Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

Miről beszéltünk korábban…

- Operációs rendszerek kialakulása
- Op. Rendszer fogalmak, struktúrák
- Fájlok, könyvtárak, fájlrendszerek
- Folyamatok
  - Folyamatok kommunikációja
  - Kritikus szekciók, szemaforok.
- Klasszikus IPC problémák
- Ütemezés
- I/O, holtpont probléma

Mi következik ma…

- Memória gazdálkodás
  - Alapvető memória kezelés
  - Csere
  - Virtuális memória
  - Lapcserélési algoritmusok
  - Lapozásos rendszerek tervezése
  - Szegmentálás

Memória kezelő

- Memória típusok
- Operációs rendszer része
  - Kernelben
- Feladata:
  - Memória nyilvántartása, melyek szabadok, foglaltak
  - Memóriát foglaljon folyamatok számára.
  - Memóriát felszabadítson.
  - Csere vezérlése a RAM és a (Merev)Lemez között

Alapvető memória kezelés

- Kétféle algoritmus csoport:
  - Szükséges a folyamatok mozgatása, cseréje a memória és a lemez között. (swap)
  - Nincs erre szükség
- Ha elegendő memória van, akkor nem kell csere (swap)!
- Van elegendő memória?
  - HT1080Z – 16 KB, C64 – 64 KB, IBM PC-640 KB
  - Ma: PC 2-4-8 GB, kis szerver 4-16 GB

Monoprogramozás

- Egyszerre egy program fut.
  - Nincs szükség „algoritmusra”. Parancs begépelése, annak végrehajtása, majd várjuk a következőt.
  - Tipikus helyzetek
    - Op.rendszer az alsó címeken, program felette (ma ritkán használt, régen nagygépes, minigépes környezetben használták.
    - Alsó címeken a program, a felső memória területen a ROM-ban az operációs rendszer (kézi számítógép, beágyazott rendszer)
    - Op.rendszer az alsó címeken, majd felhasználói program, felette a ROM-ban eszközmeghajtók. (MS-DOS, ROM-BIOS)

Multiprogramozás(Multitask)

- „Párhuzamosan” több program fut. A memóriát valahogy meg kell osztani a folyamatok között.
  1. Multiprogramozás megvalósítása rögzített memória szeletekkel.
  2. Multiprogramozás megvalósítása memória csere használattal.
  3. Multiprogramozás megvalósítása virtuális memória használatával.
  4. Multiprogramozás szegmentálással.

1. Multiprogramozás rögzített memória szeletekkel

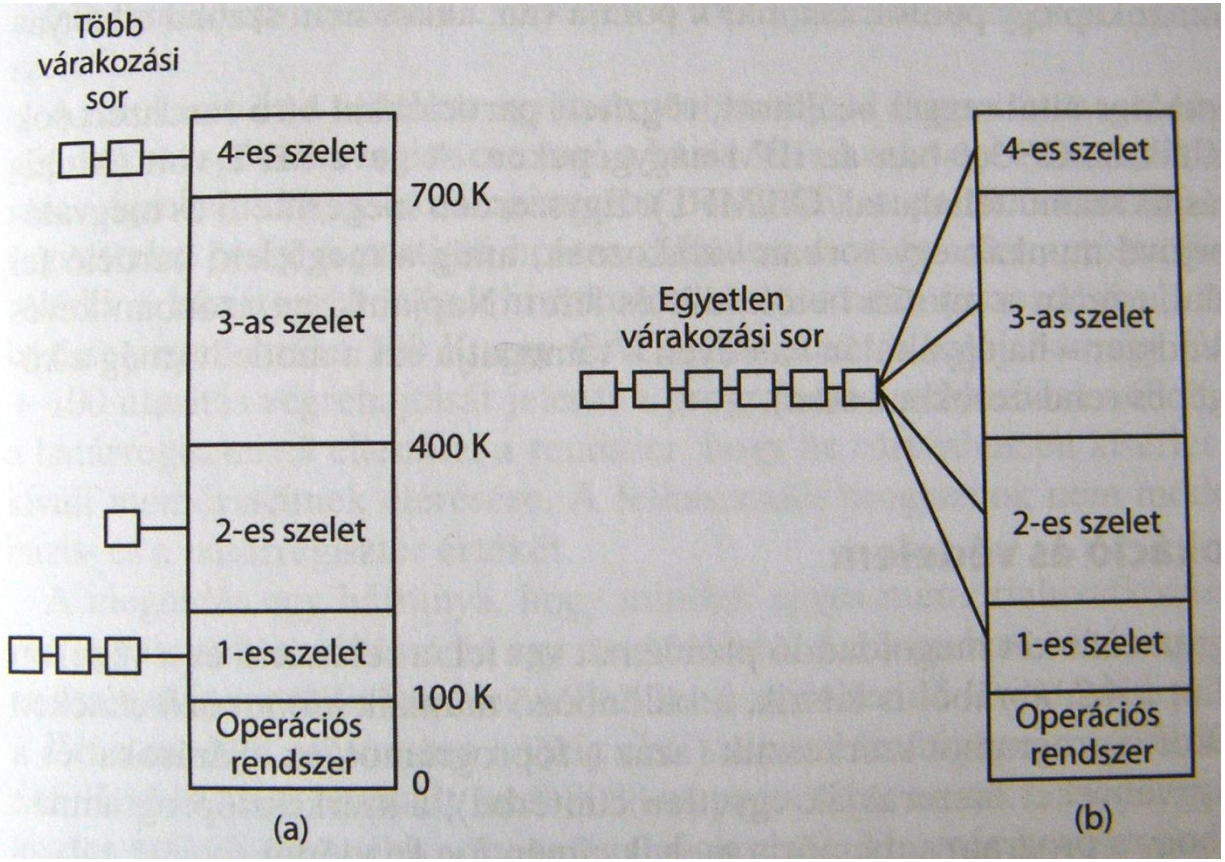
- Monoprogramozás ma jellemzően beágyazott rendszerekben van jelen (PIC)
- Ma tipikusan preemptív időosztásos rendszereken több folyamat van a memóriába „végrehajtás alatt”.
- Osszuk fel a memóriát n (nem egyenlő) szeletre. (Fix szeletek)
  - Pl. rendszerindításnál ez megtehető
  - Egy közös várakozási sor
  - Minden szeletre külön-külön várakozási sor.
  - Kötegelt rendszerek tipikus megoldása.

