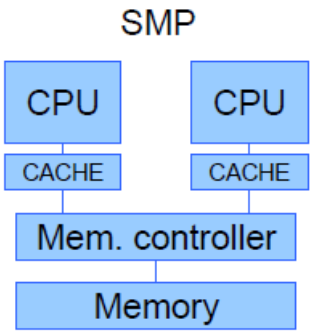
1. előadás: opre bevezető

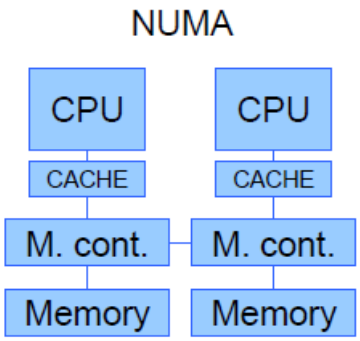
SMP:

- Minden maghoz saját cache, de közös mem. controller, közös mem



NUMA:

- Minden maghoz
  - saját cache
  - saját mem. controller
  - saját memória
- mem. controllerek összekötve



Monolitikus kernel: (embedded)

- Minden egy programba van fordítva
- Beágyazott rendszerek, ahol nem változik a HW

Moduláris kernel: (Linux, Win)

- Minimális kernel, dinamikusan betölthető modulokkal
- Egy komponens hibája → teljes OS hiba
- Újabb Linux kernelek, Windows

Mikrokernel: (OS X)

- Minimális kernel; kliens – szerver architektúrában csatlakozó szolgáltatások
- Erőforrás igényesebb
- Kernel védett a csatlakozó szolgáltatásoktól is
- Mac OS X

Exokernel: (kísérleti)

- Alkalmazások direkt hozzáférése az egyes perifériákhoz (hatékonyság)

MMU feladatai:

- Memória állapotának nyilvántartása
- Virtuális memória → leképezés → fizikai memória
- Memória védelem

OS elindulás (PC):

- Init/RESET vector (CPU)
- BIOS/EFI (firmware)
  - POST (power on self test)
  - HW perif. keresése és init
  - Boot média meghatározása
- BOOT sector (HDD típusú tároló)
- 2. szintű boot loader
- OS betöltődik és elindul
  - HW reprogramming (BIOS → device driver)
  - Kernel módban további initek

2. előadás: UNIX bevezető

UNIX kernel felépítése:

- Réteges
- Moduláris
- Lehet Mikrokernel (pl MINIX), vagy Monolitikus kernel (pl Linux)
- Végeredmény NA: moduláris monolitikus réteges
- **Szervereken:**

    - RedHat, openSUSE, Ubuntu Server, Debian, OpenBSD, FreeBSD, NetBSD, OpenSolaril
- **Klienseken:**
    - Ubuntu, Linux Mint, Fedora, OpenSUSE, Arch Linux

3. előadás: Windows operációs rendszer

Hordozhatóság:

- HW specifikus rész: HAL-ban és a kernel alsó

Kiterjeszthetőség:

- Moduláris felépítés
- Interfészek használata
- Unicode használata (kernelben is)

Megbízhatóság:

- közös címtér
- biztonsági szabványok

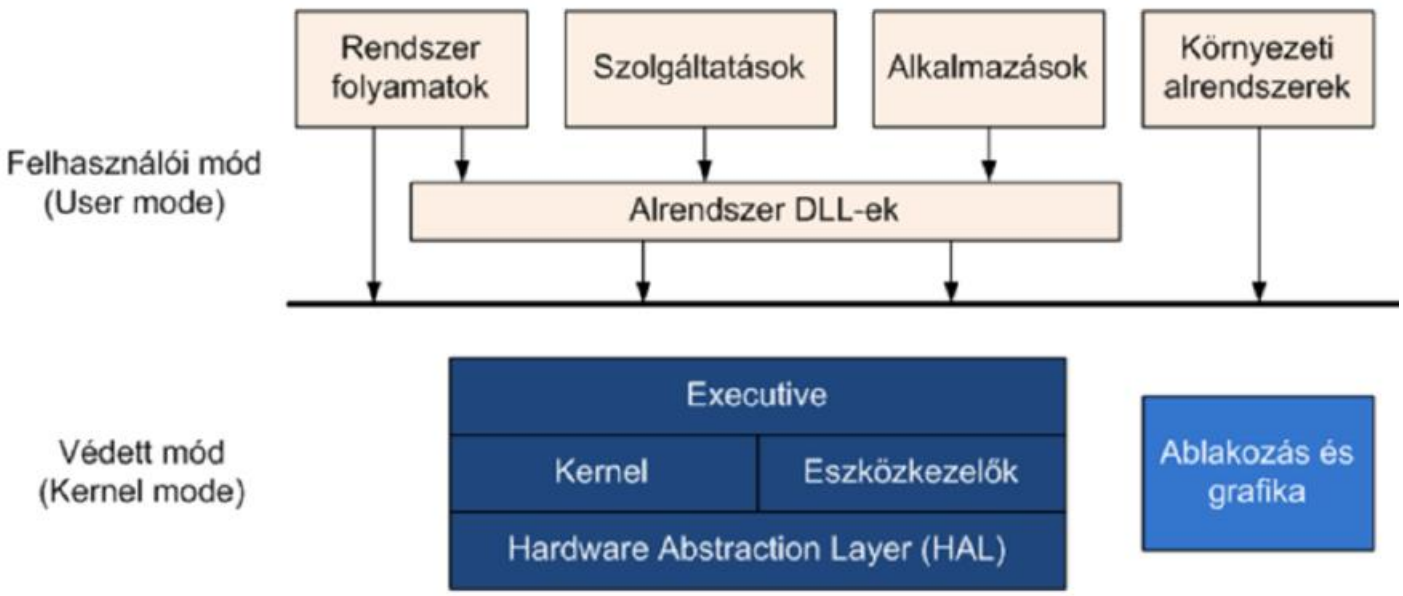
Teljesítmény:

- többszálú
- újrahívható: rendszerhívásokat több app is hívhatja egyszerre
- preemptív, SMP, aszinkron IO

Kompatibilitás:

- POSIX-szal

- Többféle környezeti alrendszer, mind az NT kernel felett
- kernel vagy felhasználói módban futhat a task



HAL:

- alapvető HW egységek elrejtése
    - megszakításkezelő, firmware, CPU detektálása
- hal.dll

Eszközkezelők:

- kernel módú modulok
- rétegzett struktúra
- fájlrendszer, hálózatkezelés is
- *.sys

Kernel:
- alapfunkciók
    - ütemezés, megszakításkezelés, multiprocesszoros sync, környezetváltás
- kevés hardverspecifikus rész
- ntoskernel.exe

Executive:
- magasabb szintű funkciók
    - memória- és folyamatkezelés, biztonság, I/O
- OOP
- ntoskernel.exe

Rendszerfolyamatok:
- alapvető funkciók:
    - inicializálás, bejelentkezés, hitelesítés
- induláskor ezek indulnak először

Szolgáltatások:
- háttérfolyamatok
- UI-tól és belépéstől függetlenül a háttérben futnak
- kibővítik az alapszolgáltatásokat
- DNS kliens, indexelés, RPC

Környezeti alrendszerek:
- folyamatok felhasználói módú kezelése
- állapottárolás
- csrss.exe
- minden környezethet külön API
- rendszerhívások egy részét kínálja

Alrendszer DLL-ek:
- API hívásokat fordítják Executive hívásaira
- pl: kernel32.dll, ntdll.dll
- NTDLL.dll:
    - executive által biztosított fv-eknek megfelelő csonkok
    - uolyan paraméterezés
    - átváltanak kernel módba
    - meghívják az executive beli fv-t.

Ablakozás és grafika:
- GUI kernel módban fut: kevesebb módváltás kelljen (erőforrás) [!= context switch!]
- felhasználói módban a konzol maradt
- ha hiba van benne, rántja a rendszert

Munkamenet: egy user bejelentkezéshez tartozó folyamatok és rendszerobjektumok

Rendszerhívás:
- **alkalmazás:** call ReadFile()
- **alrendszer DLL-ek**: **kernel32.dll:** call ZwReadFile()

- ---------- **eddig volt user módban** ----------
- **executive**: **ntoskernel.exe:** **System Service Dispatcher:** call ZwReadFile()
- **kernel**: **ntoskernel.exe:** do the operation

Windows 8:

- Windows Store appok
- WinRT: windows API-t használó új API
- Windows RT: ARM port, csak előtelepítve

MinWin:

- monolitikus kernel

POSIX a Windows DLLs felett, hogy a GUI függvényeket csak egyszer kelljen megvalósítani

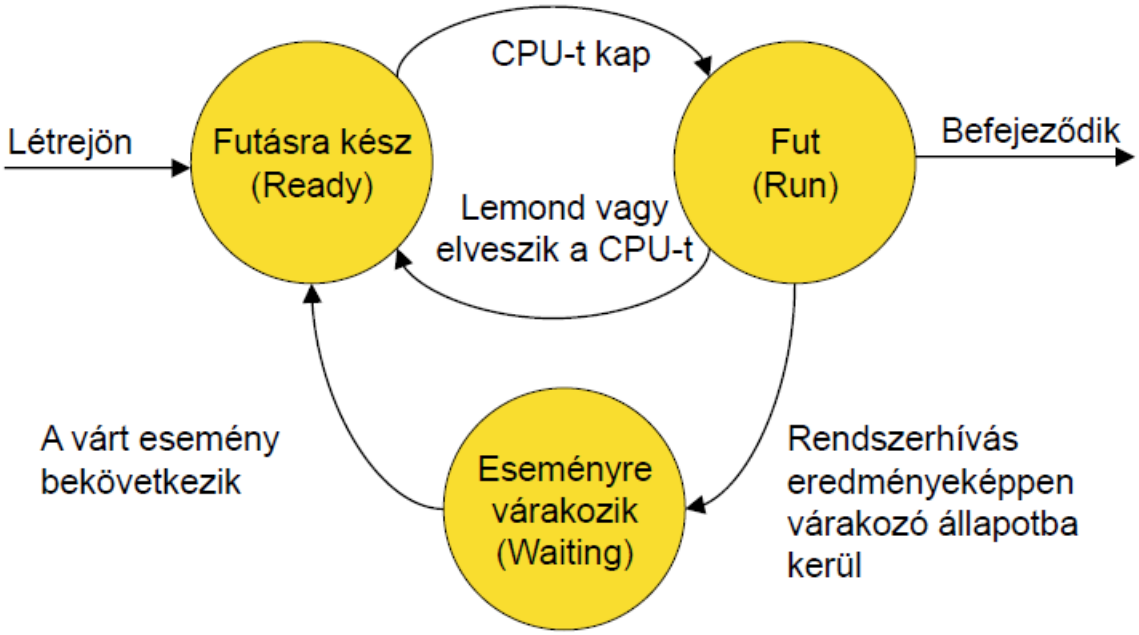
5. előadás: Taskok kezelése

Event:

- Rendszer életében lezajló változás vagy történés
- Belső: szoftver megszakítás vagy kivétel
- Külső: hardver megszakítás

Task:

- A feladatokat folyamatkén valósítjuk meg.
- Műveletek meghatározott sorrendben történő végrehajtása
- több task van, mint program
- állapotai: run(ning), waiting, ready
    - minden állapotátmenet megszakításra történik



Megszakítás vezérelt!

Kontextus váltás:

- taskok közötti váltásnál
- minden taskhoz külön: PC, CPU regiszterek, MMU állapot
- ezeket el kell menteni memóriába → overhead

Feladat leíró (Task Control Block [TCB]):

- Taskkal kapcsolatos adatok tárolása → ez alapján kezeli az OS a feladatokat
- Task ID, Állapot, feladat utolsó kontextusa (regek, PC, cache, mem)

Ütemezések:
- Rövid távú:
  - 10-20ms-enként futásra kész taskok között váltás
- Középtávú:
  - taskok egy része → háttértárra
  - háttértárról memóriába másolást ütemezi
- Hosszú távú:
  - Sokkal több feladat, mint amennyit hatékonyan párhuzamosan végre tudunk hajtani
  - Pl 3-4 nagy fájl másolása egyik HDD-ről a másikra
    - párhuzamosan vagy szekvenciálisan?

