszámláló)
    - a kernel kiolvassa és tárolja a módosításokat
  - szükség esetén tárcsere
    - túlterhelés esetén teljes taszkok kiírása

A virtuális memóriakezelés adatstruktúrái

- **kerettábla** (pfdata: page frame data) (kernel kontextus)
  - a keret sorszámával indexelt
  - **állapot** (szabad, foglalt), módosult (dirty), DMA alatt áll stb.
  - **hivatkozásszámláló** (acc): hány taszk használja a keretet

- **laptábla** (page table) (taszk kontextus)
  - lap sorszám
  - keret sorszám
  - jelzőbitek:
    „valid”, „dirty” (módosult), „accessed” (használt), „read-only”
  - állapot: memóriában / háttértáron / igény szerint kitöltendő
  - a taszk azonosítója
  - másolás-írás-esetén (COW) jelzőbit,
  - jogosultságok stb.

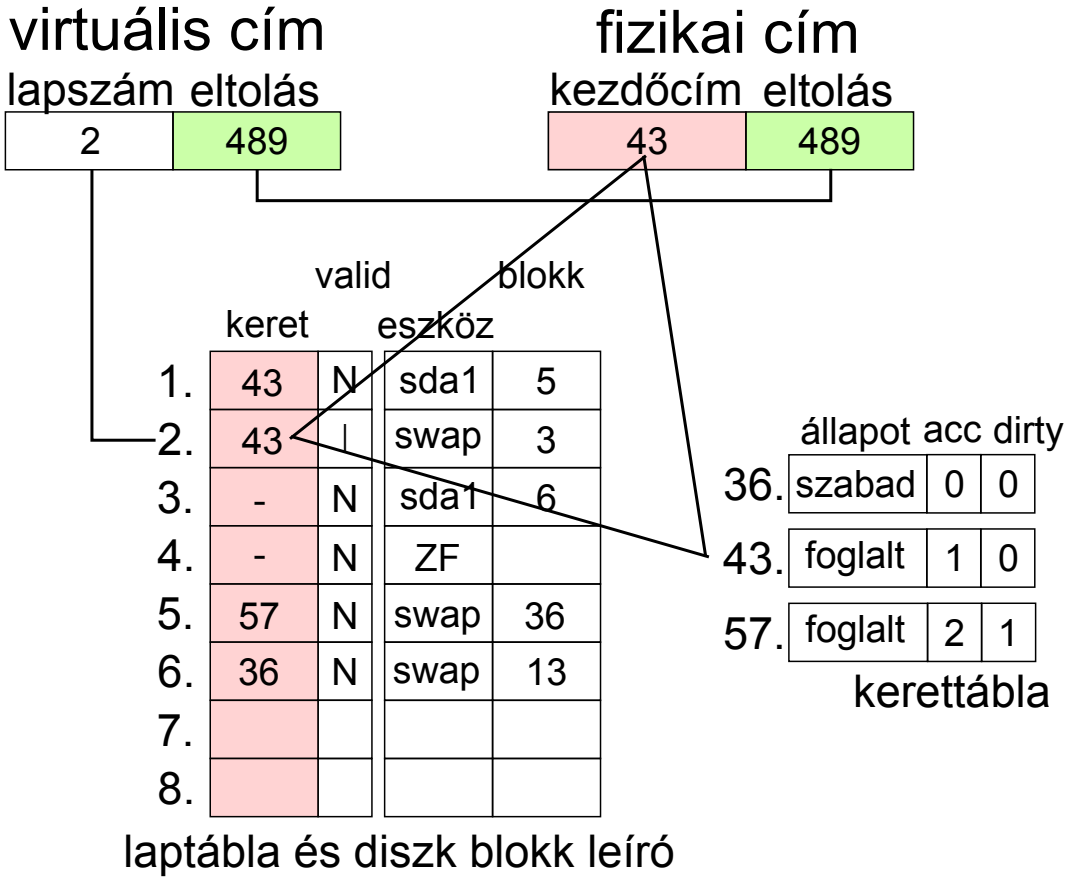
MMU-ba kerül

Hardverfüggő !

csak kernel

- **diszk blokk leíró** (kernel kontextus)
  - háttértár eszközazonosító
  - blokk sorszám
  - típus: swap (a háttértáron van), zero-fill, fill-from-text stb. (OS-függő)

Példa az adatszerkezetekre



Értelmezzük a 2., 4. és 6. lap adatait!
Mi történik, amikor a 6. lapra hivatkozik a program?
Hol található az 57. keret lapja?

