Operációs rendszerek

1. Bevezetö

Operációs rendszer: nincs jó definíció

Korai operációs rendszerek
Korai batch rendszerek:
- lyukkártyák, átlapolt feldolgozá, pufferelés
- Spooling: nagyobb kapacitású, gyors, véletlen hozzáférésü memóriák
  - több feladat, melyek egy idöben futhatnak
  - elmozdulás a multiprogramozás felé
- multiprogramozás
  - nagyobb kapacitású, gyorsabb memóriá
  - a feladatok nem beérkezés sorrendje szerint dolgozhatók f
  - optimalizáció lehetösége, megjelenik az ütemezés (scheduling)

1960-as évek vége:
- C és modernebb nyelvek, ARPANET, miniszámítógépek
- idöosztásos rendszerek
  - time sharing/multitasking
  - online felhsználók -> fontos a válaszidö
  - pl.: klasszikus UNIX erre vállalkozik

1970-es évek:
- PC, IBM PC, x86 CPU architektúra, memória, HDD, grafikus képernyö, ...
- lépés az elosztott rendszerek felé
  - decentralizálás - funkciók térbeli elosztása
  - elöny/hátrány: biztonság, skálázhatóság, megbízhatóság
- többprocesszoros rendszerek: homogén / heterogén processzorok

Operációs rendszer típusok:
- alkalmazás specifikus : kliens/server/mainframe, beágyazott, mobil OS
- tulajdonság specifikus: valós idejü, nagy megbízhatóságú, konfigurálható

Beágyazott rendszer: (PC-t használhatunk, de nem erre lettek kitalálva)
- jól meghatározott feladatra alkalmas
- külvilággal intenzív információs kapcsolat
- sokszor biztonságkritikus környezet (közlekedés, egészségügy, ...)

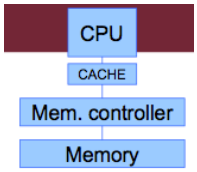
Valós idejü rendszerek:
- Adott eseményre a rendszer adott idön belül adott valséggel vászol
- *lágy valós idejü*: valség<1, ha nincs katasztrofális köv.-e a késésnek
- *kemény valós idejü*: valség=1, a rendszer nem késhet, katasztrofális köv.
- valós idejü operációs rendszerek: ~~Windows, Linux~~, sajnos az OS X sem :)

HW architertúrák:
- sokféle, x86 PC esetén is
- különbségeket elfedi az operációs rendszer, de figyelni kell
- pl.: Linux - ARM különbségek: HW közeli cserével oldatóak meg

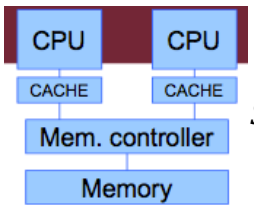
Védelmi szintek a CPU-ban: (CPU privilege levels)
- hierarchikus, 2 vagy 4 szint, általában: user/kernel mode
- alacsony szintü CPU eröforrásokhoz történö hozzáférést szab.

Memory Management Unit (MMU):
- speciális HW a CPU-ban
- memória állapotának nyilvántartása
- virtuális memória leképzése fizikai memóriára
- memória védelem

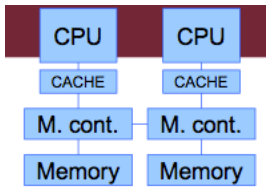
Számítógép architektúrák: (homogén többprocesszoros rendszerek esetében)



- *single CPU*
  - DMA, ha párhuzamosan kezeli a memóriát a CPU-val
  - versenyhelyzet a DMA vezérlö és a CPU között
  - CACHE koherencia sérülhet

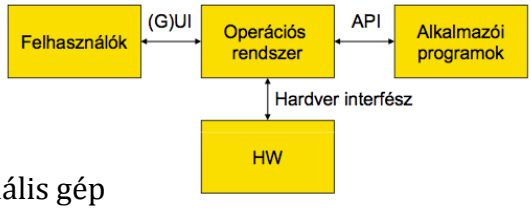


SMP (Simmetric Multiprocessing)
  - több azonos CPU, külön cache, közös buszon a memória
  - OS támogatás kell hozzá (különben 1 CPU látható)



- *NUMA* (Non-Uniform Memory Access)
  - összefüggö fizikai memória, ccNUMA (cache koherenciával)
  - csak a memória vezérlök között speciális kommunikáció

OS és környezete:
- a felhasználók, alkalmazói programok és a HW nincs direkt kapcsolatban



OS-ek átlalános belsö felépítése:
- réteges szerkezet- rétegek határán egy virtuális gép
- kernel, HW közeli réteg, rendszerprogramok, rendszerhívások fogadása

OS-ek kialakítása:
- monolitikus kernel - összes funkció
- moduláris kernel - dinamikusan betölthetö kernel modulok
- mikrokernel - minimális funkció, kliens/server arch.-ban csatlakozás

Linux: monolitukus kernel, modulárisan betölthetö kernel modulokkal
OS X: mikrokernel + Free BSD UNIX

Hardver kezelés: I/O portok, mem írás/olvasás, DMA, megszakítás (HW/SV)
Rendszerhívás: alkalmazói program - IT - kernel - alk. program, nagy overhead!
I/O müveletek: rendszerhívásokkal, kernel hajtja végre

OS elindulása (PC/server):
bootstrap process - Init/Reset vector - BIOS/EFI - boot sector - 2. szintü boot loader - OS betölt majd indul

2. UNIX bevezetö:

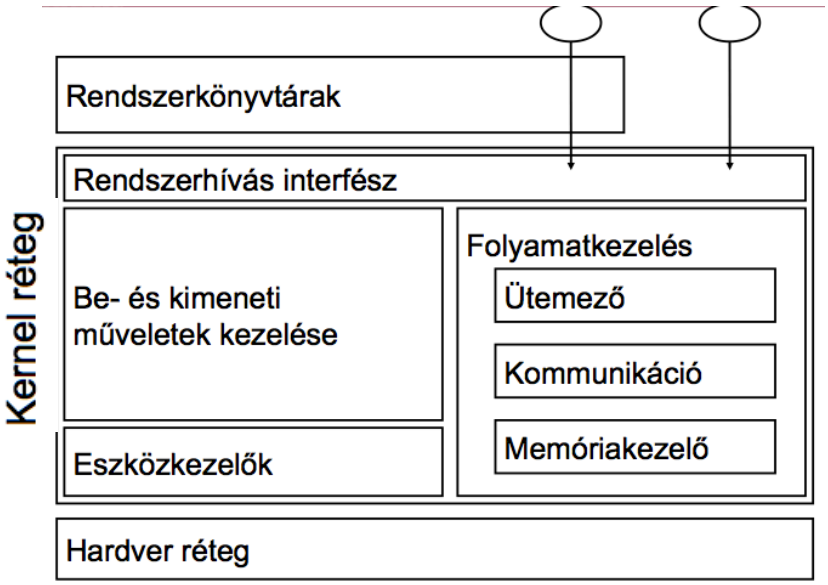
Miért UNIX?
- >30 év, nyílt forráskód, minden rétegben megtalálható

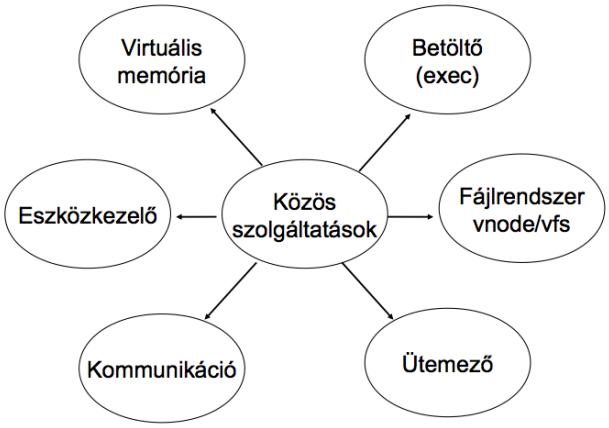
Történet:
- AT&T Bell Lab, 1969, Ken Thompson és Dennis Ritchie

UNIX fejlesztési modellje:
- iteratív, forráskódú, több fejlesztö, több HW platform
- elöny: gyorsan fejllödik/terjed, hátrány: inkompatibilis, szakértelem

Családfa és szabványosítás
- System V: AT&T változat: Sun Solaris, SCO, ...
- BSD: Berkley változat: SunOS, OpenBSD, FreeBSD, OS X :)
- szabványosítás: IEEE POSIX, AT&T SVID, X/Open, Unix95, Unix98,...
- ma: kliens elenyészö, server platformon jelentös, beágyazott terület

Felépítés:
- réteges, moduláris monolitikus kernel





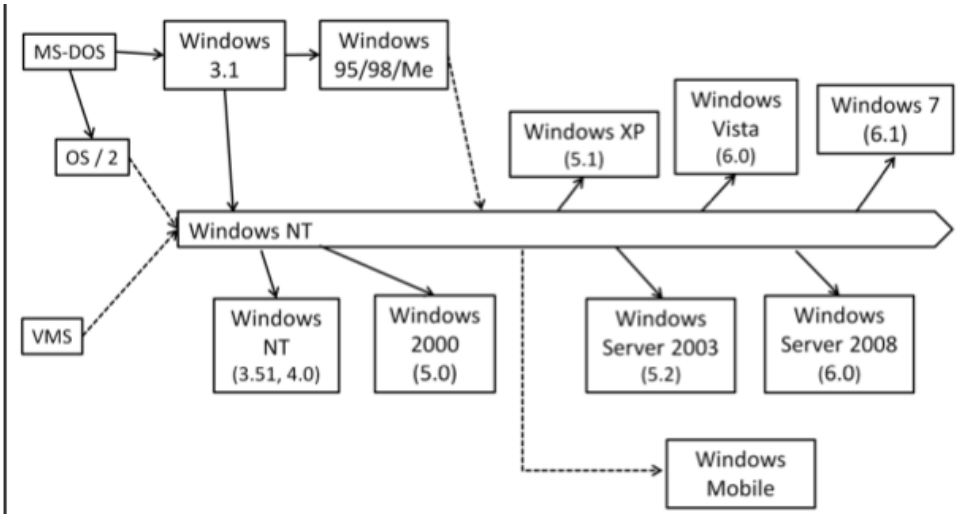
Rendszergazdai szemmel:
- karakteres (grafikus) felület, felhasználo azonosítás és hozzáférés-szab.
- naplózás, eszközkezelés, hálózati/vállalati szolgáltatások, virtualizáció, ...

Felhasználói szemmel:
- grafikus felület, könyvtárrendszer, beépített parancsok, telepített alk.-ok

Jelentösebb UNIX disztribúciók:
- server: CentOS, OpenSolaris, IBM AIX, Suse Linux E. Server, openSUSE
- kliens: Ubunti, Debian, Fedora, SUSE Linux E. Desktop, openSUSE

3. Windows bevezetö

Történelem:

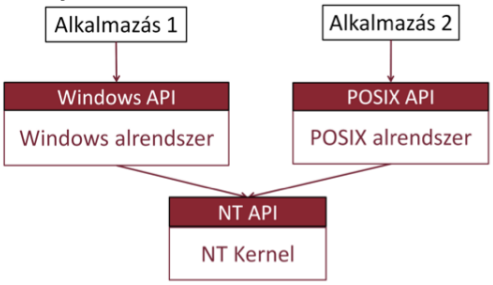


Windows felépítése:
- hordozhatóság a cél: többféle CPUra, kernel: C nyelven
- kiterjeszthetöség: moduláris, jól def. interfészek, unicode használata
- megbízhatóság(smile): biztonsági szabványok
- teljesítmény: 32 bites, preemptív, többszálú, újrahívható, SMP, aszinkron I/O
- kompatibilitás: POSIX, DOS és 16 bites Win Api támogatás, OS/2

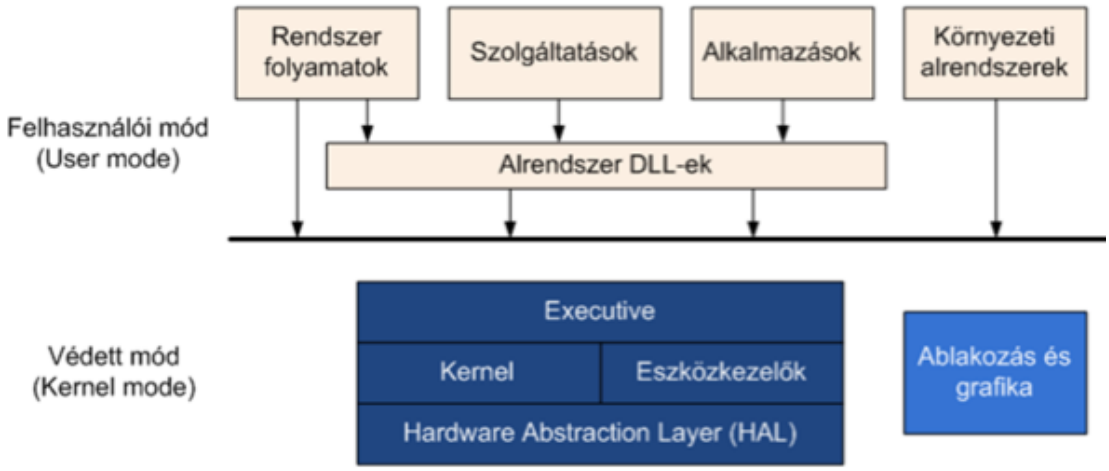
Újrahívható (reentrant): a rendszerhívásokat több alkalmazás is meghívhatja egyszerre, nem blokkódnak, ha már valakit éppen kiszolgál az adott rendszerhívás
Preemptív: egy szálat fel lehet függeszteni futás közben. A Windows kernel **teljesen preemptív**, a kernel szálak is felfüggeszthetőek.

API (Application Programming Interface): kívülröl milyen függvényeken, adat struktúrákon keresztül érhetö el egy alkalmazás (pl: fork(), select(), fopen())

Környezeti alrendszer:



Egyszerüsitett architektura:

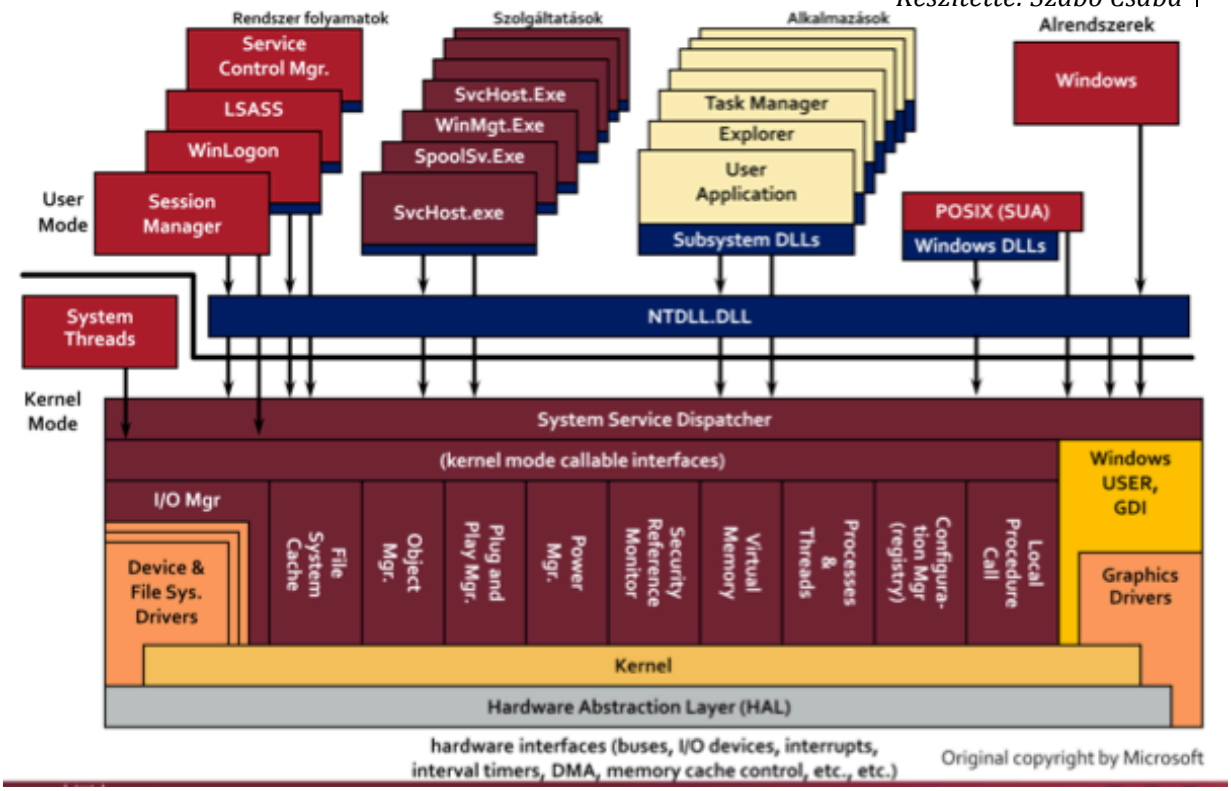


HAL: HW részletek elrejtése, egységes felület, hal.dll
Eszközkelezők (Device driver): kernel módú modulok, rétegzett struktúra, fájlrendszer, hálózatkezelés, *.sys
Kernel: Alap szolgáltatások, ütemezés, megszakítás, multiprocesszosor szinkron, ntoskrnl.exe
Executive: OS szolgáltatások, memória, folyamatkezelés, OO szemlélet, Biztonság, I/O, ntoskrnl.exe

Processor Access Mode: CPU támogatás, védelem, kernelt felh. folyamatoktól, felh. folyamatokat egymástól, más mint a környezetváltás ahol elmentjük az éppen futó szál adatait (regiszterek, program számláló, stb.), és betöltünk, majd futtatunk egy újat; itt nem változik, hogy melyik szálat hajtjuk végre.

Rendszerfolyamatok (System processes): alapvetö rendszerfunkciók, induláskor ezek
Szolgáltatások(Services): háttérben futó folyamatok, hasonló mint a UNIX daemon
Környezeti alrendszerek: folyamatok felh. módú kezelése, állapottárolás, csrss.exe
Alrendszer DLL-ek: alrendszer API hívásokat fordítják az Executive hívásaira, kernel32.dll
Ablakozás és grafika: GUI kernel módban fut, ablakkezelés, rajzolás

Munkamenet(session): több felh. bejelentkezve egy gépen, ID 0: rendszerfolyamat
Windows kernel: monolitikus



Session Manager: munkamenetek kezelése
Wininit: rendszer folyamatok indítása
Winlogon: bejelentkezés, Ctrl+Alt+Del kezelés
LSASS: biztonság
Service Control Manager: szolgáltatások elindítása, leállítása

Szolgáltatások: felhasz. bejelentkezés nélkül is futnak, SvcHost: általános futtató folyamat

Alkalmazások: legtöbb program alrendszer DLL-eken keresztül használja az OS-t

NTDLL.DLL: executive fgv csokjai, belsö függvények az alrendszereknek

System Service Dispatcher: rendszerhívások elkapása, paraméter ellenörzése, továbbítás a megfelelö komponensbe
System threads: kernel és meghajtók szálai, feladatok amikhez várni kell a háttérben futnak, ezek is felfüggeszthetöek

Objektum Kezelés:
- a Windows objektumokkal dolgozik: executive, kernel
- adatrejtés, interfész használat

Windows verziók:
- ugyanaz a kernel forrás skálázódik

4. Windows hibakeresés

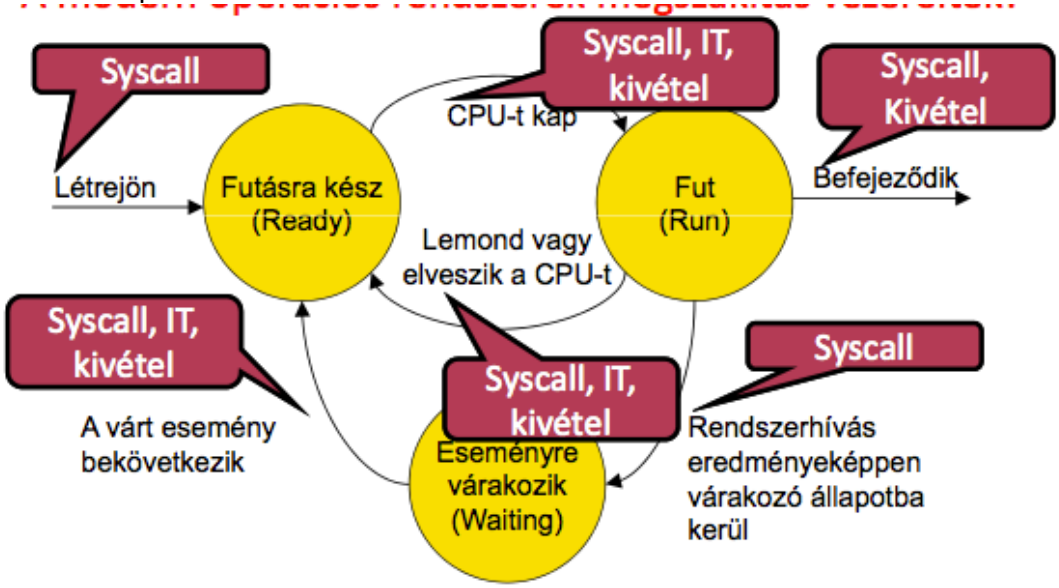
Process Monitor, eseménynapló, esemény részletei
Eventlog online help
Crash dump
Safe mode

5. Scheduling (ütemezés, task kezelés multiprogramozott OS-en)

Történeti háttér:
- batch rendszere, spolling, multiprogramozas
- multiprog:- 1 CPU - M feladat
  - feladattipusok: rendszer-, batch-, on-line feladat
  - feladat keszlet (job pool)
  - a feladatok végrehajtásához eröforrások szükségesek
- multiprog során az a cél, hogy a feladat készletet minél optimálisabban hajtsa végre a rendszer
- a feladatoknak alapesetben nem szabad tudniuk egymásról: külön "virtuális"gépen futnak, de osztozniuk kell a rendszer eröforrásain, szinkronizálódniuk kell, kommunikálniuk kell => OS feladata

Alapfogalmak:
1. Esemény: rendszer életében lezajló változás, történés, SW IT, HW IT
2. Feladat(task): folyamat, szál, process, job
   - az elnevezések keverednek
   - multiprogramozott OS legfontosabb alapfogalma
   - müveletek meghatározott sorrendben történö végrehajtása
   - a feladat több mint program: állapota van, aktív, adatszerkek adatokkal vannak feltöltve

- feladatok állapota:



- feladat leíró (Task control block, TCB):
  - adatstruktúra a feladattal kapcsolatos adatok OS-en belüli tárolására
  - ez alapján végzi az OS a feladatok kezelését
  - task ID, állpot, PC, CPU regiszterek, ...
- kontextus váltás(context switch):
  - OS feladata hogy a feladat futásakor a lementett kontextust helyreálljon

3. Feladatok ütemezése(task scheduling):
   - kiválasztani a futásra kész feladatok közül a futót
   - a feladatok feladat sorokban (task queue) vannak: pl.: FIFO
   - ütemezés idöskálái:
     - rövid távú/CPU ütemezés: futásra kész sorból futó állapotba átmenö feladat
     - hosszú távú ütemezés batch rendszerekben: sokkal több feladat mint amennyit hatékonyan párhuzamosan végre tudunk hajtani, percenként fut
     - középtávú ütemezés: swapping (memória kiírása háttértárra), majd visszatöltés

4. Preemptív ütemezés: A futó feladattól az OS elveheti a CPU-t // nem preemptív üt.
5. Mértékek:
   - mérték - algoritmusok összehasonlítására
   - CPU kihasználtság (%): $t_{CPU} = t_{CPU,munka} + t_{CPU,admin} + t_{CPU,idle}$ (adminisztráció és a henyélés is bele tartozik), CPU kihasz. = Sum($t_{CPU,munka}$) / Sum(t_{CPU}) * 100 [%]
   - átbocsátó képesség: [munka/s] v. [1/s], adott idöegység alatt elvégzett feladatok száma, rendszerfeladatokat nem számoljuk
   - várakozási idö: $t_{waiting} = t_{ready} + t_{other,\ non\text{-}running}$[s]
   - körbefordulási idö: egy feladat rendszerbe helyezésétöl a teljesítésig eltelt idö $t_{CPU,végrehajtás} + t_{várakozás}$
   - válaszidö: feladat megkezdésétöl az elsö kimenet produkálásáig eltelt idö

Ütemezés követelmények:
- ütemezö algoritmus valós idejü - alacsony overhead
- célfüggvény szerint legyen optimális

- matematikai modell, szimulációk vagy mérések alapján vizsgálódnak
- korrektség
- kiéhezés elkerülése
- jósolható viselkedés
- alacsony adminisztratív veszteségek
- max átbocsátó képesség, min várakozási idö
- eröforrás használat figyelembe vétele

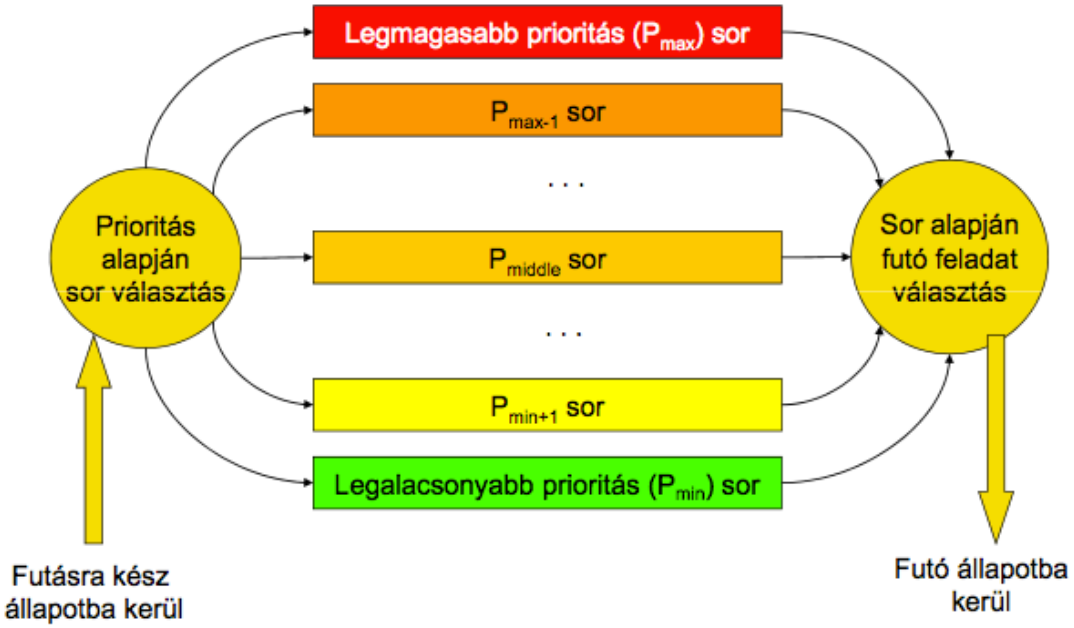
Egyéb szempontok:
- prioritás
- fokozatos leromlás/összeomlás
- statikus/dinamikus ütemezési algoritmus