Memória felosztása

Több várakozási sor:
• Kihasználatlan partíciók

Egy sor:
•Ha egy partíció kiürül, a sorban első beleférő bekerül.

IBM használta az OS/360 rendszeren: OS/MFT (Multiprogram Fix Task)



Relokáció - Védelem

- Nem tudjuk hova kerül egy folyamat, így a memória hivatkozások nem fordíthatók fix értékekre! (Relokáció)
  - OS/MFT program betöltéskor frissítette a „relokálandó” címeket.
- Nem kívánatos, ha egy program a másik memóriáját „éri el”! (Védelem)
  - OS/MFT PSW (program állapotszó, 4 bites védelmi kulcs)
- Másik megoldás: Bázis+határregiszter használata
  - Ezeket a programok nem módosíthatják.
  - Minden címhivatkozásnál ellenőrzés: lassú.

2. Multiprogramozás memória csere használattal

- A korábbi kötegelt rendszerek tipikus megoldása a rögzített memória szeletek használata (IBM OS/MFT)
- Időosztásos, grafikus felületek esetén ez nem az igazi.
- Teljes folyamat mozgatása memória-lemez között.
- Nincs rögzített memória partíció, mindegyik dinamikusan változik, ahogy az op. Rendszer oda-vissza rakosgatja a folyamatokat.
  - Dinamikus, jobb memória kihasználtságú lesz a rendszer, de a sok csere lyukakat hoz létre!
    - Memória tömörítést kell végezni! (Sok esetben (foci nézés) ez az időveszteség nem megengedhető!)