Teljesítménynövelő technikák: fill-on-demand

- Mi történik a `malloc()` kiadásakor?
  - a memória tartalma nem definiált, ezért nem foglal neki keretet vagy cserehelyet
  - a laptábla megfelelő elemei fill-on-demand (pl. zero-fill) bejegyzést kap(hat)nak
- Amikor először hivatkoznak az allokált lapra
  - az MMU megszakítást generál
  - elindul a szoftveres címleképezés
    - allokál és kitölt egy keretet ← **csak ekkor történik memóriafoglalás!**
- Eredmény?
  - allokáció során csak egy laptábla bejegyzést kell kitölteni (gyors)
  - csak akkor foglal tényleges memóriát, ha azt használják
  - jól kezeli a lefoglalt memória „szellősségét”
- hasonló módon működik a programkód betöltése is
  - fill-on-demand: fill-from-text
  - a laptáblában a kódot tartalmazó diszkblokkokat állítja be

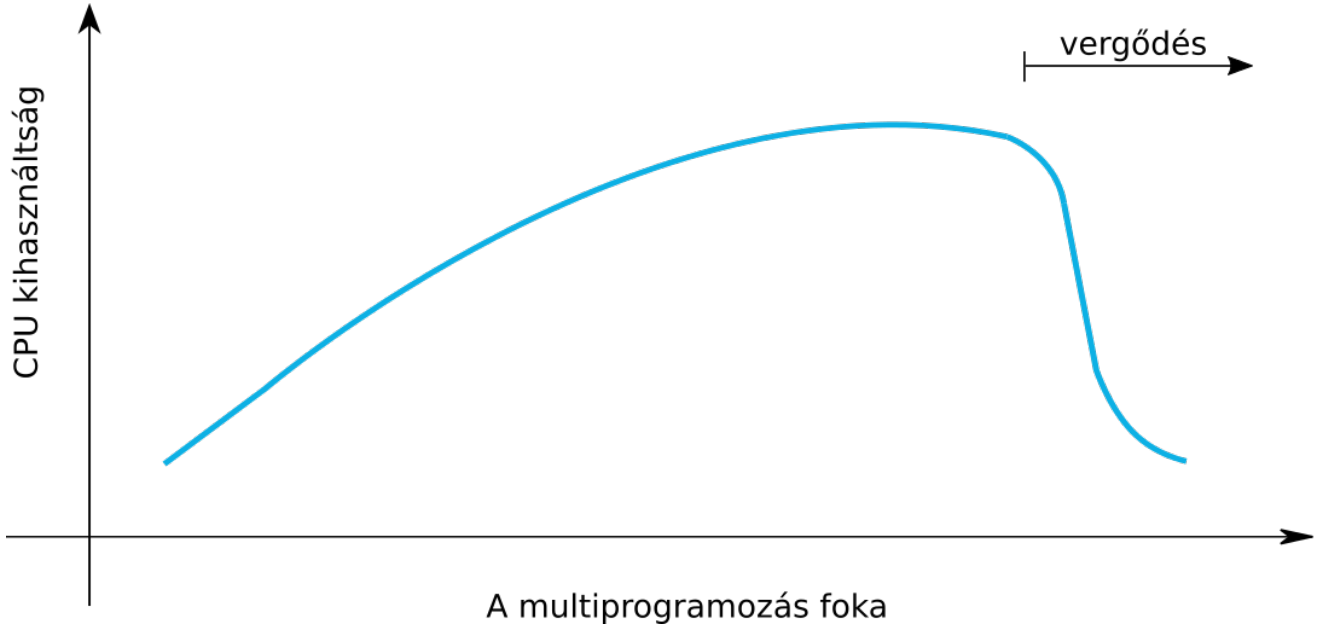
Teljesítménynövelő technikák: COW

- Emlékeztető: a fork() működése
  - több taszk ugyanazt a programkódot futtatja
- A lapok taszkok közötti megosztása
  - egy keret – több lap (több taszk)
  - olvasás: nem gond, írás: probléma
- A fork() és a copy-on-write (COW) technika
  - duplikálja a laptáblát (kereteket nem)
  - növeli a hivatkozásszámlálókat
  - beállítja a read-only (RO) és a copy-on-write (COW) jelzőbitet
  - írás esetén
    - a read-only bit miatt megszakítás
    - a kernel megszakításkezelője látja a COW bitet
    - duplikálja a lapot ← **csak itt allokál új memóriát**
    - törli a lapok RO és a COW bitjeit
    - visszatér a megszakításból
    - az MMU megismétli az írás műveletet

Mely lapok legyenek a fizikai memóriában?

- Mennyi keretet rendeljünk egy taszkhoz?
  - túl sok – jó neki, de másoknál lesz laphiba
  - túl kevés – sok taszk futhat, de mindenkinél laphibák lesznek

- A magas **laphiba-gyakoriság** (page fault frequency, PFF) hátrányos



Laphibák

- A magas **laphiba gyakoriság** (page fault frequency, PFF) hátrányos
  - lassítja a taszkok működését
    - futása megszakad, újra kell ütemezni
    - emlékeztető: **memória-intenzív taszk → I/O-intenzív**
  - nő a rezsiköltség és a terhelés → újabb laphiba keletkezhet

  Vergődés (trashing): gyakori laphibák miatt a teljesítmény jelentősen romlik
  A taszkok számának kordában tartásával (középtávú ütemezéssel) kezelhető.