CPU ütemezési algoritmusok: (egy CPU, egy feladat fut egy idöben, task quene, a feladatok CPU v. I/O löketekböl (burst) állnak)
- FIFO,FCFS
  - nem preemptiv
  - átlagos várakozási idö nagy lehet
  - kis adminisztrációs overhead
  - hosszú feladat sokáig feltartja az utána következöket
  - hosszú CPU löketü feltartja a rövid CPU löketü feladatokat
- Round-robin:
  - idöosztásos rendszer számára
  - kedvezöbb az on-line felhasználók számára
  - FIFO+egyszeri óra megszakítás: preemptiv
  - idöszelet > átlagos CPU löket -> átmegy FIFO-ba
  - idöszelet = átlagos CPU löket -> normál müködés
- Prioritásos ütemezök
  - külsö/belsö prioritás
  - statikus/dinamikus prioritás
  - jellegzetesen preemptiv, de lehet nem preemptiv is
  - nem korrekt, kiéheztetés elöfordulhat (feladatok öregítés segíthet)
  - egyszerü prioritásos ütemezö - egy szinten egy feladat
  - legrövidebb löketidejü (Shortest Job First - SJF)
    - nem preemptiv, legrövidebb löketidejüt választja futásra
  - legrövidebb hátralévö idö (Shortest Remaining Time First - SRTF)
    - SJF preemptiv változata
  - a legjobb válaszarány (Highest Response Ratio - HRR)
    - kiéhezést próbálja megoldani

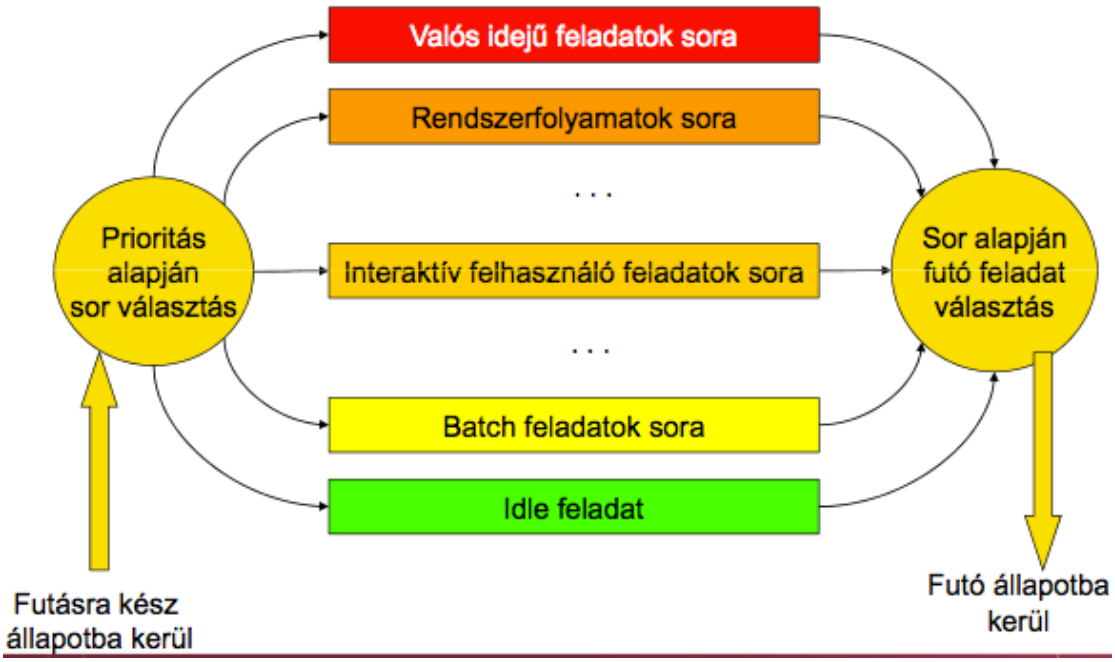
6. Scheduling (ütemezés)

Számolási feladatok ütemezéshez: (1.2)
http://home.mit.bme.hu/~micskeiz/opre/files/opre-feladatok-segedanyag.pdf

Többszintü sorok (multilevel queue):
- minden prioritási szinthez task queue
- különbözö szinteken különbözö algoritmusok lehetnek
- általában 8-16-32 prioritási szint



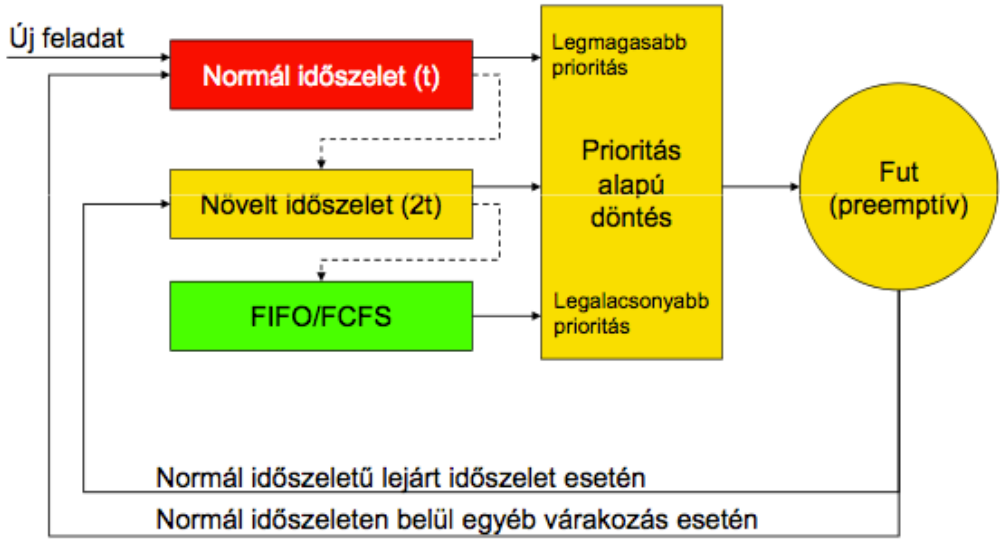
- prioritás feladathoz redelése: a prioritás a feladat jellegéhez kapcsolódik
- tipikusan RoundRobin minden szinten



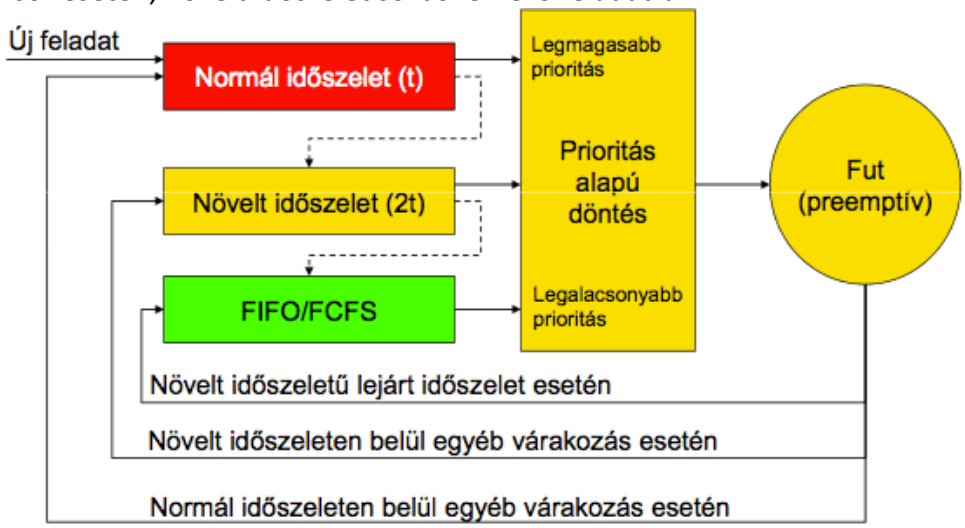
- alacsony prioritási szinten - éhezés => idöosztás prioritási szintek között, tárolunk egy futás idö nyilvántartást

Visszacsatolt többszintü sorok: (széles körben alkalmazzák, részleteiben eltérö algo.)

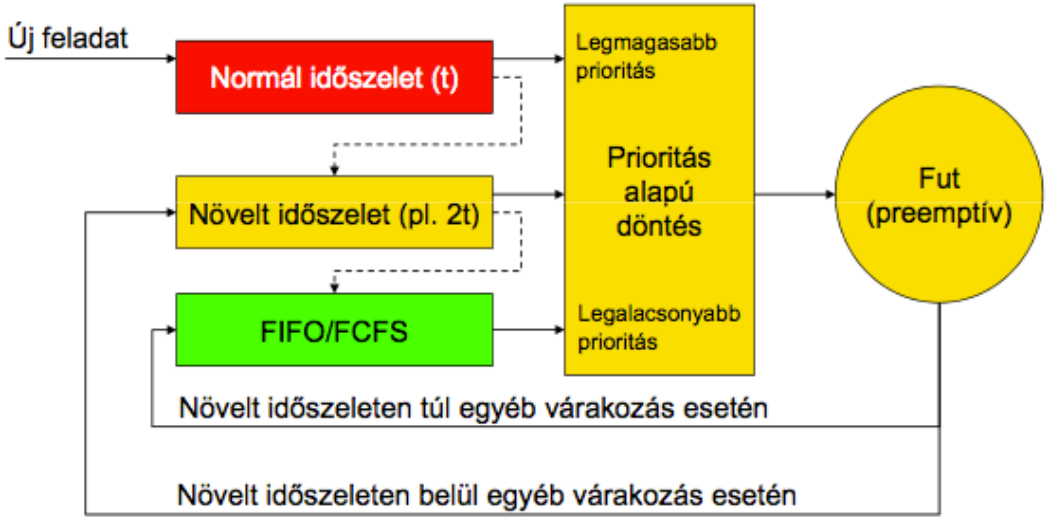
- 3 sor esetén, normál idöszelet sorból érkezö feladatra



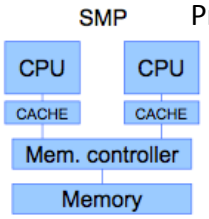
- 3 sor esetén, növelt idöszelet sorból érkezö feladatra



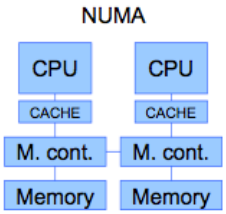
- 3 sor esetére, FIFO/FCS sorból érkezö feladat esetén



Többprocesszoros ütemezés (multiple-processor scheduling):
- homogén többprocesszoros rendszerek esetén (SMP vagy NUMA)
- két megoldás:
- egy I/O periféria egy adott CPU-hoz van rendelve (arra csatlakozik)
    - masters and slaves (egy CPU osztja ki a feladatokat)
    - self-scheduling / peering (minden CPU ütemez)



Processzor affinitás (Processor Affinity):
- a cache a futó feladat utasítás és adat tartalmának egy részét tartalmazza
- cél: a feladatot ugyanazon a CPU-n tartani: laza/kemény CPU affinitás
- SMP esetén a CPU affinitás csak a cache tárolók szempontjából érdekes
- NUMA esetén a feladat által használt fizikai memória is érdekes: a feladat CPU-hoz közeli memóriából fusson, távoli memória elérés minimalizálása



Terhelés megosztás (load balancing):
- egy globális futásra kész sor vagy processzoronként futásra kész sor
    Processzoronkénti task queue:
    - push and/or pull
    - push: OS kernel folyamat mozgatja a sorok között a feladatokat
    - pull: az idle állapotban CPU próbál a többi sorából feladatot kapni
- összefüggö, párhuzamosan futtatható feladatok optimalizálása: Gand scheduler

7. Feladatok együttmüködése

Feladat(task) fogalma:
- feladat fogalma eredetileg folyamat (process) értelemben került használatra
- folyamat = végrehajtás alatt álló folyamat
  - ugyan abból a programból több folyamat is létrehozható
  - saját kód, adat, halom (heap) és verem memória területtel rendelkezik
  - védettek a többi folyamattól (szeparácio, virtuális gép, sandbox)

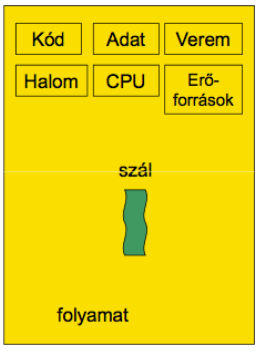
Folyamatok szeparációja:
- Virtuális CPU-n futnak: nem nem férhetnek hozzá a többi folyamat és az OS futása során elöálló processzor állapothoz
- saját virtuális memóriaterületük van: nem férhetnek hozzá más folyamatok virtuális memóriájához vagy direkt módon a fizikai memóriához - MMU feladata

Folyamatok létrehozása
- OS specifikus rendszerhívás (pl.: CreatProcess(), fork(), ...)
- szülö/gyerek viszony a létrehozó és létrehozott között => process fa, a szülö eröforrásaihoz hozzáférés többnyire konfgurálható, a szülö megvárhatja a gyerek terminálódását vagy futhat vele párhuzamosan, paraméterezhetö gyerek
- sok adminisztráció, eröforrás igényes

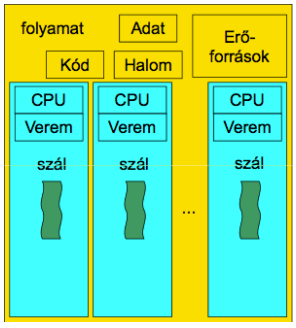
Folyamatok kommunikációja:
- a folyamatoknak együtt kell müködniük => kommunikáció kell
- két tetszöleges folyamat nem tud közös memórián keresztül kommunikálni - MMU, virtuális memória, OS rendszerhívásokon keresztül
- hatékony a védelem/szeparáció szempontjából
- nem hatékony módja a párhuzamos, erösen összefüggö feladatok megoldásának

Folyamatok befejezése:
- OS specifikus rendszerhívás (pl.: TerminateProcess(), exit(), ...)
- nyitott, használatban lévö eröforrásokat le kell zárni (pl.: fájlokat)
- szülö megkapja a visszatérési értéket (általában integer)
- a szülö megszünik de a gyerek nem: OS spec: gyerek is megszünik/default szülö
- sok adminisztráció, eröforrás igényes

Folyamatok értékelése:



- védelmi/szeparációs szempontból jó megoldás, de eröforrás igényes (létrehozás,kommunikáció, eröforrás megosztás, megszüntetés)
- megoldás: szál(thread):
  - a szál a CPU használat alapértelmezett egysége, magában szekvenciális kód
  - saját virtuális CPU-ja van, saját verem
  - kód, adat, halom, egyéb eröforrások osztozik a többi szálakkal, melyekkel azonos folyamat kontextusában fut
- folyamat = nehézsúlyú folyamat, szál = pehelysúlyú folyamat



Szálak támogatása:

- jelenleg natív módon támogatják az OS-ek a szálak létrehozását
- Windows: az ütemezö szálakat ütemez
- Linux: az ütemezö taskokat ütemez, melyek lehetnek folyamatok vagy szálak

Felhasználó módú szálak:

- korábban UNIX/Linux alatt: green threads
- az OS csak a folyamatot tudja ütemezni, ha az fut akkor azon belül a felhasználói módú szál könyvtár saját ütemezöje fut

Szálak létrehozása:

- Win32 API: CreateThread() bonyolult paraméterezéssel
- Pthreads: POSIX threads
- JAVA thread: VM a folyamat, VM-en belül szál, Thread osztályból származtatva, runnable interface megvalósítása, platform-specifikusan

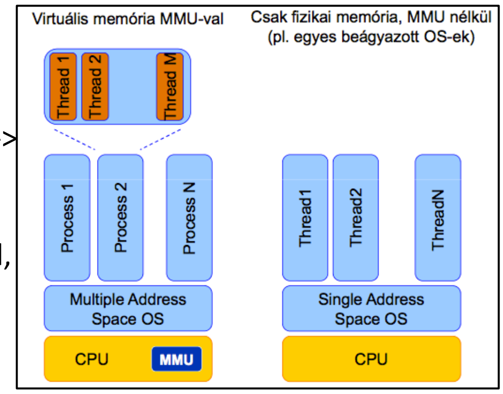
Szálak alkalmazásának elönyei:

- kis eröforrás idényüek a létrehozásuk és megszünetetésük
- alkalmazáson belüli többszálúság támogatása
- gyors kommunikáció közös memóriában az azonos folyamat kontextusában futó szálakra, csak a verem szál specifikus
- skálázhatóság

Szálak alkalmazásának következményei:

- a közös memórián keresztüli kommunikáció veszélyes, a kommunikációra használt memóriaterület konzisztenciája sérülhet
- megoldás: kölcsönös kizárás

HW támogatás ->



Coroutine és fiber (rost):

- kooperatív multitasking: folyamaton vagy szálon belül, ütemezési algoritmus a felhasználó kezében
- coroutine: programnyelvi szinten
    - subroutine általánosítása
    - subroutine: LIFO, egy belépési pont, több kilépési pont, verem használata
    - coroutine: belépési pont azonos a subroutine-nal, legutolsó kilépési ponttal tér vissza, átlépés utasítással történik, nem használhat vermet
- fiber: rendszerszintü eszköz
- verem alapú környezetben (C/C++) nehéz az implementációja a coroutine-nak
- nem szükséges osztani az eröforrásokon
- egyetlen végrehajtó egységet tudnak csak használni

8. UNIX folyamatok kezelése

Alapismeretek:

- folyamat: egy program futás alatt álló példánya
- mikor indulnak a folyamatok? : rendszerhíváskor, felhasználó által
- hogyan kezeljük a folyamatokat? : listázás (ps, top), vezérlés
- folyamatok elkülönítése a kerneltöl
    - végrehajtási mód: különbség a kernel és a folyamat programkódja között
    - kontextus: kernel és folyamat adatok közötti különbség
- végrehajtási (futási) módók:
    - kernel mód: védett (kernel) tevékenységek végrehajtása
    - felhasználói mód: a folyamat programkódjának végrehajtása
- végrehajtási környezet:
    - kernel kontextus: kernel saját feladatainak ellátásához szükséges adatok
    - folyamat kontextus (virtuális-memória kezelés): folyamat futásainak adatai, folyamat kezeléséhez, vezérléséhez szükséges adatok

Folyamatok futtatása:

- felhasználói mód:
    - folyamat kontextus: alkalmazás
    - kernel kontextus: nincs
- kernel mód:
    - folyamat kontextus: rendszerhívás
    - kernel kontextus: megszakítások, rendszer feladatok

A végrehajtási mód váltása:

- felhasználói mód és a kernel mód között átmenet lebonyolítása
- jellemzöen rendszerhívások meghívása esetén zajlik
    - a folyamat kernel módban végrehajtandó tevékenységet szeretne végezni
    - meghívja a megfelelö rendszerhívást
    - a libc kiadja a SYSCALL utasítást (megszakítast generál)
    - a CPU kernel módba vált a megszakítás hatására
    - a kernel SYSCALL kezelöje elökészíti a rendszerhívás végrehajtását
    - végrehajtódik a kernel módú eljárás
    - a kernel visszatér a megszakításból
    - a CPU felhasználói módba vált a visszatérési utasítás hatására
    - a libc visszatér a folyamat által meghívott függvényböl
- hardver megszakítások és kivételek esetén is kell módváltás

Rendszerhívások:

- nyomkövetése: strace parancs
- virtuális rendszerhívások
    - probléma: túl sok rendszerhívás, sok interrupt, sok kontextusváltas
    - az egyszerü esetekben próbáljuk lerövidíteni az utat: bizonyos kernel funkciókat felhasználói címtérben lévö eljárásokként teszünk elérhetövé
    - virtuális rendszerhívások(Linux): minden folyamat címterében megjelenik egy "kernel" lap, biztonságosnak ítélt rendszerhívások érhetöek el rajta

UNIX folyamatok kontextusa részletesebben:

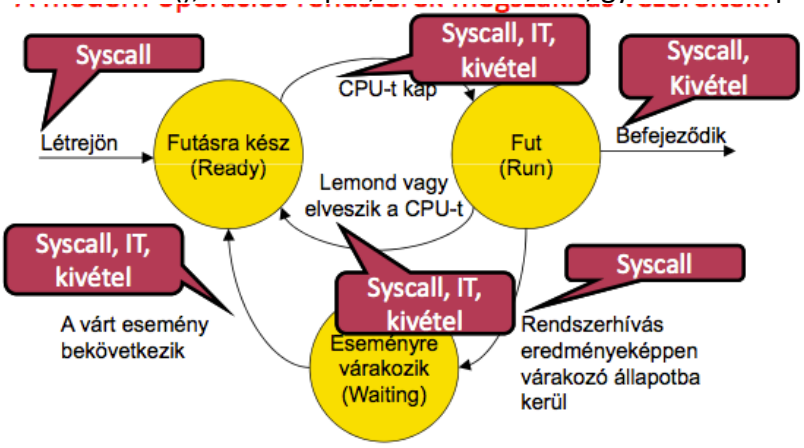
- folyamat adatok: programszöveg, adatok, vermek, ...
- hardver kontextus: cpu, mmu, fpu, ...
- adminisztratív adatok
  - folyamat futása során szükségesek (folyamat címtér része): hozzáférés-szabályozás adatai, rendszerhívások állapotai és adatai, nyitott fájj objektumok, számlázási és statisztikai adatok
  - folyamatok kezeléséhez szükségesek (kernel címtér része): azonosítók, futási állpot és ütemezési adatok, memóriakezelési adatok
- környezeti adatok (folyamat indításakor megörökölt adatok)

Folyamatok föbb adminisztratív adatai:

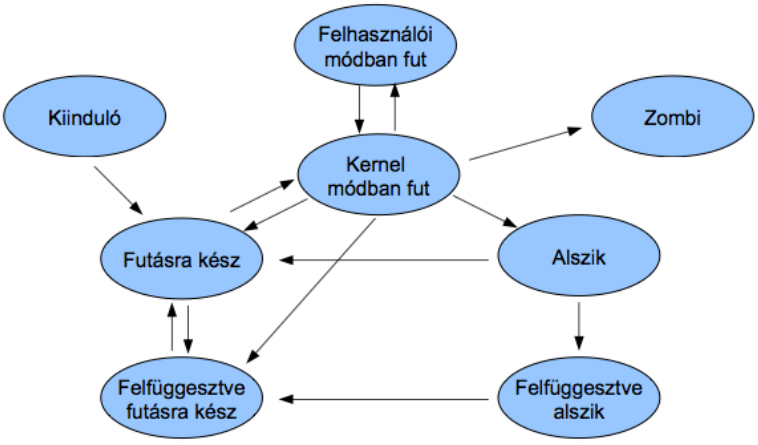
- PID (Process ID), PPID: szülö folyamat azonosítója
- folyamat állapota: fut, alszik, ... / ütemezési információk
- hitelesítök: UID(futtató PID), GUI(futtató csoport PID), valós/effektív azonosítók
- memória-kezelési adatok: címleképzési térkép
- kommunikációs adatok: fájlleírók, jelzés információk
- statisztikák: eröforrás használat

Folyamatok életciklusa:

- létrehozás: fork(), exec()
- futás elindítása, futás megállítása(várkozás, felfüggesztés, leállás, átütemezés)
- leállás: exit(), zombi állpot, szülö értesítése, gyerekek adoptálása



UNIX folyamatok állapotai



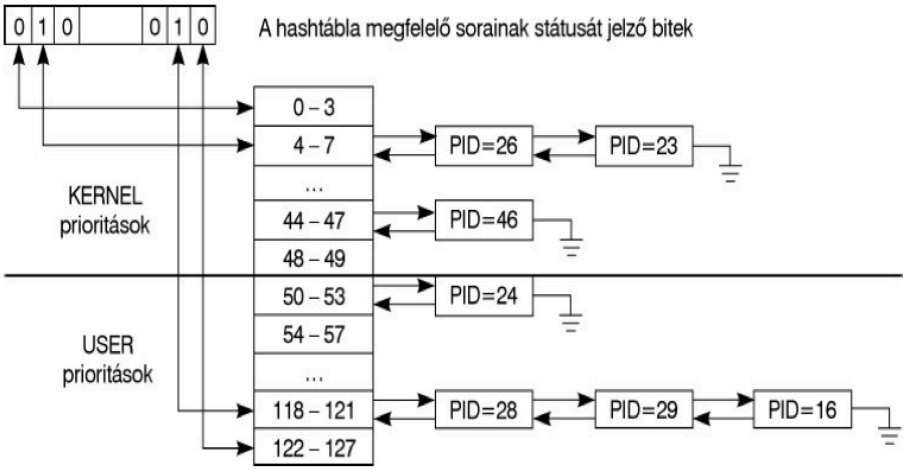
A fork() és exec() redszerhívások:
- a fork() eltérö értékkel tér vissza a szülö és a gyere esetében
- az exec() sikeres végrehajtás esetén nem tér vissza
- fork() variációk: clone():osztott címtér, vfork():exec, fork1():szál
- exec() variációk: exec1(), execv(), execle(), execve(), execvp(),...