Preemptív ütemezés:
- Futó feladattól az OS elveheti a CPU időt

CPU kihasználtság: CPU munkával töltött idő/CPU minden idő (%)
Throughput: időegység alatt elvégzett munkák száma (1/s)
Várakozási idő: összes idő, amit a task várakozással tölt (s)
Körbefordulási idő: végrehajtási idő + várakozási idő (s) [beérkezéstől a befejezésig eltelt idő]
Válaszidő: task megkezdésétől első kimenetek produkálásáig eltelt idő (s) [beérkezéstől az első kimenetig eltelt idő]

Elvárások:
- fairness (korrektség)
- no starvation (nincs éhezés)
- determinism (megjósolható viselkedés)
- low overhead (alacsony fölösleges terhelés)
- max throughput, min waiting times
- erőforráshasználat figyelembevétele

Real time:
- ismert végrehajtási idő → garantált körbefordulási idő

Graceful degradation: fokozatos leromlás:
- ha a terhelés elérte a könyökkapacitást, akkor megváltozhat a rendszer viselkedése
- további terhelésre rosszabb működéssel reagál
- elvárható, hogy ezt fokozatosan tegye (ne omoljon össze hírtelen)

Statikus ütemezés:
- tervezési időben meghatározott, hogy mikor mi fut
- legrosszabb esetre tervezés

Dinamikus ütemezés:
- futási időben dől el mikor melyik task fut
- dinamikus erőforrás kihasználás
- tervezési időben nehezen vizsgálhatók

Ütemezési algók:
- **FIFO, avagy first come first serve:**
  - magas átlagos várakozási idő
  - kis adminisztrációs overhead
  - nem preemptív (pl I/O műveletre várhat)

- **Körforgó (Round-robin):**
  - FIFO + egyszeri óramegszakítás (10-20ms) → preemptív
  - jobb válaszidő than FIFO's
  - Időszelet > átlagos CPU löket → FIFO-ként viselkedik
  - Időszelet = átlagos CPU löket → normális működés
    - CPU löketek 80%-a < időszeletnél
- **Prioritásos ütemezők:**
  - Külső prioritás: feladat maga állítja be
  - Belső prioritás: OS állítja be
  - Statikus/Dinamikus prioritás
  - általában preemptív, de nem muszáj
  - Nem fair, de ez a cél! yo!
  - Kiéheztetés: ha túl van terhelve a rendszer
  - **Shortest Job First:**
    - nem preemptív
    - optimális várakozási és körbefordulási idő
    - lökdetidő becslés WTF impossibrú
  - **Shortest Remaining Time First:**
    - SJF preemptív változata
  - **Highest Response Ratio:**
    - SJF + várakozási idő is számít
    - több várakozás → valószínűbb, hogy ütemeződik
    - (löketidő + k*várakozási_idő)/löketidő
  - Windowsban/Linuxban ofc

6. előadás: Taskok kezelése

Itt jó kis feladat van. Később megnéz! Külön jegyzet van hozzá.

Multilevel Queue:

- Minden egyes prioritási szinthez → várakozási sor
- 8-32 prioritási szint
- Időosztás minden prioritási szinten
- Real time task > rendszer task > interaktv (online) task > batch task (I/O)
- időosztás szintek között → korrektség
  - adminisztráció

Multilevel Feedback Queue:

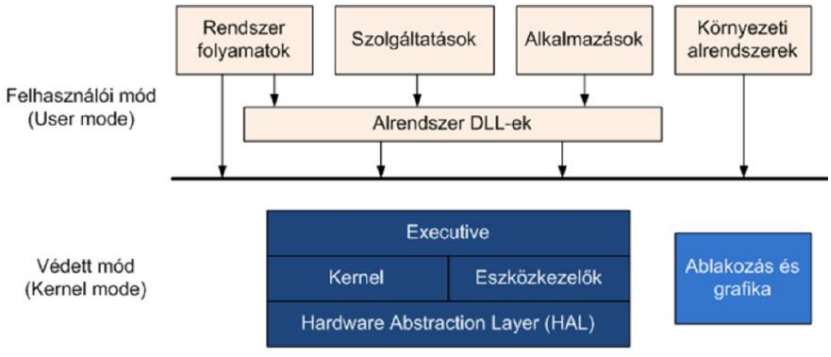
- ténylegesen végrehajtott CPU löketek alapján
- rövid löket → előny
- hosszabb löket → alacsonyabb prioritású és nagyobb időszeletű sorba
- dinamikusan
- kombinál-able másfélékkel
- Windows/Linux

Multiple - processor scheduling:

- Homogén: SMP vagy NUMA
- Master – Slaves vagy Self – scheduling

Processzor affinitás:

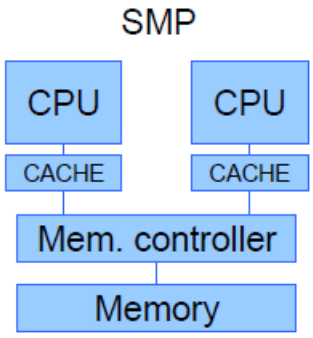
- magok cache tartalma különbözhet
- más magra kerül a task → pfff overhead
- cél: taskot ugyanazon a magon tartani
- laza: nincs gari, de törekszik az OS
- kemény: garantáltan ugyan ott marad



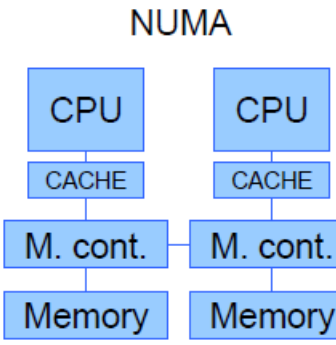
- **SMP v. NUMA:**
  - SMP: csak a cache találatok szempontjából érdekes
  - NUMA: fizikai memória is érdekes az affinitás szempontjából, összetettebb algoritmust igényel

Load balancing (terhelés megosztás):

- processzoronkénti futásra kész sor:
  - push: OS kernel folyamat mozgatja a sorok között a feladatokat
  - pull: idle CPU próbás a többi sorából feladatot szedni
  - kombinálható



7. előadás: Ütemezés windowsban



Program: végrehajtandó kód
Process (folyamat): egy végrehajtás alatt lévő program → több szála lehet, legalább egy thread
Thread (szál): ez fut a CPU-n
Ütemezés:

- preemptív (user és kernel módban is)
- 32 prioritási szint (szálaknak van, nem processzeknek) → 32 FIFO sor
- legmagasabb szál fut mindig
- azonosak között Round Robin (körforgó)
- szálak quantum ideig futnak
- prioritásuk változhat
- 16 felső: „real time”
  - time-critical, highest
- 15 alatta: dinamikusan változtathatja az OS a szintjeiket
  - high, above normal, normal, below normal, idle
- 0.: rendszerszálnak: felszabadított memórialapok kinullázása
- Vista: 5 féle prioritás az I/O kéréseknek
- várakozás végén priority boostot kap a szál
- quantum végén priority decayt
- since Vista: Média lejátszó 21-re, hogy lejátszódjon a kontent mindig
- anti – éhezés:
  - sec-enként megnézi a ready szálakat
  - aki 15-nél kisebb és 300 tick óta nem volt → 15-ös proiritás, nagyobb quantumszám 1 futásig (hosszabban futkos)

Quantum:

- óra megszakításban mérik
  - óra megszakítás periódusideje: 0.5-15.6ms (clock tick ideje)
- futó szál quantumja 3-mal csökken minden óraütéskor (Vista előtt ☹) → azóta nem óraütésben mérik az időszeletet
- quantum hossza:
  - kliens (pl. win7): 2-6 clock tick
  - szerver: 12 clock tick
- előtérszál: hosszabb quantum

Prioritás módosítása:

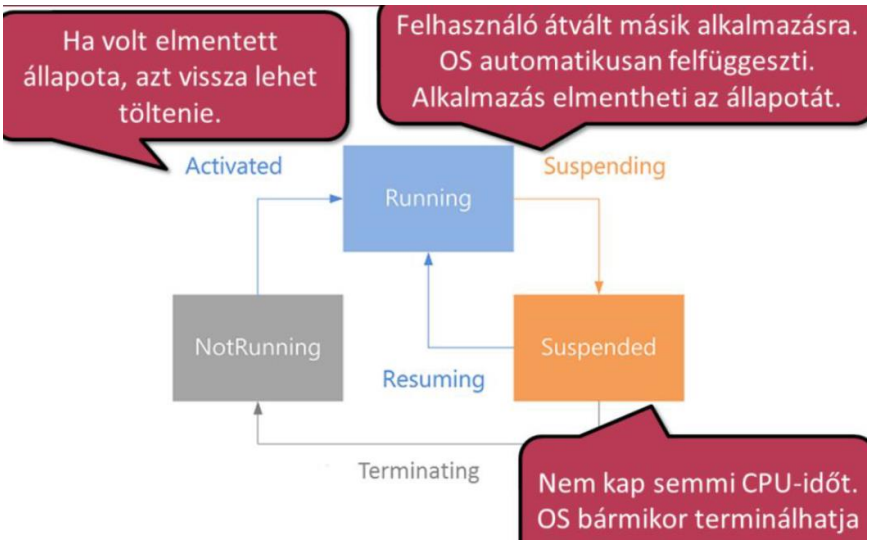
- várakozás végén prioritás növelése
- Quantum végén visszacsökken az eredetire

Éhezés elkerülése:

- másodpercenként futásra kész szálak vizsgálata
- aki 300 tick óta nem futott:
  - 15-ös prioritást kap (ha 15nél kisebb)
  - növelt quantum
  - egy quantumnyi futási

Windows Store:

- kulcsszavakban: erőforrástakarék, könnyű telepítés/frissítés, biztonság, hurka
- életciklus: Running → (Suspending [5s mentésre] →) Suspended → (Resuming →) Running → Suspended → (Terminating →) Not Running



Multiproc. ütemezés: (bit maszk-kal [1 bit – 1 CPU mag])

- soft affinity
- beállítható hard affinity

Windows 7:

- minél kevesebb CPU használata (nem használtat standby-ba rakja)
- Remote Desktor szerverkhez:
  - minden munkamenetet egy CPU keretet kap
    - ha elhasználja, csak idle CPU-t kaphat
- globális zárak megszüntetése

8. előadás: Micro - Controller Operating System II

mikroC/OS-II

- embedded OS
- forráskód rendelkezésre áll
- skálázható, multitasking
- processor specific code elkülönítve → hordozható
- preemptív ütemező, determ. futási idő
- taskonként eltérő stack méret lehet, interrupt management (255 szintű egymásba ágyazódás)
- megbízható
- doksi van
- **Task állapotai:**
  - waiting, ready, running
  - dormant: szunnyadó: memóriában van, de az ütemező hatáskörén kívül
  - ISR: interrupt megszakította.
- **Ütemező:**

    - 2D bitmap struktúrában (**OSRdyTbl** táblázatban az **OSRdyGrp** a sorok)
        - minden taskhoz 1 bit. 1 = futásra kész
        - taskoknak kötelező egyedi prioritás
        - gyors beszúrás/kiolvasás
        - egyedi prioritás
        - lookup táblázat kell: bitminta → legmagasabb prior. futásra kész
    - task kivétele a ready-kből, futásra késszé tétele, legmagasabb prioritású ready megtalálása, kontextus váltás
        - **task kivétele a ready-kből:** várakozó rendszerhívások csinálják:
            - jellemző művelet (pl: szemafor lefoglalása) elvégzése
            - az őt meghívó taskot kiveszi a ready halmazból
            - meghívja az ütemezőt
        - **futásra késszé tétele:** elengedő rendszerhívások csinálják, vagy adott idő letelte:
            - jellemző művelet (szemafor felszabadítás)
            - futássá készé teszi a task(okat)
            - megívja az ütemezőt
        - **legmagasabb prioritású ready megtalálása:** ütemező csinálja:
            - elvégzi a taskváltást, ha nem a legmagasabb prioritású fut
        - **kontextus váltás:** ütemező feladata
    - az ütemező egy függvény, amit más OS fv-ek hívogatnak
- **Időzítő:**
    - timeout OS hívások, idő mérése, időközönként, vagy adott idő múlva fv híváshoz
    - periodikus megszakítások
    - kis ciklusidő → pontosság
    - nagy ciklusidő → kis ovearheadség
- **Egyebek:**
    - prioritások: kisebb szám → nagyobb prioritás
    - OS init és OS indítása a program main fv-ében
- **Újdonságok:**
    - 255 task
    - egyszerre több eventre várás
    - időzítők
    - **OS-III:**
        - tetsz. számú task
        - egy prior. szinten több task: round robin
        - 0 közeli megszakítás tiltási idő

9. előadás: Taskok együttműködése

Task/folyamat

- Eddig **Feladat = Task = Folyamat = Process** volt. Ez továbbra is így lesz.
- **Process**: végrehajtás alatt álló program.
    - ugyan abból a programból több folyamat is létrehozható
    - saját: kód, adat, heap, stack
    - védett a többitől
- Virtuális CPU-n: nem bántja a többi taskot
    - kontextusváltás ha más folyamat kerül futásra
- Saját memóriaterület (proci MMU-ja oldja meg)
    - de lehetséges megosztott terület
- létrehozás: sok adminisztráció
    - szülő/gyermek viszony
    - process fa
    - UNIX fork()
- együttműködés: kommunikáció

    - OS syscall-okon keresztül → erőforrás igényes
    - hatékony védelem, nem hatékony erősen öfő. párh. feladatoknál
- befejezés: sok adminisztráció
    - OS syscall
    - erőforrások lezárása (pl: nyitott fájlok)
- **Szál** bevezetése:
    - CPU használat alapegysége, szekvenciális kód
    - saját: virtuális CPU és stack.
    - amivel azonos kontextusban fut, azzal osztozik ezen: kód, adat, heap, egyéb erőforrások (file)
    - Tehát folyamaton belül több szál lehet.
    - Mostantól:
        - **Szál = Thread = Könnyűsúlyú folyamat**
        - **Folyamat = Process = Nehézsúlyú folyamat**
        - **Feladat = Task = Folyamat vagy Szál**
    - Többszálas folyamatnál:
        - szálakhoz van rendelve a CPU és minden szálnak saját stack-je van
        - a szálak osztoznak a folyamat közös adatán, kódján, heap-jén.
- Windows:
    - program, vagy szolgáltatás = folyamat (process)
        - ezen belül szálak (threadek)
    - ütemező: szálakat ütemez
- Linux:
    - program, vagy daemon = folyamat (process)
        - ezen belül szálak (threadek)
    - ütemező: taskokat, ami lehet folyamat vagy szál

Szálak létrehozása:

- Win32: CreateThread() bonyolul paraméterezéssel
- POSIX (unix és linux): Pthreads, kernel és user szinten is
- Java:
    - VM a folyamat, VM-en belül szálak
    - Threadből leszármazás, vagy Runnable megvalósítás
    - one-to-one: Minden java szál → külön OS szál
    - many-to-one: Minden java szál → 1 OS szál
    - many-to-many: Néhány java szálat összevon 1 OS szálba