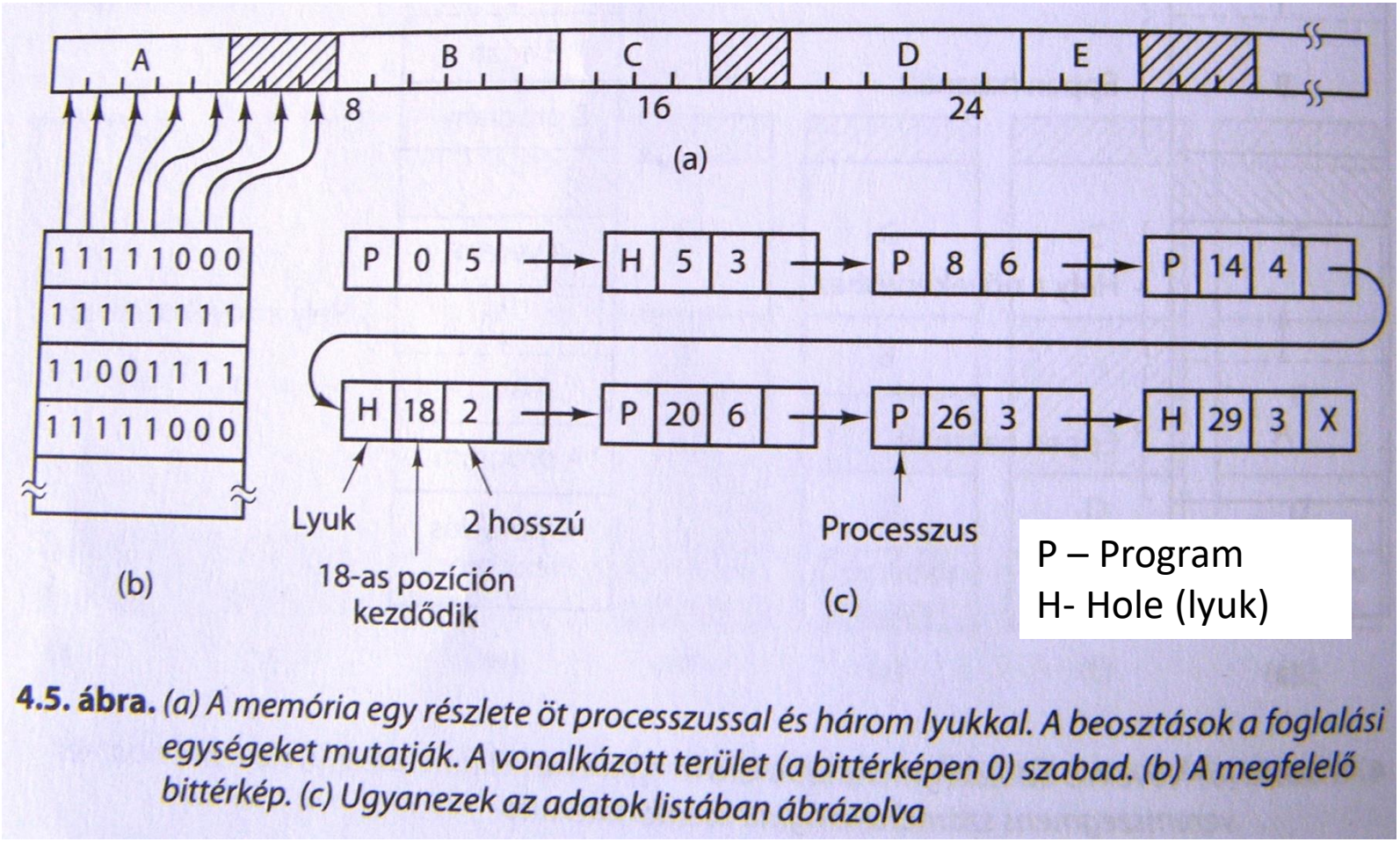
Dinamikus memória foglalás

- Általában nem ismert, hogy egy programnak mennyi dinamikus adatra, veremterületre van szüksége.
- A program „kód” része fix szeletet kap, míg az adat és verem része változót. Ezek tudnak nőni (csökkenni).
  - Ha elfogy a memória, akkor a folyamat leáll, vár a folytatásra, vagy kikerül a lemezre, hogy a többi még futó folyamat memóriához jusson.
    - Ha van a memóriában már várakozó folyamat, az is cserére kerülhet.
- Hogy tudjuk nyilvántartani a „dinamikus” memóriát?

Dinamikus memória nyilvántartása

- Allokációs egység definiálása.
  - Ennek mérete kérdés. Ha kicsi akkor kevésbé lyukasodik a memória, viszont nagy a nyilvántartási „erőforrás (memória) igény”.
  - Ha nagy az egység túl sok lesz az egységen belüli maradékokból adódó memória veszteség.
- A nyilvántartás megvalósítása:
  - Bittérkép használattal
  - Láncolt lista használattal
    - Ha egy folyamat befejeződik, akkor szükség lehet az egymás melletti memória lyukak egyesítésére.
    - Külön lista a lyukak és folyamatok listája.

Bittérkép, Láncolt lista megvalósítása



4.5. ábra. *(a) A memória egy részlete öt processzussal és három lyukkal. A beosztások a foglalási egységeket mutatják. A vonalkázott terület (a bittérképen 0) szabad. (b) A megfelelő bittérkép. (c) Ugyanezek az adatok listában ábrázolva*

Memória foglalási startégiák

- Új vagy swap partícióról behozott folyamat számára, több memória elhelyezési algoritmus ismert (hasonlóak a lemezhez) :
  - First Fit (első helyre, ahova befér, leggyorsabb, legegyszerűbb)
  - Next Fit (nem az elejéről, hanem az előző befejezési pontjából indul a keresés, kevésbé hatékony mint a first fit)
  - Best Fit (lassú, sok kis lyukat produkál)
  - Worst Fit (nem lesz sok kis lyuk, de nem hatékony)
  - Quick Fit (méretek szerinti lyuklista, a lyukak összevonása költséges)

3. Multiprogramozás virtuális memória használattal

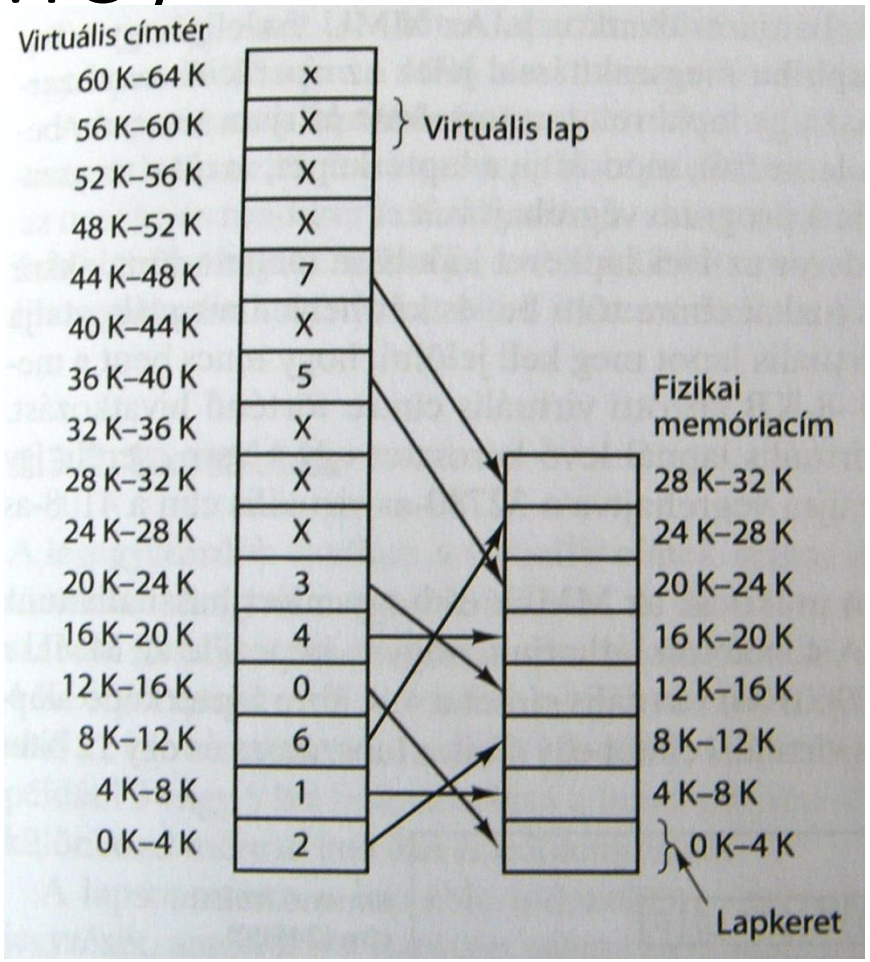
- Egy program használhat több memóriát mint a rendelkezésre álló fizikai méret.
  - Az operációs rendszer csak a „szükséges részt” tartja a fizikai memóriában.
  - Egy program a „virtuális memória térben” tartózkodik.
  - John Fotheringham (1961, ACM)
  - Az elv akár a monoprogramozás környezetben is használható.
    - 1961-ben és utána a „single task” rendszerű (kis)gépek léteztek.