A folyamatok családfája:
- folyamatot csak egy másik folyamat tud létrehozni, a szülö változhat
- a fork() megadja a szülönek a gyerek azonosítóját (PID)
- az ös folyamat (PID 1): rendszer leállásáig fut, örökli az árva folyamatokat
- a szülö értesítést kap a gyerek folyamat leállásáról

9. UNIX: folyamatok ütemezése

Feladatok ütemezése: (ism.)
- feladat: kiválasztani a futásra kész felakatok közül a következö futót
- alapfogalmak: preemptiv, mértékek(CPU kihasználtság, átbocsátó képesség, várkozási idö, körülfordulási idö, válszidö), prioritás, statikus/dinamikus
- müködés alapelve: task queue-ba rendezzük a feladatokat (FIFO)
- az ütemezö a sorokból választja ki a következö futó folyamatot
- követelmények: kis eröforrásigény, optimalitás, determinisztikus, fair, elkerüli a kiéhezést, összeomlás elkerülése

Tradicionális UNIX ütemezök:
- felhasználói módban: preemptív, idöosztásos, idöben változó prioritásos
- kernel módban: nem preemptív, nincs idöosztás, rögzített prioritás
- adatstruktúra: prioritás 0-127 között, 0 legmagasabb, 127 legkisebb, 0-49:kernel priotitási szintek, 50-127: felhasználói szintek
- az ütemezö a folyamatokat prioritásuk szerint 32 db sorba rendezi:



Ütemezés kernel módban:
- prioritás statikus, nem preemptív
- prioritás nem függ a felhsználói módban lévö prioritástól és CPU idötöl
- kernel típusú prioritást a folyamat elalvásásnak az oka határozza meg - alvási prioritás

Ütemezési felhasználói módban:
- a számított prioritás az ütemezés alapja
- ütemez minden óraciklusban: (van-e magasabb szinten folyamat)?(átütemez):(-)
- ütemez minden idöszelet végen (10 óraciklus): Round-Robin algoritmus

Ütemezés további adatai (prioritás számításhoz):
- p_pri a folyamat aktuális prioritása
- p_usrpri a folyamat felhasználói módban érvényes prioritása
- p_cpu a CPU használat mértékére vonatkozó szám
- p_nice a felhasználó által adott prioritás módosító értéke
- kernel módba váltáskor a prioritás értéke elmentödik a p_usrpri-ba

Prioritás számítása felhasználói módban:
- minden óraciklusban a futó folyamat p_cpu értékét növeli: p_cpu++
- minden 100. óraciklusban a kernel kiszámítja a prioritás értékét
  - minden folyamatnál "öregíti" a p_cpu értékét : p_cpu = p_cpu * KF
  - kiszámolja a prioritást: p_pri = P_USER + p_cpu/4 + 2*p_nice
- korrekciós faktor: KF = 1/2, 4.3 BSD: KF=2*load_avd /(2*load_avg +1)
  load_avg = futásra kész feladatok száma

Felhasználói módú ütemezés összefoglalója:
- minden óraütésnél:
  - ha van magasabb prioritási sorban folyamat, akkor átütemezés
  - a futó folyamat p_cpu értékének növelése (csökkenni fog a prioritása)
- minden 10. óraütésnél:
  - Round-Robin ütemezés a legmagasabb futási szinten lévö folyamatok között
- minden 100. óraütésnél
  - korrekciós faktor kiszámítása, az elmúlt 100 ciklus átlagos terhelése alapján
  - a p_cpu "öregítése"
  - a prioritás újraszámolása
  - szükség szerint átütemezés

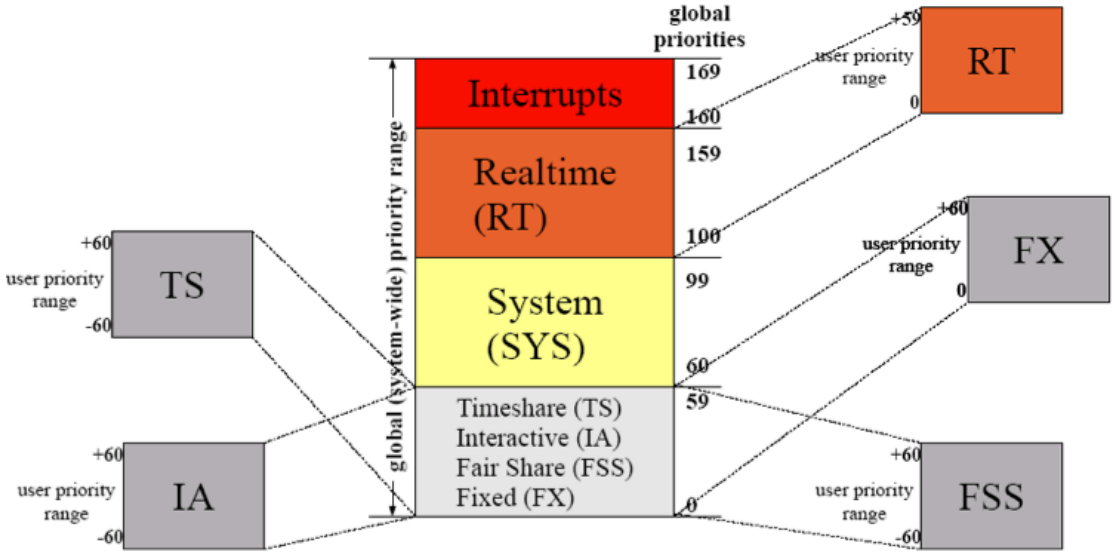
Ütemezési mintapéldák: (1.3)
http://home.mit.bme.hu/~micskeiz/opre/files/opre-feladatok-segedanyag.pdf

Tradicionális UNIX ütemezö értékelése:
+ egyszerü, hatékony
+ átlalános célú, idöosztásos rendszerben megfelelö
+ jól kezeli az interaktív és batch típusú folyamatok keverékét
+ elkerüli az éhezést
+ jól támogatja az I/O müveleteket végrehajtó folyamatokat
- nem skálázódik jól: sok folyamat -> sok számítás
- nem lehet folyamat (csoportok) számára CPU-t garantálni
- nem lehet a folyamatokat igényeik szerint eltéröen ütemezni
- nincs garancia a válaszidöre
- a többprocesszoros támogatás nem kidolgozott
- a kernel nem preemptiv

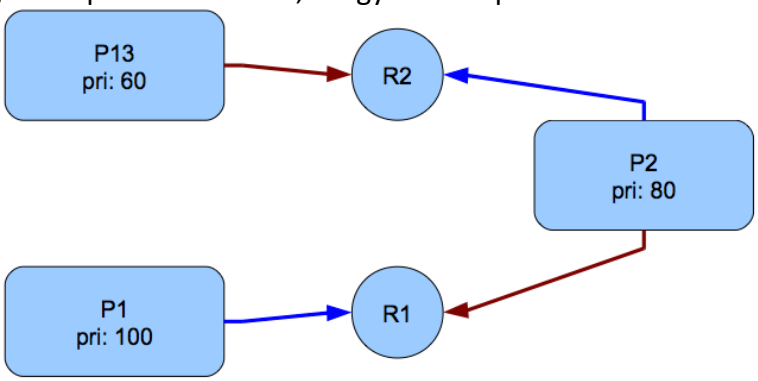
Modern UNIX ütemezök szempontjai:
- új ütemezés osztályok: speciális igényekre, "fair share", a kernelek is egyre több saját folyamatot(szálat) futtatnak, batch real-time kernel és interaktív folyamatok keveréke fut egy idöben, moduláris ütemezö rendszer szükséges
- kernel megszakíthatóság: többprocesszoros ütemezésnél elengedhetetlen
- az ütemezö eröforrásigénye: az ütemezés egyre összetettebb feladat, az ütemezési algoritmusok komplexitása lineáris, de inkább O(1) legyen
- szálak kezelése

Solaris ütemezö tulajdonságai:
- szálalapú, kernel preemptív, támogat többprocesszoros rendszert/virtualizációt
- többféle ütemezési osztály:
  - idöösztásos (TS): ütemezés alapja hogy ki mennyit várt/futott
  - interaktív (IA): TS + az éppen aktív ablakhoz tartozó folyamat kiemelése
  - fix priotirás (FX): állandó prioritású folyamatok
  - fair share (FSS): CPU eröforrások folyamatcsoportokhoz rendelése
  - real-time (RT): a legkisebb késleltetést biztosító legmagasabb szint
  - kernel szálak (SYS): az RT szint kivételével mindennél magasabb szint

A Solaris ütemezési szintjei:



Örökölt prioritások (Solaris): a prioritás inverzió problémája (kék nyíl:vár, piros:foglal)
Megoldás: prioritás emelés, avagy örökölt prioritások



Linux ütemezök története, tulajdonságai:
- elsö változatok (v2 elött) tradicionális ütemezöre épültek
- 2.4 elött:ütemezési osztályok:RT/nem-preemptív/normál, O(n) ütemezés, egy ütemezési sor, nem preemptiv kernel
- kernel v2.6: 0(1) ütemezés, processzoronkénti futási sorok, I/O és CPU terhelésü folyamatok heurisztikus szétválasztása

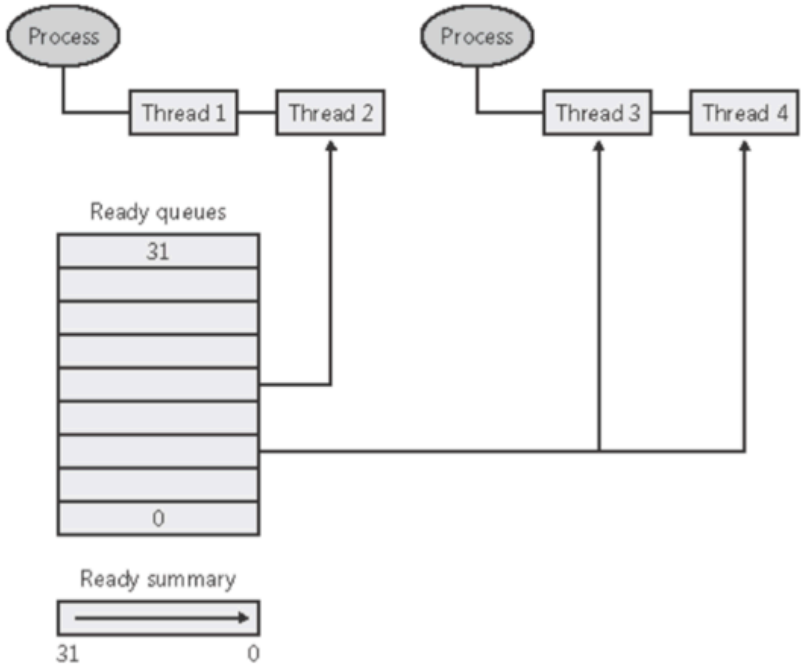
- 2.6.23: CFS (Completely Fair Scheduler): CPU idö biztonsításának kiegyensúlyozása egy speciális adatstruktúra segítségével (PF fa)

10. Ütemezés Windowsban

Ütemezési alapelvek:
- preemptív ütemezö (kernel és user módban is)
- 32 prioritási szint: legmagasabb fut mindig, azonosok között Round-Robin
- a szálak adott ideig futnak (quantum)
- nincs mindig futó központi ütemezö, ütemezést események indítják
- szálak prioritása válthozhat a futás során
- 31-16: 16" realtime"szint: nem igazi valós idejü,csak nincs prioritási szint változás
- 15-1: 15 dinamikus szint: OS változtathajta a prioritást futás közben
- 0: zero page thread: felszabadított memórialapok kinullázása
- idle szál(ak) - ha nincs futásra kész, üres idö számlálására

- szálak: 7 féle relatív prioritás
- I/O prioritás: Vista kernel módosítás, 5 féle prioritás az I/O kéréseknek, I/O sávszélesség foglalás

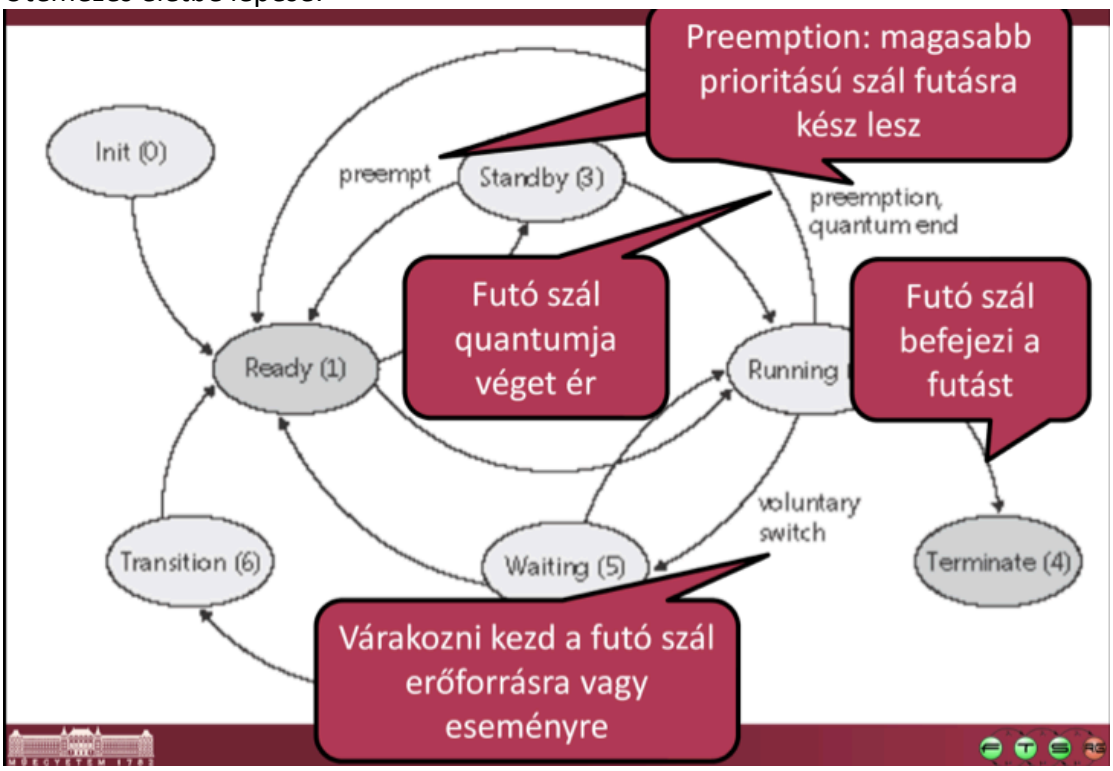
Várakozás sorok - kész szálak:



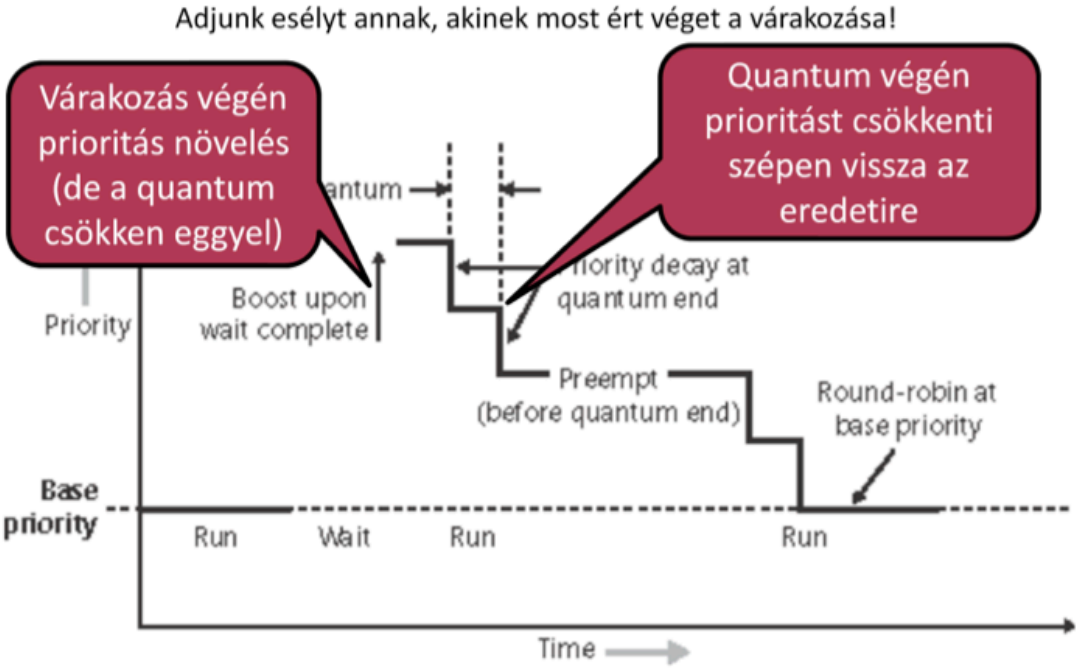
- kétszeresen láncolt listák a futásra kész szálakról, lista elejéröl veszi le az ütemezö a következö szálat -> nem O(n) komplexitású

Quantum: RR ütemezésnél az idöszelet
- óra megszaításban mérik (clock tick), tárolás: "3*clock tick száma, futó szálak quantumja 3-mal csökken minden óraütéskor (Vista elött
- Quantum hossza: kliens esetén 2-6 clock tick (elötérben lévök hosszabbak), szerver esetén mindenkinek 12

Ütemezés életbe lépése:



Prioritás módosítása:



Éhezés elkerülése:

- OS mp-enként megnézi a futásra kész szálakat

- aki nem futott már 300 óraütés óta: 15-ös prioritás, XP-n quantumját megduplázza, Serveren 4-re állítja, egy quantumnyi futásig

SMP (Symmetric Multiprocessing):
- minden CPU egyenrangú: közös memória címtér, IT-t bármelyik CPU kiszolgálhat
- CPU-k maximális sz'ma a registry-ben távolva
- implementációs limit(bitvektor hossza): 32 CPU 32bites, 64 CPU 64 bites
- Windows 7/Server 2008 R2 : logikai CPU csoportok, 4*46 CPU, NUMA támogatás

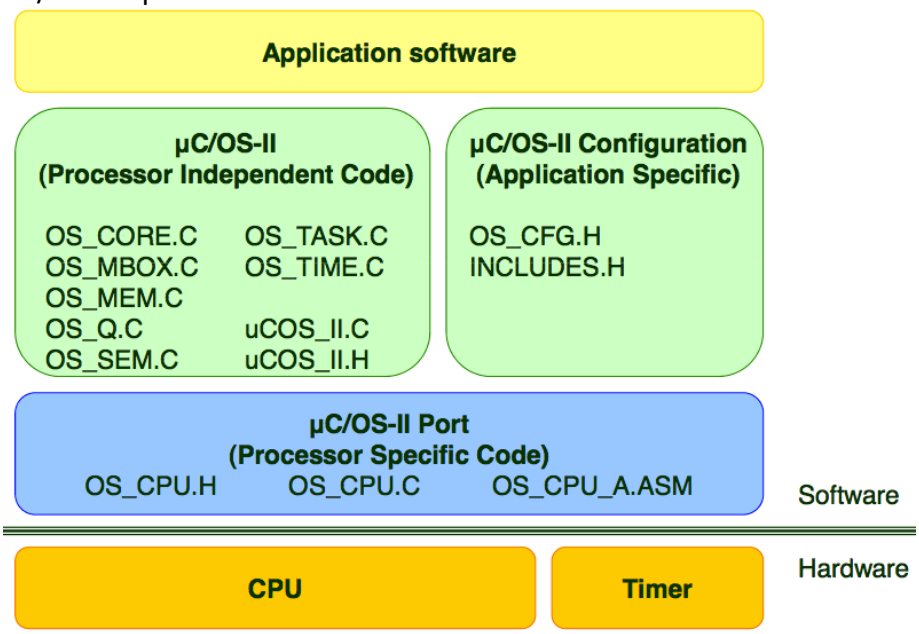
Multiprocesszoros ütemezés:
- szálak bármelyik CPU-n futhatnak, de megpróbálja az OS ugyanazon a CPU-n tartani ("soft affinity"), beállítható hogy melyiken futhat ("hard affinity")
- eloszott: bármelyik CPU bármelyik futást megszakíthatja
- ütemezési adatok tárolása:
    - Windows Server 2003 elött: rendszerszintü futásra kész sorok
    - Windows Server 2003-töl: CPU-nkénti sorok

Windows 7 módosítások:
- Core Parking(server): minél kevesebb logikai CPU használata, nem használt socketek standby üzemmódban
- time coalescing: azonos periódusú idözítök összevonása
- DFSS(Dynamis Fair Share Scheduling): remote desktop szerverekhez, minden munkamenet "CPU keretet" kap, ha elhasználja csak idle CPU-t kaphat
- globális zárak megszünetetés

11. uC/OS-II

Tulajdonságok:
- nyílt forráskód, hordozható, skálázható, multi-tasking, preemptív ütemezö
- determinisztikus futási idö, különbözö mértü stack-ek
- rendszer szolgáltatások: mailbox, queue, semaphore, idö...
- IT management, robosztus, megbízható
- jó dokumentáció, kiegészítö csomagok

uC/OS felépítése:



Taszk állapotai: Running, ISR(IT rutin), Waiting, Ready, Dormant("szunnyadó"), ha nincs futásra kész task, akkor az OSTaskIdle() fut

uC/OS ütemezöje:
- 2D bitman struktúrában tárolódnak a futásra kész taszkok
- (+): gyors beszúrás
- (-): taszkoknak egyedi prioritás -> RR kizárva, lookup táblázat a bitmap mellé
- 1. taszk kivétele a futásra készek közul
- 2. taszk futásra késszé tétele
- 3. legmagasabb prioritású, futásra kész taszk megtalálása
- 4. kontextus váltás
- az ütemezö lényegében egy függvény

Idözitö: hardver timer, ami periodikus megszakításokat generál
Egyéb szolgáltatások:
- taszt kezelés
- idö kezelés
- memória particiók kezelése
- szinkronizáció/kommunikáció

12. Feladatok együttmüködése (kölcsönös kizárás, szinkonizáció, kommunikáció)

Együttmüködés lehetöségei:
- közös memórián keresztül (RAM v. PRAM modell) : szálak esetén
- üzenetekkel

RAM modell:
- klasszikus Random Acess Memory (egy végrehajtó egység)
- tárolórekeszekböl áll, 1D-ban, rekeszenként címezhetö, írás és olvasás
- írás: teljes rekesztartalmat felülír, olvasás: nem változtatja a rekesz tartalmát

PRAM modell:
- Parallel Random Access Memory (sok végrehajtó egység)
- több végrehajtó egység olvashatja/írhatja párhuzamosan
- változtatások a RAM modellhez képest:
    - olvasás - olvasás => ugyanazt olvassák
    - olvasás - írás, írás - írás => versenyhelyzet
- gyakorlatban ezt használjuk

Eröforrás (resource):
- olyan eszköz amire a párhuzamos programnak futás közben szüksége van
- legfontosabb eröforrás a végrehajtó egység
- memória és annak tartalma, perifériák, ...

Közös eröforrás (shared resource):
- egy idöintervallumban több, párhuzamosan futó feladatnak lehet rá szüksége
- az eröforráson osztoznak a feladatok
- egy idöben egy vagy maximum megadott számú feladat tudja helyesen használni
- fontos feladat: felismerni a közös eröforrásokat, és helyesen használni
- az OS szolgáltatásokat nyújt probléma megoldására, de a megoldás a programozó kezében van

Kölcsönös kizárás (mutual exclusion):
- annak biztosítás, hogy a közös eröforrást egy idöben csak annyi magában szekvenciális feladat használja, amely mellett a helyes müködés garantálható
- a kölcsönos kizárást meg kell oldanunk a programban
- többnyire a használt eröforrást lock-oljuk (elzárjuk)

Kritikus szakasz (critical section):
- a magában szekvenciális feladatok azon kódrészletei, amely során a kölcsönos kizárást egy bizonyos közös eröforrásra biztosítjuk
- a kritikus szakaszt a hozzá tartozó eröforrásra atomi müveletként kell végrehajtani, nem megszakítható módon

Atomi müvelet (atomic operation):
- nem megszakítható müvelet,amelyet a processzor egyetlen utasításként hajt végre
- TAS, RMW, speciális CPU utasítások az IT tiltás/engedélyezés elkerülésére
- a közös eröforrás lock-olásst, kritikus szakasz megvalósítást atomi müveletekre vezetjük vissza

Közös eröforrások védelme:
- közös eröforrásokhoz hozzáférhetek: ISR(IT service routine), feladat, DMA
- lehetöségek: IT tiltás/engedélyezés, locking, ütemezö lista tiltása/engedélyezése
- a lock megoldás sokak szerint nem jó, de jobb mint bármi eddigi megoldás

Újrahívhatóság (reentrancy):
- a közös eröforrás problémájának egyfajta kiterjesztett esete egy függvényen belül is felléphet, amennyiben ezt a függvényt egyszerre többen is meghívhatják
- elöfordulása: - ugyanazt a függvenyt hivjuk taszkból is és egy IT rutinból is
  - ütemezés preemptiv, és ugyanazt a függvényt hívjuk két taszkból
- újrahívhatóság feltételei: újrahívott függvényben használt változók, perifériák, függvények, stb. közös eröforrásnak minösülnek, és ha azok, akkor azokat közös eröforrásként megfelelöen kezeli-e a függvény?
- pl:PC BIOS: nem újrahívhatóak,preemptiv OS API függvényei újrahívhatóak

Várakozás közös eröforrásra: más feladat által használt közös eröforrásra is "eseményre várakozik" állapotban várakozik a feladat, az OS nyújt a közös eröforrások védelmére szolgáltatásokat

Lock-olás és az ütemezö:
- a közös eröforrást használni kívánó feladat meg próbálja szerezni az eröforrást egy OS hívással
- az eröforrás szabad:
  - az OS lefoglalja az eröforrást, visszatér "fut"/"futásra kész" állapotba, majd CPU megkapás után a hívásból visszatérve tud tovább
  - használja az eröforrást
  - használat végén felszbadítja azt egy OS hívással
- foglalt eröforrás:
  - a feladat az adott eröforrást váró feladatok várakozasi sorába "eseményre vár" állpotba kerül
  - fut az ütemezö a következö futó feladat kiválasztásaára
  - ha egy másik feladat késöbb felszabadítja az eröforrást:
    - fut ismét az ütemezö, és kiválaszta az eröforrásra váró feladatok közül valamelyiket
    - annak számára lefoglalja az eröforrást
    - majd "futásra kész" állapotba helyezi azt
- lock-olás részletessége
  - kölcsönös kizárás megvalósítása eröforrás használattal jár, minimalizálni kell
  - túl nagy egységekben végzett kölcsönös kizárás is eröforrás pazarlással jár

Hibák:
- versenyhelyzet (race condition):
  - párhuzamos program lefutása során a közös eröforrás helytelen használata miatt a közös eröforrás vagy az azt használó program nem megfelelö állapotba kerül
  - a hibás állapot részletei erösen függenek azt használó szekvenciális részfeladatok lefutási sorrendjétöl
- kiéheztetés (starvation):
  - ha a párhuzamos rendszer hibás müködése miatt egy feladat soha nem jut hozzá a müködéséhez szükséges eröforrásokhoz, akkor ki van éheztetve
  - nem csak a CPU-ra, de más közös eröforrásokra is felmerülhet
- holtpont:
  -a közös eröforrások hibás beállítás vagy használata miatt a rendszerben a részfeladatok egymásra várnak
  - nincs futásra kész folyamat
  - nem jöhet létre belsö esemény
  - a rendszer nem tud elörelépni
- livelock:
  - többnyire a hibás holtpont feloldás eredménye
  - a rendszer folyamatosan dolgozik, de nem lép elöre

- prioritás inverzió (priority inversion):
  - prioritásos ütemezökben fordul elö,de az eröforrás használattól is összefügg
  - kedvezöbb esetben csak a rendszer teljesítménye csökken,a válszidök nönek
  - rosszabb esetben kiéheztetés, vagy akár holpont is lehet

Priotirás inverzió megoldások:
- Prioritás öröklés (PI, Priority inheritance)
  - az alacsony prioritású feladat megörökli az átlala kölcsönös kizárással feltartott feladat prioritását a kritikus szakaszából váló kilépésig
  - csak részben oldja meg a problémát
- Prioritás plafon (PC, Priority ceiling)
  - majdnem ugyan az, de az adott közös eröforrást használó legnagyobb prioritású feladat prioritását örökli meg
  - az adott eröforrást máskor használó többi feladat sem tud futni
  - az alacsony prioritású feladat akadály nélkül le tud futni
- modern beágyazott OS-ekben választható

Prioritás inverzió más szempontból:
- sokak szerint a prioritás inverzió alapvetöen egy rendszertervezési hiba eredménye
- nem speciális protokollokkal kell megoldani, hanem jól kell tervezni a rendszert

Hogyan lock-olunk:
- passzív várakozás az OS szolgáltatásaink felhasználásával
- aktív várakozás

Lock feloldásra várkozás passzívan:
- sleeplock, blocking call, ...
  - ütemezö által karbantartott várakozási sorok
    - ha az eröforrás nem lockolt, akkor megkapja az eröforrást a feladat lezárva, és fut tovább
    - ha az eröforrás lock-olt, akkor a feladat megy az eröforráshoz tarozó várakozási sorba, a futásra kész feladatok közül egy futó állpotba kerül, ezután ha az eröforrás felszabadul, akkor az eröforráshoz tartozó sor elején állo megkapja az eröforrást lezárva, és futásra kész állapotba kerül
  - eröforrás-takarékos, de van overhead-je
  - utána csak futásra kész sorba kerül a részfeladat, pontos idözítés nehezen megoldható, alsó korlátot ad
  - a processzor aludhat, ha nincs feladat: HW IT-re felébred
- livelock, spinlock, busy wait, spinning, ...
  - aktív várakozás az eröforrás felszabadulására,megszerzésére - CPU pazarlás
  - fogyasztás is nö, hiszem a CPU folyamatosan fut, nem tud aludni
  - Compiler optimalizáció kiszedheti/eltávolíthatja a kódot
  - függ a CPU sebességétöl