- Jobb, ha nem kezeljük, hanem elkerüljük
  - a lapok kiírása és behozása során legyünk körültekintőek

  Hogyan?
    - lapok behozása során → jó **lapozási stratégiával**
    - keretek felszabadítása során → megfelelő **lapcsere algoritmussal**

Lapozási stratégiák

- Mely lapokat töltsük be a fizikai memóriába?
- Ideális algoritmus: amelyikre szükség lesz
  - ha a jövőbe látna az OS, akkor pontosan tudná
- A valóságban...
  - megpróbálhat jósolni:

    előretekintő lapozás (anticipatory paging)

  - inkább csak a jelenlegi igényekkel foglalkozik:

    igény szerinti lapozás (demand paging)

Igény szerinti lapozás (demand paging)

- Működés
  - csak laphiba esetén fut
  - csak a szükséges lapot hozza be
- Értékelés
  - egyszerű
  - korábban nem használt lapokra való hivatkozás **mindig** laphibát generál
  - ez lassítja a taszk futását
- Példa: műveletek nagy adatstruktúrán
  - laponként laphiba (gyakori)
  - laphiba → I/O-ra vár a taszk → átütemezik
  - CPU-intenzív helyett I/O-intenzív
  - sok megszakítás, kontextusváltás
  - nő a rezsiköltség

Előretekintő lapozás (anticipatory paging)

- Működés
  - lapcsere során több lapot hoz be
  - „előre dolgozik”
  - megpróbálja kitalálni, mely lapokra lesz szükség:
    - lokalitási jellemzők
    - laphibák a múltból

- Értékelés
  - jó becslés → kevesebb laphiba
    - korábban nem hivatkozott lapok is a fizikai memóriában vannak
    - kevesebb megszakítás, I/O és átütemezés
  - rossz jóslás → több laphiba
    - nem a megfelelő lapokat töltötte be, nem csökken a laphibák száma
    - sok felesleges lapot tart a fizikai memóriában
    - kevesebb a szabad keret

Keretek felszabadítása

- Feladat
  - melyik keretet?
  - mikor? hogyan?

- Működés (lapcsere)
  - laphiba-megszakítás
  - felszabadítandó keret kiválasztása Hogyan? **Lapcsere algoritmus**
  - a keret tartalmának mentése a cserehelyre (ha szükséges)
  - a kerethez tartozó laptábla-bejegyzés módosítása és a keret felszabadítása

6.	36	N	swap	13

36.	szabad	0	0

  - a keret új tartalmának betöltése a cserehelyről (vagy előállítása)

36.	foglalt	1	0

  - új laptábla-bejegyzés készítése

9.	36	I	swap	43

  - MMU beállítása
  - visszatérés a megszakításból

Lapcsere algoritmusok

- **Melyik keret tartalmát írjuk ki a cserehelyre?**

- Ideális megoldás: amelyik lapra legkésőbb lesz szükség
  - ismét a jövőbelátás...

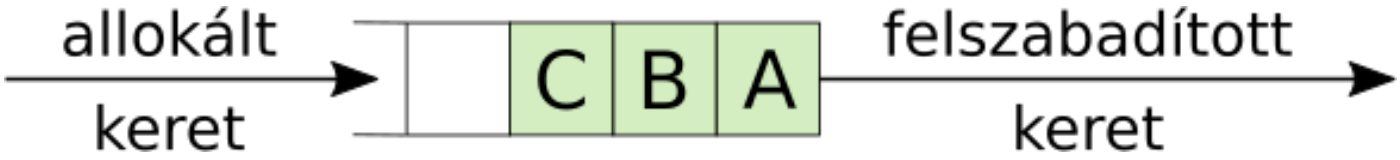
- A valóságban...

  … valamilyen módon közelítjük a jövőbelátást:
  - FIFO: amelyiket legrégebben hoztuk be
  - Újabb esély (SC): legrégebben behozott és nem hivatkozott lap
  - Legrégebben nem használt (LRU)
  - Legkevésbé használt (LFU)
  - Utóbbi időben nem használt (NRU): nem hivatkozott és nem módosított lap

Mire támaszkodhat egy lapcsere algoritmus?

- mikor allokáltak a laphoz keretet
  - milyen sorrendben
- hivatkozás jelzőbit:
  - használták-e az adott lapot „mostanában”
- mikor használták
  - milyen sorrendben
- módosított jelzőbit
  - módosult-e a keret tartalma
  - ha igen, hosszabb lehet a felszabadítása
- a választás jellege
  - az aktuális taszk címteréből: **lokális**
  - az összes lap közül: **globális**

A FIFO lapcsere



Feladatmegoldás: FIFO lapcsere 3 kerettel

Lapkérés:	1	2	3	4	1	2	5	1	2	3	4	5
Keretek A	1			4			5					5
Keretek B		2			1			1		3		
Keretek C			3			2			2		4	
Foglalás:	A1	B2	C3	A4	B1	C2	A5	-	-	B3	C4	-
FIFO	A	A	A	B	C	A	B	B	B	C	A	A
		B	B	C	A	B	C	C	C	A	B	B
			C	A	B	C	A	A	A	B	C	C
Ütem:	1	2	3	4	5	6	7	8	9	10	11	12