Szálak előnyei:

- Kis erőforrásigényű a létrehozás és megszüntetés (kb egy nagyságrenddel mint a processé)
- Appon belüli többszálúság → skálázhatóság (CPU magok kihasználása)
- Gyors kommunikáció közöttük (heap, adat, kód közös)

Közös memórián keresztüli kommunikáció veszélyes:

- memória konzisztencia sérülhet
- kölcsönös kizárás ez ellen

Coroutine és fiber: kooperatív multitasking

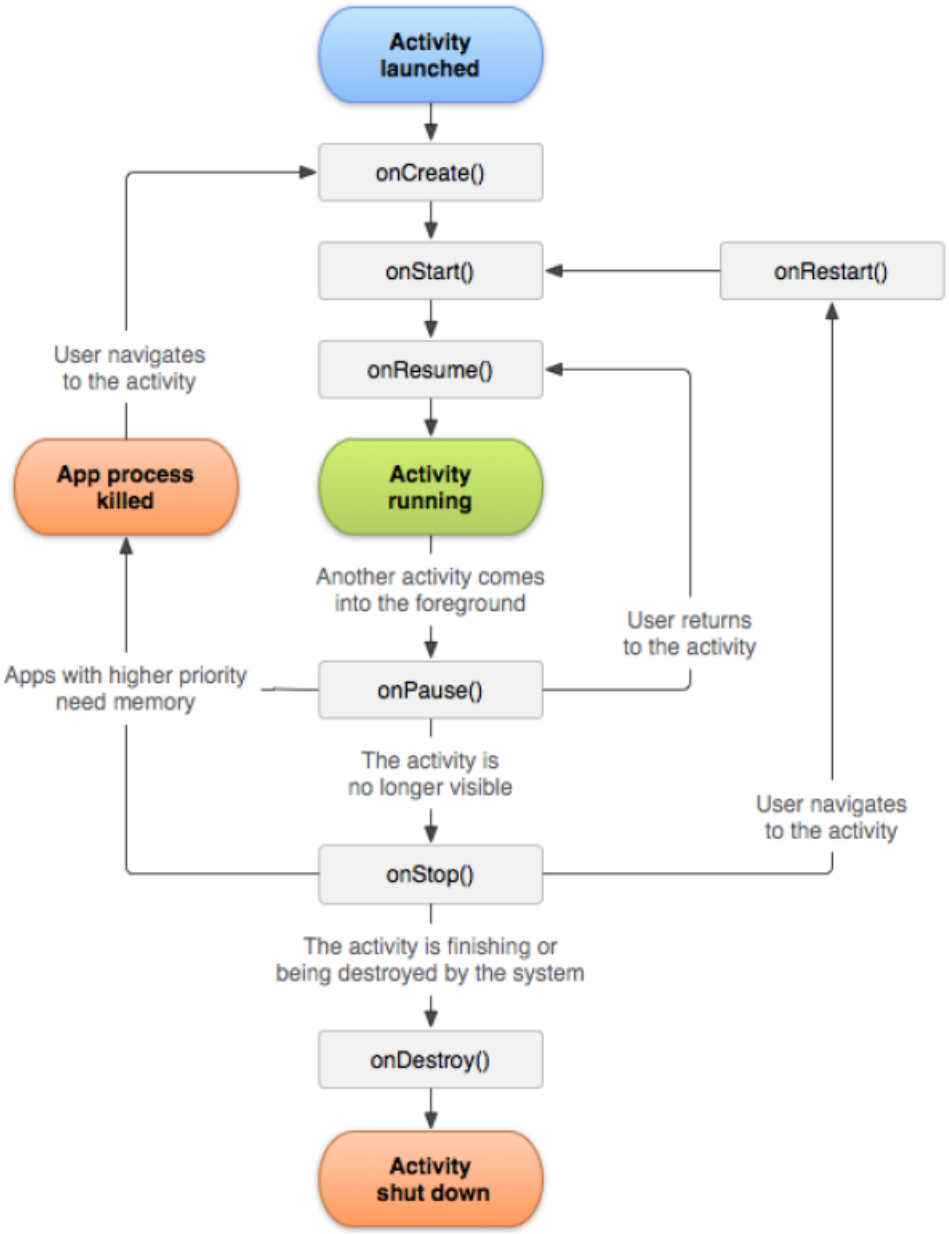
- folyamaton vagy szálon belül
- **Coroutine**: programnyelvi elem
    - Haskell, JavaScript, Perl, Python, Ruby
    - Nem használhat vermet, mert megtelne
    - Ide oda lépked, nem lép vissza, mint a subroutine
- Fiber: rendszerszintű eszköz
    - Win32 API, Symbian (lel)
- kooperatív multitasking-gal megoldható problémák esetén használják

- nem kell erőforrásokon osztozkodni
    - no OS calls
    - kisebb erőforrás igény
- csak 1 végrehajtóegységet tudnak kihasználni

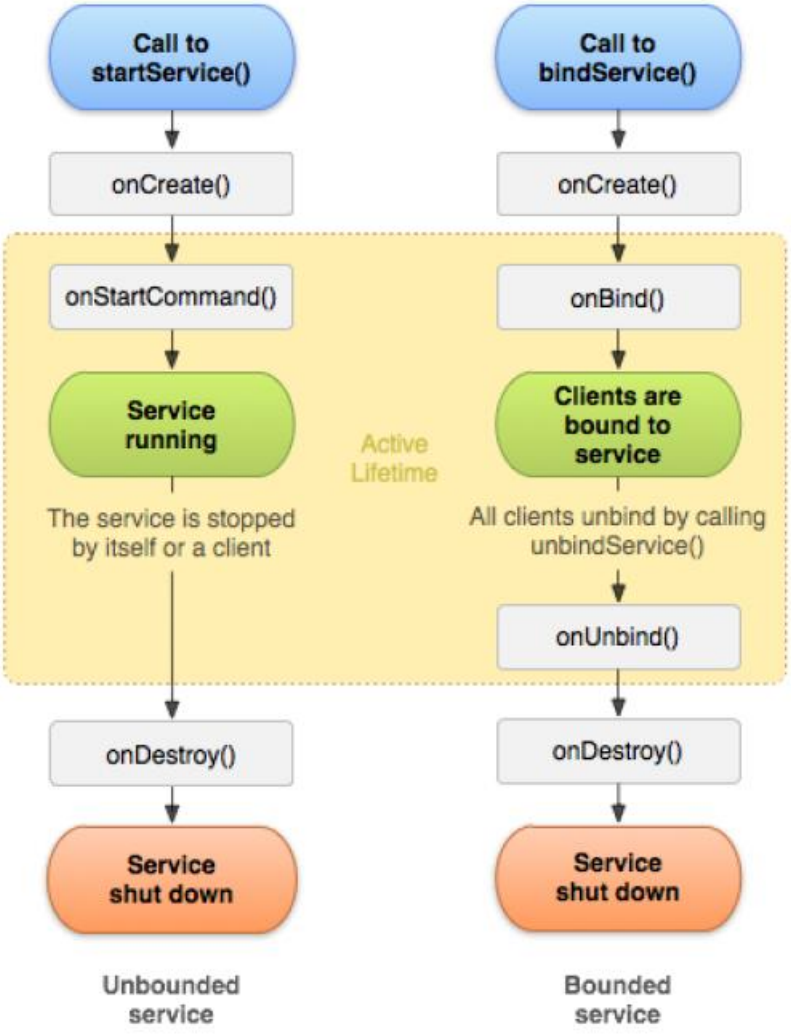
Android:

- alapja egy erősen módosított linux kernel
- ARM platformon elsődlegesen (x86-ra is le van fordítva [Intel ATOM])
- appok: UNIX folyamatban futó Dalvik VM-ben futnak
    - minden folyamatban a Dalvik VM saját példánya
- minden appnak külön Linux UID → nem tudnak egymáshoz férni
    - AndroidManifest.xml-be: futás részletei
        - szükséges jogosultságok megadása
        - minimális API level
        - használt hardverek/szoftverek
        - használt apik megadása
        - alkalmazás komponenseinek megadása
- agresszív erőforrás kezelés
    - alkalmazás bármikor terminálható
    - gyakran használtakat előre betölti a memóriába, csak nem használ CPU-t
- process prioritások
    - Aktív alkalmazás (critical)
    - Látható alkalmazás (high)
    - Elindított szolgáltatás (high)
    - Háttérfolyamat (low)
    - Üres folyamat (low)
- azonos prioritás esetén a régebbit öli meg
- Application Lifecycle Events: felül lehet definiálni őket
    - onCreate(), onLowMemory(), onTrimMemory()…

- Alkalmazás komponensek:
    - **Aktivitás**
        - Activity Stack (LIFO)
        - egy képernyő felhasználói felülettel
        - appon belül több lehet/van
        - appon belül függetlenek egymástól
        - Active/Running: képernyőn aktív
        - Paused: képernyőn inakítv, részben vmi takarja
        - Stopped: teljesen takart, leállított
        - A képernyőn lévők futnak, a többi állapota tárolva van


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

- **Szolgáltatás**
  - Háttérben futó funkció GUI nélkül:
    - pl: MP3 lejátszása
    - nem csinál saját szálat, ha CPU intenzív, akkor a fejlesztőnek a dolga csinálni hozzá
  - Started:
    - addig fut, amíg nem végzi a dolgát, túlélheti az appját
    - pl: nagy fájl letöltése
  - Bound:
    - együtt él az appal
    - interfacen keresztül érhető el az appból



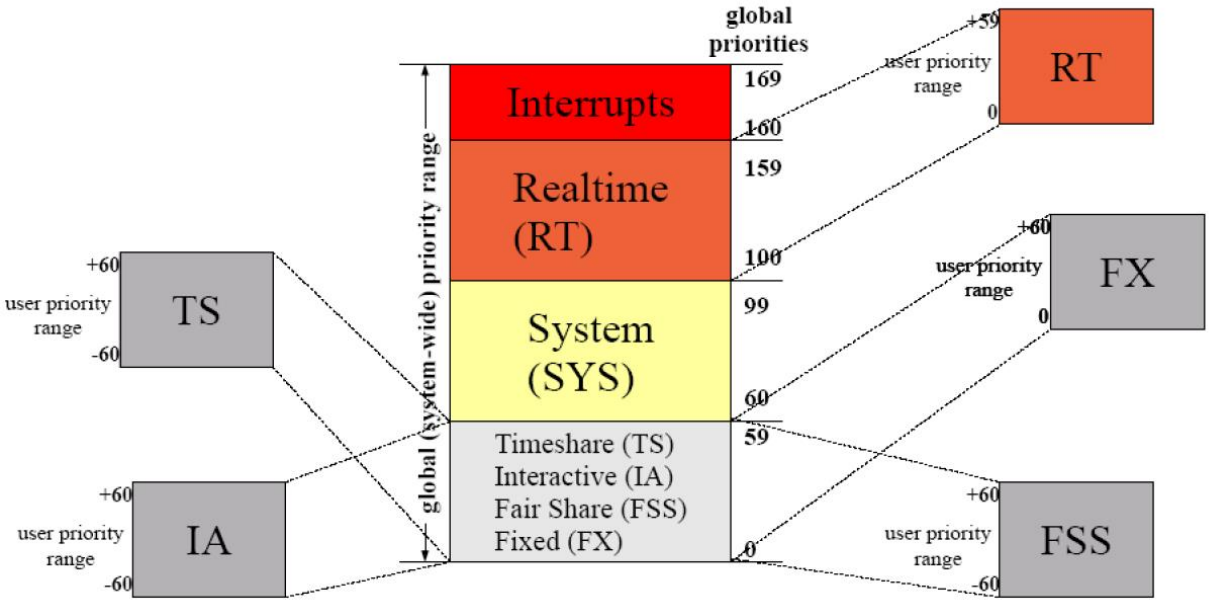
10. előadás: UNIX folyamatok ütemezése

UNIX folyamatok ütemezése

- **Folyamat adatai:**
  - PID, PPID (folyamat- és szülőfolyamat azonosító)
  - UID, GID
  - aktuális állapot: F, FK, A, …
  - ütemezési információk:
    - prioritás
    - nice érték
- prioritásos: 0 – 127 (itt 0 a legmagasabb)
- időosztásos: időszeletek vannak, h ne csak a legmagasabb prioritású fusson
- **felhasználói mód:** preemptív, időosztás, változó prioritás
- **kernel mód:** nem preemptív, nincs időosztás, rögzített prioritás → konzisztencia
- szintek: 0 a legmagasabb