Hogyan várakozzunk?
- rövididejü várkozáshoz a spinlock elkerülhetetlen: garantáltan rövid idejü kölcsönös kizárási problémák kezelésére, periféria kezelés során
- HW Timer is használható adott idötartamú várakozásra, korlátolt számú, IT overheadje megjelenik, kombinálható aktív várakozással
- nehéz jó kompromisszumot találni
- az ütemezö nem használ a feladatok ütemezése során spinlock jellegü hívásokat
- maga az OS kernel viszont gyakran használhat

Probléma: adott idejü várkozás: delay(N)
- hívásra és futásra kész állapotba kerülés között eltelt idönek a felsö korlátját adja meg többnyire
- az OS rendszeróra óraütésnek a felbontásával dolgozik

Probléma: idö mérés:
- Ha az OS timer felbontása nem elég (1-10-20 ms)
- idöintervallum mérés két esemény között
  - szabadon futó Timer periféria értékének lekérdezése és különbség képzés
  - Timerstamp counter
  - események közötti idö, stb mérhetö vele
  - idömérés többprocesszoros/eloszott rendszerben

Eszköz RAM/PRAM modell esetén:
- kölcsönös kizárás megoldására: lock bit, szemafor, kritikusz szakasz obj., mutex
- minden OS-ben hasonló eszközöket fogunk találni
- tipikus hibák kivédésére és kölcsönös kizárási probléma megoldása: monitor

Lock bit:
- legegyszerübb forma, TRUE=használhat, FALSE=nem használhat eröforrást

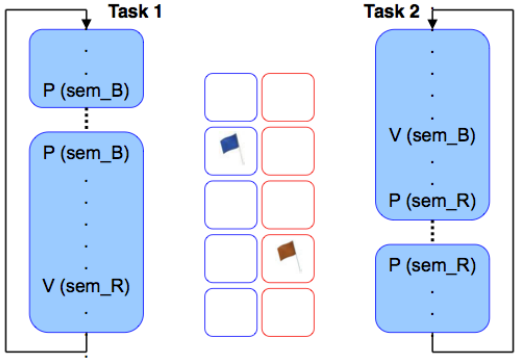
Atomi utasítások:
- a lock bit alkalmazása során egy súlyos hiba jelenhet meg:
  - IT a lock bit tesztelése során: a következö utasítás már nem fog lefutni, vagyis a kritikus szakaszba lépést nem jelezzük, pedig már abban vagyunk
  - az IT-ben vagy az utána futó más részfeladatokban az eröforrás szabadnak látszik
  - a védett eröforrás inkonzisztens állapotba kerülhet
- megoldás:
  - IT tiltás a teszt elött, IT engedélyezés a beállítás után

Szemafor:
- bináris és counter típusú: egy feladat/több feladat a kritikus szakaszban
- OS hívás, a bináris szemafor egy magas szintü lock bit
- két müvelet értelmezett rajta: belépés, kilépés
- a counter típusú esetén a belépés és kilépést számoljuk
- ha egynél több eröforrásra lenne szükségünk,akkor azokat vagy egyben mind megkapjuk, vagy a töredékeket nem foglaljuk le

Kritiks szakasz objektum és Mutex:
- bináris szemafor szerüen müködnek
- kritikus szakasz objektum: CiritcalSection obj: létrehozás, Enter(), Leave()
- Mutex (Mutal Exclusion): Acquire(), WaitOne(), Release()

Miért lock-olunk még?
- kölcsönös kizárást lezártuk
- részfeladatok közötti szinkronizáció visszavezetése kölcsönös kizárásra: nincs védendö objektum igazából, részfeladatok együttmüködése a cél
- pl.: randevú
  - két vagy több feladat összehangolt végrehajtása
  - ebben az esetben az OS által nyújtott szolgáltatásokat használunk, feladatok paszívan várnak egymásra
  - a szemaforok alapesetben foglaltként vannak inicializálva ebben az esetben
- memórián keresztül történö kommunikáció

Kétoldalú randevú szemaforral, B és R foglaltként inicializálva:



Kommunikáció: Védett memóriaterület:
- szálak közötti kommunikáció során:
    - folyamatok így nem tudnak kommunikálni (MMU)
    - UNIX SystemV shared memory nem valódi osztott memória, OS szolgáltatás, folyamatok közötti kommunikációt tesz lehetövé
- tömb, struktúra, objektum
- egy vagy kétirányú kommunikáció
- kölcsönös kizárást lock-bit, szemafor, mutex,kritikus szakasz objektum oldja meg

Lock-olás során tipikusan elkövetett hibák:
- belépés/kilépés elmarad
- többszöri be-/kilépés
- más eröforrás lefoglalása
- az eröforrás indokolatlanul hosszan történö lezárása
- prioritás inverzió
- deadlock, livelock

Monitor (hibák elkerülésésre):
- lokalizáljuk a lock-olással kapcsolatos feladatokat a közös eröforrást körülvevö API-val
- monitor fejlödése: Hoare és Mesa szemantika (müködés/jelentés): Mesa programozási nyelv például
    - Hoare szemantika: azonnal az eröforrást megszerzö részfeladat fut, eröforrás igényes, és nem kompatibilis a preemptiv ütemezökkel
    - Mesa szemantia: a futásra kész feladatok közé kerül és késöbb az ütemezö futtaja

Szemafor, Kritikus szakasz objektum, Mutez OS-en belül kerül megvalósításra:
- Egy folyamaton belül futó szálak kommunikációja közös memórián keresztül (gyors, alacsony erőforrás igény).
- kölcsönös kizárás és szinkronizáció üzenetekkel
- van overheadje: próbálkozások

13. Feladatok együttmüködése (üzenet alapú kommunikáció)

Üzenetek:
- nem azonos a hálózaton küldött üzenettel, annál általánosabb
- lehet:rendszerhívás,TCP/IP kapcsolat vagy üzenet gépen belük/gépek között
- többnyire az alkalmazás kódjában OS api függvényként jelenik meg
- OS valósitja meg szolgáltatásiaval

Üzenettovábbítás tulajdonságai:
- nagyobb késleltetés, kisebb sávszélesség
- a csatorna nem megbízható elosztott rendszerek esetén: a közös memória 1 valószínüséggel megbízható, de az OS-en belül történö üzenetküldésre ez nem igaz (túlterhelés), a számítógép hálózaton történö üzenetküldés definíció szerint nem megbízható
- minimum egy küldö folyamat
- minimum egy vevö folyamat: vétel után törlödik/megmarad
- lehet tulajdonosa: OS / folyamat

Üzenetek címzése:
- számítógép hálózatok...
- egy adott folyamat (unicast cím), egy folyamat (anycast cím)
- minden folyamat (broadcast cím): teljesítmény menedzsment események
- folyamatok csoportja (multicast cím)

Direkt kommunikáció:
- szimmetrikus üzenet alapú kommunikáció
  - send(P,message), recieve(Q,message), P/Q folyamat azonosítók
- asszimetrikus üzenet alapú kommunikáció
  - send(P,message), recieve(id,message), id visszatérési érték, bárkitöl fogadhat

Indirekt kommunikáció:
- köztes szereplö: postaláda (mailbox), üzenetsor, port, ...
- interface: létrehozás és megszüntetés + send(A,msg), recieve(A,msg), A:postaláda

Blokkolás:
- nem blokkoló hívás = aszinkron
  - az eredmények és mellékhatások visszatéréskor még nem jelentkeznek
  - csak a végrehajtás kezdödik el a hívásra
  - visszatérési érték kezelése csak más értesítés után lehetséges
- blokkoló hívás = szinkron
  - az eredmények és mellékhatások a visszatérés után jelentkeznek
  - visszatérési érték kezelés egyszerü

Blokkolás a küldö oldalon:
- blokkoló send(): visszatér ha az üzenetet vették/postaládába került/idötúllépés

- nem blokkoló send(): azonnal vissza tér, és értesül hogy megkapták-e

Blokkolás a fogadó oldalon:

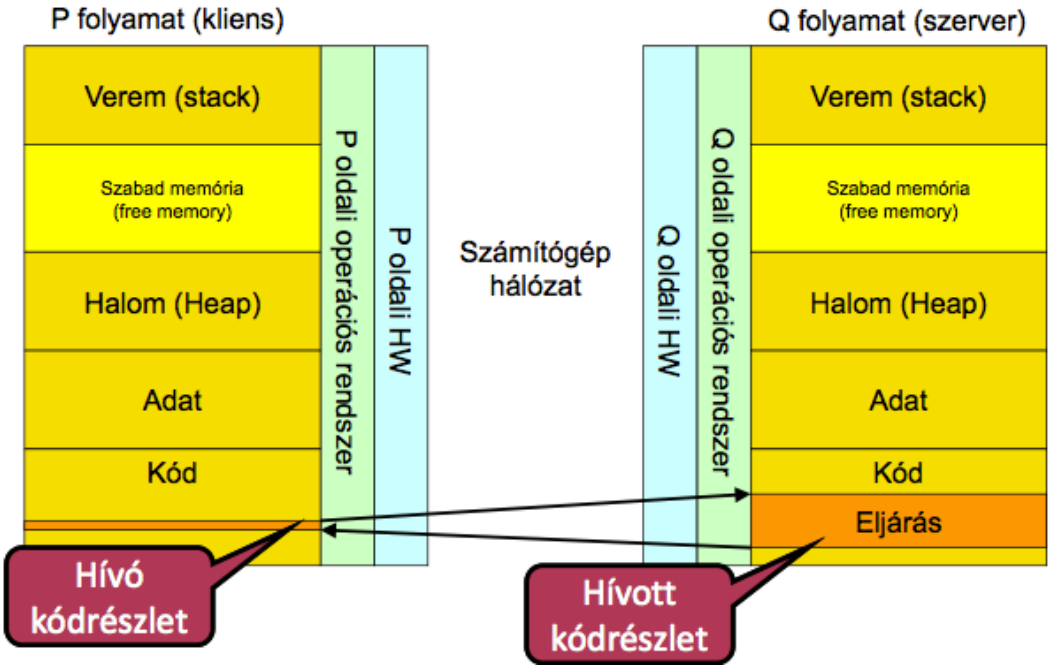
- blokkoló recieve(): nem tér vissza amíg nem érkezik üzenet, pl.: listen()
- nem blokkoló recieve(): azonnal visszatér ha van üzenet, ha nincs üzenet akkor azt jelzi, ha nincs üzenet akkor végtelen ciklusban arra vár

Implementációk:

- mailbox
  - indirekt kommunikáció
  - egy vagy több üzenet tárolása, de véges számú
  - OS szintü támogatás
- MessageQueue
  - indirekt kommunikáció
  - szinte végtelen számú üzenet tárolására alkalmas
  - üzenet alapú middleware-ek
- TCP/IP TCP vagy UDP port:
  - direkt kommunikáció
  - socket interface
  - gépen belül a localhost-on (127.0.0.1/8)
  - alacsony szintü, számos más middleware alapul rajta:
    - távoli eljárás hívás
    - távoli metódus hívás
    - üzenet alapú middleware-ek
- különbözö folyamatok és csövezetékek
- System V Shared Memory (UNIX, Linux): direkt, memóriaszerü interface

Távoli eljárás hívás (Remote Prucedure Call, RPC):

- alapja a napjainkban használt távoli metódus hívást alkalmazó megoldásnak
- egy másik folyamat memóriaterületén elhelyezkedö függvény hívás
- a hívó fél blokkolva vár a távoli hívás lefutására
- a meghívott függvény az öt tartalmazó folyamat egyik vezérlési szálaban fut



Az RPC a programozó számára:
- hívása azonos egy "lokális" függvény hívással: lényegében egy programkönyvtárban található függvény
- a tényelegesen végrehajtott függvény megírása sem különbözik egy lokális végrehajtandó függvény megírásától

RPC müködése:
- típusos a hívás és a visszatérési érték
  - struktúrált adatküldés
  - platform függetlenség megvalósítása az OS és a környezet feladata
- a kliens program egy teljesen normális lokális függvényt hív meg
  - automatikusan generált csonk (stub)
  - feladata a kommunikáció részleteinek elrejtése a hívó fél elöl
  - feltételezzük hogy a hívó fél megtalálja a szervert
- a kiens oldali csonk feladata a hívás során:
  - a hívási paraméterek összecsomagolása platform független formára és elküldése a végrehajtó folyamatnak
  - ehhez felhasználja a lokális OS és a hálózat szolgáltatásait
- a szerver oldalon az RPC-t használó szolgáltatás megkapja a szabványos formátumban megérkezö hívást
  - platform független formától lokális formára hozás
  - lokális függvény hívása
  - visszatéréskor a visszatérési érték platform függetlenné alakítása
  - a visszatérési érték küldése a kliensnek
- a kliens oldali csond feladata a hívás visszatérése során:
  - a szerverröl megkapott visszatérési érték konvertálása lokális formára
  - visszatérés a lokális csonkból a visszatérési értékkel
- a kliens szár az RPC hívás során várakozik
  - eseményre várkozik a hívás során
- a szerver szál vár a beérkezö hívásokra, és amikor van kiszolgálandó hívás, akkor fut, eseményre várkozik a hívás beérkezéséig

14. Feladatok együttmüködése (holtpont)

Holtpont (deadlock)

- a holtpont a legsúlyosabb hiba, ami multiprogramozás során elöfordulhat
- Def.: egy rendszer feladatainak H részhalmaza holtponton van, ha a H halmazba tartozó valamennyi feladat olyan eseményre vár, amelyet csak egy másik, H halmazbeli feladat tudna elöállítani
- nehéz felismerni: versenyhelyzet formájában fordul elö
- szükséges feltételek:
  - 1. Kölcsönös kizárás (Mutal Exclusion): vannak olyan feladatok melyeket csak kizárólagosan lehet használni, de azonat több feladat használná
  - 2. Foglalva várakozás (Hold and Wait): olyan feladat mely foglalva tart eröforrásokat, miközben más eröforrásokra várkozik
  - 3. Nincs eröszakos eröforrás elvétel a rendszerben
  - 4. Körkörös várakozás

Holtpont kezelése:

- *1. holtpont észlelése és feloldása*
  - észlelés:
    - megkülönböztetjük az egypéldányos/többpeldányos esetet
    - egypéldányos:Wait-for gráf:azt rögzítik melyik feladat vár melyikre
    - többpéldányos: Coffmann-féle holtpontdetektáló algoritmus
    - mikor fusson az algoritmus?
      - egyik szélsöség: amikor az eröforrás kérelem nem kielégíthetö azonnal
      - másik szélsöség: periodikusan, alacson CPU kihasználtság idején, amikor sor feladat vár eröforrásra
    - gyakorlatban használt algoritmusaok nagy futás idejüek
  - feloldás
    - többféle megoldás: radikális, kiméletes
    - feladatok: áldozatok kiválasztása, feladatok visszaállítása
    - el kell kerülni hogy ne mindig ugyanazt a feladatot állítsuk vissza

- *2. holtpont megelözése tervezési idöben*
  - foltpont feltételeiböl legalább egy ne teljesüljön
  - nincs futás idejü eröforrás használat
  - foglalva várakozás kizárása
    - az eröforrást birtokló feladat kér újabb eröforrást
    - minden szükséges eröforrást rendszerhívással egyben kell foglalni
    - eröforrás kihasználás romlik
  - eröszakos eröforrás elvétel lehetövé tétele: eröforrás menthetö állapottal, nem kell rollback feladat szinten
  - körkörös várakozás elkerülése
    - fejlesztés során az eröforrás használat korlátozása
    - teljes sorrendezés algoritmus

- *3. holtpont elkerülése futási idöben*

- eröforrás igény kielégítése elött mérlegelni kell, hogy az eröforrás igény kielégítésével nem kerülhet-e holtpontba a rendszer? Bankár algoritus

Bankár algoritmus, feltételezések:
- N feladat, M eröforrás (többpéldányos)
- a feladatok elözetesen bejelentik hogy az egyes eröforrásokból maximálisan mennyit használnak futásuk során: MAX NxM-es mátrix
- a feladatok által lefoglalt eröforrások száma: FOGLAL NxM-es mátrix
- szabad eröforrások száma: SZABAD M vektor
- MAXr az egyes eröforrásokból rendelkezésre álló maximális példányszám
- FOGLALr az egyes eröforrásokból foglalt példányszám
- egy feladat által még maximálisan bejelenthetö kérések száma: MÉG = MAX - FOGLAL NxM-es mátrix
- a feladatok várakozó kérései: KÉR NxM-es mátrix

Bankár algoritmus, müködése:
- 1. lépés: Ellenörzés (van-e elég eröforrás)
- 2. lépés: Állapot beállítása
- 3. lépés: Biztonság vizsgálata
  - 1. lépés: Kezdöérték beállítása
  - 2. lépés: Továbblépésre esélyes folyamat keresése:
    - a maximális igény kielégíthetö a jelenlegi eröforrásokkal
    - ha igen: lefuthat, és az eröforrásai visszakerülhetnek SZABAD-ba
    - ha van még feladat, akkor 2. pont, egyébként 3. pont
  - 3. lépés: Kiértékelés: feladatok listája, amelyek holtpontra juthatnak
- 4. lépés: Ha nem biztonságos a rendszer, az állapot visszaállítása

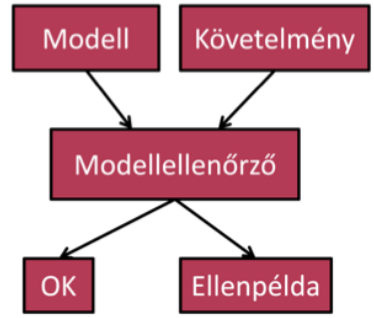
Bankár algoritmusra feladatok: (2.1)
http://home.mit.bme.hu/~micskeiz/opre/files/opre-feladatok-segedanyag.pdf

Bankár algoritmus értékelése:
- a holtpont lehetöségét bizonyítja, nem biztos hogy holtpontra is
- az algoritmus a legkedvezötlenebb esetet vizsgálja
- MAX mátrix nem mindig áll rendelkezésünkre

Holtpont kezelése a gyakorlatban:
- többnyire tervezési idöben történik, holtpont megelözése
- a holtpont észlelése és megszüntetése többnyire humán operátorokkal
- okok: a futás idejü algoritmusok komplexek, eröforrás-igényesek, és a gyakorlatban nem feltétlenül állak rendelkezésre a futásukhoz szükséges adatok

15. Windows: Feladatok együttműködésének ellenőrzése

Lehetnek-e ketten egyszerre a kritikus szakaszban?
- Hyman algorimusa
- Peterson algoritmusa



Algoritmusok helyességének ellenőrzése:
- **Modellezökkel**
  - modell: - rendszer müködésének leírása
    - tipikusan állpotgép szerü
  - követelmény:
    - mit akarunk ellenörizni (holtpont, kölcsönös kizárás)
    - logikai kifejezés
  - modellezö: fekete doboz, automatikus, TRUE/FALSE
- **UPPAAL modellezö:**
  - globális változók, automata (állapot, átmenet, órák), rendszer
  - átmenet kiválasztása, változók állapota, automaták képe
  - trace: szöveges/grafikus
  - követelmény: logikus forma
  - elemei: állapotra hivatkozás, NOT, AND, OR

Példa: Étkezö filozófusok.

További eszközök:
- *Java Pathfinder*
- *CHESS*: .NET, párhuzamosságból adódó hibákra
- *jchord:* java, versenyhelyzet és holtpont detektálásra
- *Static Driver Verifier*: Windows eszközmeghajtók ellenörzésére

16. UNIX folyamatok kommunikációja

Ismétlés:

- kernel
- felügyel a folyamatokat, menedzseli az eröforrásokat
- elkülöníti egymástól a folyamatokat (külön virtuális gép)
- réteges, moduláris felépítésü
- a folyamatok a kernellel rendszerhívásokon keresztül kommunikálnak

- folyamatok
- a felhasználoi programok futás alatt álló példányai
- UNIX alatt szülö-gyerek kapcsolatban állnak egymással

- kommunikáció elmélete
- közös memórián keresztül, üzenetekkel, távoli eljárással (RPC)
- eröforrások védelme, kritikusz szakasz, szemafor
- blokkoló (szinkron), nem blokkoló (aszinkron)

UNIX jelzések:

- cél: egy folyamat értesítése
  - a kernelben, más folyamatokban, önmagában bekövetkezett eseményröl
  - szinkronizálása más folyamatokhoz
- jelzés típusa (SIGINT, SIGCHLD, SIGKILL)
  - rendszer: kivételek, hibák, kvóta, riasztás, értesítés
  - felhasználói: emberek (CTRL+C,CTRL+V), folyamatok
- a müködés áttekintése
  - jelzést keltése (rendszerhívással / esemény bekövetkeztéve)
  - a kernel értesíti a címzetteket a jelzésröl
  - a címzett egy jelzéskezelö eljárásban fogadja a jelzést
- problémák a megválósítással:
  - a keltés és a kézbesítés idöben szétválik
  - sokféle implementáció, némelyik nem túl megbízható
- jelzés keltése

```
#include <signal.h>
kill(pid, SIGUSR1); // jelzés küldése
```

- jelzések kezelése:
  - többféle kezelési eljárás lehetséges, bizonyos keretek között állítható
    - core: core dump és leállás (exit())
    - term: leállás (exit())
    - ign: figyelmen kïvül hagyás
    - stop: fefüggesztés
    - cont: visszatéreés a felfüggesztett állapotból
  - az alkalmazás saját kezelö függvényt is megadhat

```
void alarm(int signum) { ... }
signal(SIGALRM, alarm);
```

  - a jelzés típusától függ, hogy a fentiek közül mi az alapértelmezett, illetve minek a beállítása engedélyezett: pl.: a SIGKILL nem hagyható figyelmen kívül és nem definiálható rá jelzéskezelö függvény

UNIX jelzés példa:

```
#include <signal.h>                 // signal(), kill()
#include <unistd.h>                 // getpid()
#include <sys/types.h>              // pid_t
pid_t pid = getpid();               // saját PID
kill(pid, SIGSTOP);                 // STOP jelzés küldése
signal(SIGCLD, SIG_IGN);            // nem foglalkozunk a gyerekekkel
signal(SIGINT, SIG_IGN);            // nem foglalkozunk a CRTL+C jelzéssel
signal(SIGINT, SIG_DFL):            // alapértelmezett kezelö beállítása
void alarm(int signum) { ... }      // az eljárás
signal(SIGALARM, alarm);            // jelzéskezelö függvény beállítása
alarm(30);                          // alkalmazás: ALARM jelzés 30 mp múlva
```

UNIX csövezetékek: pipe()

- cél: folyamatok közötti adatátvitel
- jellemzök:
  - csak szülö-gyerek (leszármaztatott, testvér) viszonylatban
  - adatfolyam (nincs üzenethatár, tipizálás)
  - egyirányú adatfolyam (egy/több író -> olvasó)
  - limitált kapacitás
- a megvalósítás:
  - egy folyamat létrehoz egy csövezetéket (pipe())
  - a kernel létrehozza az adatstruktúrát és olvasási/írási leírókat ad vissza
  - (a folyamat továbbadja a leírókat a gyerekeinek)
  - a leírók segíségével kommunikálnak a folyamatok(read(), write())
- korlátok, problémák:
  - nincs címzés, tipizálás, üzenethatár
  - csak a "rokonságban" müködik

UNIX elnevezett csövezetékek (named pipe, FIFO):

- az egyszerü csövezetékek legkomolyabb problémájának megoldása
  - független folyamatok kommunikációja
  - avagy egy már létezö csövezeték elérése egy másik folyamat által
- jellemzök
  - nem csak szülö-gyerek viszonylatban müködik
  - ugyanúgy müködik, mint a csövezeték
  - a létrehozás a csövezeték segítségével történik
  - lehetséges a kétirányú kommunikáció (olvasás és írás)

UNIX System V IPC:

- cél: folyamatok közötti "szabályos", egszséges kommunikáció
  - adatátvitel és szinkronizáció
- közös alapok (fogalmak)
  - eröforrás: a kommunikáció eszköze
  - kulcs: azonosító az eröforrás eléréséhez
  - közös kezelöfüggvények: *ctl(), *get(... kulcs ...)
  - jogosultsági rendszer: létrehozó, tulajdonos és csoportjaik / a szokásos hozzáférés-szabályozási rendszer(felhasználó, csoport, mások)

- eröforrások: szemaforok, üzenetsorok, osztott memória

UNIX System V IPC: szemaforok:

- cél: folyamatok közötti szinkronizáció: P()/V(), szemaforcsoportok kezelése
- használat:

  sem_id = semget(kulcs, szám, opciók);

  - adott számú szemaforhoz nyújt hozzáférést (adott kulccsal és opciókkal)
  - az ops struktúrában leírt müveletek végrehajtása

    status = semop(sem_id, ops, ops_méret);
  - egyszerre több müvelete, több szemaforon is végrehajtható
  - blokkoló és nem blokkoló P() operáció is lehetséges
  - egyszerü tranzakciókezelésre is lehetöség van

UNIX System V IPC: üzenetsorok:

- cél: folyamatok közötti adatátvitel:
  - diszkét/tipizált üzenetek
  - nincs címzés, üzenetszórás
- használat:

  msgq_id = msgget(kulcs, opciók);

  - adott kulcsú üzenetsorhoz nyújt hozzáférést (adott opciókkal)
  - üzenetküldés: msgsnd(msgq_id, msg, méret, opciók);
  - vétel: msgrcv(msgq_id, msg, méret, típus, opciók);
  - a típus beállításával szürést valósíthatunk meg:

    = 0 a következö üzenet

    > 0 a következö adott típusú üzenet

    < 0 a következö üzenet, amelynek a típusa kisebb-egyenlö

UNIX System V IPC: osztott memória:

- cél: folyamatok közötti egyszerü és gyors adatátvitel
  - a kernel helyett közvetlen adatátviteli csatorna
  - a fizikai memória elkülönített része, amely közösen használható
- használat:

  shm_id = shmget(kulcs, mérte, opciók);

  - adott kulcsú osztott nyújt hozzáférést (adott opciókkal)
  - hozzárendelés saját virtuális címtartományhoz: az adott változót hozzákötjük a visszakapott címhez

    változó = (típus) shmat(...);
  - lecsatolás: shmdt(cím);
  - a kölcsönös kizárást meg kell valósítani (pl: szemaforokkal)

UNIX "hálózati" (socket) kommunikáció:

- cél: címzéssel és protokolokkal támogatott adatátvitel
  - tetszöleges folyamatok között kliens - szerver akchitektúra
  - többféle célra (folyamatok közötti / gépek közötti)
  - sokféle protokollt támogat
  - többféle címzés
- fogalmak
  - hálózati csatoló avagy azonosító (socket): a kommunikácio végpontja
  - IP cím és portszám

- használat
  sfd = socket(domén, típus, protokoll)
  szerver: bind(sfd, cím, ...);
  kliens: connect(sfd, cím, ...);
  szerver: listen(sfd, sor_mérete);
  szerver: accept(sfd, cím, ...);
  send(sfd, üzenet, ...);
  recv(sfd, üzenet, ...);
  shutdown(sfd);

Távoli eljárás UNIX módra: (SUN) RPC:

- a socket kommunikációra épülö "elosztott rendszer" infrastrutúra
- cél:
  - folyamatok közötti magas szintü kommunikáció
  - távoli eljárások meghívása
  - programozói támogatás: interfész leírás + programgenerálás
- fogalmak
  - RPC nyelv: a hívható eljárások és típusaik (interfész) leírása
  - azonosítók: a leírásban megadott egyedi számok (program, eljárás)
  - portmapper: a programazonosítók és a hálózati portok összerendelése
  - rpcgen: a leírásból C programkódot generaló program
- a Sun RPC technológia részei:
  - interfész leírás
  - programgenerátor
  - kommunikációs infrastruktúra

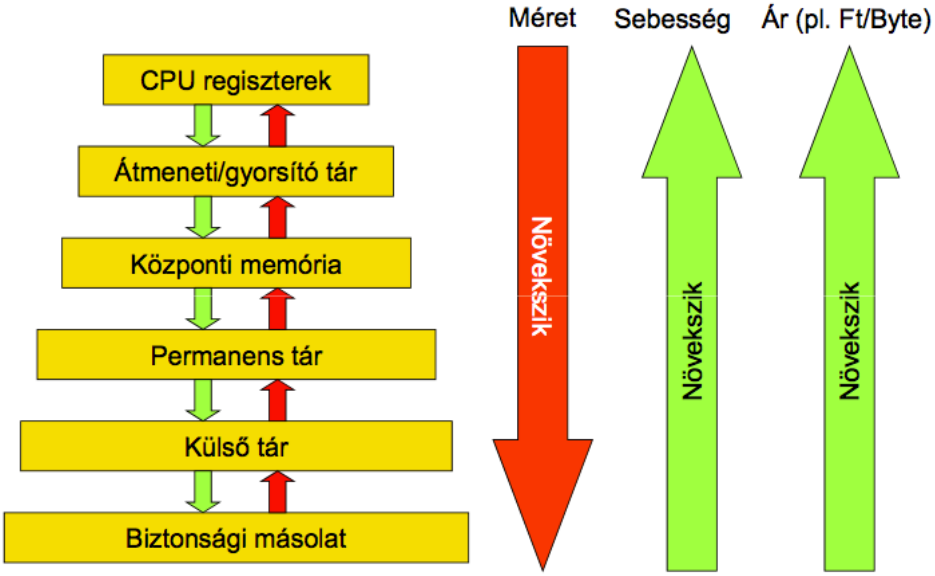
RPC interfész leírás és programgenerátor:

- RPC nyelv (date.x)

```
program DATE_PROG{
    version DATE_VERS{
        long BIN_DATE(void) = 1;        // eljárás azon = 1
        string STR_DATE(long) = 2;      // eljárás azon = 2
    } = 1;                              // verziószám = 1
} = 0x31234567;                         // program azon= 0x31234567
```

- kódgenerátor: rpcgen
  - rpcgen date.x eredményei:
    date.h: adattípusok deklarációja
    date_clnt.c: a kliens kódjában felhasználható date_... függvények
    date_src.c: a szerver date implementációját meghívó függvények
    (...)

