Operációs Rendszerek

MindMap Kidolgozás

0.2.2

2010-06-03

Előszó

A kidolgozás a MIT portálról letölthető MindMap alapján készült, a váza egy-az-egyben tartalmazza azt. A segédanyag egy "rövid", gyors kidolgozás, nem helyettesíti sem az előadásokat, sem a diák átolvasását, sem a napi rendszeres étkezést! **Komolyan nem!** A doksi ~2-3 óra alatt készült, előfordulhatnak benne hibák, annak ellenére, hogy a hivatalos diákból dolgoztunk.
Mérete miatt ajánlott ZIP-ben olvasni (mert úgy sokkal kisebb)!

A készítők:

A készítők Ad-Hoc módon szerveződtek egy vizsga előtti éjjelen, kérjük hogy olvasás előtt - minden alkalommal - egy perc néma csönddel emlékezz a készítők fáradhatatlan önfeláldozására!

- Sir **Maros Viktor** Fizika Négyes - Ötletgazda, Access Point
- **Horváth Gergely J.** Ottó - Vizsgafanatikus, OpenOffice technikus
- **Tóth Gábor** - Lelkes aviátor
- **Gubek Andrea** - Az egyetlen nő - A NŐ!
- **Koczka Tamás** Hosszú Még Az Este - Különben is, mindenhol ott van

(A sok hülyeséget Ottó írta, éjfél és 2:55 között... Csaki ötletei alapján... Nemis!)

Jelmagyarázat

- ***Diában szerepelt, itt nem.***
- ***Nincs kidolgozva, nem tudom stb.***
- Több törp törp, mint egy!

TODO

- Kiegészíteni, átolvasni, finomítani :)

ChangeLog

- 0.2.2 – Normális színezéseket és a felesleges szóközöket, sortöréseket
- 0.2.1 – Minden kérdés-válasz párosítás kész
- 0.2.0 – Helyreraktam a fejezeteket és a szintezést minden maradék fejezetben
- 0.1.1 – OpenOffice félig formázott változat, eltűntetve a sok szar RTF formázást
- 0.0.1 – Importálás RTF-be, onnan meg tűrhető OpenOffice-ba
- 0.0.0 – Google Docs

Bevezető

Operációs rendszer definíciója:

Az operációs rendszer egy interfész a hardware és a felhasználó közt, ami a feladatok koordinációjával és menedzsmenjével, illetve a számítógép erőforrásainak megosztásával foglalkozik. Kiszolgálóként viselkedik az alkalmazások számára, amik a gépen futnak.
(An operating system is an interface between hardware and user which is responsible for the management and coordination of activities and the sharing of the resources of a computer, that acts as a host for computing applications run on the machine.)