Memória kezelő feladatok

- Betölti az alkalmazást a memóriába
- Nem minden részen engedélyezzük a végrehajtást
  - DEP- Data Execution Prevention
- Osztott memória használat
  - Közös kód (könyvtár) részlet
    - Pl: shared.dll
  - Közös adat terület, közös lap
    - COW – Copy On Write
      - Mindaddíg az adatterület közös, amíg a folyamat nem módosítja azt. Módosítás esetén először saját példányt készít, majd azt módosítja.
- Dinamikus (virtuális memória kezelés)

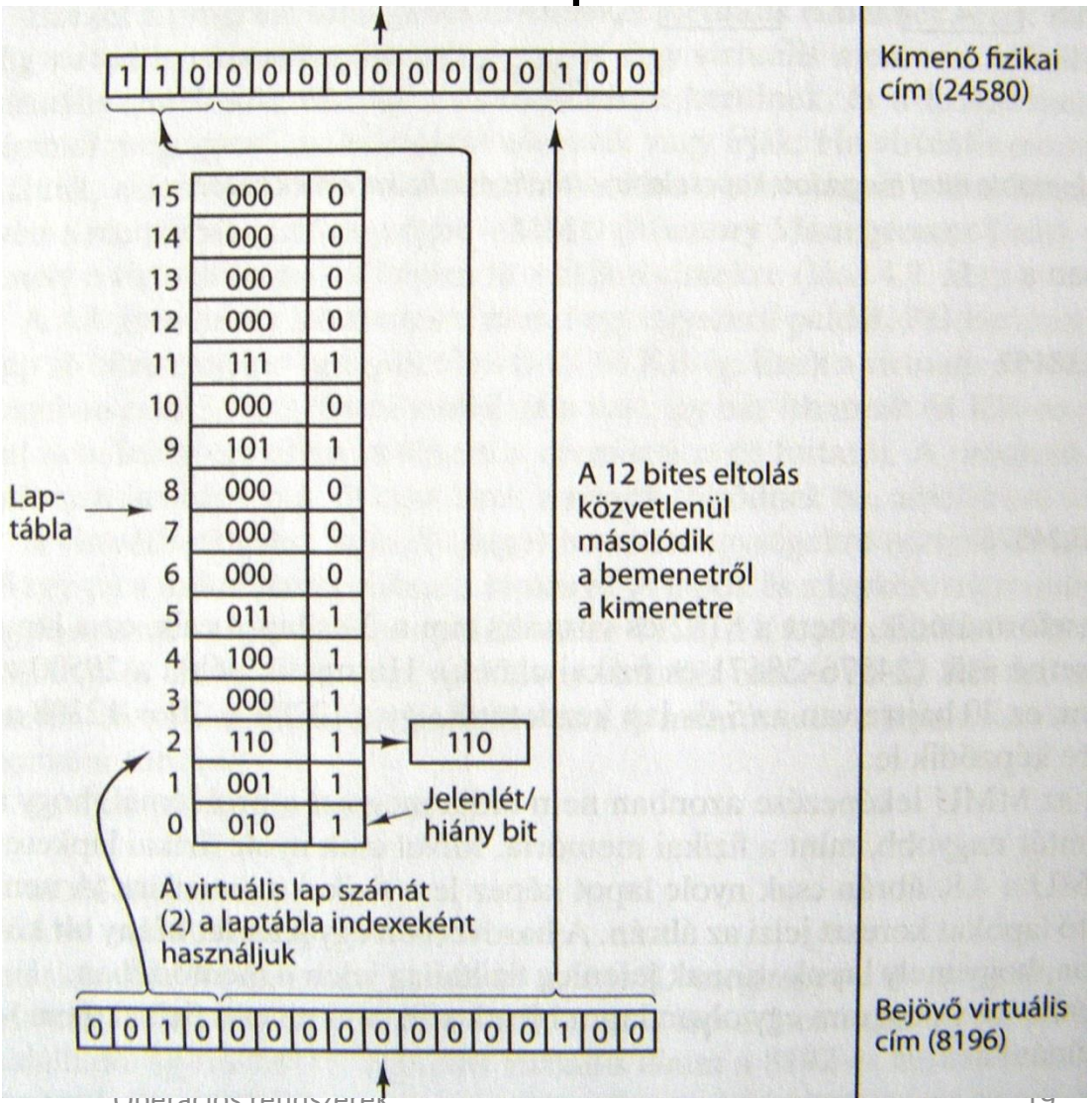
Memória Menedzsment Unit (MMU)

- A virtuális címtér „lapokra” van osztva. (Ezt laptáblának nevezzük)
- Több információt tárol a lapokra nézve.
- Pl: Jelenlét/hiány bit
- Virtuális-fizikai lapok összerendelése
- Ha az MMU látja, hogy egy lap nincs a memóriában, laphibát okoz, op.rendszer kitesz egy lapkeretet, majd behozza a szükséges lapot.