17. Memóriakezelés



CPU regiszterek:
- jellegzetes D tároló: 10-100 gépi szó, nem átlalános felhasználású egyik része
- sebesség: az utasítás végrehajtása alatt akár többször is elérhetö (pl: ADD)
- ár: nehezen kategorizálható, architektúra függö
- fejeltö memória (tápfeszültség)

Átmeneti/gyorsító tár (Cache):
- jellegzetes SRAM, többnyire többszintü (1.:64-128KB|2.:1-8MB|3.:4-32MB)
- sebesség: n*10GB/s (1.: egy/néhány órajel ciklus|2-3.: 10-n*10 órajel ciklus)
- ár: magas, az SRAM cella nagy félvezetö területet foglal
- felejtö memória (tápfeszültség nélkül)

Központi (fizikai) memória:
- jellegzetes DRAM, n*10 MB - n*100 GB
- sebesség:
  - memory wall, max. és random access
  - max.: n*1 GB/s - 10 GB/s
  - késleltetés: n*10 ns (worst case)
- ár: erösen sebesség, méret, technológia és piac függö
- felejtö memória: tápfeszültség + idö

Permanens/háttér tár (HDD, Flash):
- HDD/flash memória, mérete: n*1MB - n*100 TB
- fájl alapú elérés: blokk elérésü eszköz, fájl-ként látjuk a tartalmat,kivéve NOR
- sebesség:
  - max. és random access más, olvasás és írás más
  - max.: n*10MB - n*1GB
  - késleltetés: n*10 ns - 10 ms
- ár: erösen sebesség, méret, technológia és piac függö
- nem felejtö, de meghibásodhat: HDD: MTBF, Flash: véges számszor írható

További lépések:
- központi memória kezelése
  - fizikai, logikai, virtuális memória fogalma
  - ezeknek a megfeleltetése és kezelése
- permanens tár kezelése
  - a virtuális tárkezelésen keresztül kapcsolódik majd a központi memória kezeléséhez
  - a permanens tár kezelése, szerkezete, partíció, fájlrendszer, fájl fogalma
  - RAID, NAS, SAN

Memória:
- a CPU használja: utasítások betöltése és adatok írása/olvasása
- egymás után következö memória müveletek folyamáról van szó
  - olvasások és írások adott sorrendben
  - ahogy ezt a program kód elöirja
- folyamat támogatás (szeparáció)
- tekintsünk el az átmeneti tárolótól (cache): hozzáférés sebességén befolyásol

Logikai cím (logical address): A CPU generálja a folyamat futása közben.
Fiziki cím(physical address): egy adott memória elem címe, ahogy az a fizikai memória buszon megadásra kerül a memória vezérlö által.
Logikai/Fizikai címtartomány: egy adott folyamathoz tartozó logikai/fizikai címek összessége
Logikai cím = Virtuális cím, ha a fizikai és logikai cím eltér, ebben az esetben a futás idejü leképzést az MMU valósítja meg.

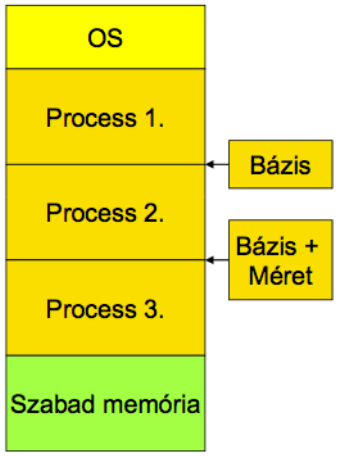
Címképzés: mikor történik meg a fizikai és logikai címek közötti leképzés?
- fordítási idejü (compile/link time)
  - ismert a betöltés helye, abszolút címzés
  - a program fizikai címeket tartalmaz
- betöltési idejü (load time)
  - áthelyezhetö kód
  - betöltés során oldódik fel a leképzés
  - a program fizikai címeket tartalmaz
- futás idöben is változhat a leképzés:
  - folyamat számára tanszparens módon más fizikai címterületre kerülhet
  - speciális HW (MMU) szükséges ehhez
  - a folyamat nem látja (láthatja) hogy melyik fizikai címeken található a memória, amin elfér: mindig logikai/virtuális címeket használ
- a fizikai címekhez kötött logikai címek használata alapvetö a modern OS-ek müködése szempontjából
- dinamikus betöltés: bizonyos funkció nem töltenek be amíg nem használjuk
  - gyors program indulás
  - alacsonyabb memória használat
  - a program feladata a dinamikus betöltés megvalósítása: OS nem tud róla

Dinamikus kapcsolatszerkesztés (dynamic linking)

- a dinamikus betöltés operációs rendszer támogatással
  - dynamically linke/loaded library (Windows *.dll)
  - shared object (UNIX/Linux *.so)
  - a dinamikusan beszerkesztett programkönyvtárak több program számára is elérhetöek (code sharing)
- megválósítás:
  - a program csak egy csokot(stub) tartalma
  - a csonk feladat a dll/so megtalálása és betöltése az OS felhasználásával
  - egy adott funkciójú dll/so több eltérö verzióban is lehet a rendszerben

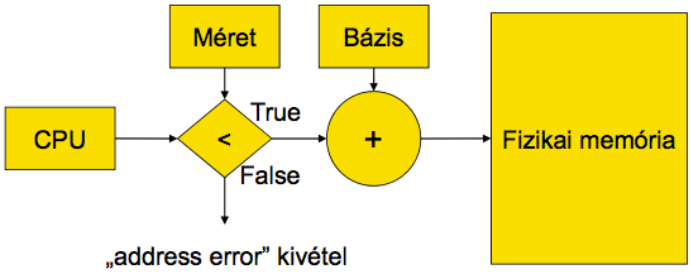
Alapmegoldás (változó mértü partíciók):

- minden egyes folyamat egy összefüggö, statikus, elöre kiválasztott fizikai címtartományba kerül
  - báziscímtöl kezdödik
  - adott mértü
  - bázisrelatív címzés
- a folyamatnak ebbe a memóriaterületbe kell beleférni élettartama során
- nem túl hatékony, de elsö iterációnak jó



Egyszerü megvalóstítás:

- implementációhoz szükséges
  - folyamatonként 2 védett módban állítható regiszter
  - 1 összeadás és egy vizsgála
- ha a folyamat kicímez a rendelkezésre álló memóriából, akkor kivétel jelzés
  - a CPUnak küldjük, és azt a CPU kernel módban kezeli



Problémák:

- külsö tördelödés: Process 2 kilép, a helye felszabadul, helyére egy kisebb Process 4 töltödik, Process 4 és Process 3 közötti memória terület nem használható: nem fér bele folyamat, elveszik az OS számára
- belsö tördelödés: ugyanaz mint a külsö, csak P. 4 az egész területet megkapja
- a programok által igényelt memória dinamikusan változik, nehéz felsö korlátot adni
- a programok rendelkeznek az alábbi tulajdonságokkal:
  - a programkód kis része használja az adatok kis részét nagyrészt
  - egyes részek soha nem kerülnek végrehajtásra: dinamikus kapcsolatszerkesztés, csak a ténylegesen használt kódot töltjük be

Foglalási stratégiák:
- First fit: tár elejéröl indulva az elsö elégséges méretü területet foglalja le
- Next fit: legutolsó allokált területtöl kezdi a keresést
- Best fit: legkisebb szabad hellyel járó helyre tesszük be, minimális külsö tördeléses
- Worst fit: a legnagyobb szabad hellyel járó helyre tesszük be, talán a maradék helyre még fér be valami

Foglalási stratégiák értékelése:
- külsö tördelést befolyásolják
- "First/Next/Best fit" kb a teljes memória terület 33%-át, a használt memória 50%-át nem tudja allokálni, az kihasználatlanul marad
- a "worst fit" algoritmus esetén a teljes memória 50%-a kihasználatlan marad
- "first/next fit" a legkevésbé eröforrás igényes, másik 2 teljes listát végignézni

Számolási feladatok: (3.1)
http://home.mit.bme.hu/~micskeiz/opre/files/opre-feladatok-segedanyag.pdf

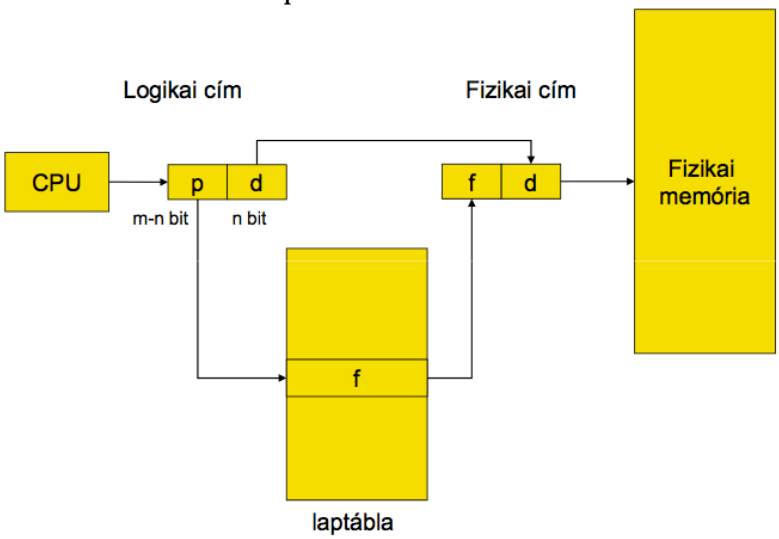
Szabad helye tömörítése(Compaction, Garbage collection):
- megoldja a külsö és belsö tördelést
  - átrendezi a futó folyamatokat az optimális memória használat elérésére
  - egy, összefüggö szabad területet hoz létre
- nagyon eröforrás igényes

További lehetöségek:
- fizikai memória kezelést általánosabban, hatékonyabban kell megvalósítani
- tárcsere (swapping)
- szegmens szervezés
- lapcsere

Tárcsere (swapping):
- a teljes folyamat háttértárolóra is kiírható
- ha nincs futó állpotban és nincs folyamatban lévö I/O müvelet (kivéve az OS területén lefoglalt pufferekbe/pufferekböl végzett I/O)
- ha a címképzés futási idöben történi: akár még más fizikai címre is kerülhet a visszatöltött folyamat
- a tárcserével összekapcsolt kontextus váltás nagyon idöigényes: a teljes folyamatot ki kell írni, majd visszatölteni

Lapszervezés (paging):
- a folyamathoz tartozó fizikai memória terület lehet nem folytoson is
- a fizikai memóriát keretekre (frame) osztjuk
- a logikai memóriát lapokra (page) osztjuk
- minden egyes logikai címet két részre osztunk: lap szám (p), lap eltolás (d)
- a lap szám a laptábla indexelésére szolgál: a laptábla az egyes lapokhoz tartozó fizikai báziscímet tárolja (f)
- a keret tábla tartja nyilván az üres kereteket

Címtranszformáció laptáblán:

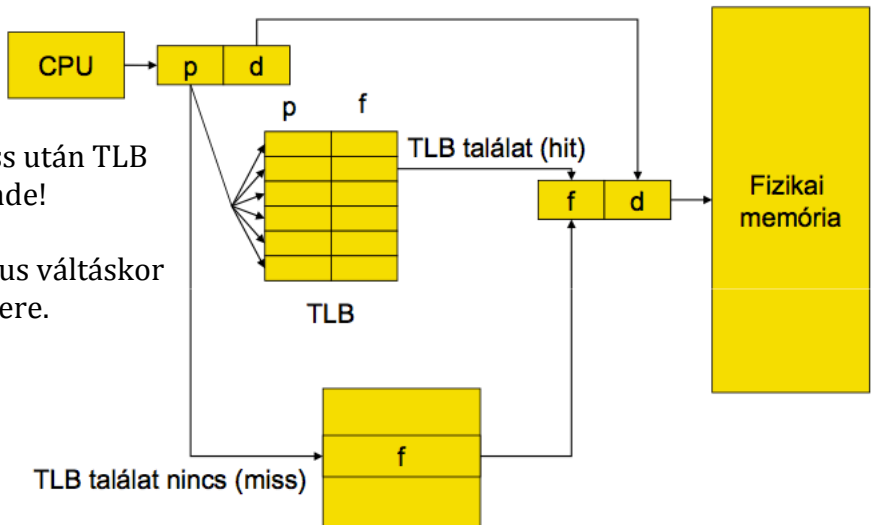


Lapszervezés tulajdonságai;
- a leképzést hardver végzi
- szeparáció megvalósul (folyama létrehozható)
  - a folyamat által látott logikai címtartomány, és a tényleges használt fizikai címtartomány teljesen elkülönülnek
  - a folyamatok nem férnek hozzá egymás lapjaihoz
  - az OS kezelheti a lap- és kerettáblát
- külsö tördelés nincs, belsö tördelés átlagosan fél lapnyi
- a modern gépekben sok memória van -> nagy lapméret lenne jó

Keresés a laptáblában:
- a lap- és kerettábla nagy lehet a modern OS-ekben
  - 4KB-os lapok, 32 bites címzés -> 2^{20} bejegyzés van -> min 4MB memóra
  - többféle lapméret támogatás
- megoldás:
  - többszintü laptáblák (hierarchical paging)
  - hash-elt laptáblák (hashed page table)
  - inverted page table
- a keresés a laptáblában lassú:
  - 2x annyi idö (laptába indexelés, olvasás majd adatelérés)
  - asszociatív lekérdezés (Translation look-aside buffer, TLB:

TLB miss után TLB upade!

Kontextus váltáskor teljes csere.



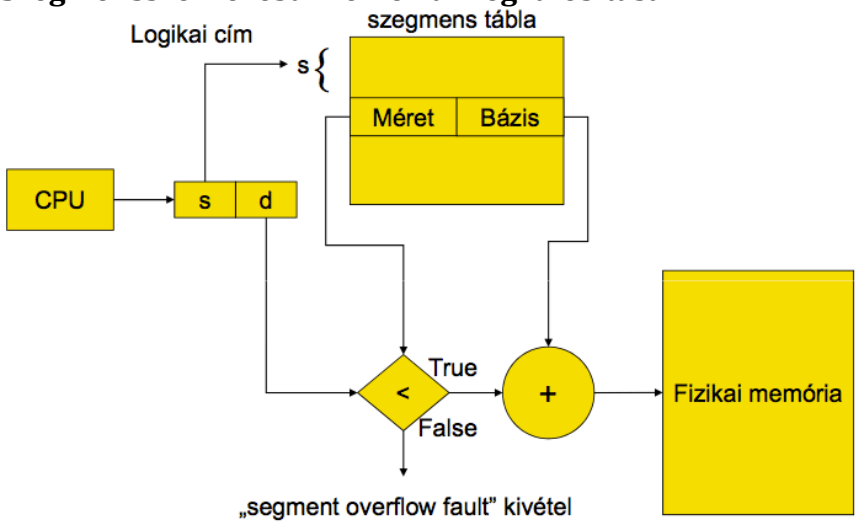
TLB hatása:
- a kérdés a találati arány (hit ratio):
  - a TLB méretétöl és a végrehajtott kódtól függ
  - tipikusan 80-90% körül van
- hatása a teljesítményre: ADM Phenon
  - a TLB hibás, a BIOS-ból kikapcsolható, eltérés: 20% teljes, 14% alkalmazás, memória benchmarkok esetén akár 50% is

Egyéb információk a laptáblában:
- kiegészítö bitek a laptáblában:
  - valid/invalid bit: benne van-e az adott lap a fizikai memóriában
  - read/read-write bit, execute bit: végrehajtható memória müveletek
  - referenced/used bit: memóriamüvelet esetén az MMU beállítja
- a HW(MMU) beállítja/ellenörzni, ha a szabályokkal ellentétes használatot talál, akkor kivételt generál, amit az OS kezel
- az OS is felhasználja ezeket

Lapok megosztása:
- folyamatok memória tekintetében szeparálva vannak egymástól
- de teljesítmény okokból nem teljes a szeparáció
- OS által kontrollált módon megosztható memória:
  - közös kód lapok
  - Copy on Write (COW):
    - szülö-gyerek kapcsolatokban lévö folyamatok esetén
    - a két folyamat elöször egy közös fizikai lapot használ
    - ha azt bármelyik írni próbálja, akkor a gyereknek létrejön egy saját lapja, az írási kísérlet elötti tartalommal
    - közösen olvashatóak és írhatóak (UNIX System V Shared Memory)
- HW (MMU) támogatás szükséges a megvalósításhoz

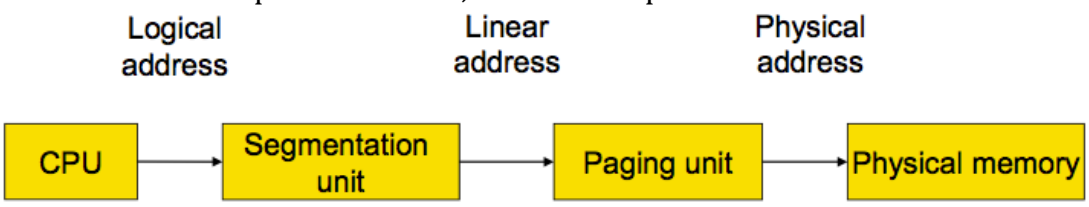
Szegmensszervezésü memória:
- a programozó:
  - adat, kód, bizonyos programkönyvtárak, stack, heap, stb.
  - nem egy összefüggö lineáris memória területben gondolkozik
- a szegmensszervezesü memória ennek az elképzelésnek felel meg:
  - a logikai címtartomány szegmensekre van osztva
  - a programozó egy szegmenst, azon belül egy szegmens ofszetet ad meg
    - a lapozásnál egy címet ad meg, és azt a HW bontja ketté
    - itt két részt ad meg, és azt a HW rakja össze
- a szegmensek mérete adott, azt is tárolni kell
  - adott szegmensen kívüli címzés esetén "segment overflow fault" kivétel
- a szegmensek folytonosan tárolhatóak, belsö törde lödés nincs
- a szegmenseket a fordító és a linker állítja össze, és adja meg a programot betöltö loader-nek

Szegmensszervezésü memória megvalósítása:



Szegmens és lapszervezés együtt:

- egyes rendszerek mint a kettöt támogatják
- a szegmensszervezést minimálisan használják a x86 HW-ét támogató OS-ek
- a lapozás viszont alapvetö CPU szolgáltatás
  - x86 HW esetén 4KB (2 szintü tábla) és 4MB-es lapok (1 szintü tábla)
  - a Linux 3 szintü laptáblát használ, ebböl a középsö üres x86 HW esetén



Virtuális tárkezelés (Virtual memory):

- logikai cím != fizikai cím (nem erröl lesz szó)
- össze szokták keverni a swapping-gel (nem erröl lesz szó)
- a korábban ismert memória menedzsment módszere alapján kidolgozott komplex memória menedzsment megoldás a virtuális tárkezelés

Alapgondolatok, megfigyelések:

- a folyamatok fizikai memóriában történö végrehajtása:
  - szükségesnek és célszerünek tünik
  - komoly és kellemetlen következményekkel jár

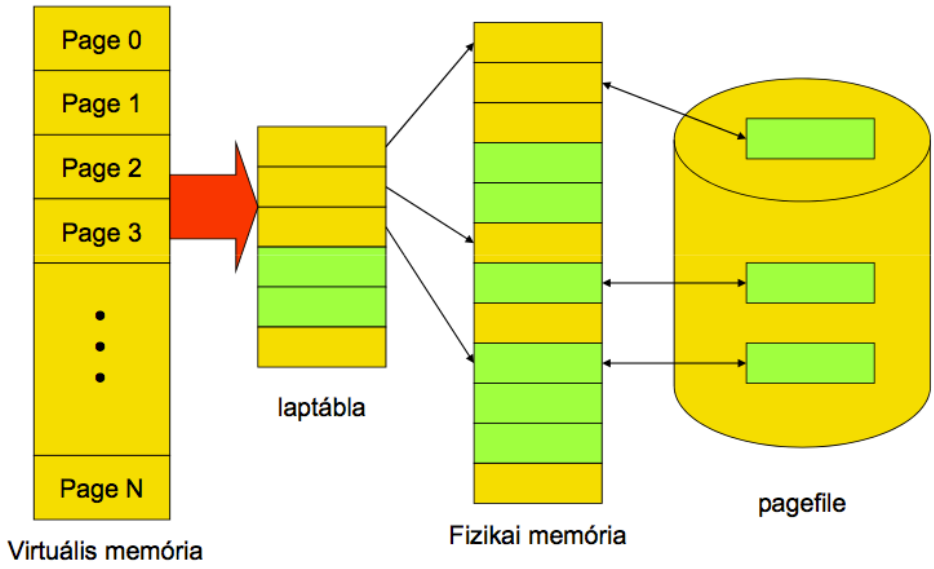
1. A teljes folyamat nem szükséges annak végrehajtásához, többnyire az aktuális utasítás számláló "környezete", és az éppen használt adatszerkezetek elégségesek (lokalitás)
2. A folyamatok kódrészleteinek nagy részét soha nem hajtjuk végre vagy nagyon ritkán hajtjuk végre (error handling, software/feature bloat)
3. A folyamatok elindulásához nem szükségs a teljes program betöltése
4. Egyes kódrészletek, eröforrások megoszthatóak a folyamatok között
5. A párhuzamosan futó folyamatok egyes, már használt kódrészleteire sokáig vagy akár soha többé nincs szükségük, míg más folyamatoknak nem elég a memória

Elvárások:

- jó lenne nagyobb memóriát használó folyamatokat futtatni, mint amennyi fizikai memória rendelkezésre áll
  - virtuális memória, a programozónak nem kell foglalkozni a rendelkezésre álló memóriával
  - architektúra függő határig lehetséges
  - következmények: komplexitás és sebesség
- ha a folyamataink csak a tényeleges szükséges fizikai memóriát tartják a fizikai memóriában, több folyamatot tudunk egy idöben betölteni
- a programok gyorsabban induljanak el, csak a szükséges kódrészleteket töltse be az OS
- képes legyen osztozni a közös kód és adatszerkezeten, eröforrásokon

Virtuális tárkezelés:

- az alapjai a lapozás: a folytonos virtuális címteret egy tábla (memory map, page table, laptábla) képezi le
- nem direkt módon fizikai memóriára történik leképzés, hanem:
  - részben fizikai memóriára
  - részben a permanens táron (HDD, Flash tár) kialakított speciális terület
    - pagefile (windows), swap file (UNIX/Linux)
- a folyamat egy összefüggö virtuális címteret lát



Egyéb tulajdonságok:

- folyamatok megoszthatnak memóriaterületeket olvasásra/írásra
  - hozzáférési jogosultságok
  - az ilyen memória területek több folyamat virtuális címtartományába vannak belapozva
- módosítás nyilvántartása (modified/dirty bit)
  - minden laphoz tartozik egy HW által kezelt bit: betöltéskor tölrik, módosításkor beállítódik
- hivatkozások nyilvántartása (referenced/used bit)
  - OS adott idönként és/vagy adott eseményekre törli
  - használat esetén beállítják

Következmények:
- a folyamat virtuálisan összefüggö memóriát lát (virtuális memória)
- valójában az összefüggö memóriaterület ritka (sparse address space)
    - nagy része mögött nincs fizikai memória vagy pagefile bejegyzés
    - ha az szükséges, akkor dinamikusan kerül mögé memória, amit a folyamat elérhet

Müködés:
- ha egy éppen használt virtuális memória lapon található cím bent van a fizikai memóriában, akkor a kód vegrehajtható
- ha nincs bent (valid/invalid bit)
    - laphiba ("page fault") kivételt generált az MMU
        - érvényes, de éppen nem fizikai memóriában lévö lap
        - nem hiba, normális müködés
    - az OS ezt kezeli, lehetöségek:
        - HHD-re ki van írva: be kell hozni a pagefile-ból
        - soha nem lett betöltve: be kell tölteni
        - kérdés: hová, föleg ha tele van a fizikai memória?
    - az OS visszaadhatja a vezérlést a megszakított folyamatnak
    - a hozzárendelés során a folyamat passzívan várkozik

Lapozási stragégiák (fetch strategy):
- igény szerinti lapozás (demand paging)
    - csak laphiba esetén, és csak a laphibát megszüntetö lapot hozza be a fizikai memóriába
    - csak a szükséges lapok vannak a fizikai memóriában
    - új lapra vonatkozó hivatkozás mindig hosszú várkozást eredményez (be kell hozni)
- elöretekintö lapozás (anticipatory paging)
    - elöre tekintve (becslés) az OS megpróbálja kitalálni, hogy mely lapokra lesz szükség, és azokat is behozza
    - feltétel: szabad eröforrások (CPU, HDD, fizikai memória)
    - ha a behozott lapokra tényleg szükség lesz, akkor csökken a laphibák száma
    - ha nem, akkor feleslegesen használtunk eröforrást