    - 0 – 49: kernel
    - 50 – 127 user
    - 32 db FIFO sorba vannak osztva
- **Kernel mód:**
    - prioritás rögzített
    - prioritás meghatározása: folyamat elalvásának oka
        - diszk I/O-ra vár: 20
        - inputra vár terminálról: 28
- **User mód:**
    - mindig a legnagyobb prioritásút ütemezi be
    - ha van magasabb, mint az épp futó → átütemez
    - minden óraciklusban megnézi van-e magasabb folyamat: átütemez
    - minden 10. óraciklusban (minden 1 időszelet végén) megnézni van-e megegyező prioritású?
        - ha igen, átütemez
            - a futási jogától megfosztott a sor végére kerül
        - round robin
- **Üzemezés adatai:**
    - p_pri: aktuális prioritása → minden 100. ciklusban kiszámolja
    - p_usrpri: user mód beli prioritása (kernel módba váltásnál elmenti)
    - p_cpu: korábbi CPU használat mértéke → minden óraciklusban ++ (fair)
        - minden 100. ciklusban öregíti
    - p_nice: felhasználó által adott prioritást módosító érték
- **Felhasználói összefoglalva:**
    - Minden óraütésnél:
        - ha van magasabb → átütemez
        - p_cpu++;
    - Minden 10. óraütésnél:
        - Round Robin a user folyamatok között
    - Minden 100. óraütésnél:
        - p_cpu öregítése: p_cpu = p_cpu * KF (korrekciós faktor)
        - prioritás újraszámolás: p_pri = P_USER + p_cpu/4 + 2*p_nice
            - P_USER = 50 konstans
        - ha kell, átütemez
- **Értékelése:**
    - **pozitív:**
        - egyszerű, hatékony
        - általános célra jó
        - interaktív és batch típusú folyamatok keverékét jól kezeli
        - no éhezés
        - I/O műveletet végző folyamatok jól támogatva
    - **negatív:**
        - sok folyamat → sok számítás (rossz skálázódás)
        - nem lehet CPU-t garantálni
        - nem lehet igény szerint ütemezni
        - válaszidőre nincs garancia
        - többprocesszoros támogatás nem kidolgozott
        - kernel nem preemptív
- **Modern UNIX ütemezők:**
    - moduláris
    - több féle osztály a kül. alkalmazási igényekhez
    - többprocesszoros rendszerek, jobb erőforrás-allokáció, szálak ütemezése
- **Solaris ütemező:**
    - szál alapú
    - kernel is preemptív
    - többprocesszoros rendszerek támogatása

- többféle ütemezési osztály:
  - időosztásos (TS): ki mennyit futott/várt
  - interaktív (IA): aktív ablakhoz tartozó kiemelése
  - fix prioritásos (FX)
  - fair share (FSS): CPU erőforrások →folyamat csoportohkoz
  - real time (RT)
  - kernel szálak (SYS)



- **CFS:** Completely Fair Scheduler
  - időben rendezett piros-fekete fa segítségével
  - eddigi O(1) helyett O(logn)
  - lista helyett fa
  - törlés és beszúrás véges idejű (?)

11. előadás: UNIX folyamatok kezelése

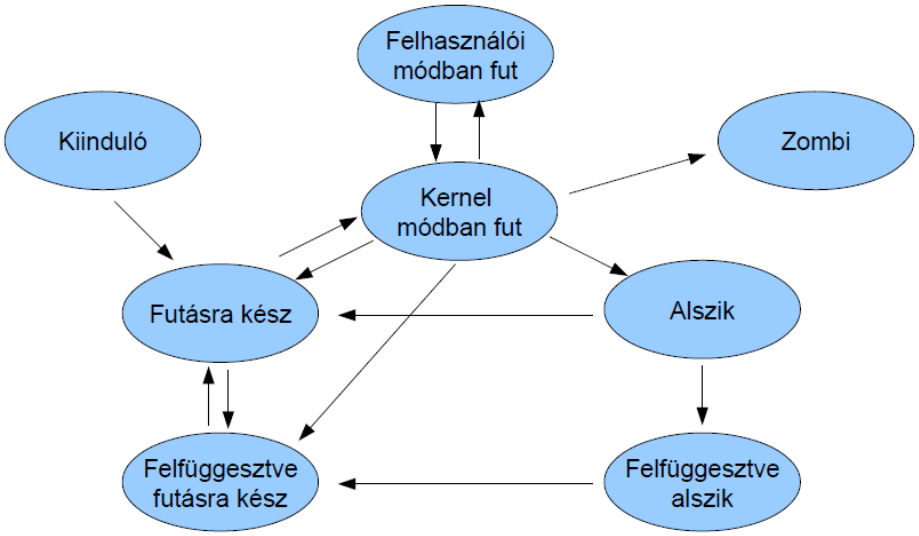
Folyamatok a UNIX-ban:

- kernel folyamatai: kjournald, kswap, init (PID = 1, minden folyamat őse)
- rendszerszolgáltatások folyamatai (**daemon process**): időkezelés, fájlrendszerek ellenőrzése, hálózati interfészek konfigurálása, stb
  - init indítja őket
- felhasználói folyamatok: shell, appok
- kernel indítja el az első folyamatot: **init**
  - indítja és leállítja a kernelszolgáltatásokat
- **rendszer futási szintje**: működési mód, aktív szolgáltatások köre
  - 0: teljes leállás
  - 1, vagy S: single user
  - 2-5: multi user, GUI-val vagy nélüle
  - 5 az alapérzelmezett: teljes GUI
  - 6: restart

Folyamatok állapotai:

- Kiinduló → Futásra kész → Kernel/User módban fut → Alszik → Felfüggesztve alszik → Felfüggesztve futásra kész → Futásra kész → Kernel/User módban fut → Zombi
- létrehozás: fork() syscall → fát épít. ős az init folyamat
- új végrehajtandó kód betöltése: exec()
- vezérlés alapvetően jelzésekkel
- **leállás (exit):**
  - zombi állapotba lép

- szülő értesül a gyerek leállásáról
- leálló folyamat gyerekeit az init adoptálja



- **Ősfolyamat** (PID 1: init, upstart, systemd…)
  - rendszer leállásáig fut
  - örökli az árvákat
  - figyel bizonyos rendszerfolyamatokra

Végrehajtási módok:

- Kernel: védett tevékenység
- Felhasználói: folyamat kódjának végrehajtása

Végrehajtási környezetek (kontextusa):

- Kernel kontextus: saját feladathoz adatok
- Folyamat kontextus: folyamatok vezérléséhez szükséges adatok, folyamat futásának adatai

- **folyamatok kezelése:**
  - ps: listázás
  - kill: leállítás
  - nice: prioritás állítás

- **folyamatok adatai:**
    - kód, adatok, vermek
    - kontextus: cpu, mmu, fpu
    - **adminisztratív adatok**:
        - **u-terület**: folyamat címtér része
            - **futása során szükségesek:**
                - hozzáférés-szabályozás adatai, rendszerhívások állapotai, nyitott fájl objektumok
        - **proc struktúra**: kernel címtér része
            - **kezeléséhez szükségesek:**
                - azonosítók, futási állapot, ütemezési adatok, memóriakezelési adatok, u-terület címe
    - **Környezeti adatok**: indításkor örökölt tulajdonságok
- **Végrehajtási mód váltása:**
    - rendszerhívások esetén (pl: fájlt akar írni)
    - meghívja a rendszerhívást (libc beli fv hívódik meg)
        - libc: SYSCALL utasítás (megszakítást generál)
            - CPU kernel módba lép
                - kernel előkészíti a végrehajtást (a SYSCALL kezelője)
                    - végrehajtódik a védett módú utasítás
                - kernel visszatér a megszakításból (iret, sysexit)
            - CPU user módba lép
        - libc visszatér
- **/proc fájlrendszer:**
    - folyamatok és kernel adatok érhetőek el fájlrendszeren keresztül
- **virtuális rendszerhívások:**
    - folyamat címterében egy kernel lap
        - biztonságos rendszerhívások vannak raja
    - nincs kontextusváltás, módváltás
    - user program nem lát különbséget

12. előadás: Taskok együttműködése

Lockolás atomi művelet kell legyen

Erőforrás:

- Minden olyan eszköz, amire a párhuzamos programnak futás közben szüksége van.
- legfontosabb: CPU
- memória és annak tartalma (adatstruktúrák)
- perifériák

Közös erőforrás:

- Egy időintervallumban több, párhuzamosan futó feladatnak lehet rá szüksége.
- erőforráson osztoznak a feladatok
- egy időben megadott számú feladat tudja helyesen használni
- felismerni és helyesen használni őket!

Kölcsönös kizárás:

- Annak biztosítása, hogy a közös erőforrást egy időben csak annyi magában szekvenciális feladat használja, amely mellett a helyes működése garantálható.
- programban kell megoldani
- többnyire a használt erőforrást lock-oljuk (elzárjuk)

Kritikus szakasz:

- A magában szekvenciális feladat azon kódrészletei, amely során a kölcsönös kizárást egy közös erőforrásra biztosítjuk.
- atomi műveletként kell megvalósítani
- atomi művelet: nem megszakítható

Atomi művelet:

- Nem megszakítható művelet, amelyet a processzor egyetlen utasításként hajt végre.
- egyprocesszoros rendszerben:
    - IT teljes tiltása
    - műveletsor végrehajtása
    - IT engedélyezése
- lockolás is ilyen kell legyen

Közös erőforrások védelme:

- Interrupt rutin, task, DMA férhet hozzájuk
- védelemre lehetőség:
    - IT tiltás/engedélyezés
    - üzemező tiltás/engedélyezés
    - Locking

Újrahívhatóság:

- A közös erőforrás problémájának egy kiterjesztett eset.
- Egy függvényen/objektumon belül is felléphet, amennyiben ezt a függvényt egyszerre többen is meghívhatják.
- Pl:
    - ugyan azt a függvényt hívjuk egy taskból és egy megszakítási rutinból is
    - preemptív esetben 2 külön taskból
- Újrahívható ha: használ-e közös erőforrást?
    - Ha igen:
        - megfelelően kezeli-e?
- Pl:
    - PC BIOS hívások nem!
    - preemptív OS SysCalljai igen.
    - API fv-ek lehet igen és nem

Hibák:

- **Versenyhelyzet**:
    - párhuzamos program futása során közös erőforrás nem megfelelő használata inkonzisztenciát okoz az erőforrásban, és a használó programban pedig hibát
- **Kiéheztetés:**
    - wating állapotban nagyon sokáig (örökké) vár a szál az erőforrásra
    - a párhuzamos rendszer hibás működése miatt
    - nem csak CPU-ra merülhet fel
        - CPU: „Futásra kész” állapot
        - más erőforrás: „Eseményre vár” állapot
- **Holtpont (deadlock):**
    - feladatok egymásra várnak a közös erőforrás hibás használata miatt
    - nincs futásra kész folyamat, nem jöhet létre belső esemény →
    - rendszer nem tud előrelépni
- **Livelock:**
    - egyszerre akarja 2 folyamat használni az erőforrást és „elállják” egymás útját
    - általában hibás deadlock feloldás eredménye

- **Prioritás inverzió:**
  - 3 statikus prioritású task
    - Task 3: magas prioritású
    - Task 2: közepes prioritású, CPU intenzív
    - Task 1: alacsony
  - **T3** vár **B-re**, miközben fut
  - utána **T3** nem fut, **T1** fut és megszerzi **A** erőforrást
  - utána **T1** nem fut, **T3** fut és megszerzi **B** erőforrást
  - **T3** futás közben elengedi **B** erőforrást és **A** erőforrást akarja használni
    - lényegében **T1**-re vár
  - **T1** ismét tud futni és használja **A**-t
  - közben **T2** futásra kész állapotba kerül és mivel magasabb prioritású, futó állapotba kerül
    - hosszú („végtelen”) ideig fut
  - **Eredmény:**
    - **T3** nem tud továbblépni, mert A-ra vár (eseményre vár, waiting)
    - **T1** nem tud továbblépni, mert CPU-ra vár, hogy befejezze a munkáját és **A** felszabadulhasson (futásra kész, ready)
    - **T2** intenzíven használja CPU-t sokáig (fut, running)
  - **megoldás:**
    - **prioritás öröklés (PI):**
      - alacsony prioritású az általa feltartott task prioritását örökli a kritikus szakaszából való kilépésig
      - csak részben oldja meg
    - **prioritás plafon (PC):**
      - az adoitt közös erőforrást használó taskok közül a legnagyobb prioritású task prioritását örökli meg
    - Sokak szerint a prioritás inverzió tervezési hiba a rendszerben

Lock feloldására várakozás passzívan (sleeplock, blocking call):

- ütemező által karbantartott várakozási sorok
- Ha az erőforrás nem lockolt:
  - a feladat megkapja az erőforrást lezárva és fut tovább
- Ha az erőforrás lockolt:
  - a feladat megy az erőforráshoz tartozó várakozási sorba
  - a futásra kész feladatok közül egy futó állapotba kerül
  - ha az erőforrás felszabadul, akkor az erőforráshoz tartozó sor elején álló feladat megkapja az erőforrást lezárva és futásra kész állapotba kerül
- kontextus váltás: overhead
- erőforrás-takarékos
- pontos időzítés nehezen emgoldható
- a processzor aludhat ha nincs feladat

Lock feloldására várakozás aktívan (spinlock):

- aktív várakozás a felszabadulására
- más feladat nem tud futni, mert a várakozó fut!
- folyamatos fogyasztás
- garantáltan rövid idejű lock kezelésére
- ütemező nem csinál ilyet, de OS kernel csinálhat

Kölcsönös kizárás módjai:

- **Lock bit:**
  - erőforráshoz tartozik egy boolean:
    - false: nem használt
    - true: használt
  - IT tiltás teszt előtt, IT engedélyezés beállítás után