MMU működés
Példa: 16 darab- 4kb lap esetén

- Kettő hatvány méretű lapok
- A 16 bites virtuális címből az első 4 bit a lapszám, a többi az offset.
- 1 bit a jelenlét/hiány jelzésére.
- Kimenő 15 bit kerül a fizikai címsín-re.



Laptábla problémák

- A korábbi példa (16 bit virtuális címtér, 4 bit laptábla, 12 bit,4 KB, offset, azaz lapméret) egyszerű, gyors.
- Egy mai processzor vagy 32 vagy 64 bites.
  - 32 bit virtuális címtérből 12 bit (4kb) lapméretnél 20 bit a laptábla mérete. (1MB, kb. 1millió elem)
    - Minden folyamathoz saját virtuális címtér, saját laptábla tartozik! Ilyen méretű laptábla még elképzelhető!
  - 32 bites rendszerben 2 vagy 3GB lehet a virtuális címtér.
  - 64 bites virtuális címtér esetén ilyen méretű laptábla megvalósíthatatlan!
    - Helyette a CPU architektúra 48 bites laptábla mutatót támogat.

Többszintű laptáblák

- Már 32 bites virtuális címzésnél is gond az egyszerű laptábla használat.
- Használjunk kétszintűt:
  - Lap Tábla1= 10 bit (1024 elem)- felső szintű laptábla
  - Lap Tábla2= 10 bit - Második szintű laptábla
  - Lapon belüli Offset= 12 bit
  - Előny: mivel egy folyamathoz szükséges a laptábla memóriában tartása is, itt csak 4 db 1024 elemű táblára van(LT1-re, meg a program, adat és verem LT2 táblájára) szükség (nem egymillióra)
- Tábla és offset bitszámokra klasszikus értékeket választottunk, ezeken módosíthatunk, de lényegi változás nem lesz.
- A két szintet többszintűre is cserélhetjük! (Bonyolultabb)

Egy táblabejegyzés jellemző szerkezete

- Tartalmazza a lapkeret számát (fizikai memória és az offset mérete mondja meg, hogy hány bit)
- Jelenlét/hiány bit
- Védelmi bit, ha =0 írható, olvasható, ha =1 csak olvasható.
  - 3 védelmi bit: írás, olvasás, végrehajtás engedélyezése minden lapra.
- Dirty bit (módosítás). Ha 1 akkor módosult a lapkeret memória, azaz lemezre íráskor tényleg ki kell írni!
- Hivatkozás bit, értéke 1 ha hivatkoznak a lapra (használják). Ha használnak egy lapot, nem lehet lemezre tenni!
- Gyorsító tár letiltás bit. Ott fontos, ahol a fizikai memória(bizonyos területe) egyben I/O eszköz adatterület is.