Feladatmegoldás: FIFO lapcsere 4 kerettel

Lapkérés:	1	2	3	4	1	2	5	1	2	3	4	5
Keretek A	1				1		5				4	
Keretek B		2				2		1				5
Keretek C			3						2			
Keretek D				4						3		
Foglalás:	A1	B2	C3	D4	-	-	A5	B1	C2	D3	A4	B5
FIFO	A	A	A	A	A	A	B	C	D	A	B	C
		B	B	B	B	B	C	D	A	B	C	D
			C	C	C	C	D	A	B	C	D	A
				D	D	D	A	B	C	D	A	B
Ütem	1	2	3	4	5	6	7	8	9	10	11	12

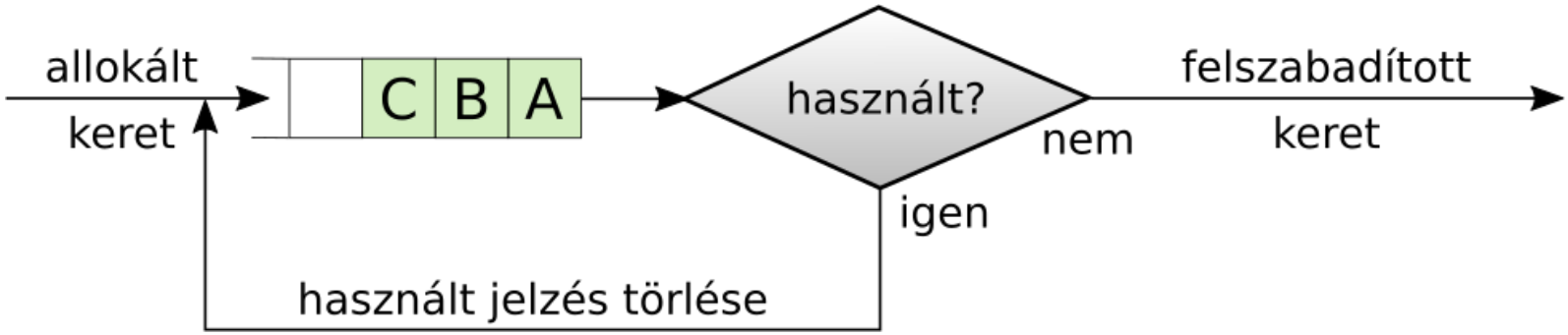
A FIFO lapcsere értékelése

- Tulajdonságai
  - egyszerű algoritmus és adatstruktúra (FIFO)
  - előrenéző (nem érdekli a múlt)
- Komplexitás
  - O(1)
- Rezsiköltség
  - minimális
- Előnyök, problémák
  - könnyű megvalósítani
  - a lapok jövőbeli használatát (optimális algoritmus) gyengén becsli
  - nem figyeli a módosítást (a kiírás sokáig tarthat)

  Mi történik, ha növeljük a rendelkezésre álló keretek számát?
  - csökken a laphibák száma
  - **időnként** nem, sőt, nő!!

  Bélády-féle anomália (Bélády László, IBM)