Lapcsere stratégiák (replacement strategies):
- tele van a fizikai memória, és laphiba történik
    - ki kell választani a permanenst tárra kikerülö lapot
    - ennek a helyére kerül be a lap a permanens tárról
- algoritmusok:
    - optimális algoritmus
    - legrégebbi lap (FIFO)
    - újabb esély algoritmus (second chance)
    - legrégebben nem használt (LRU, last recently used)
    - legkevésbé használt (LFU, laft frequently used)
    - utóbbi idöben nem használt (NRU, not recently used)

Optimális algoritmus:
- elöre néz, és teljes információval rendelkezik a jövöben használt lapokról
- nem realizálható, de jó összehasonlítási alap

FIFO:
- a behozott lapokat egy FIFO-ba rendezi, és mindig a legrégebben behozott lapot cseréli
- egyszerü, a múlt alapján dönt, hátranézve
- azokat a lapokat is cseréli amiket a folyamat gyakran használ
- Bélády anomáisa: több folyamat, több laphiba

Second chance algoritmus:
- a sor elején lévö lapot csak akkor cseréli le, ha arra nem hivatkoztak: used bit
- a used bitet (referenced bitet) az MMU állítja be, ha a lapot használták
- a used bitet törli, ha az be van állítva:
    - ezért újabb esély a neve
    - ezután a lapot a sor végére rakja
    - egyébként végtelen ciklusba kerülhetne
- bonyolultabb mint a FIFO, de jobb algoritmus
- hátra néz, és a behozás sorrendje, és a használat alapján dönt
    - a program lokalitását figyelembe veszi

Legrégebben nem használt (LRU):
- bonyolult, de jól közelíti az optimális algoritmust: a lokalitás miatt jó a közelítés
- hátrafelé néz
- megvalósítása:
    - számláló: minden laphoz egy "last used" timespamt kerül
    - láncolt lista: a lista végére kerül a legutoljára használt
    - kétdimenziós tömb: NxN-es mátrix, N a lapok száma
- sokszor a közelítéseit szokták számolni

Legkevésbé használt (LFU):
- a közelmúltban gyakran használt lapokat a lokalitás miatt nagy valószínüséggel újra fogjuk használni
- a ritkán használtakat kis valószínüséggel fogjuk használni
- az R bit értkékét hozzádaja idönként egy laphoz tartozó számlálóhoz, és törli az R bitet
- a kisebb számláló értéket tartalmazó lapokat cseréljük le
- az algoritmus nem felejt
- az algoritmu a frissen behozott lapokra fogja lecserélni - azokat be kell fagyasztani a memóriába kis idöre

Utóbbi idöbe nem használt (NRU):
- a hivatkozott és módosított biteket is használja
- R törölhetö, viszont M-et meg kell hagyni
- prioritást rendel a lapokhoz M és R alapján:
    - 0. prioritás : M=0, R=0 (legalacsonyabb)
    - 1. prioritás: R=0, M=1
    - 2. prioritás: R=1, M=0
    - 3. prioritás: R=1, M=1 (legmagasabb)
- mindig a legkisebb priotirású csoportból választ, ahol még van lap

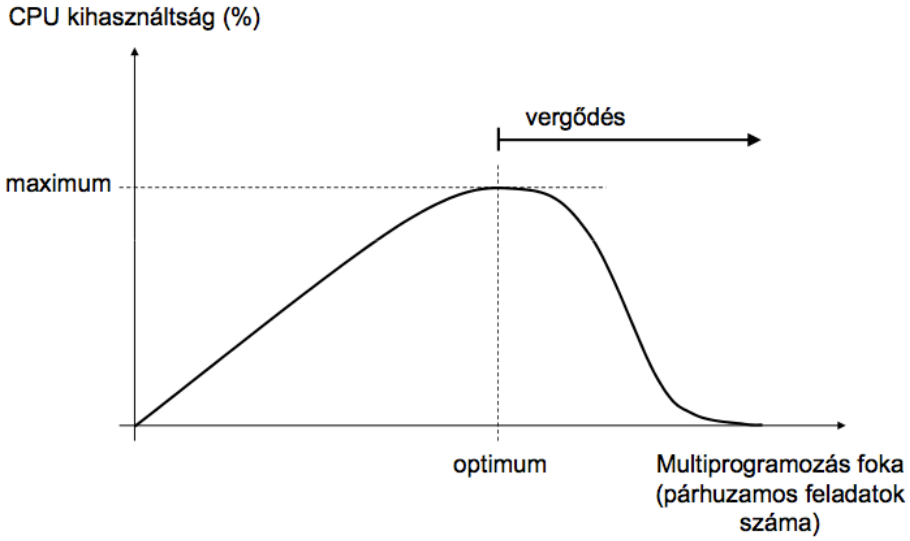
Lapcsere szempontok:
- globális csere: a teljes fizikai memória potenciálisan cserélhetö
- lokális csere: a folyamat által használt fizikai memória lapok között történik a csere
- lapok tárba fagyaztása (lock bit):
    - I/O müveletek hivatkoznak rá
        - ott fizikai címet kell használnunk
        - lehet kernel szinten pufferelni, és akkor ez is megkerülhetö
    - LFU algoritmus frissen behozott lapjai

Számítási feladat: (3.2)
http://home.mit.bme.hu/~micskeiz/opre/files/opre-feladatok-segedanyag.pdf

Virtuális memória teljesítménye:
- fizikai memória sebessége:
    - n*1GB/s adatátviteli sebesség
    - n*10ns késleltetés
    - ha cache-elve van, akkor még gyorsabb
- permanens tároló sebessége:
    - tipikusan 100MB/s adatátviteli sebesség, de random elérés esetén és sok párhuzamos felhasználó esetén sokkal lassabb
    - 10 ms legrosszabb hozzáférési idö (fejmozgás)
    - Flash esetén az írás(törlés) lassú és problémás, hamar tönkremegy
    - több nagyságrend a különbség
- ha egy lap nincs bent a fizikai memóriában:
    - több nagyságrenddel lassabb a permanenst tárolón lévö lapok elérés, mint egy fizikai memóriában lévö lap elérése
    - az elérhetö sebességet alapvetöen a laphibák gyakorisága befolyásolja
        - gyakori laphiba => teljesítmény drasztikusan csökken
        - sikeres elöretekintö lapozás javíthat a teljesítményen

Vergödés:
- hogyan válasszuk meg, hogy egy folyamathoz hány memória keretet rendeljünk a fizikai memóriából?
    - kevés: nagyszámú/állandó laphiba (trashing)
    - sok: más feladatnak nem marad memória
- DEF: a gyakori laphibák által okozott rendszer teljesítmény csökkenését vergödésnek (trashing) nevezzük
    - a laphiba kezelése során újabb laphiba jelenik meg

- a laphibák felgyülnek a háttértár várkozási sorában
- a CPU a laphibák megoldására vár
- a hosszú távú ütemezö (ha van), ezt I/O intenzív folyamatként is értelmezheti, újabb folyamatokat beengedve a rendszerbe
- ábra:



Vergödés elkerülése:
- cél: alacsonyabb laphiba gyakoriság (PFF, Page Fault Frequency)
- egy laphiba kezelése során ne jöjjön létre újabb laphiba: a háttértár várakozási sora csak csökkenhet
- lokális lapcsere stratégia:
  - a folyamatok nem tudják egymástól elvenni a fizikai memória keretet
  - nem terjedhet át a hatás más feladatokra
  - csökkenti a problémát, de nem oldja meg
- hány keretre van szükség a fizikai memóriában egy folyamatnak a hatékony futáshoz?

Lokalitás:
- statisztika: egy idöintervallumban a folyamatok a címtartományuk csak egy kis részét használják: idöbeli / térbeli
- vergödés: megfelelö számu fizikai memória keret allokálása
  - nincs vergödés
  - laphibák a lokalitás váltásakor

Munkahalmaz (Working-set):
- a lokalitáson alapul
- a folyamat azon lapjainak halmaza, amelyekre egy idöintervallumbana folyamat hivatkozik
- a munkahalmaz alapján megadható az adott folyamat munkahalmaz mérte (WSS)
- a futó folyamatok teljes fizikai keret memória igénye kiszámolható (D)

$$D = SUM(WSS_i)$$

Alkalmazása:
- az OS méri a WSS-t folyamatonként
    - ha van szabad memória keret:
        - akkor az igények kielégíthetöek
        - új folyamatok engedhetöek be a rendszerbe
    - ha nincs szabad memóriakeret:
        - akkor ki kell választani egy "áldozat" folyamatot
        - azt fel kell függeszteni
        - az áldozat folyamat fizikai memória kereteit fel lehet használni
- a vergödés elkerülhetö a multiprogramozás fokát közel optimálisan tartva
- a gyakorlatban eröforrás igényes a megvalósítasa

PFF alapú optimalizáció:
- vergödés: magas PFF érték
- folyamatonként egyszerüen mérhetö PFF érték
    - alacsony: túl sok fizikai memória keret
    - magas: túl kevés fizikai memória keret
- felsö és alsó PFF hatérértékek megadása
    - felsö határértéket túllépi: kap egy fizikai memória keretet
    - alsó határértéket túllépi: egy fizikai memória keret elvonása

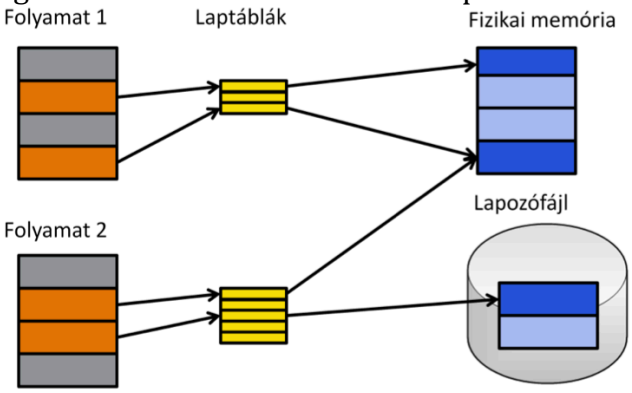
Beágyazott rendszerek:
- nagyon eltérö a memóriakezelés
- a beágyazott rendszerek CPU-inak egy részében nincs MMU:
    - lapozás, virtuális memória kezelés, szeparáció, stb ki van zárva
- ha van MMU, és használják:
    - a háttértár korlátos, a pagefile-nak nincs hely sem
    - elsösorban a feladatok szeparációja és virtualicáció a feladat

18. Memóriakezelés a Windowsban

A Windows memóriakezelésésnek alapelvei:

- virtuális tárkezelés
  - lapszervezés (4KB/2KB mértü lapok, x86:2 szintü, x86+PAE:3 szintü, x64:4 szintü)
  - lapozófájl használata
- hatékonyság
  - igény szerinti lapozás + lustering + prefetxh
    - lapozás: kezdetben igény szerinti lapozás
    - clustering: a kért lap környékén lévö lapokat is behozza (lokális elv)
    - prefetch: rögzíti, hogy a programok induláskor miket szoktak igényelni, es azokat elöre behozza
  - memória megosztás, copy-on-write
  - fájl cachelés memóriában (memory mapped file)
- biztonság
  - minden folyamatnak külön címtartomány
  - elérés leírókon keresztül (hozzáférési token)
- x86 (32 bites rendszer)
  - maximum 4 GB fizikai memória, de néha kevesebbet lát az OS
  - server 2008 enterprise esetén max 64 GB fizikai memória
  - alapesetben egy felhasználó folyamat maximum 2 GB-os címtartományt használhat fel, a címtartomány másik részén a rendszer memória van
  - boot.ini-ben /3GB kapcsolóban 3GB-ra növelhetö (kényszer megoldás)
- x64 (64 bites rendszer)
  - lényegesen nagyobb memória: max. 8/192 GB (Win 7 basic/prof)
  - server 2008 enterprise esetén max 2048 GB fizikai memória
  - 8GB felhasználói folyamat tartomány / 6657 GB rendszer taromány
  - Phyisical Address Extension (PAE): 32 helyett 36 címbit van, akár 64 GB memóriát is képes így kezelni
- Virtual Address Space (VAS)
  - felhasználó címtartomány:
    - egyedi minden folyamatra
    - a futó alkalmazás (.exe és .dll-ek)
    - felhasználói módú verem minden szálnak
    - alkalmazás adatstruktúrái
  - rendszer tartomány:
    - rendszerszinten közös
    - executive, kernel és a HAL
    - rendszerszintü adatstuktúrák
    - laptáblák (virtuális -> fizikai leképzés, folyamatonként különbözö)
    - executive heap-ek (pools)
    - védett módú eszköz meghajtók
    - védett módú verem minden folyamat minden szálának
- folyamatok memóriafoglalása (két lépésben)
  - reserve: virtuális címtartomány lefoglalása
  - commit: virtuális memória lefoglalása
  - elöny: csak annyit foglal, amennyi ténylegesen kell a folyamatnak

Logikai és fizikai címek közötti leképzés:



x86 PAE címfordítás:
- hiába van PAE, a folyamatunk továbbra is csak 2^{20} fizikai memória keretet tud megcímezni, így a 12 bites offsettel együtt is 4 GB memóriát kezel. A különsbség ott van a PAE-nél, hogy a fizikai memórialap címzésére 24 bitet használunk -> 64 GB memóriát tud kezelni

x64 PAE esetén a PTE:
- 64 bites, 24 bit a lap címének
- flagek: P(present), A(access), D(dirty), U/S(user/system), W/R(write/read)

Munakészlet (Working set)
- egy folyathoz tartozó fizikai memóriában lévö lapok
- ezeket éri el laphiba nékül
    - soft laphiba (soft page fault): memóriában volt még a lap
    - hard laphiba (hard page fault): lemezröl kell beolvasni

Working set limit:
- ennyi fizikai memóriát birtokolhat egyszerre
- ha eléri, lapcsere kell
- ha a szabad memória lecsökken: trimming (egy rendszer szál megvizsgálja a folyamatot, és akinek sok fizikai memórialapa van vagy régóta nem használta öket, azoktól elvesz párat)

Lapozófájl (page file):
- módosított adat kerül bele, kód nem, ha:
    - van szabad memória
    - folyamatok nem foglalhatnak bármennyi memóriát
    - tartalék az új/többi folyamatnak
- meghajtóként egy darab, ajánlott nem a rendszerlemezre rakni, de maradjon egy kicsit ott is a memory dumpnak
- ajánlott méret: 1 vagy 1,5-szer a fizikai memória

Optimalizáció: Prefetch (Win XP)
- egy program indulásakor sok laphiba van
- mindig ugyanazonat kell betölteni
- Prefetch: elsö tíz másodperc hozzáféréseit megjegyzi
- alkalmazás következö induláskor: hivatkozott lapok betöltése aszinkon módon
- bootolás figyelése is

Optimalizáció: Superfetch (Vista)
- 8 priotirás a memórialapokhoz: standby listából 8 darab ennek megfelelö
- lapok használatának követése
- memória felhasználása esetén lassan visszahoz lapokat a standby listára, amik kellenek még

19. Virtualizáció

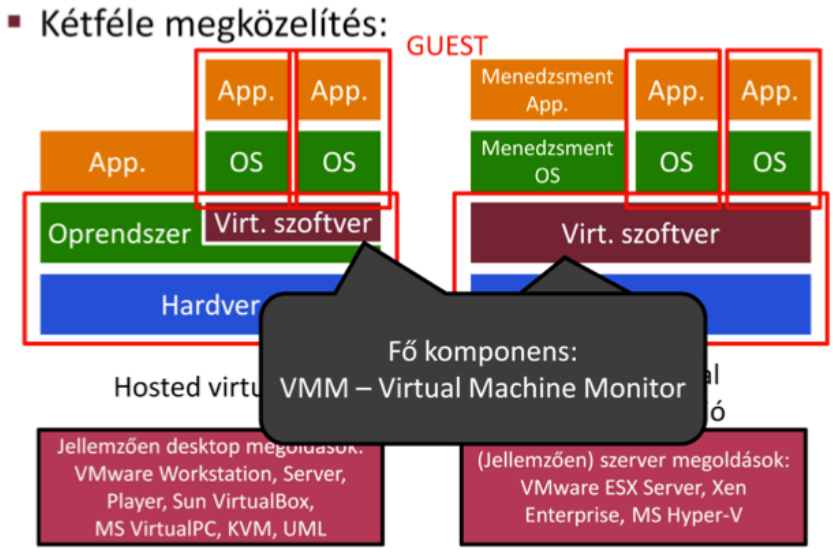
Virtualizáció alkalmazásai:
- vékony kliensek
- egygépes termékek
- OS szintü virtualizáció
- alkalmazás becsomagolás
- teljes számítógép virtualizálás
- tárolórendszer felépítésének elfedése
- dinamikus adatközpont, életciklus kezelés, konvertálás, telepítés

Virtuális gép osztályozása:
- System VMs : VM csak egy hardvert lát
- Process VMs: VM egy ABI-t (Application Binary Interface) lát

Platform virtualizáció:
- teljes számítógép virtualizálása, egy gépen több OS futhat
- elemek:
  - gazda gép (host machine) = fizikai gép
  - vendég gép (guest machine) = virtuális gép
  - Virtual Machine Monitor (VMM) = virtuális gépeket kezelö program

Miért jó a platform virtalizáció?
- tesztrendszer kiépítése
- HM kondszoldáció: alacsonyabb kihasználtságú szervereket, amiket a rajtuk lévö egymással nem komplatibilis alkalmazások miatt nem lehet egy fizikai gépre rakni, összevonja egy gépre
- Régi rendszer (regacy system): pl.: DOS, régi UNIX
- On-demand architektúra / dinamikus adatközpont
- rendelkezésre állás, katasztrófa védelem
- hordozható alkalmazások
- új terület: mobil virtualizáció

Platform virtualizáció:



1. Elméleti alapok, követelmények egy virtualizációs megoldástól:

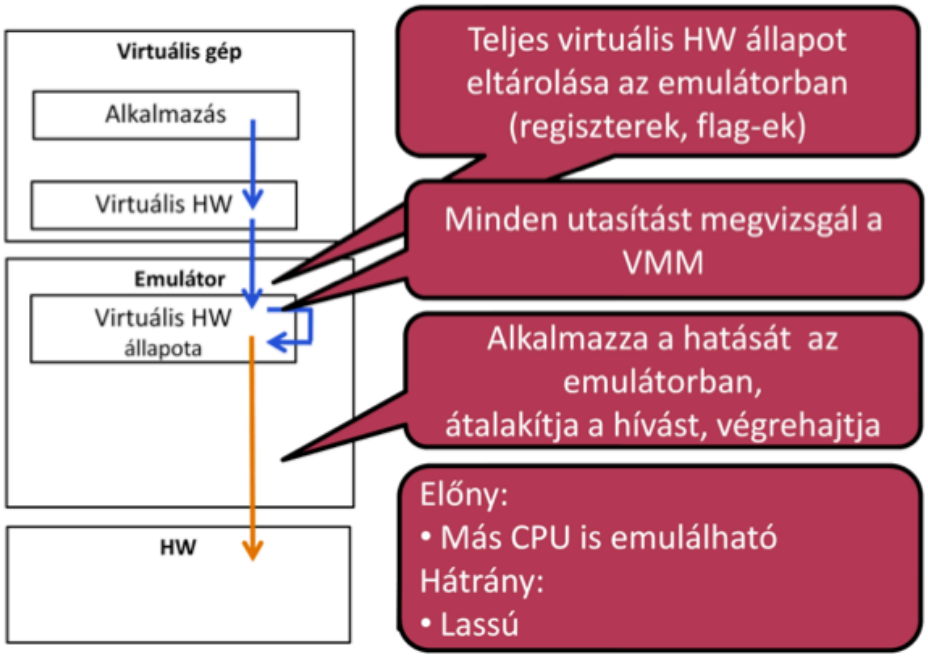
- *azonosság:* virtuális gépen futtatott programok ugyanazt az eredményt adják
- *biztonságosság*: a VMM kezeli az össze hardver eröforrást
- *hatékonyság*: a vendég gép utasításainak nagy része beleavatkozás nélkül fut

Alapvetö probléma:

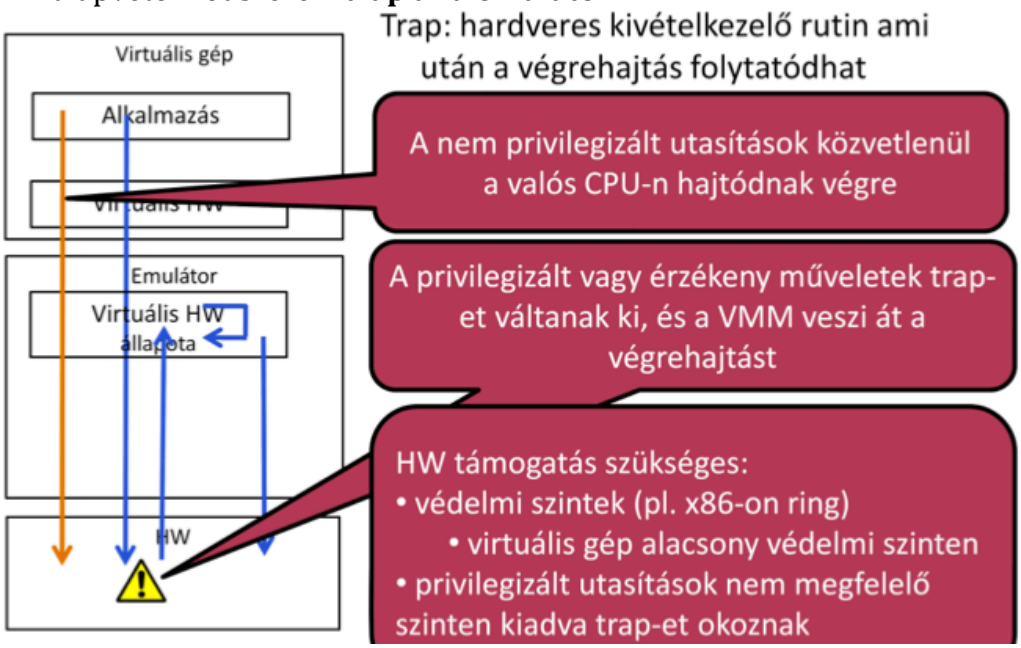
- vendég gépektöl védeni kell a rendszert
- megoldás: BMM felügyeli a vendég utasításait
  - privilegizált (érzékeny regisztert/flaget módosít) utasítások kezelése

2. Elméleti alapok, CPU virtualizáció:

- alapvetö módszerek: **tiszta emuláció**

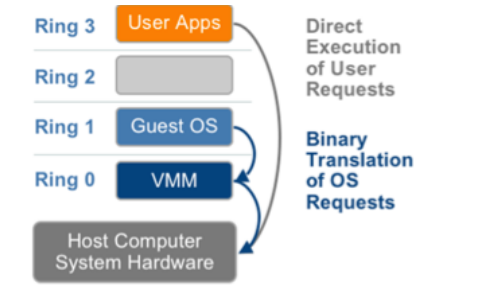


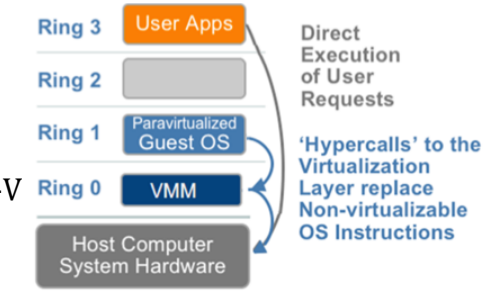
- alapvetö módszerek: **trap and emulate**

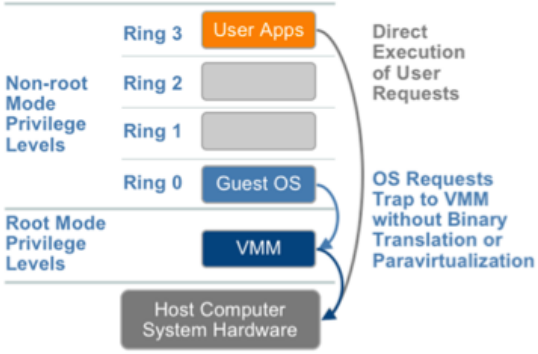


x86 virtualizációk korlátai:
- egyes architektúrák könnyen virtualizálhatóak, az x86 nem ilyen volt
- kb. 250 utasításból 17 megsérti a klasszikus feltételeket
- következmény: nem használható a trap&emulate módszer klasszikus x86-on

Megoldások az x86 CPU virtualizációra:
- **binary transation (szoftveres)**
  - utasítások nagy része közvetlenül fut
  - privilegizált utasítások átírása futás közben
  - nem igényel forráskódot
  - átírt változatot eltárolja
  - vendég OS nem tud arról, hogy virtualizált
- **paravirtualizáció**
  - vendég OS forrásának módosítás
  - problémás utasítások lecserélése
  - hypercall: CMM-et hívja közvetlen
- hardveres virtualizáció
  - 2005: Intel Virtualization Technology/AMD-V
  - HW támogatás: root mode, VMCS
  - müködik a trap&emulate módszer



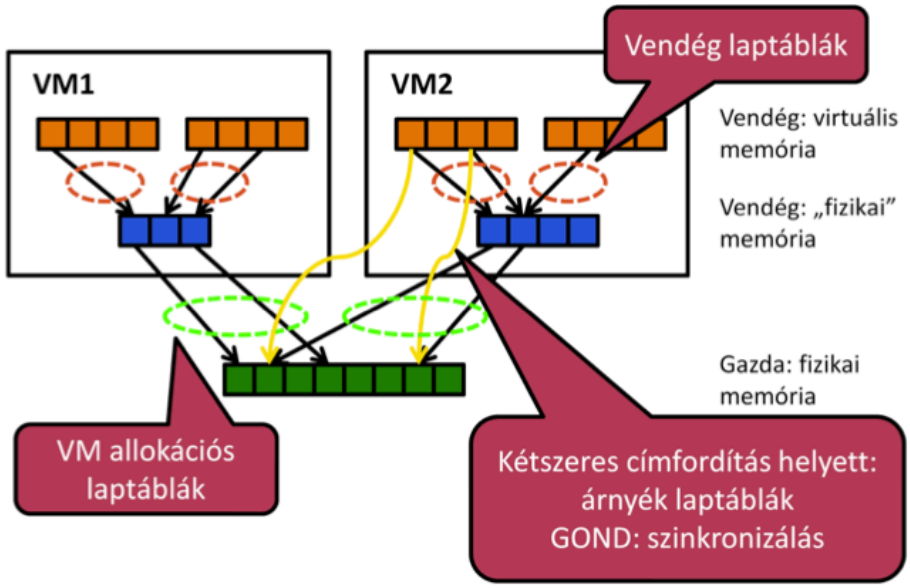




Melyik a legjobb/leggyorsabb módszer?
- folyamatosan változik
  - környezettöl, terheléstöl is függ
  - HW virtualizáció kiforratlanabb kezdetben
- összemosódnak a határok
  - több módszert használnak vegyesen

3. Elmélteli alapok, memória virtualizáció:

Memória virtualizálása - szoftveresen:

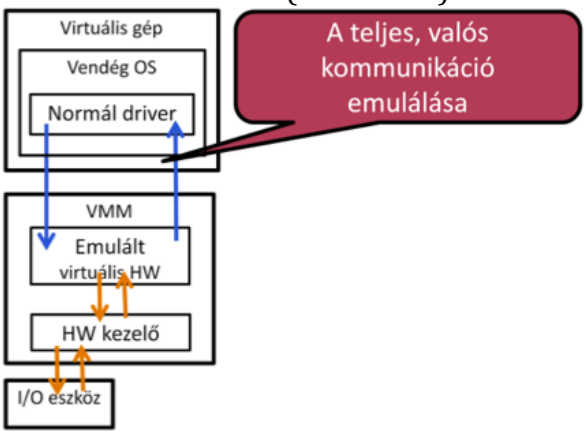


Kétszeres címfordítás kell, így kétféle laptáblákat kell fenntartani.
A régebbi hardverekben csak egy leképzés támogatás van, fenntartanak egy harmadik féle struktúrát is (árnyék laptáblák). Így viszont a VMM-nek gondoskodnia kell arról, hogy az árnyék laptáblák szinkronban legyen a vendég laptáblákkal.