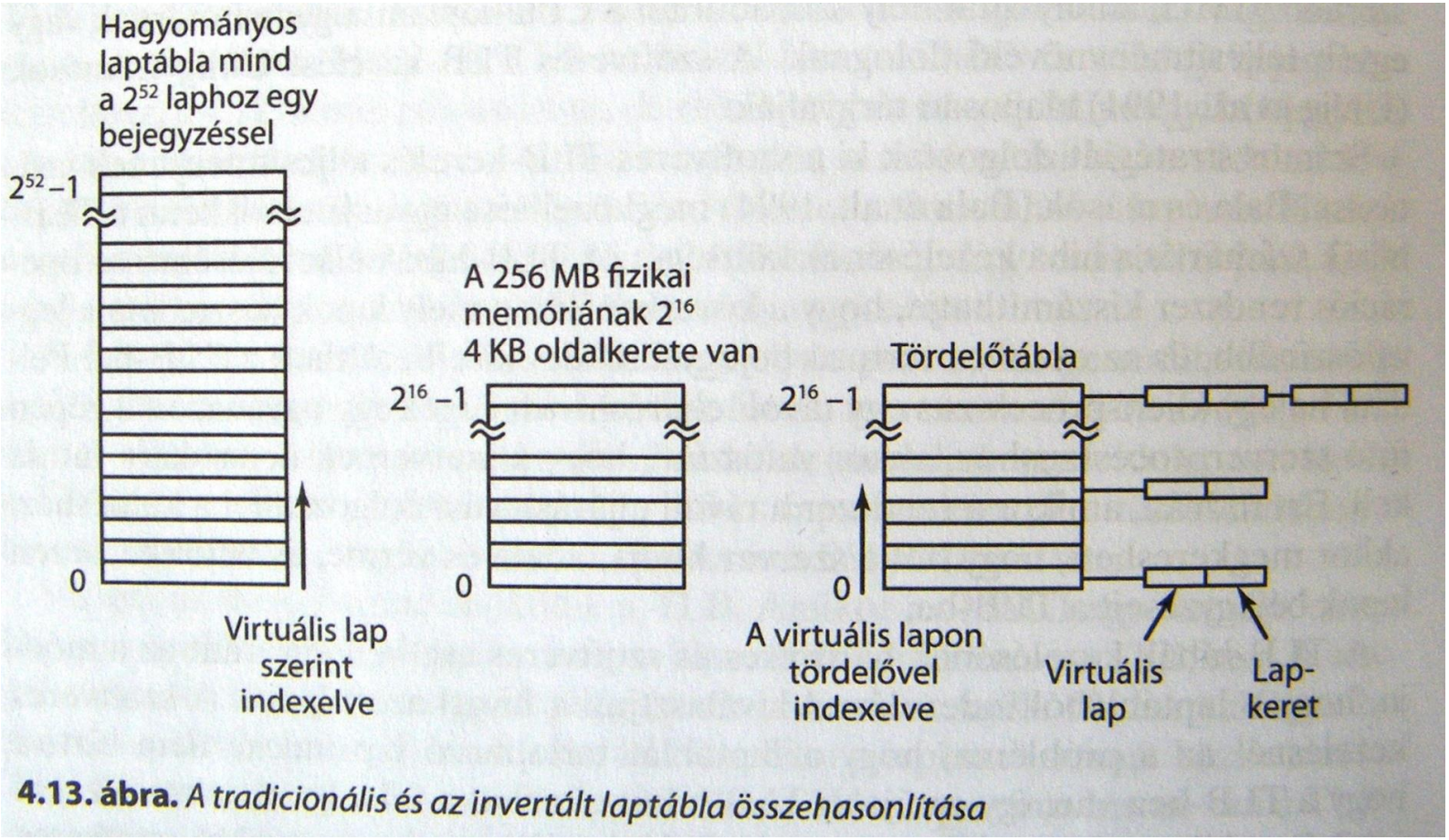
Lapozás helyett(mellett) TLB (Translation Lookaside Buffer)

- A klasszikus MMU eszközünk lassú, egy memória hivatkozáshoz legalább 1 táblahivatkozás kell, így minimum felére csökken a memórai műveleti idő.
- Tegyünk az MMU-ba egy kis HW egységet (TLB-asszociatív memória), kevés bejegyzéssel (eleinte 64-nél nem többet)
  - Ma a Nehalem esetében kétszintű TLB van, a TLB2 512 elemű.
- Szoftveres TLB kezelés
  - 64 elem elég nagy ahhoz, hogy kevés TLB hiba legyen, így a hardveres megoldás kispórolható.
  - Ilyen pl. a Sparc Mips, Alpha, HP PA, PowerPC

Invertált laptáblák

- Induljunk ki a valós memória méretből.
- A laptábla annyi elemet tartalmaz, amennyi a fizikai memóriából következik.
  - Sok helyet takarít meg, de nehezebb a virtuális címből a fizikai megadása. (Nem lehet automatikus index használatot végezni!)
  - PL: 4 kb lapméret, 1GB RAM, 2^{18} darab bejegyzés.
- Kiút: Használjunk TLB-t, ha hibát ad, akkor az invertált laptáblában keresünk, és az eredményt a TLB-be rakjuk.

Invertált tábla példa



4.13. ábra. *A tradicionális és az invertált laptábla összehasonlítása*

Címfordítás 32 vagy 64 biten

- 32 biten:
  - 2 bit Directory pointer(map) tábla
  - 9 bit Directory tábla
  - 9 bit Page tábla
  - 12 bit Laptábla
- 64 biten:
  - 9 bit Directory map tábla
  - +1 Directory tábla, összesen 2 Directory+1 Page tábla
  - 64 bitből 48 használt
  - Max méret: 256 TB, 128TB User space, 128 TB System space

Memória kezelés mai rendszerekben

- 64 bites architektúra- 4 szintű laptábla +Page Table Entry (PTE)
- 48 bites virtuális címtér – 12 offset
- A ma használt lapcsere módszer: LRU vagy FIFO (clock)
- Lokális, globális lapcsere
- CPU CR3 regiszter
- PTE-Page Table Entry
  - 64 bit, bit jelentések

Current implementations of the x64 architecture limit virtual addresses to 48 bits. The components that make up this 48-bit virtual address and the connection between the components for translation purposes are shown in Figure 5-20, and the format of an x64 hardware PTE is shown in Figure 5-21.

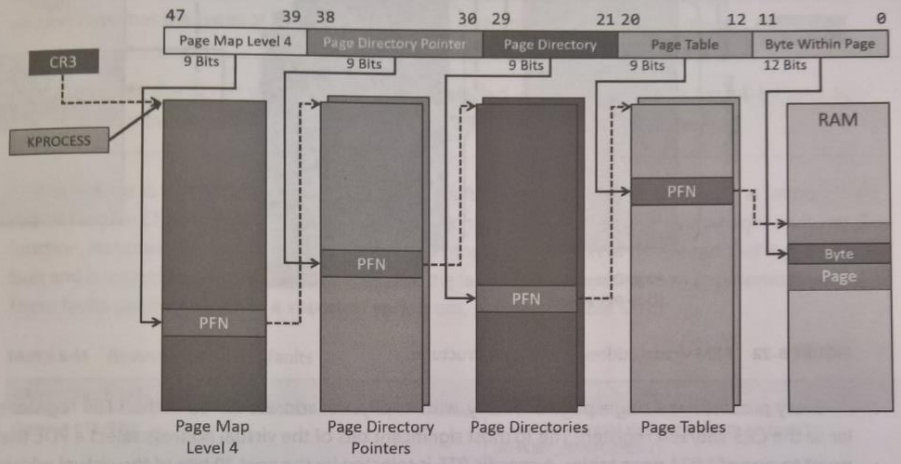


FIGURE 5-20 x64 address translation.