- **Semaphore:**
  - **bináris:** magas szintű lock bit lényegében; egy feladat a kritikus szakaszban
  - **counter típusú:** több feladat a kritikus szakaszban
    - Belépés: P(), Wait(), Pend()
    - Kilépés: V(), Signal(), Post()
    - belépés és kilépés számmal lehetnek paraméterezve
    - ha 1-nél több erőforrásra kell, akkor vagy egyben mind megkapjuk őket, vagy a töredékeket nem foglaljuk le
      - paraméter: szükséges erőforrások száma
      - egyenként foglalva versenyhelyzetet okozna…
- **Kritikus szakasz objektum és Mutex:**
  - Bináris szemaforra hasonlítan
  - **Kritikus szakasz objektum:**
    - létre kell hozni CriticalSection objektumot
    - Használata:
      - Enter(): belépés a kritikus szakaszba
        - Blokkol ha már vannak a kritikus szakaszban (sleeplock)
      - Leave(): kilépünk a kritikus szakaszból
  - **Mutex (Mutual Exlusion):**
    - Acquire()/WaitOne()
    - Release()
  - **Multiple read single write mutex (readers-writer lock):**
    - writer addig nem lép be, amíg reader van
      - writer starvation
        - writer saját másolatot kap
- **Lockolás célja még lehet:**
  - Randevú: két vagy több feladat összehangolt végrehajtás
    - pl: termelő - fogyasztó eset
  - memórián keresztüli kommunikáció
- **Monitor:**
  - lokalizáljuk a lockolással kapcsolatos feladatokat
    - a közös erőforrást körülvevő API-val
  - a lockolás nem szétszórva történik
    - compiler valósítja meg a kizárást mutex vagy semaphore alkalmazásával
  - Hoare: azonnal az erőforrást megszerző feladat fut
    - szar
  - Mesa: a futásra kész feladatok közé kerül
    - ütemező futtatja
    - lehetséges notifyAll üzenet küldése is
    - java: synchronized
    - c#: lock

13. előadás: Taskok együttműködése

Taskok kommunikációja:

- Szemafor, kritikus szakasz objektum, mutex
  - memórián keresztül ugyebár
- **Üzenetekkel**
  - **OS valósítja meg szolgáltatásaival**
  - **Memóriához képest:**
    - nagyobb késleltetés, kisebb sávszélesség
    - nem megbízható

- **unicast, broadcast, multicast, anycast**
- **Direkt kommunikáció:**
  - Szimmetrikus, Aszimmetrikus
  - Send(P, message)
  - Recieve(Q, message) [szimm.] Recieve(id, message) [aszimm.]
    - P, Q, id folyamat azonosítók
- **Indirekt:**
  - köztes szereplőn keresztül: proxy tervezési minta
  - Send(A, message)
  - Recieve(A, message)
    - A: postaláda azonosító
- **vétel után törlődik/megmarad**
- **tulajdonosa:**
  - OS: őt használó folyamattól függetlenül létezhet
  - folyamat: csak folyamattal együtt lézhet
- **Nem blokkoló hívás:**
  - eredmények a visszatéréskor még nem jelentkeznek
  - csak a végrehajtás kezdődik el a hívásra
  - visszatérési érték kezelése csak más értesítés után lehetséges
  - **Nem blokkoló send():**
    - üzenet az elküldés után azonnal visszatér
    - vagy nincs hibakezelés, vagy callback útján értesül róla
  - **Nem blokkoló recieve():**
    - azonnal visszatér
    - ha van üzenet azzal, ha nincs, akkor végtelen ciklusban üzenetre várás busy waitinget eredményez
- **Blokkoló hívás:**
  - az eredmények a visszatérés után jelentkeznek
  - egyszerűbb a kezelése…
  - **Blokkoló send():**
    - send() nem tér vissza, amíg nem vették az üzit
    - időtúllépés esetén hibával tér vissza
  - **Blokkoló recieve():**
    - nem tér vissza, amíg nem érkezik üzenet
    - pl: TCP/UDP socket listen()
- **Implementációk:**
  - **Mailbox:**
    - indirekt
    - véges számú üzenet tárolása
    - OS szintű támogatás
  - **MessageQueue:**
    - indirekt
    - többnyire végtelen számú üzenet tárolása
    - middleware-ek (3rd party programok által): pl: MSMQ
  - Beágyazott OS-ekben e 2-t erőltetik
  - **TCP/UDB port:**
    - direkt
    - socket interfész
    - gépen belül localhoston (127.0.0.1/8)
    - alacsony szintű, számos middleware alapul rajta: pl: COBRA
      - Távoli eljárás hívás (Remote Procedure Call, RPC)
  - **Folyamok és csővezetékek:**
    - indirekt
    - UNIX pipe, Windows Named Pipe

        - **System V Shared Memory** (UNIX, Linux):
            - direkt
            - memória interfészű
            - memóriában tárolt adatstruktúrára a kölcsönös kizárást biztosítani kell
    - **Távoli eljárás hívás (RPC):**
        - Másik folyamat kód-memóriaterületén lévő fv. meghívása üzenetek felhasználásával.
        - hívó fél blokkolva vár a távoli hívás lefutására
        - meghívott függvény az őt tartalmazó folyamat szálában fut le
        - programozó számára azonos egy „lokális” fv. meghívásával
            - meghívott fv. megvalósítása is egyszerű
        - Működése:
            - rétegelt, absztrakciók.
            - kliens szál a hívás során eseményre várakozik
            - szerver szál a hívás beérkezéséig eseményre várakozik

14. előadás: Taskok együttműködése

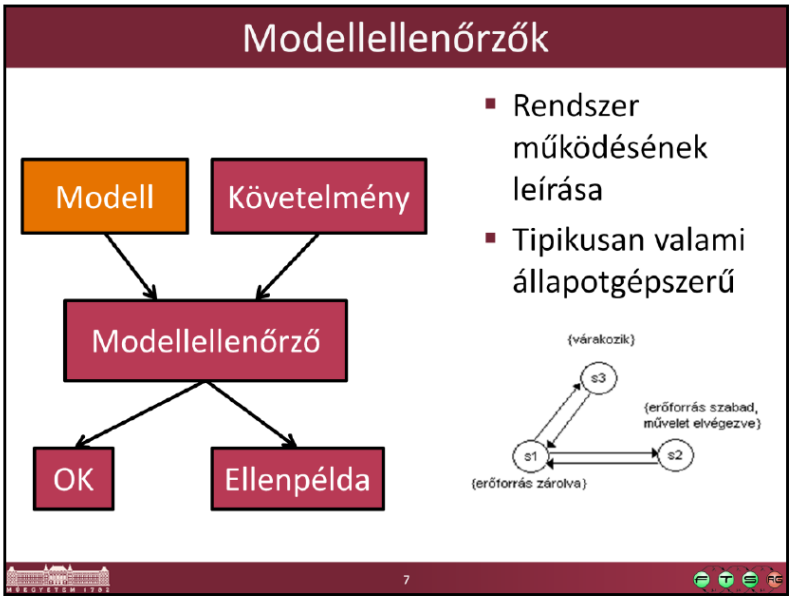
Holtpont:

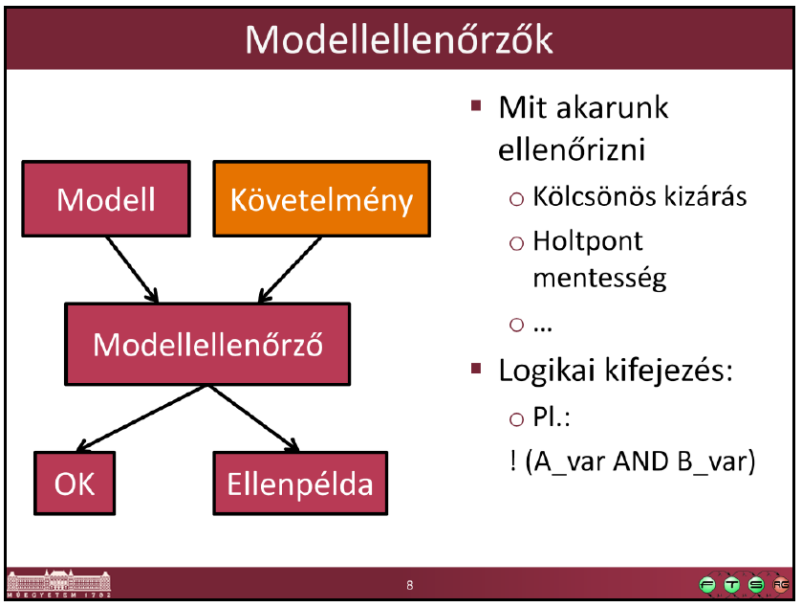
- Egy rendszer feladatainak egy H részhalmaza holtponton van, ha H halmazba tartozó valamennyi feladat olyan eseményre vár, amelyet csak egy másik, H halmazbeli feladat tudna előállítani.
- Nehéz felismerni
    - versenyhelyzet formájában jelentkezik: hol előll, hol nem
- Más feladatokat is befolyásolhat: H részfeladatok által lefoglalt erőforrásokon keresztül
- **Létrejöttéhez a feltételek:**
    - **Kölcsönös kizárás:** Vannak olyan erőforrások, amiket csak kizárólagosan lehet használni és azt többen is használnák
    - **Foglalva várakozás:** Legyen olyan feladat, amelyik lefoglalva tart erőforrásokat, miközben másikra várakozik
    - **Nincs erőszakos erőforrás elvétel a rendszerben:** Feladatok csak önszántukból szabadítják fel az erőforrásokat
    - **Körkörös várakozás:** körben az egymás által lefoglalt erőforrásokra várakoznak a feladatok
- **Erőforrásfoglalási gráf:**
    - pontok: erőforrások, vagy feladatok
    - élek: igény, vagy létező foglalás
    - irányított kör: lehet benne holtpont, de nem biztos, hogy kialakul
- **Kezelése:**
    - **Holtpont észlelése és feloldása:**
        - Észlelés (mikor fusson?):
            - amikor egy erőforrás kérelem nem elégíthető ki azonnal
            - amikor alacsony a CPU kihasználtság (sok feladat vár erőforrásra)
                - reálisabb, mert nagy futási idejűek az algók
        - Feloldás:
            - Radikális: összes holtpontban lévő feladat felszámolása
            - Kíméletes: egyes feladatok felszámolása, és eldönteni, hogy sikerült-e
        - Feladatok:
            - áldozatok kiválasztása
            - áldozatok visszaállítása

- **Holtpont megelőzése tervezési időben:**
  - Holtpont szükséges feltételeiből legalább 1 nem teljesül.
  - PL:
    - **Nincs futási idejű várakozás:** (embedded)
      - drasztikus as fuck
    - **Foglalva várakozás kizárása:**
      - minden szükséges erőforrást egyben kell lefoglalni, egyetlen syscallal
      - erőforrás-kihasználtság romlik
    - **Erőszakos erőforrás elvétel:**
      - erőforrás menthető állapottal
      - nem kell rollback feladat szinen
    - **Körkörös várakozás elkerülése:**
      - pl: teljes sorrendezéssel
      - programozónak be kell tartania
        - esetleg forráskód automatikus vizsgálata
- **Holtpont elkerülése futási időben:**
  - Igény kielégítése előtt:
    - holtpontba kerül-e a rendszer?
  - **Bankár algoritmus:**
    - **N** db feladat
    - **M** db erőforrás
    - **MAX:**
      - Egyes feladatok az egyes erőforrásokból maximálisan mennyit használnak a futásuk során
      - NxM mátrix
    - **FOGLAL:**
      - A feladatok által foglalt erőforrások száma
      - NxM mátrix
    - **SZABAD:**
      - szabad erőforrások száma
      - M elemű vektor
    - **MAXr:**
      - Az egyes erőforráskból rendelkezésre álló max. szám
      - N elemű vektor
    - **FOGLALr:**
      - Az egyes erőforrásokból a foglalt példányszám
      - N elemű vektor
    - **MÉG:**
      - Egy feladat által még maximálisan bejelenthető kérések száma:
      - MÉG = MAX – FOGLAL
      - NxM mátrix
    - **KÉR:**
      - A feladatok várokozó kérései
      - NxM mátrix
    - **Biztonság vigyálat:**
      - 1. lépés: kezdőérték beállítás
      - 2. lépés: továbblépésre esélyes folyamat keresése
        - a max igény kielégíthető a rendelkezésre álló erőforrásokkal
        - ha igen, akkor lefut, és az erőforrásai visszakerülnek a SZABAD készletbe
        - ha van még feladat, akkor 2. lépés, egyébként 3.
      - 3. lépés: kiértékelés
        - feladatok listája, amelyek holtpontra juthatnak

- **Kezelése a gyakorlatban:**
  - tervezési időben
  - észlelés és feloldás többnyire ember által
  - futási idejű algók komplexek, szükséges adatok nem állnak rendelkezésre
    - pl: bankár: maximumok.