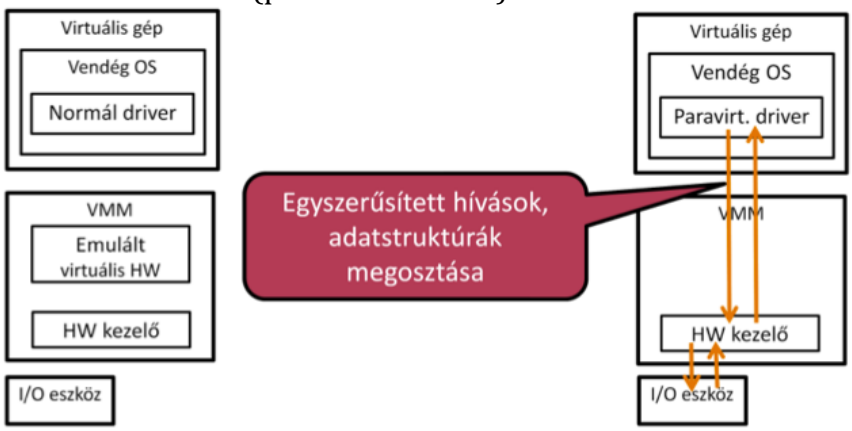
Memória virtualizálás - paravirtualizáció:
- árnyék laptáblák
- vengéd OS forrásának módosítása
- ha a vengéd módosítja a laptábláit, akkor értesítse a VMM-tet is erről

Memória virtualizálás - harveres:
- HW támogatás az újabb CPU-kban (AMD Rapid Virtualization Indexing, Intel Extended Page tables)
- beágyazott laptábla (Nested page table)
  - vendég fizikai -> gazda fizikai leképzés eltárolása
  - cím leképzési rutin ezt is bejárja
- TLB bejegyzések azonosítóval ellátása
- nagy teljesítménynövekedés

4. Elméleti alapok, I/O virtualizáció:
- I/O eszközök kezelése (szoftveres)



- I/O eszköz kezelése (paravitrualizáció)



  - speciális csomag telepítése a vendégben
    - VMware Tools, Virtual PC Additions
    - mindig telepítsük a vendég gépen!

I/O eszközök kezelése (hardveres)
- hardveres támogatás: Intel VT-d, AMD IOMMU
  - PCI szabvány kiegészítése: I/O Virtuaization (IOV)
- I/O eszközök: megosztás VM-ek között, közvetlen hozzárendelés VM-hez

Cloud computing:
- amazon, windows azure, google apps, dropbox, rackspace, ...

Számítási felhö rétegei:
- IaaS (Infrastructure as a Service)
  - virtuális gépet kapunk
  - Amazon EC2, RackSpace, ...
- Paas (Platform az a Service)
  - futtatható környezetet kapunk: java konténer, .NET adatbázis, ...
  - MS Azure, Google AppEngine, ...
- Saas (Software az a Service)
  - szolgáltatást érünk el
  - Google Docs, SalesForce CRM, ...

Virtual Application:
- adott célra összeállított viruális gép
- elöny:
  - nincs telepítés, függöség
  - csak a feltétlen szükséges komponensek vannak telepítve
- JeOS: Just Enought Operation System

20. dTrace:

A feladat: hibakeresés, diagnosztika, optimalizáció
- végzetes hibák
- tranziens hibák
- percepcionális "hibák" ("lassú a gép") okainak felderítése

Korábbi megoldások: gyenge megfigyelhetőség és invazív technikák
- egyszerű, nem túl rugalmas megfigyelő parancsok (vmstat, iostat, stb.)
- bináris vagy forráskódú beavatkozások szükségesek a hibakereséshez
- rendszerkönyvtár és kernel debug változatok használata szükséges
- erőszakosak, nehézkesek, időigényesek, korlátozottak, élő rendszeren nem alkalmazhatók, új hibákat vihetnek a rendszerbe

A DTrace összetevői:
- mérőrendszer: mérőpontok függvény be- és kilépési pontokon és adatfeldolgozók (fogyasztók)
- a mérőrendszer programozási nyelve (D)
- a megvalósító kernel modul

A mérési helyek (ún. provider-ekben):
- felhasználói függvények
- rendszerhívások
- kernel függvények
- összesen > 50 ezer mérőpont

Provider-ek:
- lista: dtrace -l
- fbt: kernel függvények (~45 ezer)
- syscall: rendszerhívások (~ 400 db mérőpont)
- I/O, processz, ütemezés, zárolás, stb.

Fogyasztók:
- DTrace parancs (szkriptek)
- programozási nyelvekben
- stb.

Működési mód:
- dinamikus mérőkód beszúrás a megfelelő fv. be/kilépési pontokra
- kernel szintű adatgyűjtés a meghatározott mérőpontokról
- a fogyasztók (pl. "dtrace" parancs) lekérdezik az adatgyűjtőt

A DTrace programozási nyelve: D

- C, awk, perl keverék
- szokásos adattípusok
- speciális típusok: asszociatív tömb
- aggregációs műveletek
- beépített változók (pid, ppid, execname, stb.)
- szálakra lokális változók: this->...
- nanoszekundum felbontású időmérés
- hozzáférés a megfigyelt függvények argumentumaihoz és visszatérési értékeihez
- megjelenítés: printf, pinta (asszociatív tömbökre)
- akciók (destruktívak is, pl. stop(), panic(), ha engedélyezzük)

20. A permanens tár kezelése

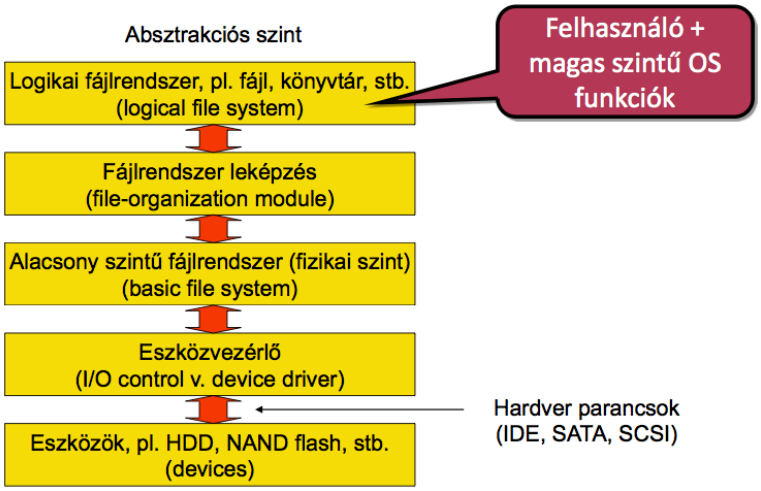
Permanens tár vagy háttértár:

- a központ memóriával összehasonlítható
- nagyságrendekkel nagyobb tároló terület
- nagyságrendekkel lassabb: adatátviteli sebessés, késleltés
- nem felejtö tároló
- blokk alapú szervezés
  - az OS ebben kezeli, ennél kisebb egységekben nem gondolkozik
  - blokkonként olvasható, írható, törölhetö
  - kivéve egyes beágyazott rendszereket
    - NOR flash memória szervezésü
    - NOR flash-böl direkt módon futtatható az OS és a programok

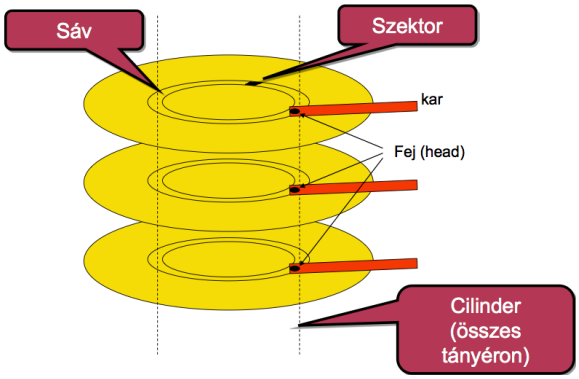
Fájl leképzése a HW tároló elemre:

- a fájl a permanens táron az adattárolás logikai egysége
  - névvel rendelkezik, a nevével lehet rá hivatkozni
- az OS feladata a logikai egységes (fájlok) leképzése valódi fizikai egységekre
- ezt az OS egy többszintü hierarchikus/réteges rendszer, különbözö absztrakciós szintekkel oldja meg
- legalacsonyabb szinten többnyire valamilyen speciális HW van (HDD, SSD, ...)
  - kivétel az u.n. RAM drive

Absztrakciós szintek (egyszerüsített):



HDD szerkezete:



HDD (merevlemez):
- forgó mágnesezhetö tányérok (platter)
- egy forgatható karra (arm) vannak szerelve író/olvasó fejek (head)
- a tányérokat cilinderekre (cylinder), és azokat sávokra (track) azokat meg szektorokra (sector) osztjuk
- a cilinder, sáv és szektor együtt azonosítja az írható/olvasható adatblokkot
- gyakorlatban már a fizikai eszközök egy logikai leképzést alkalmaznak (Logical Block Adressign, LBA: 48 vagy 64 bit napjainkban)
- 512B / 4KB szektor méret áttéres jelenleg folyik

A HDD tényleges sebessége:
- eröseg függ attól hogy éppen hol helyezkedik el a fej, és ahhoz képest az elérendö adatblokk (szektor), és milyen sebességgel forognak a tányérok
- több szintü optimalizáció:
  - diszt ütemezés (disk scheduling)
  - HDD szintjén (SATA NCQ, SCSI)
  - OS szintjén a párhuzamos írások/olvasások ütemezése során
  - prefetch
- több szintü cache:
  - HDD szintjén (16-64 MB jelenleg)
  - OS szintjén: disk cache, dinamikusan változó méret

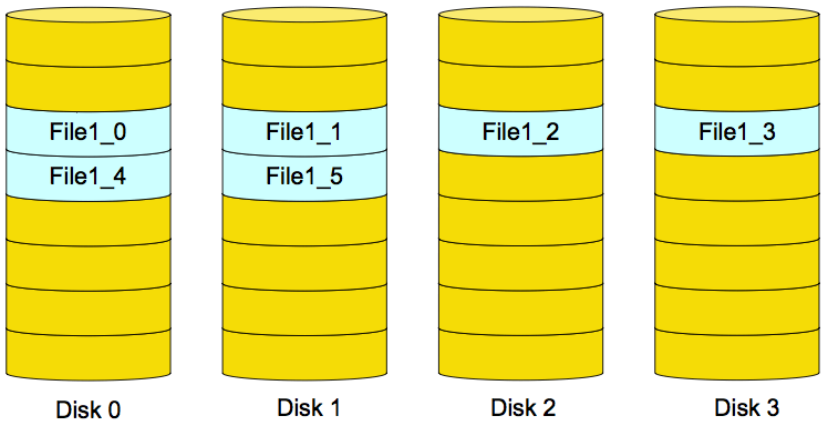
NAND Flash tároló:
- alacsony szintü interface azonos a merevlemezével
  - Solid State Disk (SSD) SATA vagy IDE interfésszel
  - PEN drive USB interfésszel: kártya olvasók is így müködnek, cserélhetö tárolóval, más-más tároló foglalattal
  - olvasás gyors, független az adatot tároló blokk elhelyezkedésétöl
    - nem kell fejeket mozgatni és szektort pozícióba forgatni
    - RAM jelleggel érthetö el
  - az írás (valójában a törlés) problémás
    - véges számú alkalommal törölhetö egy blokk
    - az írás/törlés lényegesen lassabb: párhuzamosítható több blokk írása gyorsíthat, ha az eszköz támogatja

Eszköz csatlakozás:
- a hoszt számítógéphez csatlakozó megoldások (Host Attached Storage):
  - direkt csatlakozás: SATA/eSATA, IDE, SCSI, SAS, stb.
  - indirekt csatlakozás:
    - USB, Firewire alapú alagút (tunnel)
    - RAID (Redundant Array of Inexpensive Disks)
- hálózati tároló eszközök (Storage-Area Networks, SAN)
  - hálózati alagút (tunnel) a hoszt és a tároló eszköz között
  - speciális protokollok: Fibre channel
  - Ethernet és/vagy TCP/IP alapú: iSCSI, AoE
- A NAS (Network-Attached Storage, File megosztás, stb.) nem ezen a szinten valósulnak meg

USB (USB mass storage deive class):
- SCSI transzparens parancs készlet kerül átküldésre az USB buszon
- az OS számára egy SCSI buszon keresztül csatlakozó eszköznek tünik
- az USB csupán egy alagutat képez az eszköz és az OS között

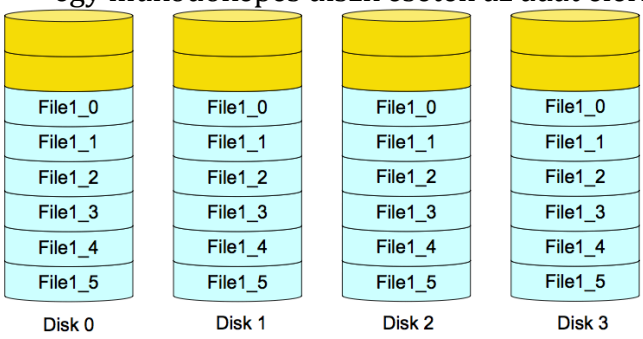
RAID:
- tények:
  - a merevlemezek olcsók
  - nem megbízhatóak (morgó alkatrészek, érzékenyek)
  - lassúak
- ötlet: használjuk belöle többet egyszerre
  - több redundáns alkalmazása növeli a megbízhatóságot
  - több párhuzamos használata növeli a sebességet
  - hozzunk létre egy virtuális diszket a fizikai diszkekböl (OS majd azt kezeli)
- megvalósítások:
  - HW RAID vezérlök
  - SW RAID vezérlök
    - alaplapi megoldások is ilyenek szinte kivétel nélkül
    - szerver alapokban esetleg van HW RAID
- szintek:
  - RAID 0-6 és egymásba ágyazott (nested) szintek
  - RAID 0-1 szabványok átlalában SW implementációval es kevés(2) diszkkel
  - RAID 2-4 szabványokat ritkán használjuk
  - RAID 5 és 6 alkalmazása tipikus nagyobb számú diszk esetén(4 vagy több)
  - egymásba ágyazott szintek: RAID 1+0, RAID 0+1
  - vannak gyártó specifikus nem szabványos megoldások

RAID level 0 (striped disks):
- több diszk párhuzamos használata
- a file részei N diszkre kerülnek, az egyes részek egymástól függetlenül érhetöek el
  - a diszkek tároló kapacitás összeadódik
  - N azonos diszk esetén a RAID 0 virtuális diszk olvasási és írási adatátviteli sebessége maximum N-szeres közelében nö
  - a hozzáférési idö közel eléri egy diszk hozzáférési idejét
  - bármelyik diszk meghibásodás esetén elveszik az adat



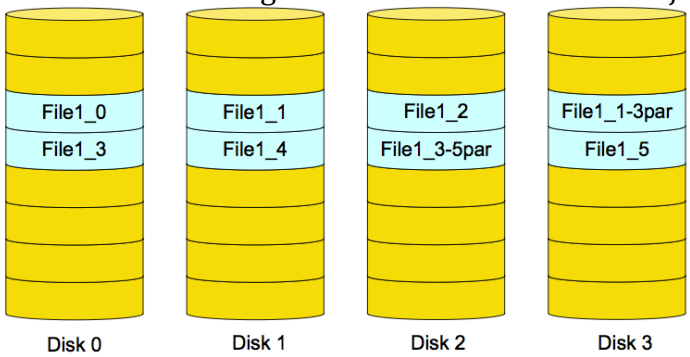
RAID level 1 (mirroring):

- több diszk redundáns használata
- a file minden része minden (N) diszkre kikerül
- azonos diszkeket feltételezve a tároló terület egy diszk tároló területével azonos
- az adatátviteli sebesség lassabb mint egy diszk sebessége
- a hozzáférési idö nö
- speciális esetben az olvasási sebesség N-szeresre nöhet
- egy müködöképes diszk esetén az adat elérhetö



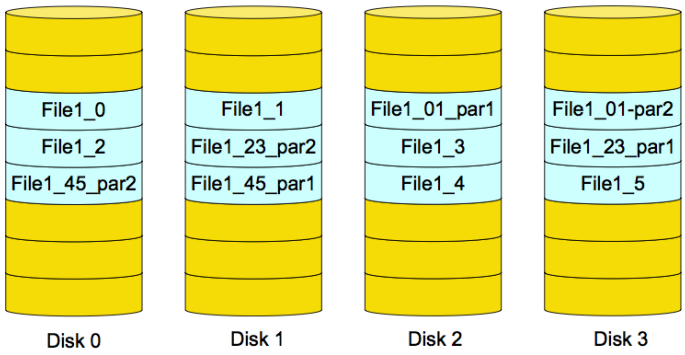
RAID level 5 (block interleaved distributed partity)

- több diszk redundáns és párhuzamos használata
- adat és paritás elosztása N+1 diszkre:
  - a sebesség tekintetében közel áll az N diszket használó RAID 0-hoz (HW támogatás esetén)
  - 1 diszk meghibásodása esetén az adat elérhetö
  - 2 vagy több diszk meghibásodása esetén az adat elveszik
  - az adat nem feltétlenül állítható helyre
    - csendes/néma hibák (silent error)
    - a 2. meghibásodás észlelése a tömb újraépítése során



RAID level 6 (block interleaved dual distributed paritity):

- több diszk redundánd és párhuzamos használata
- adat és paritás elosztása N+2 diszkre
  - a sebesség tekintetében közel áll az N diszket használó RAID 0-hoz (HW támogatás esetén)
  - 2 diszk meghibásodása esetén az adat elérhetö
  - 3 vagy több diszk meghibásodása esetén az adat eveszik
  - az adat nagyobb valószínüséggel állítható helyre a RADI 5-höz képest



RAID kritikája:

- hamis biztonságérzet:
  - csak a merevlemez egyedi, véletlen meghibásodása ellen véd
  - nem véd a SW hibáktól, illetéktelen hozzáféréstöl, stb.
  - nem pótolja a biztonsági másolatokat, csak a rendelkezésre állási idöt és a sebességet növelheti
- A HW RAID vezérlök drágák:
  - 8 portos SATA RAID RAID5 és RAID 6 támogatással kb 200e FT
  - drágább, mint a hozzá csatlakozó diszkek
  - komplett gépet lehet venni ekkora összegböl
- a SW raid megoldások elsösorban RAID 0 és RAID 1 esetén alkalmazhatóak
  - lassú a RAID 5 és 6 bonyolult kódolásának SW megvalósítása

RAID elönyei:

- a RAID 1 és RAID 5,6 megvéd a tipikus véletlen HDD hibák által okozott azonnali rendszer leállástól
  - a HDD a leggyengébb láncszem
  - SMART hatásos hibák elörejelzésére, de nem jelez biztosa elöre hibákat
- a RAID 0 és RAID 5,6 gyorsíthatja a diszk hozzáférését

Hálóztai tároló eszközök (SAN):

- Hálózati "tunnel"a hosst és a tároló eszköz között
  - alacsony, blokk szerintü megoldás
  - többnyire SCSI parancsokat küldenek át
  - a tároló eszköz virtualizációja:
    - teljes értekü, tetszölegesen skálázható, partícionálható, bootolható,...
    - mintha lokálisan lenne a merevlemez csatlakoztatva
    - átlalában egy géphez csatlakoztathatók csak (kivéve fürtök/cluster)
- megoldások:
  - speciális protokollok: Fibre channel (drága, dedikált HW)
  - Ethernet és/vagy TCP/IP alapú: iSCSI, AoE
    - olcsó/ingyenes, részben SW, de legalább firmware támogatás is kell
- konvernciók:
  - target: hálózati tároló eszköz, amihez a fizikai tároló eszközök direkt módon,vagy további SAN szinteken keresztül csatlakoztatva vannak
  - initiator: a kliens, ami használja a tároló eszközöket
  - elnevezési konvenciók a használandó tároló eszköz azonosítására
  - initiator szintü hozzáférés ellenörzése

Alacsony szintü fájlrendszer:
- fizikai diszk blokkok írását és olvasását végzik
- egyben feladata a müködés során használt információk cache-elése is
  - puffer cache (buffer cache): a lapozás során külön cache szolgál a pagefile cache-elésre, és külön a blokkok cache-elésére (beleértve a pagefile cache tartalmát is)
  - egységes puffer cache (unified buffer cache): a cache a blokk szinten müködik, nincs külön pagefile cache
  - egységes virtuális memória (unified virtual memory)
    - a lapozás és a fájlrendszer szinten megvalósított, OS egészére vonatkozó disk cache összevonása
    - a file lényegében virtuális memóriára van leképezve
    - pl.: Linux ezt használja

Fájlrendszer leképzése:
- logikai blokkok leképzése fizikai blokkokra (allocation)
  - folytosonos allokáció (contiguous allocation)
  - láncolt listás allokáció (linked allocation)
  - lindexelt tárolás (indexed allocation)
- az üres helyek menedzselése (free-space management)
  - bit vektor (bit vector)
  - láncolt lista (linked list)
  - szabad helyek csoportjainak listája (grouping)
  - számlálás (counting)
  - egybefüggö szabad területek nyilvántartás (space maps)

Folytonos allokáció (contiguous allocation):
- a fájl egy folytonos fizikai blokk sorozatot foglal el
  - hozzáférés egyszerü és gyors HDD esetén
  - Növekvö mértü fájloknak a helyfoglalás problémás: milyen mértü szabad helyet foglaljunk?
  - új fájlok számára megfelelö szabad hely megtalálása nehéz, külsö tördelés lép fel
    - fájl tölés után a méretének megfelelö számú blokk felszabadul
    - erre a helyre kisebb vagy egyenlo mérteü fájl írható
      - first fit, next fit, best fit, worst fit
- növekedö fájlok
  - a best fit allokáció stratégia különösen veszélyes
  - a fájl nagyobb szabad helyre másolása
- külsö tördelödés csökkentése:
  - teljes másolás egy üres diszkre majd vissza (off-line)
    - rendszerleállással jár
    - hosszú ideig tart és eröforrás igényes
  - futás idjeü (on-line) töredezettség csökkentés (defragmentation)

Láncolt listás allokáció (linked allocation):
- a könyvtárakat leíró adatstruktúrák tartalmazzák az elsö és az utolsó blokk azonosítóját
- minden blokk tartalmazza a következö blokk azonosítóját
- a fájlhoz tartozó blokkok tetszöleges helyen lehetnek a diszken
- nincs külsö töredezettség
- problémák
  - szekvenciális fájl elérésére alkalmas, a fájlban indexelni nehéz
  - a blokkban levö azonosítók helyet foglalnak
  - sérülékeny (azonosítók füzik a blokkokat össze)
  - sok fejmozgást okoz (seek), ha a blokkok el vannak szórva a diszken
- pl.: FAT fájlrendszer ezt használja
- töredezettség mentesség ebben az esetben mást jelent:
  - cél a fejmozgás minimalizálása egy file olvasása során
  - a fájlok egymás utáni blokkokon történö tárolását tüzi ki célul
  - ezt is defragmentation-nak hívják
  - ilyen célból ajánlott idönként töredezettségmentesítést futtatni
    - írást és olvasást is jelentösen gyorsíthat

Indexelt tárolás (indexed allocation):
- index blokkok használata: egyes blokkokat fájlokhoz tartozó indexek tárolására allokálunk
- szekvenciális és indexelt elérésre is alkalmas
- sérülékeny (az index blokkok sérülése a fájlt elérhetetlenné teszi)
- sok fejmozgást okoz (seek), ha ablokkok el vannak szórva a diszken
  - itt is lehet töredezettségmentesítést futtatni

Logikai fájlrendszer:
- OS specifikus: OS specifikus API a tetjén (tipikus API függvények: create, delete, read, write, set/get attrubutes, stb.)
- metaadat tárolása (minden, kivéve a tényleges adat)
- fájlok:
  - absztrakt adattípus (objektum, fájl mutató)
  - adat, név, típus tulajdonságok (attributes)
  - kölcsönös kizárás (file locking)
- könyvtárak (Directory/folder)
- kötetek (volume/drive)

Fájl:
- a fájl a permanens táron az adattárolás logikai egysége
- tulajdonságok: név, típus, tulajdonságok, jogosultságok, hozzáférési idö

Könyvtárak:
- az információ hierarchikus tárolására
- kialakítások az idö múlásával:
  - egyszintü => kétszintü => fa => aciklikus irányított gráf => ált. gráf

Aciklikus irányított gráf struktúra: (pl: UNIX/Linux hard és symbolic links)
- egy fájl vagy alkönyvtár több könyvtárban megtalálható, de csak egy példányban létezik

Általános gráfk struktúra:
- OS-eken ritkán használjuk
- gondok: keresés, keresés leállási pontja

Kötetek:
- a logikai fájlrendszer szintén legmagasabb egység
- megfelel egy fizikai vagy logikai partíciónak a fizikai tárolón
- Windowsban C: | UNIX/Linux mount

Adatszerkezetek az eszközön:
- alacsony szintü adatstruktúrák
- boot szektor (boot control block)
- partíciós tábla (volume control block)
- fájlrendszer specifikus információ (könyvtárstruktúra leírói, fájl leírók)

Adatvesztés:
- a fájlok egy idöben a memóriában és a permanens táron is jelen vannak
  - eltérö állapotban lehetnek
  - a metaadatokat, az allokációs struktúrákat is módosítják
  - meghibásodás/tápfeszültség elvesztése inkonzisztenciát okozhat
- konzisztencia ellenörzése
- a konzisztencia visszaállítása: tranzakció orientált fájlrendszerek
- biztonságos rendszerleállítás, szünetmentes táplálás (UPS)
- adatmentés és visszaállítás
  - a mestésböl a helyreállítást tesztelni kell
  - ha nincs helyreállítási teszt, nem beszélhetünk adatbiztonságról

Széles körben elterjedt fájlrendszerek:
- FAT (File Allocation Table)
  - 8+3 karakteres fájlnév, a hosszú fájlnév külön fájlba tárolva
  - FAT16: max 2GB partíció, 32767 könyvtár bejegyzés
  - FAT32: 2TB partíció méret, file méret: 4GB-1Byte
- NTFS (New Technology System)
  - 2^{64} Byte(16 EB) - 1KB max fájlméret, 2^{32}-1 fájl
  - 256 karakter hoszzú fájlnév
  - tranzakció alapú
  - töredezés mentesítés ennél is szükséges
- EXT2
  - alapértelmezett Linux fájlrendszer korábban
  - van Windows driver is, lehet Windows alatt is használni
  - max fájlméret: 16GB - 2TB, max 10^{18} fájl
  - max fájlnév hossza: 255 Byte, max partíció: 2-32 TB, lassú töredezés
- EXT3
  - EXT2 javított, tranzakció kezeléssel kiegészített verziói
  - Htree alapú indexelés: több könyvtárat tesz lehetövé
  - ez a javasolt Linux fájlrendszer, kivéve a flash eszközöket
  - EXT2 és EXT3 között egyszerü a konverzió
- EXT4: további bövítések (nagyobb tárak kezelése, extents, stb.)
- CD-ROM/DVD fájlrendszerek

NAS (Network-Attached Storage):

- fájlrendszer szintü hálózati fájlmegosztás (többnyire nyomtató is)
- pl: Network File System (NFS)
- fájlrendszer szintü megosztás
  - a hálózaton a könyvtárakra és fájlokra voantkozó utasításokat küldünk át
  - jellegzetesen párhuzamosan több felhasználó érheti el
  - felhasználó szintü jogosultságok
  - problémák a kliens és szerver fájlkezelési konvenciójának eltéréséből
- HTTP nem ilyen, komplett fájl osztható meg, nem lehet indexelni

21. UNIX fájlrendszerek alapismeretei

Ismétlés:
- a folyamatok
  - a felhasználói programok futás alatt állópéldányai
  - a programokat permanens tárból töltjük
- a permanens tárak
  - nem felejtö, nagyságrendekkel nagyobb és lassabb a memóriánál
  - blokkos fizikai tárolás és fájl-alapú logikai szervezés
  - többféle megoldás egyedi jellemzökkel (HDD, flash, usb, RAID, SAN)
- a kernel
  - kezeli a hardver eröforrásokat (köztük a permanens tárakat)
  - a hardverkezelö réteg felett többszintü fájlrendszer réteg található
    - háttértár kezelés, fájlrendszer szervezés, logikai felépítés
  - adminisztrálja a fájlok blokkjait és az üres helyeket a permanens tárban
  - elvégzi a fizikai és logikai szervezés közötti leképzést
  - programozói interfészt nyújt az alkalmazásfejlesztök számára

Alapfogalmak:
- fájl : adattárolási hely
- fájlrendszer: fájlok tárolásának szervezése, hozzáférés biztosítása
- fájlrendszerek felhasználói felülete:
  - programozói (API, rendszerhívások)
  - parancssori (illetve grafikus)
- fájlrendszerek szervezési felülete: diszk szervezés

UNIX fájlrendszer történelmi áttekintése:
- System V elsö fájlrendszere: s5fs
- 4.2 BSD Fast File System (FFS)
  - megnövelt teljesítmény
  - új szolgáltatások
  - akkori diszk hardver felépítéshez optimalizált rendszer
- virtuális fájlrendszerek (vnode/vfs)
  - moduláris, objektum-oritentált
  - cserélhetö szervezési modulok, akár hálózati is
- elosztott fájlrendszerek
  - NFS: transzparens hálózati fájlrendszer RPC megvalósítással
- modern fájlrendszerek:
  - ext3, ext4, xfs, ReiserFS, Solaris ZFS
  - fehasználói fájlrendszerek gnome-vfs, fuse: ftp, smb, dav, stb. célra
  - klaszter fájlrendszerek, pl. Red Hat GFS

A fájlrendszer felhasználói szemmel:

- OS felhasználó:
  - parancssori grafikus felület
  - könyvtárszervezés, speciális könyvtárak
  - fájlok és könyvtárak kezelése, attribútumaik
  - fájlrendszerek menedzselése (rendszergazda)
- Programozó (alkalmazás fejlesztö):
  - programozói interfészek (rendszerhívások, rendszerkönyvtárak)
  - fáljleírok, nyitott fájl objektumok és kezelésük
  - zárolási módszerek: kötelezö, ajánlott

Felhasználói intefész:

- diszkek, partíciók, fájlrendszerek
- tipikus UNIX könyvtárszerkezet
- fájlrendszerek csatlakoztatása a könyvtárszerkezethez
- fájl attribútumok:
  - típus (- d p l b c s)
  - linkek (hard, szoft)
  - eszköz, inode, méret, stb.
  - idöbélyegek (ctime, mtime, atime)
  - azonosítási és hozzáférés-szabályozási adatok
  - listázási parancs: ls, ls -all, ls -al, ...