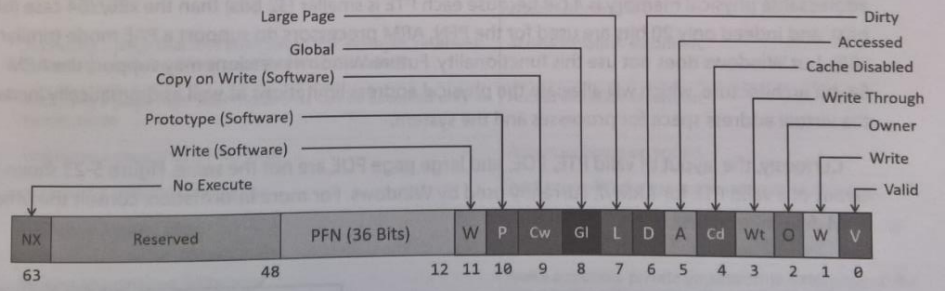


FIGURE 5-21 x64 hardware PTE.

Lapcserélési algoritmusok

- Ha nincs egy virtuális című lap a memóriában, akkor egy lapot ki kell dobni, berakni ezt az új lapot.
  - Kérdés hogyan?
  - Véletlenszerűen? Jobb azt a lapot kirakni amelyiket nem használtak az „utóbbi időben”.
- A processzor gyorsító tár (cache) memória használatnál, vagy a böngésző helyi gyorsítótáránál is hasonló a helyzet.

Optimális lapcserélés

- Címkézzünk meg minden lapot azzal a számmal, ahány CPU utasítás végrehajtódik mielőtt hivatkozunk rá!
- Dobjuk ki azt a lapot, amelyikben legkisebb ez a szám!
- Egy baj van, nem lehet megvalósítani!
- Kétszeres futásnál tesztelési célokat szolgálhat!

NRU (Not Recently Used) algoritmus

- Használjuk a laptábla bejegyzés módosítás (Modify) és hivatkozás (Reference) bitjét.
  - A hivatkozás bitet időnként (óramegszakításnál, kb. 0.02 sec) állítsuk 0-ra, ezzel azt jelezzük, hogy az „utóbbi időben” volt-e használva, hivatkozva.
  - 0.osztály: nem hivatkozott, nem módosított
  - 1.osztály: nem hivatkozott, módosított
    - Ide akkor lehet kerülni, ha az óramegszakítás állítja 0-ra a hivatkozás bitet.
  - 2.osztály: hivatkozott, nem módosított
  - 3.osztály: hivatkozott, módosított
- Válasszunk véletlenszerűen egy lapot a legkisebb nem üres osztályból.
  - Egyszerű, nem igazán hatékony implementálni, megfelelő eredményt ad.

FIFO lapcserélés, Második Lehetőség.

- Egyszerű FIFO, más területekről is ismert, ha szükség van egy új lapra akkor a legrégebbi lapot dobjuk ki!
  - Listában az érkezés sorrendjében a lapok, egy lap a lista elejére érkezik és a végéről távozik.
- Ennek javítása a Második Lehetőség lapcserélő algoritmus.
  - Mint a FIFO, csak ha a lista végén lévő lapnak a hivatkozás bitje 1, akkor kap egy második esélyt, a lista elejére kerül és a hivatkozás bitet 0-ra állítjuk.

Óra lapcserélés

- Óra algoritmus: mint a második lehetőség, csak ne a lapokat mozgassuk körbe egy listába, hanem rakjuk körbe őket és egy mutatóval körbe járunk.
- A mutató a legrégebbi lapra mutat.
- Laphibánál ha a mutatott lap hivatkozás bitje 1, nullázzuk azt és a következő lapot vizsgáljuk!
- Ha vizsgált lap hivatkozás bitje 0 kitesszük!

LRU (Least Recently Used) algoritmus

- Legkevésbé (legrégebben) használt lap kidobása.
- Hogy valósítom meg?
  - 1. Lapok listában, lista végére kerül az utoljára használt. Elején van a legrégebben nem használt. Lista kezelés kell!
  - 2. Vegyünk egy számlálót ami minden memória hivatkozásnál 1-el nő. Minden laptáblában tudjuk ezt a számlálót tárolni. Minden memóriahivatkozásnál ezt a számlálót beírjuk a lapba. Laphibánál megkeressük a legkisebb számlálóértékű lapot!
  - 3. LRU bitmátrix használattal, n lap, n x n bitmátrix. Egy k. lapkeret hivatkozásnál állítsuk a mátrix k. sorát 1-re, míg a k. oszlopát 0-ra. Laphibánál a legkisebb értékű sor a legrégebbi!