Az újabb esély (second chance, SC) lapcsere

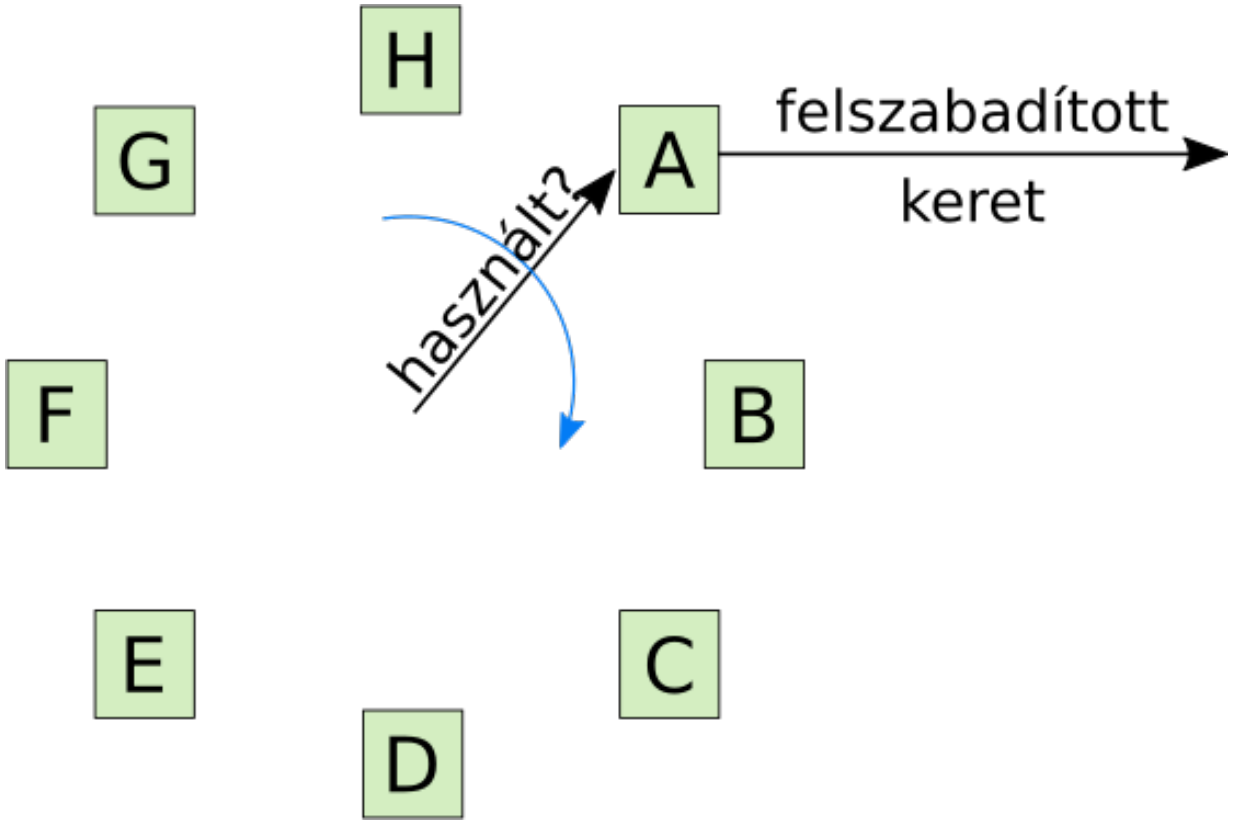


A „használt” avagy „hivatkozott” (referenced) jelzést az MMU állítja be.

Az „újabb esély” lapcsere értékelése

- Tulajdonságai
  - egyszerű algoritmus és adatstruktúra (FIFO)
  - hátranéző
  - a használt lapokat nem szabadítja fel
- Komplexitás
  - O(1)
- Rezsiköltség
  - minimális
- Előnyök, problémák
  - könnyű megvalósítani
  - jobban becsüli a jövőbeli használatot a FIFO-nál
  - nem figyeli a módosítást (a kiírás sokáig tarthat)
  - állandóan mozgatja az adatokat a FIFO-ban

Van jobb adatstruktúránk az „újabb esély” számára?



Óra (clock) lapcsere

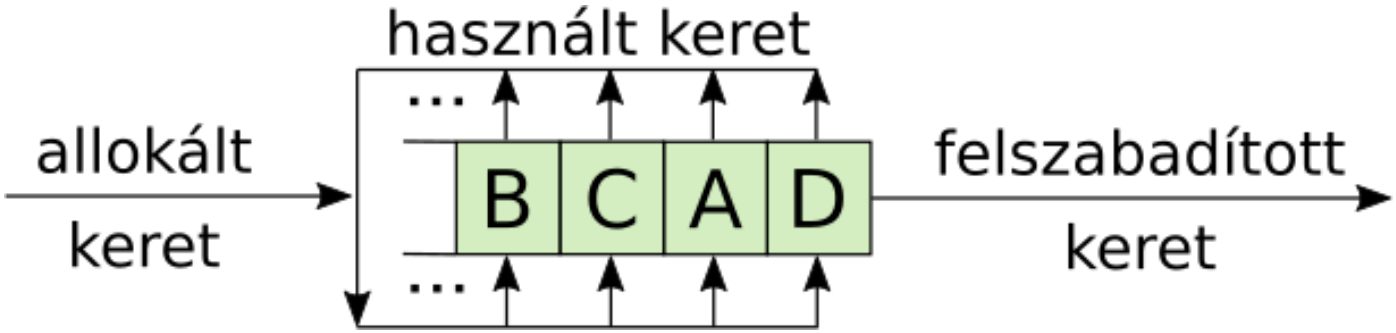
Az óra lapcsere-algoritmus értékelése

- Tulajdonságai
  - nem túl bonyolult algoritmus és adatstruktúra (lista + óramutató)
  - hátranéző
  - a használt lapokat nem szabadítja fel
- Komplexitás
  - O(1)
- Rezsiköltség
  - minimális
- Előnyök, problémák
  - ugyanolyan jó, mint az újabb esély
    - könnyű megvalósítani
    - jobban becsüli a jövőbeli használatot a FIFO-nál
    - nem figyeli a módosítást (a kiírás sokáig tarthat)
  - de megspórolja az adatok mozgatását a FIFO-ban

Lehet-e még jobban jósolni?

- Az eddigi algoritmusok...
  - a FIFO nem nagyon foglalkozik a múlttal (minél „öregebb”, annál jobb)
  - az SC és Clock a múltat 1 bitbe zsúfolták...
- Nincs több információnk a keretekről?
- Van...
  - módosítás
  - hányszor használták
  - mikor használták
  - hány taszk használta

Legrégebben nem használt (least recently used, LRU)



A lapokat a használati gyakoriságuk szerint rendezi sorba.