Programozói interfész:

- fájlok megnyitása (létrehozása)
  - open() rendszerhívás és paraméterei
  - a fájlleíró és a nyitott fájl objektum
  - több folyamat által megnyitott fájl és fork()
- írás és olvasás: read(), write()
- fájlok zárolása:
  - kötelezö (mandatory): fcntl(), lockf()
  - ajánlott (advisory): flock()
- fájlok lezárása: close()

Fájlrendszerek szervezése:

- csatlakoztatás:
  - csatlakozási pont
  - eldefés
- szervezés a háttértáron:
  - blokkos tárolás
  - fájlok leírói (diszk inode)
  - szabad helyek kezelése
- szervezés a memóriában
  - csatlakoztatás nyilvántartása
  - fájlok leírói (memória inode)
  - kapcsoalt a nyitott fájl objektummal

A tárolás megvalósítás:
- a diszken elhelyezett fájlrendszer részei:
  - szuperblokk (fájlrendszer metaadatok)
  - inode lista (fájl metaadatok)
  - tárolt adatok
- szuperblokk
  - a fájlrendszer mérte
  - szabad blokkok jegyzéke
  - zárolási információk
  - módosítás jelzöbit

Az index node (inode)
- hitelesítési információk (UID, GID)
- típus, hozzáférési jogosultságok, idöbélyegek, méret
- adatblokkok elhelyezése (címtábla)
  - 10-15 db direkt blokkcím
  - 1x, 2x, 3x indirekt blokkcímek
- incode a memóriában:
  - a nyitott fájl objektumhoz kapcsolódik
  - diszk incode tartalma bekerül a memóriába
  - az aktív használat információval bövül
    - státus (zárolt, módosított, stb)
    - háttértár eszköz (fájlrendszer) azonosítója
    - hivatkozás számlálók (fájlleírók)
    - csatlakozási pont adminisztrációja, ...

Allokáció a diszken:
- szempontok: teljesítmény, megbízhatóság
- Cilinder (blokk) csoport (FFS, ext2, ...)
- allokációs elvek
  - szuperbolokk másolása minden csoportba
  - inode lista és szabad blokkok csoportonként kezelve
  - egy könyvtár - egy csoport
  - kis fájlok egy csoportba
  - nagy fájlok "szétkenve" több csoportba
  - új könyvtárnak egy új, kevéssé foglalt csoportot keres

A virtuális fájlrendszer:
- impelentáció-független fájlrendszer absztrakció
  - a modern unix fájlrendszerek alapjai
- célok:
  - többféle fájlrendszer egységes egyidejü támogatása
  - egységes kezelés a csatlakozás után (programozói feladat)
  - speciális fájlrendszerek (hálózati, processzor, stb)
  - modulárisan bövíthetö rendszer
- absztrakció:
  - inode -> vnode
  - fs -> vfs

A vnode absztrakció:

- adatmezök
  - közös adatok (típus, csatlakoztatás, hivatkozási száml.)
  - v_data: állományrendszertöl függö adatok (inode)
  - v_op: az állományrendszer metódusainak táblája
- virtuális függvények
  - állományrendszertöl független: vop_open, vop_read, ...
  - a tényleges metódusokra helyettesíthetödnek be
- segédrutinok, makrók

A vfs absztrakció:

- adatmezök
  - közös adatok (fájlrendszer típus, csatlakozás, hivatkozás, vfs_next)
  - vfs_data: állományrendszertöl függö adatok
  - vfs_op: az állományrendszer metódusainak táblája
- virtuális függvények
  - állományrendszertöl független: vfs_mount, vfs_umount, vfs_sync, ...
  - a tényleges metódusokra helyettesíthetödnek be
- segédrutinok, makrók

22. Felhasználó- és jogosultságkezelés

Fontos!
- Személyes adatok megvédése. Vissza lehet velük élni.
- Banki, üzelti szféra.
- beágyazott, zárt rendszerekben: megpróbálnak majd belepiszkálni
- alkalmazások: felhasználó által elvárttól eltérö viselkedés

Mikor fontos a számítógépes biztonság?
- szoftverfejlesztésben minden szinten foglalkozni kell a biztonsággal
- *"ha egy rendszert nem terveztek biztonságosra, akkor később lehetetlen azzá tenni."*
- a rendszer biztonsága a leggyengébb láncszem biztonságával azonos. *"Az OS nem csodaszer, rosszul megírt alkalmazás ellen nem véd"*

Biztonság fogalma:
- cél: garantálni, hogy a rendszer midnig az elvárt módon viselkedjen
- egy technológia magában kevés
- a biztonság mindig csak a célkitüzés függvényében értelmezhetö
- bizalmasság, sértetlenség, rendelkezésre állás

Biztonságot biztosító eszközök:
- kriptográfia: kommunikáció sértetlenségéhez, bizalmassághoz kell
- platform szintü behatolás elleni védelem:
  - rendszeren futó alkalmazások sértetlensége
- hálózati behatolás elleni védelem
- redundancia, újrakonfigurálás: rendelkezésre állás
- hitelesítés, engedélyezés

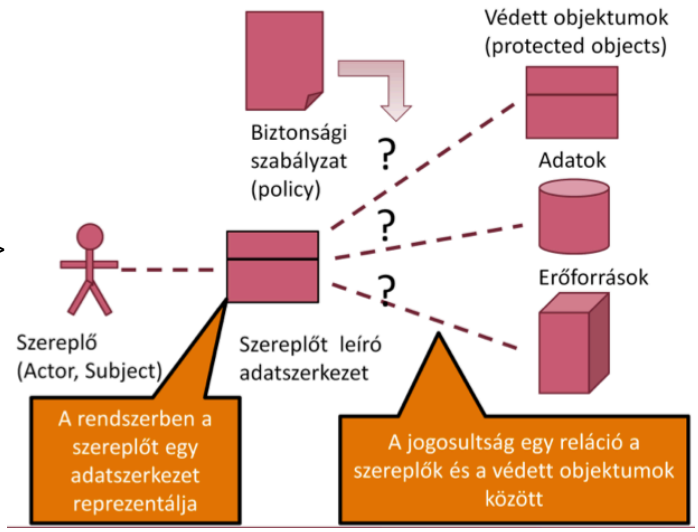
Tartalom:
- számítógépes biztonság bevezetö
- felhasználó kezelés, hitelesítés: UNIX/Windows alatt
- engedélyezés:
  - engedélyezés általános sémái
    - szerep alapú hozzáférés-vezérlés
    - hozzáférési jogosultság listák
  - engedélyezés UNIX, Linux alatt
  - engedélyezés Windows alatt
    - biztonsági alrendszer alapok
    - központosított hozzáférés-vezérlés

Hitelesítés:
- mi alapján döntjük el, hogy ki kicsoda?
  - amit tud (pl.: jelszó)
  - amije van (pl.: kulcsa, belépökártya)
  - ami ö (pl.: ujjlenyomat, arckép)
- ezek alapján egy (sértetlen) gép el tudja dönteni ki a személy aki elötte ül
  - mi a helyzet, ha nem sértetlen a gép?
  - mi a helyzet a gép-gép közötti szolgáltatásokkal?
- hitelesítés 3 szinten kerül elö:
  - ember és gép közötti interakció
  - gép és gép között valamilyen hálózaton át
  - gépen belül futó alkalmazások valamint az OS között
- hitelesítési protokollok kellenek
  - gépen belül ill. gépek között csak az "amit tud" séma lehetséges
  - itt már feltételezhetö bonyolult kriptográfiai számítás elvégzése

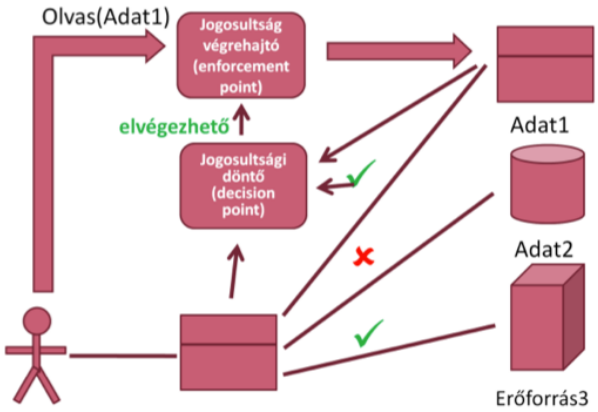
Miböl áll egy felhasználói fiók:
- a rendszer száma'ra a felhasználó egy objektum
- a fehasználói fiókot azonosítja:
  - Linux, UNIX alapú rendszerek alatt UID
    - interger (root 0, felhasználó 1000-...)
    - /etc/password,shadow,group tárolja az attribútumokat
    - (csoporthoz is van hozzárendelt jelszó)
  - mik ezek az attribútumok:
    - login név
    - jelszó (megváltoztathatóság, lejárati idö)
    - home könyvtár
    - alapértelmezett shell (illetve shell belépés megtiltása)
    - alapértelmezett csoporttagság
    - komment (valódi név)

Azonosítás Linux alatt:
- gépen belül
  - felhasználó UID-név hozzárendelés feloldása gyakran kell -> /etc/passwd-t mindenkinek kell tudnia olvasni
  - jelszó ezért nem itt van, hanem /etc/shadow alatt, hash-elve
- többi kód között?
  - pl.: ssh-nál?
  - felhasználó név/jelszó
  - kriptográfiai kulcs alapján

Engedélyezés általanos sémái ------->



Hozzáférés végrehajtása:



Jogosultságkezelés alapjai:

1. A rendszer müködése során:

- a szereplök müveleteket kezdeményeznek
- a müveletek kontextusa tartalmazza a szereplö azonosítóját, a célobjektumok és az elvégzendö müvelet fajtáját
- a jogosultsági döntö komponens kiértékeli a kontextust
  - engedélyezi vagy megtiltja a müveletet
- a jogosultsági végrehajtó komponens biztosítja, hogy a döntö által hozott döntés érvényre jusson

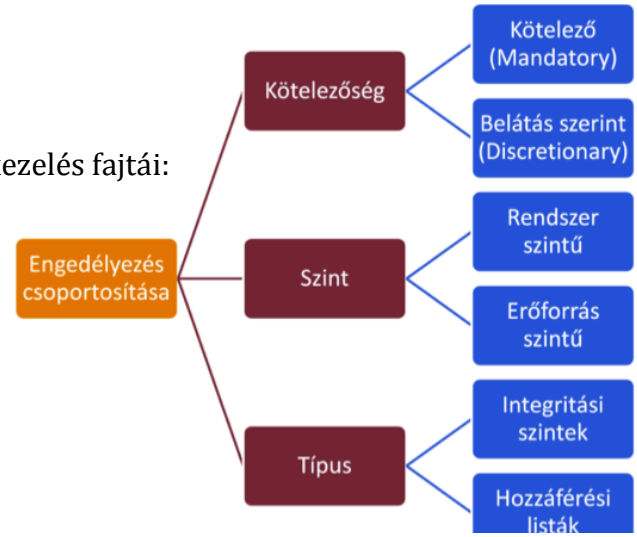
2. A rendszer karbantartása során:

- jogosultságok beállítása, módosítása történik
- a jogosultságot leíró adatszerkezet maga is egy védett objektum
  - tehát lehetnek olyan jogosultságok, amik saját magára hatással vannak
- általában a rendszer folyamatosan üzemel, nincs elkülönített kartantartási idö, a jogosultság módosótások azonnal érvényre jutnak

Jogosultságkezelés gyakorlati kihívási:

- sok szereplöt kell kezelni a rendszerben
  - különbözö rendszerek különbözöképpen azonosítják öket
- sok védett objektumot kell kezelni
  - különzöbö rendszerek ezeket is különbözöképp azonosítják
- jogosultsági relációk:
  - (szereplök) X (objektumok) X (müvelettipusok)
  - az ilyet teljes *hozzáférési mátrixnak* nevezzük
  - manuálisan (automatizáltan is) kezelhetetlen mérteü adathalmaz

jogosultságkezelés fajtái:



Felhatalmazás fajtái - kötelezöség:
- kalasszikus fogalmak (US DoD szbvány)
- kötelezö (mandatory)
    - csak központi jogosultság osztás
    - felhasználók nem módosíthatják a házirendet
- belátás szerint (discretionary)
    - megfelelö jogú felhasználó továbboszthatja a jogokat

Felhatalmazás fajtái - típus:
- integritás védelem
    - objektum címkézése: alacsony, közepes, magas... integritási szint
    - ellenörzés: alacsonyabb szintü nem olvashat/írhat magasabb szintüt
    - Bell LaPadula (bizalmassági) és Biba (sértetlenségi) modellek
- Hozzáférés vezérlési listák
    - objektum -> (szereplök, engedélyek)
        - engedély: adatok írása, attribútumok olvasása...
    - hozzáférési masz (access mask) tartalmazza, hogy pontosan melyik müveletre vonatkozik az engedély
    - szerep alapú hozzáférés-vezérlés (RBAC, Role-based Access Control)
        - a szerep foalom hierarchikus szereplö csoportosítási lehetöséget ad
        - a szükséges engedélyek száma kezeletö szintre csökken

Felhasználói fiók: A felhasználó csoporttagság valójában egy RBAC megvalósítási lehetöség.

Engedélyezés Linux alatt: POSIX fájlrendszer jogosultságok
- alapelvek:
    - szereplö: user
    - szereplö hierarchia: group
    - minden user tetszöleges sok group tagja lehet
    - minden group tetszöleges sok usert tartalmazhat
    - group további groupot nem tartalmazhat
- jogok
    - 3x3 bit, olvasás, írás, végrehajtás (könyvtárba belépé)
        - elsö a tulajdonos felhasználónak
        - második a tulajdonos csoportnak
        - harmadik mindenkinek
    - speciális bitek
        - setuid, setgid: futtatásnál átveszi a file tulajdonos uid-, gid-jét
        - sticky: újonnan létrejött fájlok tuladonosát állítja
    - az execute bit tiltó hatása impilcit módon öröklödik a könyvtárakon, más öröklés nincs

Fájrendszeren kívüli engedélyek:

- speciális kiváltságok root felhasználó nevében futó folyamatoknak
    - kérhetnek válós idejü ütemezési prioritást
    - hozzáférhetnek közvetlenül a perifériákhoz
        - kell elötte memória, illetve I/O tartomány allokáció
        - a közelmúltig így müködtek a grafikus felületet adó X Window server eszközmeghajtó programjai
    - 1024 alatti TCP/UDP porton hallgathatnak
    - kernel bizonyos konfigurációs beállításait megváltoztathatják, új modult tölthetnek be, stb.
- nem elönyos, ha ezek nem szabályozhatóak külön-külön
    - legkevesebb jog elve
    - POSIX Capabilities mechanizmus - globális rendszerszintü eröforrásokra vonatkozó jogosultságok (ún. privilégiumok)

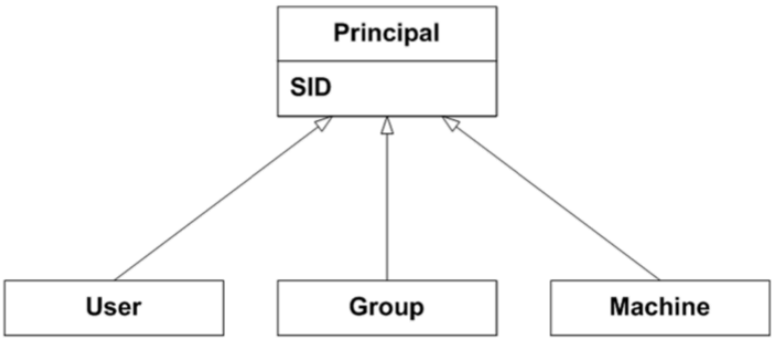
Kitekintés: finomabb felbontású jogosultságkezelés végrehajtható fájlokra:

- platform szintü behatolás elleni védömechanizmusok támogatása (PAX, grsecurity)
- a védömechanizmusok számos egyébként sértetlen programot tesznek müködésképtelenné (JavaVM)
- speciálisan kivételezni kell az ilyen alkalmazásokat fájlrendszerbe írt címkékkel (SELinux Security Labels)
- alkalmazásokhoz hozzárendelt rendszerhívás profilok (AppArmor) - felfedi ha a "szokásoshoz" képest megváltozik az alkalmazás futása

23. Biztonsági alrendszer a Windowsban

Biztonsági feladatok a Windowsban:
- 1. azonosítás (authentication)
  - birtok/tudás/biometria
  - pl. bejelentkezési képernyö, hitelesítö ablakok
- 2. engedélyezés (authorization)
  - alapelv: mindig csoportnak osztunk jogot
  - pl. biztonsági házirend, fájl ACL
- 3. auditálás
  - biztonsági naplózás

Azonoítás (authentication):



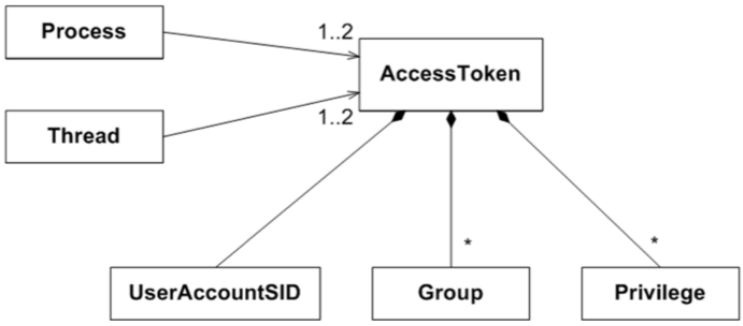
Principal: biztonsági rendszer által kezelt entitások összefoglaló neve

Security Identifier (SID):
- felhasználó / számítógép azonosítója
- felhasználó, csoportok
  - <Gép SID>-<RID>
  - RID: relative identifier
- jól ismert SID-ek
  - everyone: S-1-1-0, administrator: S-1-5-domain-500
- vista: szolgáltatások is kapnak SID-et

1. Azonosítás:
- belépés:
  - winlogon saját ablakán keresztül
  - Secure Attention Sequence: Crtl+Alt+Del
- jelszavak tárolása:
  - hash a registry-ben
- hálózati azonosítás
  - NTLM: NT LAN Manager
  - Kerberos: Windows 2000 óta, tartományi (domain) környezetben

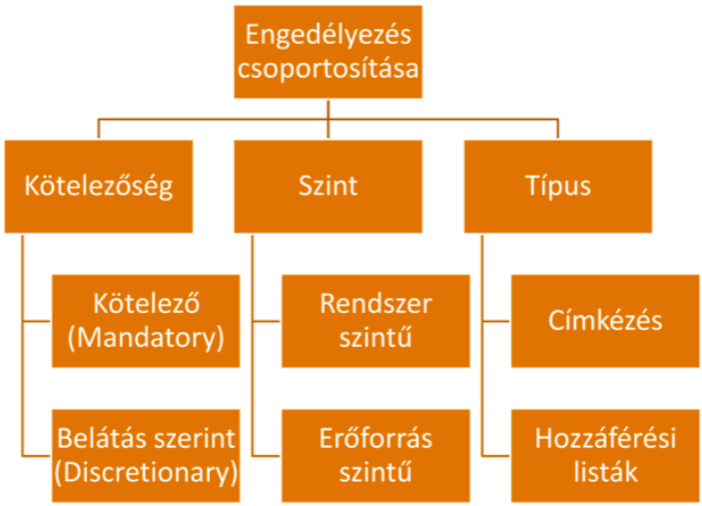
Azonosítás - Hozzáférési token:

- megszemélyesítés
- belépéskor hozzárendel egy hozzáférési token a felhasználóhoz, későbbi müveletek során az ebben tároltakat ellenörzi a rendszer



2. Engedélyezés

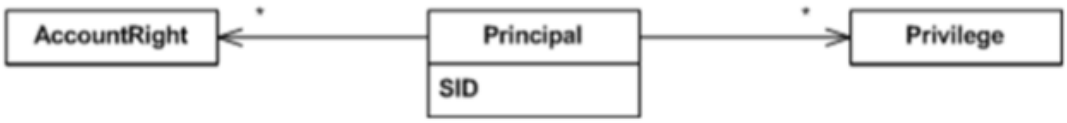
- engedélyezés fajtái



- a csoportosítás esetleges, rengeteg egyéb szempont van, ezek csak a Windowsban szerpelö fogalmak

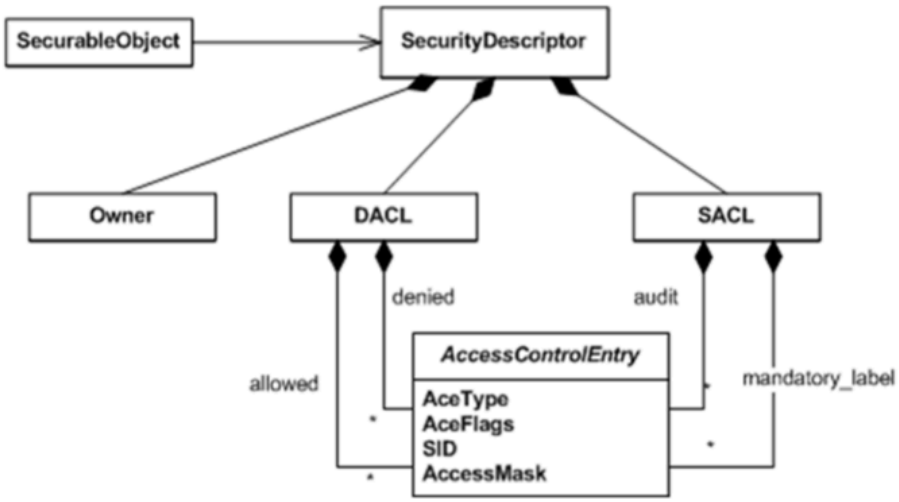
Engedélyezési lehetöségk a Windowsban:

- rendszerszintü jogosultságok



  - jogosultság (privilege)
    - OS szintü jog
    - pl.: számítógép leállítása, eszközmeghajtó betöltés
    - név: SeShoutdownPrivilege, SeLoadDriverPrivilege
  - fiók jog (account right)
    - ki hogyan léphet be / nem léphet be
    - pl.: interaktív, halózaton keresztül...

- Discretionary Access Control
  - belátás szerinti, eröforrás szintü, hozzáférési lista



  - SecurableObject: Windowsos objektum, pl.: fájl, registry kulcs, pipe...
  - SecurityDescriptor: összefogja a többi elemet
  - Owner: megváltoztathatja az objektum engedélyeit, akkor is ha nincs explicit joga
  - DACL: Discretionary Access Control List, hozzáférés szabályozása
  - SACL: System Access Control List, biztonsági naplózás szabályozása
  - AccessControlEntry:
    - típus: megengedö, tiltó, audit
    - flag: pl.: öröklödés
    - SID: kire vonatkozik
    - maszk: végrehajtás | törlés | tulajdonos írása...

  - hozzáférési listák:
    - öröklödés flag: konténer típusú objektumnál, gyerek objektum megkapja azt az ACE-t
    - kiértéklés menete:
      - egy SID-re több ACE is érvényes lehet
      - ACE-kból kapott engedélyek UNIÓJA számít
      - kiveve a tiltást, az mindig magasabb prioritású

- Manadatory Integrity Control
  - kötelezö, eröforrás szintü, címkézés

3. Auditálás (biztonsági másolat)

Eseménynapló:
- rendszeres és alkalmazás üzenetek
- bejegyzés: típus, idö, forrás, ID, leírás
- napló felülírása: ciklikus, idönként, soha