NFU (Not Frequently Used) algoritmus

- Minden laphoz tegyünk egy számlálót. Minden óramegszakításnál ehhez adjuk hozzá a lap hivatkozás (R) bitjét.
- Laphibánál a legkisebb számlálóértékű lapot dobjuk ki. (A leginkább nem használt lap)
- Hiba, hogy az NFU nem felejt, egy program elején gyakran használt lapok megőrzik nagy értéküket.
- Módosítsuk: Minden óramegszakításnál csináljunk jobbra egy biteltolást a számlálón, balról pedig hivatkozás bitet tegyük be (shr).(Öregítő algoritmus)
  - Ez jól közelíti az LRU algoritmust.
  - Ez a lap számláló véges bitszámú (n), így n időegység előtti eseményeket biztosan nem tud megkülönböztetni.

Lapozás tervezési szempontok I.

- Munkahalmaz modell
  - A szükséges lapok betöltése. (Induláskor-előlapozás) A folyamat azon lapjainak fizikai memóriában tartása, melyeket használ. Ez az idővel változik.
  - Nyilván kell tartani a lapokat. Ha egy lapra az utolsó N időegységben nem hivatkoznak, laphiba esetén kidobjuk.
  - Óra algoritmus javítása: Vizsgáljuk meg, hogy a lap eleme-e a munkahalmaznak? (WSClock algoritmus)
- Lokális, globális helyfoglalás
  - Egy laphibánál ha az összes folyamatot (globális), vagy csak a folyamathoz tartozó (lokális) lapokat vizsgáljuk.
  - Globális algoritmus esetén minden folyamatot elláthatunk méretéhez megfelelő lappal, amit aztán dinamikusan változtatunk.
  - Page Fault Frequency (PFF) algoritmus, laphiba/másodperc arány, ha sok a laphiba, növeljük a folyamat memóriában lévő lapjainak a számát. Ha sok a folyamat, akár teljes folyamatot lemezre vihetünk.(terhelosztás)

Lapozás tervezési szempontok II.

- Helyes lapméret meghatározása.
  - Kicsi lapméret
    - A „lapveszteség” kicsi, viszont nagy laptábla kell a nyilvántartáshoz.
  - Nagy lapméret
    - Fordítva, „lapveszteség” nagy, kicsi laptábla.
  - Jellemzően: n x 512 bájt a lapméret, XP, Linuxok 4KB a lapméret. 8KB is használt (szerverek)
- Közös memória
  - Foglalhatunk memóriaterületet, amit több folyamat használhat.
  - Elosztott közös memória
    - Hálózatban futó folyamatok közti memória megosztás.

Szegmentálás

- Virtuális memória: egy dimenziós címtér, 0-tól a maximum címig (4,8,16 GB, …)
- Több programnak van dinamikus területe, melyek növekedhetnek, bizonytalan mérettel.
- Hozzunk létre egymástól független címtereket, ezeket szegmensnek nevezzük.
  - Ebben a világban egy cím 2 részből áll: szegmens szám, és ezen belüli cím. (eltolás)
  - Szegmentálás lehetővé teszi osztott könyvtárak „egyszerű” megvalósítását.
  - Logikailag szét lehet szedni a programot, adat szegmens, kód szegmens stb.
  - Védelmi szint megadása egy szegmensre.
  - Lapok fix méretűek, a szegmensek nem.
    - Szegmens töredezettség megjelenése. Ez töredezettség összevonással javítható.

Pentium processzor virtuális címkezelése I.

- Sok szegmens lehet, egy szegmens virtuális címtere: 2^{32} bájtos (4GB)
  - XP, Linuxok egyszerű lapozásos modellt használnak, egy folyamat-egy szegmens- egy címtér.
  - Szegmensek száma: Pentium-ban 16000, a TSS szegmens értékéből, processzor tulajdonság.
- LDT- Local Descriptor Table
  - Minden folyamatnak van egy sajátja.
- GDT- Global Descriptor Table
  - Ebből 1 van ezt használja mindenki.
  - Ebben találhatók a rendszerszegmensek (os. is)
- IDT- Interrupt Descriptor Table
  - Ebből is egy van.
- Fizikai cím: Szelektor + offset művelet elvégzése

Pentium processzor virtuális címkezelése II.

- Szegmens elérése: 16 bites szegmens szelektor
  - Ennek alsó 0-1. bitje a szegmens jogosultságát adja
  - 2. bit=0=GDT-ben, 1=LDT-ben van a szegmens leíró.
  - Felső 13 bit a táblázatbeli index. (8192 elemből áll mindkét tábla)
- Az LDT, GDT tábla elem 32 bites, ebből kapjuk a bázis címet, amihez az eltolást hozzáadjuk. (eredmény is 32 bit!) Ez adja a lineáris címet.
- Ha a lapozás tiltott, ez a valós fizikai cím, ha nem tiltott akkor kétszintű laptábla használat: lapméret 4KB, 1024 (10 bit) elemű laptáblák.
- A gyorsabb lapkeret eléréshez TLB-t is használ.

Pentium processzor védelmi szintjei

- 0- Kernel szint
- 1- Rendszerhívások
- 2- Osztott könyvtárak
- 3- Felhasználói programok
- Alacsonyabb szintről adatok elérése engedett.
- Magasabb szintről alacsonyabb szintű adatok (0-1 szint) elérése tiltott.
- Eljárások hívása megengedett, csak ellenőrzött módon (call szelektor)
- Felhasználói programok osztott könyvtárak adatait elérhetik, de nem módosíthatják!

Köszönöm a figyelmet!