többféle megvalósítás, amelyek ötvözik a

- a használati idő szerinti
- hivatkozások száma szerinti

rendezést

hardverfüggő

Az LRU értékelése

- Tulajdonságai
  - bonyolultabb algoritmus és adatstruktúra (láncolt lista)
  - hátranéző
  - a használt lapokat nem szabadítja fel
- Komplexitás
  - jellemzően O(N)
- Rezsiköltség
  - hardvertámogatással kicsi, anélkül nagy
- Előnyök, problémák
  - a legjobban becsli a lapok jövőbeli használatát (az optimális algoritmust)
    - nyilvántartja a lapok múltbeli használatát, ami egy jó becslő
  - **a frissen behozott lapot nagy eséllyel lecseréli (miért?), ami nem jó**
  - csak hardvertámogatással érdemes megvalósítani
  - nem figyeli a módosítást (a kiírás sokáig tarthat)

A lapok tárba fagyasztása (page locking)

- Frissen behozott lapoknak nincs múltja
  - a jövőjük nem becsülhető
  - könnyen kiírásra választhatja őket a lapcsere
- I/O művelet alatt álló lapot ne szabadítsunk fel
  - az I/O műveletek fizikai címeket használnak
  - akár a CPU-t megkerülve, DMA vezérlő segítségével is módosíthatják a memóriát
  - a lapcserének erre is figyelnie kell
- Megoldás: **lapok tárba fagyasztása**
  - page lock bit jelzi a zárolt (fagyasztott) állapotot
  - az ilyen lapok nem lehetnek a lapcsere „áldozatai”
  - I/O művelet esetében annak végéig tart a zárolás
  - az első hivatkozás feloldja a zárolást

A laplopó taszk

- Sokféle név alatt létezik:
  - Page daemon, kswapd, Working Set Manager
- Feladata: üres keretek biztosítása
  - rendszeres időközönként felébred vagy a kernel felébreszti
  - a szabad keretek számát igyekszik két határérték között tartani
    - ha egy minimum szint alá esik, elkezd kereteket „lopni”
    - a maximum szint elérésekor alvó állapotba lép
- A laplopó végezheti az összes lapcserét
  - ha nem volt elég „ügyes”, és mégis elfogytak a szabad memóriakeretek
  - kiválaszt egy lapot és levezényli a lapcsere folyamatát
- Egyéb feladatai lehetnek:
  - `referenced` bit törlése
  - használati számlálók öregítése

Legkevésbé használt (least frequently used, LFU)

- Not frequently used (NFU) néven is ismert.

- Az LRU egyszerűsített változata (közelítése)
  - az OS időnként növel egy használati számlálók a referenced = 1 lapokra
  - a számláló alapján választja ki a felszabadítandó keretet

- Az algoritmus értékelése
  - közepes rezsiköltségű, periodikusan igényli a számlálók növelését
  - a laplopó taszkkal jól kombinálható
  - viszonylag jól becsli a lapok jövőbeli használatát (az optimális algoritmust)
    - közelítőleg nyilvántartja a lapok múltbeli használatát, ami egy jó becslő
  - a frissen behozott lapot nagy eséllyel lecseréli, ami nem jó
  - a számláló túlcsordulhat (a probléma **öregítéssel** kezelhető)
  - nem tud különbséget tenni a módosított és változatlan lapok között, ezért a lapcsere diszk művelettel is járhat, ami jelentősen lassítja a működést

Mostanában nem használt (not recently used, NRU)

- A második esély (SC) algoritmus finomított változata
  - a `referenced` jelzőbit mellett a `dirty` (módosult) bitet is figyeli
  - a két bit segítségével egy „prioritást” rendel a lapokhoz:
    `ref=0 dirty=0` → a prioritás 0, nem hivatkozott, nem módosított
    `ref=0 dirty=1` → a prioritás 1, nem hivatkozott, módosított
    `ref=1 dirty=0` → a prioritás 2, hivatkozott, nem módosított
    `ref=1 dirty=1` → a prioritás 3, hivatkozott, módosított

- Amikor egy szabad keretre van szükség
  - véletlenszerűen választ a legkisebb prioritású lapok közül

- Az algoritmus értékelése
  - kis rezsiköltségű, jól kihasználja a hardvertámogatást
  - a SC-nél jobban becsli a lapok jövőbeli használatát (az optimális algoritmust)
    - a hivatkozások mellett a módosításokat is figyeli
  - különbséget tesz a módosított és változatlan lapok között

Érdekességek

- Miért történik hiba, ha a `0x00000000` címre hivatkozik egy program?
  - tipikus hiba („invalid / NULL pointer”) inicializálatlan mutatóval
  - miért okoz hibát?
  - kapcsolódó: miért nem kezdődik a `0x00000000` címen a programkód?