15. előadás: Feladatok együttműködésének ellenőrzése





16. előadás: Időkezelés

Kalendáriumok: idő felosztása emberi léptékű intervallumokra

- Gergely
- UTC: Coordinated Universal Time
  - lehetőség szerint mindenhol ezt használjuk
  - folyton nő, 1 perc lehet 59 vagy 6 sec is
  - minden globális idővel foglalkozó rendszer ezt használja
- International Atomic Time

Órák: az időt mérik

- Mindig kezdeti időponttól méri az időt
- **Részei:**

    - impulzusforrás
    - számláló
    - kijelző
- **Jellemzői:**
    - stabilitás (impulzusforrás frekvencia-eltérése)
    - pontosság
    - felbontás
- **Pontatlanság okai:**
    - kezdeti számláló érték hibás
        - üzem közben nem korrigálható, sérül az idő folytonossága!
    - impulzusforrás frekvencia hibája:
        - gyártási hiba
        - fizikai hatásokra
    - frekvencia hiba akkumulálódik:
        - egyre többet késik vagy siet
        - frekvencia kompenzáció hatására nem sérül az idő folytonossága!
- **Impulzusforrások és oszcillátorok:**
    - **Real-Time Clock (RTC) és oszcillátora:**
        - időt méri, amikor a PC ki van kapcsolva
        - alacsony fogyasztású, elemes táplálású
        - pontatlan
        - lassú kapcsolat
        - képes felébreszteni az OS-t
        - BCD kódban
    - **Timer IT és az OS származtatott rendszeróra:**
        - OS induláskor inicializálja az RTC-ből
        - OS leállásakor RTC-t beállítjuk ez alapján
        - pontatlan
        - felépítés:
            - HW rész: N MHz órajel leosztása 10-20ms óraütésre, ami IT-t kér
            - SW számláló az idő mérésére
    - Hálózati interfész óra
    - Hangkártya órajele
- **Tipikus hibák számszerűen:**
    - 70-80 ppm (parts per million)
    - túl nagyok:
        - órát szinkronizálni kell a referencia időhöz
    - Megoldások:
        - Out of Band:
            - külön komm. hálózat az órák beállítására
            - GPS, DCF77 (77.5 kHz rádió), IGIR (professzionális)
        - In Band:
            - kommunikációra használt csatornán belül
            - TCP/IP felett
            - NTP
                - redundáns
                - internethez optimalizált
            - PTP
                - lokális hálózatra optimalizált
                - master-slave
                - nagy pontosságú (mikro s alatti)

17. előadás: UNIX folyamatok kommunikációja

Kommunikáció csatornái:

- **Jelzések**:
  - Cél:
    - egy folyamat(csoport) értesítése
    - szinkronizálás más folyamatokhoz
  - Típusai:
    - rendszer: kivételek (pl: hibák), riasztás, értesítések (egy gyerek leállt)
    - felhasználói: emberek (ctrl + c/z [kill]), folyamatok (tetszőleges céllal)
  - Működés:
    - létrejön (rendszerhívás, vagy esemény bekövetkezése)
    - kernel értesíti a címzettet
    - címzett egy jelzéskezelő eljárásban fogadja a jelzést
  - Jelzések kezelése
    - többféle lehetséges eljárás, bizonyos keretek közt állítható
    - saját kezelőfüggvény is megadható bizonyos jelzésekre
- **Csővezetékek** pipe():
  - Cél: folyamatok között adatátvitel
  - csak szülő – gyerek viszony
  - adatfolyam
  - egyirányú
  - limitált kapacitás
  - megvalósítás:
    - folyamat létrehoz egy csővezetéket (pipe())
    - kernel az adatstruktúrákat és olvasási/írási leírókat ad vissza
    - folyamat továbbadja a leírókat a gyerekeinek
    - leírók segítségével kommunikálnak a szálak (read(), write())
  - nincs címzés, csak rokonságban működik
  - **Elnevezett csővezetékek:**
    - ftlen folyamatok kommunikációjára
    - meglévő csővezeték elérésére másik folyamat által
    - létrehozás fájlrendszer segítségével
- **UNIX System V IPC:**
  - egységes kommunikációs keretrendszer
  - folyamatok között egységes kommunikáció
    - adatátvitel, szinkronizáció
  - kulcs: azonosító az erőforráshoz
  - közös kezelőfüggvények
  - erőforrás: kommunikáció eszközei:
    - **Szemaforok:**
      - szinkronizáció
      - P(), V() operátorok
      - adott számú szemaforoz hozzáférés
      - egyszerre több művelet egyszerre több szemaforon
      - blokkoló és nem blokkoló P() is lehetséges
    - **Üzenetsorok:**
      - adatátvitel
      - nincs címzés, üzenetszórás van
      - adott kulcsú üzenetsorhoz van hozzáférés
      - szűrés állítható be
    - **Osztott memória:**
      - egyszerű és gyors adatátvitel

        - kernel helyett közvetlen csatorna
        - fizika memória elkülönített része
        - adott kulcsú osztott memóriához van hozzáférés
        - hozzárendelhető saját virtuális címtartományhoz
        - kölcsönös kizárást kell biztosítani
- **Hálózati (socket) kommunikáció:**
    - címzéssel és protokollal támogatott adatátvitel
    - kliens szerver architektúrában
    - sokféle célra, protokoll és címzés van
    - socket: kommunikáció végpontjai
- **Távoli eljáráshívás (RPC):**
    - magas szintű kommunikáció
    - távoli eljárások meghívása (másik folyamatban vagy akár másik gépen)
    - programozói interfész + leírás

18. előadás: Memóriakezelés

Logikai cím:

- CPU generálja a folyamat futása közben
- Virtuális címnek hívjuk, ha eltér a fizikaitól
- MMU végzi a futási idejű leképezést

Fizikai cím:

- Egy adott memóriaelem címe, ahogy az a fizika memória buszon megadásra kerül a vezérlő által

Címképzés:

- **Fordítási idejű:**
    - abszolút címzés
    - a program fizika címeket használ
    - embedded
- **Betöltési idejű:**
    - áthelyezhető kód
    - a program fizika címeket használ
- **Futási idejű:**
    - a program logikai címeket használ
    - transzparens módon változhat a fizikai címterülete

Egyszerű megoldás:

- Csak betesszük a folyamatokat
    - báziscím és méret
- külső tördelés
    - használhatatlanul kicsi memóriaterületek
- belső tördelés
    - használhatatlanul kicsi memóriaterületeket processzeknek adjuk
- Allokációs stratégiák:
    - First fit
    - Next fit
    - Best fit
    - Worst fit
- szabad helyek tömörítése: Compaction, Garbage Collection
    - nagyon erőforrásigényes
        - báziscímek átállítása
        - fizikai memória másolása

Tárcsere (swapping):

- Teljes folyamat háttértárra írása
- ha nincs futó állapotban és nincs folyamatban lévő I/O művelet
- más fizikai címre is visszakerülhet
- nagyon lassú a kontextus váltás, ha a teljes folyamatot ki kell írni és visszaolvasni

Lapszervezés (paging):

- Folyamat fizikai memóriaterülete nem folytonos
- Fizikai memóriát → keretekre osztjuk (frame)
- Logikai memóriát → lapokra osztjuk (page)
- Logika cím: lapszám + eltolás
- Laptábla:
    - lapszámmal indexelhető
    - egyes lapokhoz tartozó fizikai báziscímek
- Kerettábla:
    - üres kereteket tartja nyilván
- Leképezést hardver végzi
- Logikai és fizikai címtartomány teljesen elkülönül
- Nincs külső tördelés
- Belső tördelés fél lapnyi
- **Keresés a laptáblában:**
    - nagyon sok bejegyzés ellen:
    - hierarchikus lapszervezés
    - hash-elt laptáblák
    - inverted page table
    - **TLB: Translation Look-aside Buffer**
        - találati arány 80-90%
- Kiegészítő bitek:
    - valid/invalid bit: benne van-e a lap a fizikai memóriában
    - read/read-write bit
    - referenced/used bit

Szegmensszervezés:

- logikai címtartomány szegmensekre osztva
- programozó: segment ID és segment offset
- nincs belső tördelés
- fordító és linker állítja össze a szegmenseket
- szegmensen kívül címzés: segment overflow fault

Szegmens és lapszervezés együtt:

- PC
- lapozás alapvető CPU szolgáltatás:
    - x86: 4KB (2 szintű) és 4MB (1 szintű) méretű lapok

Virtuális tárkezelés:

- Komplex memóriakezelési módszer
- Okai:
    - lokalitás: nem kell a teljes folyamat a fizikai membe
    - bizonyos kódrészek sose futnak le
    - indításhoz nem szükséges a teljes program betöltése
    - kódrészletek, erőforrások megoszthatóak
- Elvárások:
    - Nagyobb folyamatok, mint a tényleges fizikai mem
    - Több folyamat lehet a memóriában
    - Gyorsabb indítás
    - Osztozkodás közös kódon, adatokon
- Alapja a lapozás
    - részben fizikai memóriára, részben háttértárra képez le (pagefile)
    - Öfő virtuális címtér
- Bitek:
    - modified/dirty bit
    - referenced/used bit
    - valid bit

- Ha nincs a fizikai memben: valid bit jelzi
  - Laphiba kivételt generál az MMU
  - Behozni a pagefile-ból
- **Lapozási stratégiák:**
  - Igény szerinti lapozás:
    - csak laphiba esetén tölti be a lapokat
    - csak a szükségesek vannak a memóriában
  - Előretekintő lapozás:
    - megbecsli mely lapokra lesz szükség és betölti azokat
    - ehhez szabad erőforrások kellenek
- **Lapcsere stratégiák:**
  - Tele van a memória és laphiba történik.
  - **Optimális algoritmus:**
    - Jövőbe néz, és teljes információval rendelkezik a jövőben használt lapokról
  - **Legrégebbi lap (FIFO):**
    - behozott lapok FIFO-ba
    - legrégebben behozottat cseréli
    - Bélády anomália:
      - több memóriakeretet kap a folyamat
      - mégis növekszik a laphibák száma
  - **Újabb esély (Second Chance, SC):**
    - FIFO elején lévő lapot akkor cseréli, ha arra nem hivatkoztak (referenced/used bit)
    - referenced bitet MMU állítja be, ha a lapot használják
    - Ha van referenced bit, de a FIFO elején van:
      - referenced bitet törli
      - lapot a sor végére rakja
  - **Legrégebben nem használt (Least Recently Used, LRU):**
    - Megvalósítás:
      - minden laphoz egy last used timestamp
        - mikor volt utoljára használva
      - láncolt lista végére kerül a legutoljára használt
      - kétdimenzios tömb
    - azt cseréli, amelyik legrégebben volt használva
  - **Legkevésbé használt (Least Frequently Used, LFU):**
    - minden lapra számoljuk, hogy hány hivatkozás történt rá
    - az algoritmus nem felejt
    - friss lapokat dobná ki:
      - azokat befagyasztja kis időre
  - **Utóbbi időben nem használt (Not Recently Used, NRU):**
    - hivatkozott és módosított biteket használja
    - R törlődik egy idő után, M nem
      - 0. prioritás: R = 0, M = 0
      - 1. prioritás: R = 0, M = 1
      - 2. prioritás: R = 1, M = 0
      - 3. prioritás: R = 1, M = 1
    - hivatkozás többet ér, mint a módosítás!
- **Vergődés (Thrashing):**
  - Gyakori laphibák által okozott teljesítménycsökkenés.
    - laphiba kezelése során laphiba
    - laphibák felgyűlnek a háttértár várakozási sorában
  - egy folyamathoz hány memóriakeretet rendelünk?
    - kevés: sok laphiba → vergődés (magas PFF = Page Fault Frequency)
    - sok: más feladatnak nem marad memória
  - Elkerülése: lokális lapcsere stratégia:
    - folyamatok nem tudják egymástól elvenni a memória kereteket

        - csökkenti a problémát
- **Munkahalmaz (Working-set):**
    - lokalitáson alapul
    - A folyamat azon lapjainak halmaza, amelyekre egy időintervallumban a folyamat hivatkozik
    - mérete: WSS
    - Alkalmazása:
        - OS méri a WSS-t
        - Ha van szabad memória:
            - új folyamat engedhető be
        - Ha nincs szabad memória:
            - ki kell választani az áldozat folyamatot
            - felfüggeszteni és a memóriáját felhasználni
        - vergődés elkerülése
        - gyakorlatban erőforrásigényes
            - **helyette PFF alapú optimalizció**
                - folyamatonként mérhető a PFF
                - alsó határ, felső határ

19. előadás: Memóriakezelés a Windowsban

Virtuális tárkezelés:

- Lapszervezés (4KB / 2MB méretű lapok, 2/3/4 szintű)
- lapozófájl

Hatékonyság:

- Igény szerinti lapozás + clustering + prefetch
- Memória megosztás, copy-on-write
- Fájl cachelés memóriába (memory mapped file)

Biztonság:

- Minden folyamatnak külön címtartomány
- Elérés leírókon keresztül

Felhasználói folyamatok tartománya és rendszertartomány

- fele-fele
- x86: 2-2GB
- x64: 8-8TB
- Minden folyamatnak maximális virtuális tartomány van
- **Felhasználói:**
    - exe-k, dll-ek
    - alkalmazások adatstruktúrái
- **Rendszertartomány:**
    - executive, kernel és HAL
    - rendszerszintű adatstruktúrák
    - laptáblák

További dolgok:

- **Folyamatok memóriafoglalása:**
    - **Reserve:** virtuális címtartomány lefoglalása
        - foglalási igény jelzése
    - **Commit:** virtuális memória lefoglalása
        - lefoglalt, commited memóriának a helyet fenn kell tartani
- **PAE:**
    - folyamatok továbbra is 4GB-os címteret látnak
    - de 64GB-os címtéren tudja őket elhelyezni az OS
- **Working Set:**
    - A folyamathoz tartozó fizikai memóriában lévő lapok

- **Working Set limit:**
  - Ennyi fizikai memóriát birtokolhat egyszerre
  - ha eléri, lapcsere
  - ha a szabad memória csökken: trimming
- **Lapozófájl:**
  - csak módosított adat, kód nem
  - akkor is kerülhet bele, ha van szabad mem. (többi folyamatnak fenntartja)
  - meghajtónként 1
  - 1-1.5x fizikai memória mérete
- **Memóriahasználat megfigyelése:**
  - Working Set: megosztott lapok is
  - Working set (private): megosztott lapok nélkül
  - Commit size:
    - privát, lefoglalt virtuális memória
    - ez bekerülhet a lapozófájlba
- **Optimalizációk:**
  - **Windows 8:**
    - Memory combining: azonosságok keresése
    - Szolgáltatások csökkentése: start on demand
  - **XP: Prefetch**
    - program első tíz másodpercének hozzáféréseit jegyzi meg és tölti be induláskor a hozzá tartozó lapokat
  - **Vista: Superfetch**
    - 8 prioritás a memórialapokhoz
    - lapok használatának követése

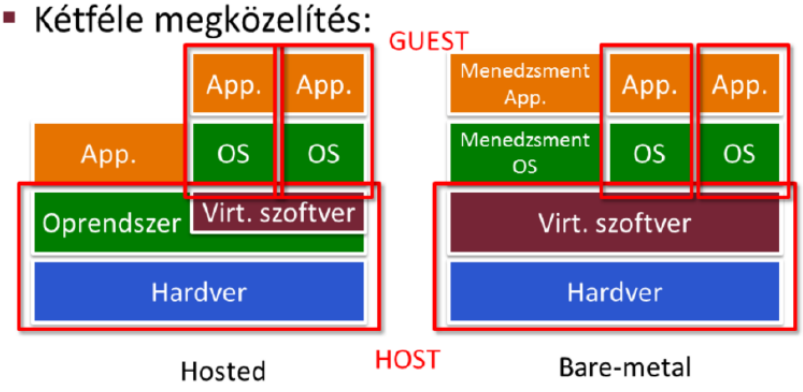
20. előadás: Virtualizáció

- **Definíció:**
  - Az erőforrás tényleges fizikai tulajdonságainak elrejtése a felhasználója elől.
  - Lényege az absztrakció, azaz, hogy elfedi az alatta levő réged valamilyen jellemzőjét.