OS fejlődése

- **Korai számítógépek:**
  Korai számítógépek huzalozott programmal rendelkeztek. Egy feladatot tudtak végrehajtani egy időben, ezek váltása rendkívül időigényes volt. Az erőforrásokkal történő optimálisabb gazdálkodás emberi erővel történik.
- **Korai Batch rendszerek:**
  Programírás -> Futtatás -> Eredmények. Teljesen on-line, szekvenciális periféria működés. Hasonló erőforrás igényű munkák csoportosítása -> sok különálló, ismétlődő lépésre nincs szükség. On-line perifériás műveletekről áttérés off-line műveletekre, I/O processzor, nő a komplexitás. Egyszerű monitor (resident monitor) a munkák ütemezésére: Egyiknek vége, automatikusan kezdődik a másik.
- **Átlapolt feldolgozás:**
  Az I/O processzorok elfedik a periféria specialitásait. Szabványos, absztrakt interfész - Logikai I/O perifériák megjelenése. Pufferelés az I/O perifériák és a központi egység közötti kapcsolat megoldására, így az Input –> CPU –> Output átlapolódik. A hibakeresés még mindig nehézkes.
- **Spooling (Simultaneous peripherial operation on-line):**
  Nagyobb kapacitású RAM háttértárolók megjelenése -> Több feladat egy időben (Egy fő program plusz az I/O-val kapcsolatos feladatok) -> A feladatok jobban átlapolódhatnak. Eredmény: elmozdulás a multiprogramozás felé.
- **Multiprogramozás:**
  Még nagyobb kapacitású, gyors RAM jellegű háttértárak -> A feladatok nem csak beérkezésük sorrendjében dolgozhatók fel (egy időben több befér a memóriába) -> Optimalizáció lehetősége/igénye futási időben -> Job pool (lehetséges feladatok készlete) -> Megjelenik az ütemezés (scheduling).
- **Miniszámítógépek:**
  Kisebb csoportok számára elérhető, olcsóbb gépek -> Sokkal többen férnek hozzá egy géphez. MULTICS, majd UNIX - C és egyéb modern nyelvek megjelenése.
- **Időosztásos/time sharing/multitasking rendszerek:**
  Több ember tudja egy időben használni a gépet (10-100). Fontos a válaszidő (n*10 ms), a gép ne legyen üresjáratban. A feladatok virtuálisan egy időben, szeletekben futnak, egy óra által periodikusan időzítve/váltva. A háttérben egy batch rendszer, a megmaradó időszeletek kihasználására. Pl. klasszikus UNIX erre vállalkozik.
- **Személyi számítógépek:**
  1970-es évek közepétől az egy felhasználó egy gép összerendelés lehetséges. x86 CPU architektúra, memória + HDD, hálózat (LAN majd Internet) stb. Lépések az elosztott rendszerek megjelenése felé. Új követelmény a Felhasználó barátság.
- **Elosztott rendszerek:**
  Decentralizálás, funkciók térbéli elosztása.
  - Előnyök: Teljesítőképesség, sebesség és megbízhatóság hőhet
  - Hátrányok: Biztonság, skálázhatóság, megbízhatóság, fejlesztési kérdések

- ***Heterogén, homogén. GPU, sokmagos CPU stb.***

OS kategorizálása

- **Alkalmazás-specifikus**
  - **Kliens, szerver, mainframe:**

    Kliens egyértelmű. Sokprocesszoros szerver/mainframe: 8-256 CPU, 10-100 GB RAM, magas rendelkezésre állás, redundancia, működés közben történő alkatrész csere és particionálás és HW támogatott virtualizáció. Pl.: IBM System Z10, Sun Fire X4600
  - ***Adatközpontok (grid, cloud, etc.)***
  - **Beágyazott:**

    Olyan speciális számítógépes rendszerek, amelyeket egy jól meghatározott feladatra találtak ki. Ezen feladat ellátása érdekében a külvilággal intenzív információs kapcsolatban állnak. (Ez nem zárja azt ki, hogy pl. PC-ét használjunk beágyazott rendszerekben, de akkor - alkalmazástól függően - annak legalább részben dedikált feladata lesz. Akár standard Windows vagy Linux operációs rendszerrel is.) Sokszor biztonságkritikus a környezet, nNem elviselhető kockázat, így el kell kerülni, bizonyítani kell, hogy a technológia adott szintjén mindent megtettünk az elkerülésére. Nincs 100%-os biztonság! Valós idejű működés, megbízhatóság, rendelkezésre állás.
  - **Mobile:**

    Összemosódik a kliens operációs rendszerekkel. Követelmények: speciális GUI, multitouch, gyorsulásérzékelő stb. Telep/akkumulátor élettartam maximalizálása, limitált erőforrások, részben valós idejű funkciók (alacsony szintű kommunikáció). Heterogén architektúra: User CPU, kommunikációs DSP.
- **Tulajdonság-specifikus**
  - **Valós idejű:**

    A rendszer adott eseményekre adott időn belül adott valószínűséggel válaszol (egyébként hibás, hiába funkcionálisan jó a válasz). Felépítéséből következően megadott szolgáltatásai képesek valós időben működni. (Pl.: a HW megszakítás után a megszakításhoz tartozó kód futása adott időintervallumon belül.) A futtatott alkalmazásnak is valós idejűnek kell lennie, a valós idejű OS csak lehetővé teszi a valós idejű működést. (A Linux és a Windows nem ilyen, nem adható ilyen garancia - Mindkettőhöz létezik viszont ilyen garanciák megadását lehetővé tévő kiterjesztés (RTLinux, Ardence RTX))
    - **Lágy valós idejű (soft real-time):**