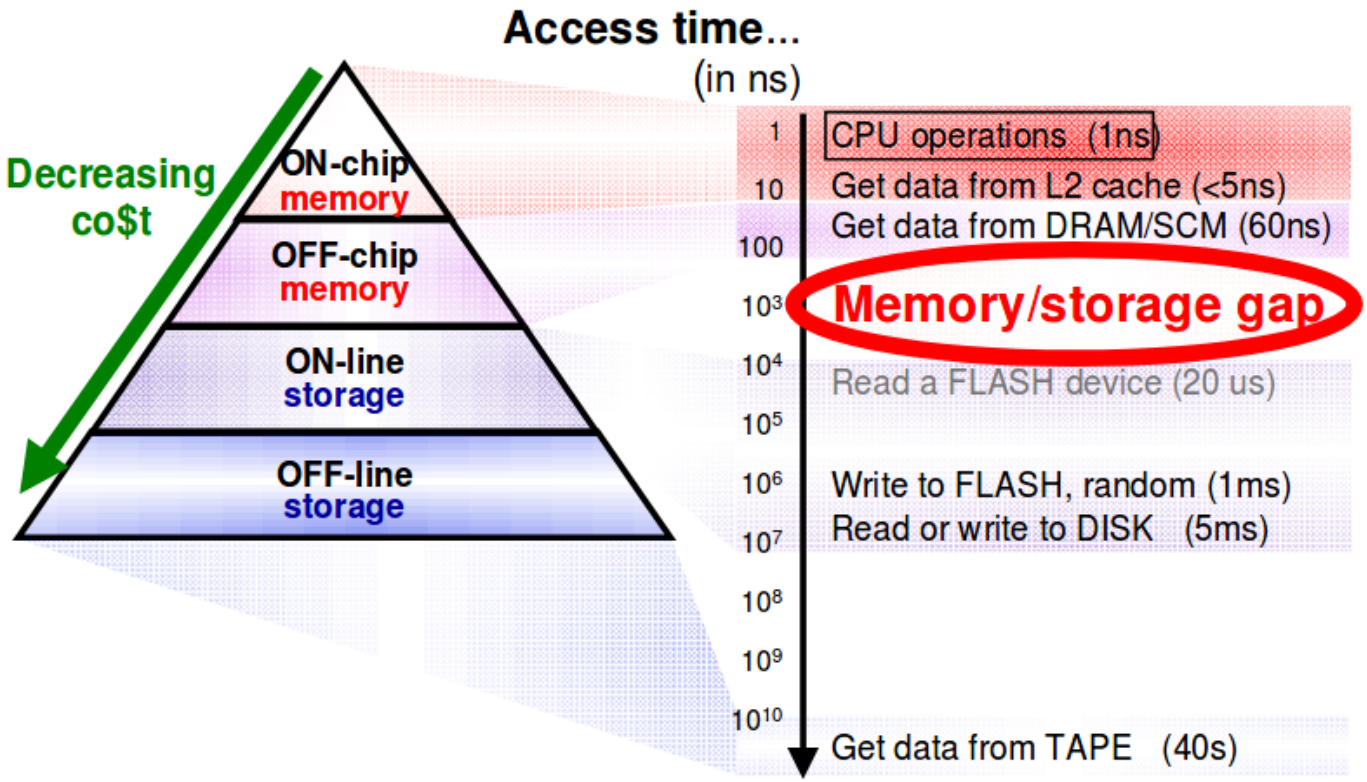
    Tipp

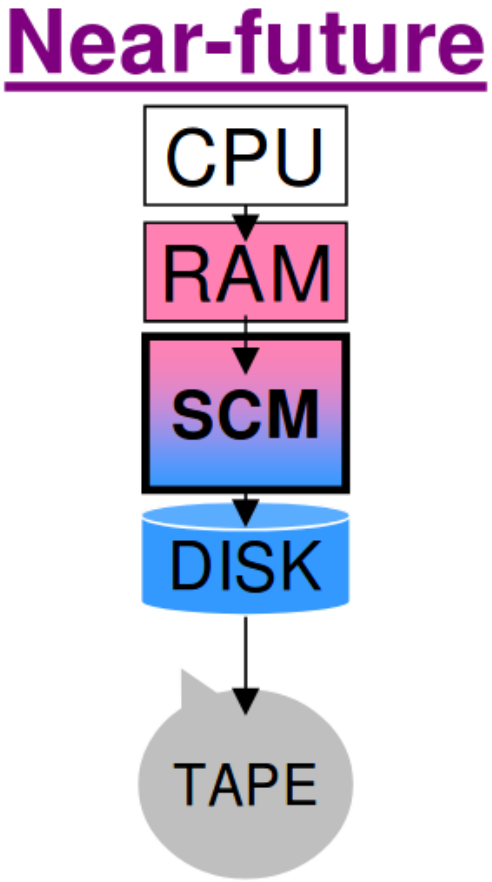
- Hogyan akadályozható meg, hogy a verem túl nagyra nőjön?
  - nem érdemes statikusan lefoglalni a teljes lehetséges méretét
  - ha viszont dinamikusan nőhet, hogyan állítható meg a növekedése?
  - akár bajt is okozhat: lásd Stack Clash bug (helyi root jogot ad)

    Tipp

- Storage Class Memory (SCM)
  - **nem felejtő**, gyors (az I/O már nem lassú), „olcsó” / GB, nagy (TB)
  - koncepcióváltás: számítás-orientált → adat-centrikus (lásd gépi tanulás)
  - többprocesszoros, heterogén és elosztott rendszerek közös adattárolással
  - IBM, Gen-Z, ACM

    (lásd következő fóliák)