Platform virtualizáció:

- Teljes számítógép virtualizálása, egy gépen több OS futtatása.
- **Elemek:**
  - Gazdagép (host) = fizikai gép
  - Vendéggép (guest) = virtuális gép
  - Virtual Machine Monitor (VMM) = a virtuális gépeket kezelő program
- **Mire jó:**
  - tesztrendszer kiépítése
  - HW konszolidáció
  - régi rendszerek futtatása
  - on-demand architektúra
  - rendelkezésre állás

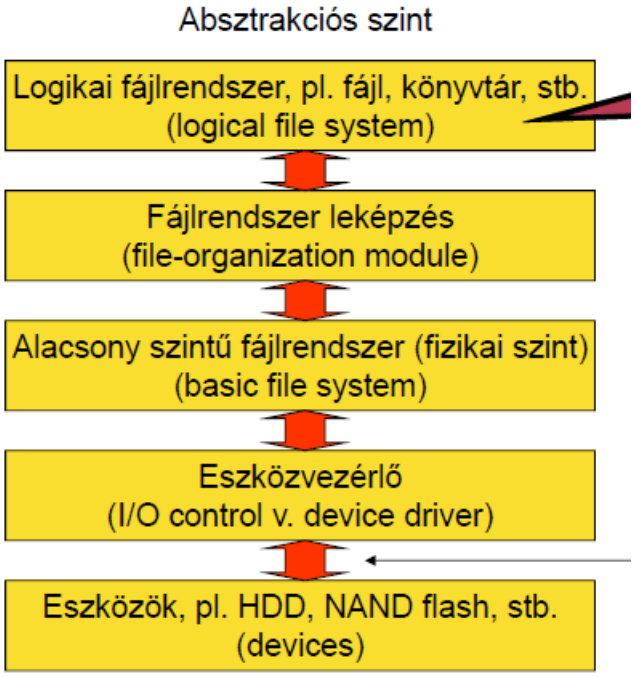
- Kétféle megközelítés:



- **Architektúrái:**
  - **Bare-metal:** a VMM kezeli az alapvető HW erőforrásokat
  - **Hosted:** az OS kezeli az alapvető HW erőforrásokat
- **Követelmények:**
  - **Azonosság:** a virtuális gépen futtatok programok ugyan azt az eredményt adják
  - **Biztonság:** a VMM kezeli az összes hardver erőforrást
  - **Hatékonyság:** a vendég gép utasításainak nagy része beavatkozás nélkül fut
- **Privilegizált utasítások:** VMM által felügyelt utasítások, hogy ne tegyenek kárt (pl: kikapcsolás)
- **CPU virtualizáció:**
  - **Tiszta emuláció:**
    - emulátorban teljes HW állapot eltárolása
    - VMM megvizsgálja az utasítás hatását és végrehajtja
    - emuláció != virtualizáció
    - Előny: más CPU is emulálható
    - Hátrány: kurva lassú
  - **Trap and Emulate:**
    - a nem privilegizált utasítások közvetlenül a CPU-n
    - privilegizált utasítások hardveres kivételt generálnak (**Trap)**
      - ekkor a VMM veszi át a végrehajtást
    - virtuális gép alacsonyabb védelmi szinten fut
  - **Binary Translation (szoftveres):**
    - utasítások nagy része közvetlenül fut
    - privilegizált utasítások átírása futás közben (eltárolja)
    - nem igényli a guest OS forráskódját
    - guest OS nem tudja, hogy virtualizált
  - **Paravirtualizáció:**
    - módosítják a vendég OS forrását:
      - problémás utasítások cseréje
    - VMM-et hívja közvetlen: **Hypercall**
  - **Hardveres virtualizáció:**
    - hardveres támogatása a virtualizációnak
    - Virtual Machine Control Structure
    - VMLAUNCH
    - VMCALL
- **Memória virtualizáció:**
  - **Szoftveres:**
    - kétszeres címfordítás kell, de a HW-ek csak 1-et támogatnak
    - ehelyett: **árnyék laptáblák**
      - vendég virtuális címeket közvetlen gazda fizika címekre képez
      - TLB ennek az elemeit cacheli
    - VMM-nek gondoskodnia kell róla, hogy az árnyék laptáblák szinkronban legyenek a vendég laptáblákkal
  - **Paravirtualizáció:**
    - árnyéklaptáblák
    - vendég OS forrását módosítják:
      - ha a vendég módosítja a laptábláit, akkor értesíti róla a VMM-et
  - **Hardveres:**
    - HW támogatás az újabb CPU-kban
    - beágyazott laptábla
      - vendég fizikai → gazda fizikai leképezés eltárolása
    - TLB bejegyzések azonosítóval ellátása
- **I/O virtualizáció:**
  - **Szoftveres:**
    - egy létező, gyakori HW-t emulál a VMM
      - normál eszközmeghajtóval használja a guest OS

        - lassú, mert minden úgy kell csinálna, mintha tényleg a lassú buszon lenne az eszköz
    - **Paravirtualizáció:**
        - speciális driver csomag telepítése a vendég gépen
        - fiktív eszközzel kommunikál a vendég OS
    - **Hardveres:**
        - I/O eszközök megosztása a virtuális gépek között
            - vagy közvetlen hozzárendelése 1 géphez
        - PCI szabványnak is létezik ilyen kiegészítése
- **Cloud Computing:**
    - **IaaS:** Infrastructure as a Service
        - virtuális gépet kapunk
        - pl: Amazon EC2
    - **PaaS:** Platform as a Service
        - futtatókörnyezetet kapunk
            - Java VM, .NET, adatbázis
        - pl: MS Azure, Google AppEngine
    - **SaaS:** Software as a Service
        - szoftver szolgáltatást kapunk
        - pl: Google Docs, SalesForce CRM

21. előadás: A permanens tár kezelése



- blokk alapú szervezés
- **fájl: adattárolás logikai egysége**
    - név, névvel lehet rá hivatkozni
    - tetszőleges méret (fájlrendszer függvényében)
    - OS képzi le fizikai egységekre
        - ehhez többszintű réteges rendszer
        - legalján a HW
- **HDD:**
    - cilinder, sáv, szektor
    - Logical Block Addressing (LBA)
- **NAND Flash:**
    - véges számú írás
    - írás/törlés lassabb is
- **Eszköz csatlakozás:**
    - Direkt: SATA, IDE, SCSI, SAS
    - Indirekt:
        - USB: alagutat képez, de SCSI parancsok
        - RAID
    - **Hálózati tároló eszközök (Storage-Area Networks, SAN)**
        - hálózati alagút a hoszt és a tároló eszköz között
        - speciális portok: fibre chanel
        - Ethernet és/vagy TCP/IP alapú
- **RAID:**
    - merevlemez lassú, nem megbízható és olcsó
    - egyszerre többet használunk
    - RAID 0-1 SW implementáció és kevés (2 db) diszkkel
    - RAID 2-4 ritkán használva
    - RAID 5-6 HW implementáció és sok (>4 db) diszkkel

- **RAID level 0: (Stripped Disks)**
  - **A file részei N diszkre kerülnek.**
  - Több diszk párhuzamos használata.
  - Tárolókapacitás összeadódik
  - Sebesség N-szeres közeli
  - Hozzáférési idő 1 diszkké
  - Bármely diszk hibája adatvesztést okoz
- **RAID level 1: (Mirroring)**
  - **A file minden része minden (N) diszkre kerül.**
  - Több diszk redundáns használata.
  - Tárolókapacitás 1 diszkké
  - Sebesség lassabb, mint 1 diszkké (de lehet N-szeres is)
  - Hozzáférési idő nagyobb, mint 1 diszkké
- **RAID level 5:**
  - **Adat és paritás elosztása N+1 diszkre.**
  - Több diszk redundáns és párhuzamos használata.
  - Sebesség lehet közel N-szeres (HW támogatás)
  - Kapacitás N-szeres
  - 1 diszk meghibásodása esetén az adat elérhető
  - Adat nem feltétlenül állítható helyre
    - 2. meghibásodás jelentkezése
- **RAID level 6:**
  - **Adat és paritás elosztása N+2 diszkre.**
  - Több diszk redundáns és párhuzamos használata
  - Sebesség lehet közel N-szeres (HW támogatás)
  - Kapacitás N-szeres
  - 2 diszk meghibásodása esetén elérhető az adat
  - 1 hiba esetén azonnal javítani!
- **Hamis biztonságérzet:**
  - tápegység hibájától, sw hibától nem véd
  - nem pótolja a biztonsági másolatokat
  - HW vezérlők drágák
  - SW csak RAID 0 és 1-nél
- RAID 1, 5, 6 megvéd a tipikus HDD hibák ellen
- RAID 0, 5, 6 gyorsítja a hozzáférést

- **Hálózati tároló eszközök (SAN):**
  - hálózati tunnel a hoszt és a tároló között
  - SCSI parancsok
  - tároló eszköz virtualizációja:
    - mintha lokálisan lenne csatlakoztatva
    - általában csak 1 géphez
  - **Megoldások:**
    - Speciális protokoll: Fibre Channel → drága
    - Ethernet és/vagy TCP/IP alapú
  - **Elnevezések:**
    - Target: hálózati tároló eszköz
    - Initiator: a kliens aki használja a tároló eszközöket
- **Alacsony szintű fájlrendszer:**
  - Fizikai diszk blokkok írását végzi.
  - Információk cachelése:
    - **buffer cache:**
      - külön cache a pagefilenak és a blokkok cachelésének
      - kétszeres cachelés
    - **unified buffer cache:**
      - cache csak blokk szinten

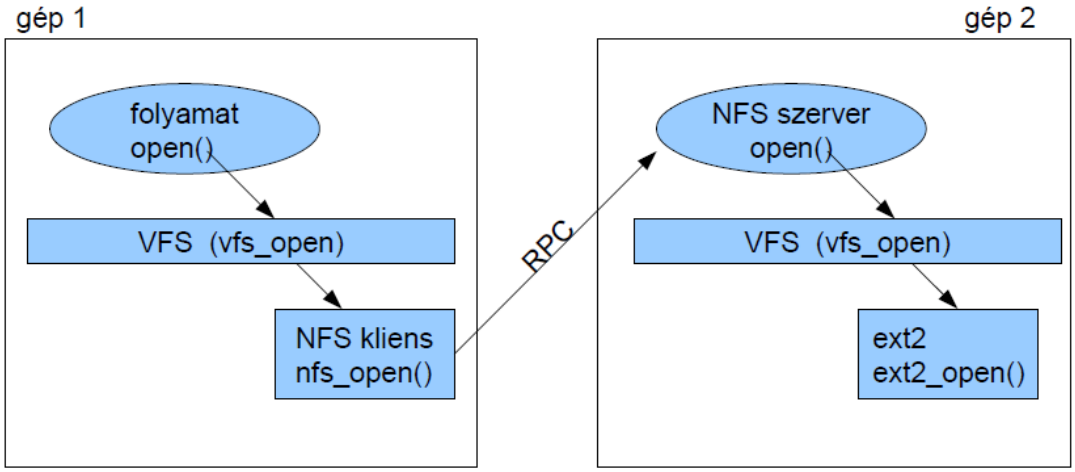
        - **unified virtual memory:**
            - lapozás és a diszk cache összevonása
            - file virtuális memóriára van leképezve
            - Linux, Windows
- **Fájlrendszer leképezés:**
    - Logika blokkok leképezése fizikai blokkokra.
    - **Folytonos allokáció:**
        - A fájl folytonos fizikai blokk sorozatot foglal el.
        - egyszerű és gyors hozzáférés
        - új fájloknak szabad hely megtalálása nehéz
        - külső tördelés
            - teljes másolás üres diszkre és vissza (offline)
            - futási idejű töredezettség mentesítés
        - szekvenciális és indexelt elérés is
    - **Láncolt listás allokáció:**
        - Könyvtárak leírói tartalmazzák az első és utolsó blokk azonosítóját.
        - minden blokk tartalmazza a következő blokk azonosítóját
        - blokkok tetszőleges helyen a diszken
        - nincs töredezettség
        - szekvenciális elérésre alkalmas
        - fájlba indexelni nehéz (n. blokk direkt elérése)
        - sérülékeny
        - sok fejmozgás
        - pl: FAT
        - töredezettség mentesítés: fejmozgás minimalizálása
    - **Indexelt tárolás:**
        - Index blokkok használata:
            - Egyes blokkokat a fájlokhoz tartozó indexek tárolására allokálunk.
        - szekvenciális és indexelt elérés is
        - sérülékeny
        - sok fejmozgás
- **Üres helyek menedzsmentje:**
    - **Bit vektor:**
    - **Láncolt lista:**
- **Logikai fájlrendszer:**
    - OS specifikus.
    - OS specifikus API a tetején
    - Metaadatok tárolása:
        - fájl mutató objektum
        - adat, név, típus, tulajdonságok
        - file locking
    - **Fájl:**
        - Permanens táron az adattárolás logikai egysége.
        - Tulajdonságok:
            - Név: elnevezési konvenciók
            - Típus: kezelése módja
            - tulajdonosok, jogosultságok
            - hozzáférési időpontok
    - **Könyvtárak:**
        - információ hierarchikus tárolása
        - **Aciklikus irányított gráf:**
            - a fájl több könyvtárban található meg
                - de csak 1 példányban létezik
            - pl: UNIX, Linux
        - **Általános gráf struktúra:**