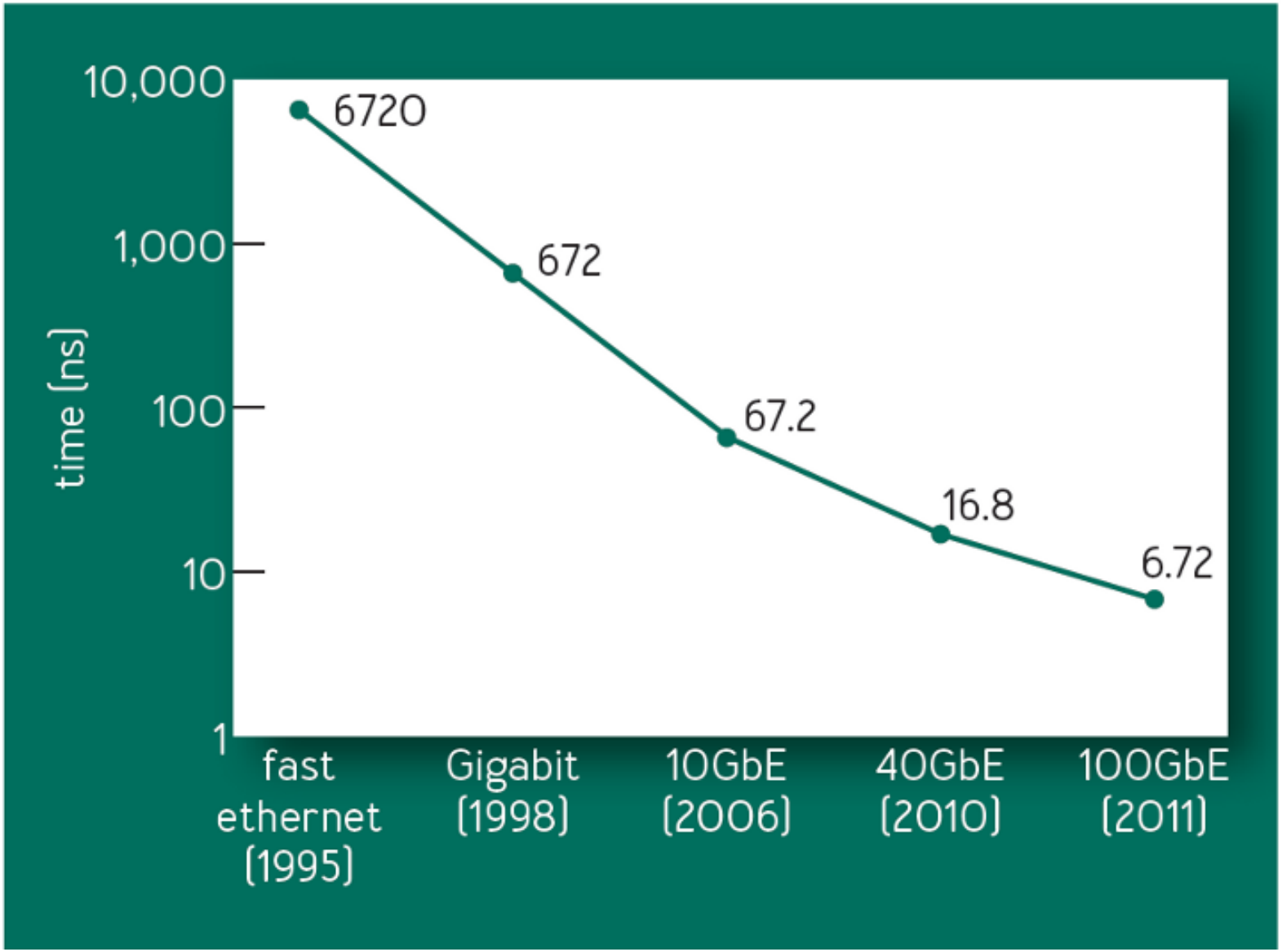




Research into new solid-state non-volatile memory candidates
– originally motivated by finding a "successor" for NAND Flash –
has opened up several interesting ways to change the memory/storage hierarchy...

Forrás: IBM

FIGURE 1: **PER-PACKET PROCESSING TIME WITH FASTER NETWORK ADAPTERS**



Forrás: ACM

Összefoglalás

- Absztrakt virtuális gép + lapkezelés
  - a taszkok egy lapokra bontott, virtuális címtartományt használnak
  - a kernel kereteket rendel a lapokhoz (címtábla) – **erőforrás-allokátor**
- A memóriakezelés
  - alapvetően az MMU végzi a logikai – fizikai címleképezést
  - ha egy lapot nem ér el (**laphiba**) a kernel szoftveres címleképezése segíti
  - a háttértárral bővíti a fizikai memóriát
  - gondoskodik a taszkok szeparációjáról és védelméről
- Lapok betöltése:
  - igény szerint jellemző, korlátozottan előretekintő módon
- Szabad keretek biztosítása (laplopó taszk)
  - lapcsere algoritmusok
  - az ideális megoldás nem érhető el, csak közelíthető