                - hogyan lehetne keresni benne?
                - OS-ekben nem.
                - WEB a kis linkjeivel ilyesmi
            - **Fa struktúra:**
                - Windows
    - **Kötetek:**
        - Logikai fájlrendszerben a legmagasabb egység.
        - fizikai partíció
        - pl: Windows C:
- **Adatszerkezetek eszközökön:**
    - **Boot szektor:**
        - ezt tölti be a BIOS/EFI
        - ez alapján tölt be az OS
    - **Partíciós tábla:**
        - partíció specifikus adatok
        - méretei, használt/szabad hely, azokra referenciák
    - **Fájlrendszer specifikus információ:**
        - könyvtárstruktúra leírói
        - fájlok leírói (File Control Block)
- **Hirtelen leállásnál a konzisztencia megőrzése a legfontosabb.**
- **Mentésből helyreállítást tesztelni kell.**
- **Fájlrendszerek:**
    - **FAT (File Allocation Table):**
        - FAT16
        - FAT32
            - 2TB partícióméret
            - 4GB – 1B fájlméret
        - nincs külön kis- és nagybetű
    - **NTFS (New Technology File System):**
        - tranzakció alapú
        - nincs külön kis- és nagybetű
        - kell töredezettség mentesíteni
    - **EXT2:**
        - ritkán kell töredezettség mentesíteni, de akkor lassú
        - minden nagy
    - **EXT3:**
        - tranzakciókezelés
        - Htree indexelés: több könyvtár
        - egyszerű oda-vissza konverzió EXT2-vel
    - **EXT4:**
        - nagyobb tárak, stb
        - újabb Linux disztrók ezt teszik fel
- **Network-Attached Storage (NAS):**
    - Fájlrendszer szintű hálózati megosztás.
    - NFS (Network File System)
    - hálózaton fájlrendszer utasítások
    - több user párhuzamosan

22. előadás: UNIX fájlrendszerek alapismeretei

- s5fs: 80-as évek
- BSD Fast File System (FFS) (ext2 alapja)
    - megnövelt teljesítmény
    - új szolgáltatások
- Virtuális fájrendszerek
    - moduláris, OOP
- Elosztott fájlrendszerek
    - NFS
- Modern fájlrendszerek
    - ext2, ext3, ext4
    - xfs, ReiserFS, Solaris tfs
- Alapvető parancsok: ls, cp, mv, rm, cd, pwd, rmdir
- Menedzsment parancsok: mount, umount
- **Fájl- és könyvtár-attribútumok:**
    - típus
    - linkek (hard, soft)
    - eszköz, inode, méret
    - időbélyegek
    - **hozzáférési jogosultságok:**
        - **POSIX jogosultságok**, pl: 740
        - 3x3 bit: tulajdonos, csoport, mások
        - értékek:
            - 4: olvasás
            - 2: írás
            - 1: futtatás
        - beállítás: chmod parancs
        - ACL (Access Control List)
            - usereknek és groupoknak külön jogosultságok
- **Programozói interfész:**
    - open(), read(), write()
    - zárolás:
        - kötelező: lockf()
        - ajánlott: flock()
    - lezárás: close()
    - opendir(), readdir(), closedir()
- **Tárolás megvalósítása:**
    - **Szuperblokk:**
        - fájlrendszer metaadatok
        - fájlrendszer típusa, mérete
        - szabad blokkok jegyzéke
        - inode lista információk
        - zárolási infók, módosítás jelzőbit
        - másolatok helye
    - **inode lista:**
        - fájl metaadatok
        - hitelesítési infók (UID, GID)
        - típus
        - hozzáférési jogok
        - időbélyegek, méret
        - adatblokkok:
            - 10-15db indirekt blokkcím
            - 1x, 2x és 3x indirekt blokkcímek
    - **tárolt adatok**

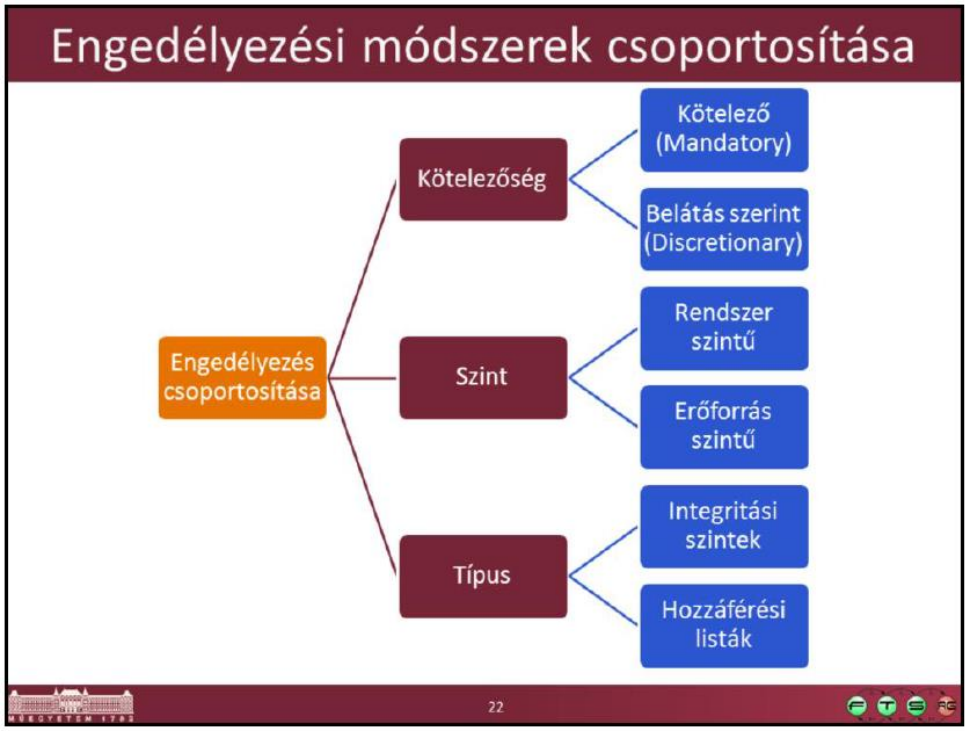
- **inode lehet a memóriában is:**
  - \+ infók:
    - státusz: zárol/módosított
    - háttértár eszköz azonosítója
    - hivatkozás számláló
- **Virtuális fájlrendszer:**
  - implementáció-független fájlrendszer absztrakció
  - többféle FS egyidejű, egységes támogatása
  - egységes programozói interfész
  - modulárisan bővíthető
  - speciális FS-ek támogatása
  - Absztrakció:
    - inode → vnode
    - fs → vfs
  - **vnode:**
    - közös adatok
    - **v_data:** FS-től függő adatok (inode)
    - **v_op:** FS metódusainak listája
    - **FS független függvények:** vop_open, vop_read
  - **vfs:**
    - közös adatok
    - **vfs_data:** FS-től függő adatok (inode)
    - **vfs_op:** FS metódusainak listája
    - **FS független függvények:** vfs_mount, vfs_umount



23. előadás: Hitelesítés és engedélyezés

- **Mikor foglalkozunk vele:**
  - tervezés
  - implementáció
  - üzemeltetés
  - Minél korábban pls
- **Miből áll a biztonság:**
  - CIA: Confidentiality, Integrity, Availability
  - **Bizalmasság:**
    - titoktartás 3. féllel szemben
  - **Sértetlenség:**
    - az legyen a feladó mezőben, aki a feladó
    - az legyen tartalma, ami eredetileg volt
  - **Rendelkezésre állás:**
    - főleg biztonságkritikus rendszerekben
- **Eszközök:**

    - kriptográfia
    - behatolás elleni védelem
    - redundancia
    - hitelesítés, engedélyezés
- **Hitelesítés (Authentication):**
    - ki vagyok? az vagyok-e, akinek mondom magam?
    - Ez alapján dől el:
        - amit tud (jelszó):
            - gépen belül csak ez
        - amije van (kulcs)
        - ami ő (ujjlenyomat)
    - Három szinten:
        - ember és gép között
        - gépen belül, alkalmazások között
        - gép és gép között
    - protokollok
- **Engedélyezés (Authorization):**
    - mihez férhetek hozzá? mit csinálhatok vele?
- **Hitelesítés Linux alatt:**
    - **User:**
        - egy objektum
        - UID:
            - root: 0
            - userek: 500-tól, vagy 1000-től kezdődően
        - name, password
        - shell, home directory, comment
        - expiry date
        - egy felhasználó több csoportba is tartozhat
        - egy csoportba több felhasználó lehet
    - **Group:**
        - GID, name
        - password (ritkán)
    - Folyamat identitása menet közben változhat:
        - Real UID vs. Effective UID
        - su:
            - váltás adott felhasználóra
            - adott user jelszavát kéri
        - sudo:
            - parancs végrehajtása más nevében
            - saját jelszó + jog kell
    - Pluggable Authentication Modules (PAM):
        - hitelesítési keretrendszer
    - Kerberos:
        - hálózati hitelesítés
- **Engedélyezés:**
    - **szereplők:** műveleteket kezdeményeznek
    - **műveleteknek:** kontextusa van
    - **jogosultsági döntő komponens:** kiértékeli a művelet és engedélyezi vagy megtiltja
    - **jogosultsági végrehajtó komponens:** döntő döntését érvényre hozza
    - sok szereplő, sok objektum → kezelhetetlen adathalmaz



- **Kötelezőség:**
  - **Kötelező:**
    - jogosultságok osztása központilag
    - userek nem módosíthatják
  - **Belátás szerinti:**
    - megfelelő jogú user továbboszthatja a jogokat
- **Típus:**
  - **Integritási szintek:**
    - címkék definiálása a szereplőkre és objektumokra
      - ezek alapján kiértékelés
    - No read up: nem olvashatok magasabb szintűt
    - No read down, No write up, No write down
  - **Hozzáférési listák:**
    - objektumhoz → (szereplők, engedélyek) listáját rendeli
    - szereplőnek az engedélye egy maszk
    - engedély ←→ objektum: több a többhöz hozzárendelés
- **Szerep alapú hozzáférés-vezérlés (RBAC):**
  - szereplőkhöz szerepeket rendelünk → kevesebb hozzárendelés
- objektumok hierarchiába szervezése
- engedély öröklés
- **user csoportok: valójában RBAC megvalósítás**
- **Engedélyezés Linux alatt:**
  - POSIX jogosultságok:
    - read, write, execute
    - 3x3 bit
    - sticky bit: csak a tulaj törölheti a fájlt
  - szereplő: user
  - szereplő hierarchia: group
  - groupba nem lehet group
  - speciális kiváltságok root nevében futnak
    - 1024 alatti UDP/TCP porton hallgathatnak
    - perifériákhoz hozzáférhetnek

24. előadás: Biztonsági alrendszer a Windowsban

- **Principal:** biztonsági alrendszer által kezelt entitások összefoglaló neve
- **SID:** user/PC azonosítója
    - **csoportokkal**: <Gép SID> - <R(elative) ID>
    - Everyone: S-1-1-0
    - Administrator: S-1-5-<domain>-500
- **Hitelesítés:**
    - belépés: ctrl + alt + del
    - jelszavak Hash-e a registryben
    - Win 8:
        - Microsoft account
        - felhő, szinkronizálás
        - picture password
    - belépéskor userhez rendelődik egy hozzáférési token
        - ezt ellenőrzi a rendszer
        - indított folyamatok ezzel a tokennel vannak
            - ideiglenesen kaphatnak más tokent is
            - pl: fájlszerver végrehajtószála
- **Engedélyezés:**
    - csoportnak adunk jogot
    - **Rendszer szintű jogosultságok:**
        - pl gép kikapcsolása
    - **Discretionary Acces Control:**
        - **belátás szerinti, erőforrás szintű, hozzáférési lista**
        - **SecurableObject:** Windows objektum
        - **SecurityDescriptor:** írja le, hogy kik hogyan férhetnek az objektumhoz
            - összefogja a többi elemet:
            - owner
            - access list
            - naplózás
            - kikre vonatkozik, típus, öröklődés flag (pl könyvtárnál)
    - **Mandatory Integrity Control:**
        - **kötelező, erőforrás szintű, címkézés**
        - DAC nem véd az általunk indított programoktól
        - folyamataink megkülönböztetése (pl: böngésző)
        - címkék: System, High, Medium, Low
        - No write up
- **Auditálás:**
    - Naplózás.
    - eseménynapló készül a rendszer és alkalmazás üzenetekről
- Run as… parancs
- letöltött fájlok blokkolása