      A valószínűség < 1, de elég közel van egyhez. Akkor használjuk, ha a határidők nem teljesítésének nincsenek katasztrofális következményei, azok bizonyos szintig elviselhetőek. Service Level Agreement (a szolgáltatás minősége). NEM feltétlenül prioritásos a működése, de leggyakrabban az!
    - **Kemény valós idejű (hard real-time):**

      A valószínűség = 1. Ha nem válaszol időben, a válasz rossz. A határidők nem teljesítésének katasztrofális következményei vannak. A rendszer NEM késhet! A válaszra megadott időintervallum nagyságára a definíció nem mond semmit, ez erősen függ az alkalmazástól.

- ***Nagy megbízhatóságú***
- ***Konfigurálható***

HW alapok

- **Védelmi szintek:**

  Egyes processzorok támogatják az alacsony szintű CPU erőforrásokhoz történő hozzáférés szabályozását. Pl. I/O, memória bizonyos része, CPU konfigurációs regiszterek, stb. 2 vagy 4 szint van. Jellegzetesen 2-őt használunk: User mode (real mode) és Kernel mode (protected mode). (Esetleg egy 3. is van: Driver és/vagy kernel service mode).
- **Memory Management Unit (MMU):**

  Speciális HW a CPU-ban. Feladatai:
  - Memória állapotának nyilvántartása: Tulajdonos folyamat azonosítója, hozzáférési jogosultságok (ACL), CACHE-elhetőség, ha van CACHE (pl. DMA).
  - Virtuális memória leképzése fizikai memóriára: Pl. Translation lookaside buffer (TLB), kontextus váltásnál ezt is kezelni kell (ha van), Pagefile vagy SWAP (HDD).
  - Memória védelem: Tiltott memória hozzáférés megakadályozása vagy legalább jelzése (ACL alapján), General Protection Fault (GPF) a Windows-ban.
- **Uniprocessor:**

  Egy végrehajtó egység (single CPU). Teljesítményben skálázhatóak (architektúra+órajel) egyelőre. DMA, ha van, párhuzamosan kezeli a memóriát a CPU-val: Versenyhelyzet a DMA vezérlő és a CPU között.
  A CACHE koherencia sérülhet! Megoldások:
  - Egész CACHE érvénytelenítése/tiltása (a transzfer alatt) – Egyszerű, de katasztrofális a hatása a teljesítményre.
  - DMA-val kezelt memóriaterületre tiltani kell a CACHE-t (pl. MMU).
  - CACHE koherens DMA (HW támogatás).
- **SMP (Symmetric MultiProcessing):**

  Több, azonos végrehajtó egység (Több CPU, vagy CPU mag, pl: AMD Phenom vagy Athlon X2, Intel C2D/C2Q). Esetleg saját CACHE hierarchiával, a CACHE koherencia biztosításával. Közös buszon/vezérlőn a memória, így minden végrehajtó egység azonos tulajdonságokkal (sebesség, késleltetés) éri el. Muticore MCU-k megjelenése, pl: ARM11 MPCore, ARM Cortex-A9 MPCore -> Beágyazott területen is szerepet kap az SMP. OS támogatás kell hozzá (egyébként 1 CPU látható).
- **NUMA (Non-Uniform Memory Access):**

  A memória egyes részei "közelebb" vannak egyes végrehajtó egységekhez mint másokhoz:
  - Összefüggő fizikai memória
  - CACHE koherens (ccNUMA) és nem CACHE koherens változat is
  - A memóriavezérlők egy speciális kommunikációs felületen csatlakoznak (QPI az Intelnél, vagy Hypertransport az AMD-nél)
  - Pl. AMD Opteron vagy a Intel Core i7 alapú szerver processzorok több CPU esetén ccNUMA architektúrát is használhatnak (CPU-n belül SMP van)
  - OS támogatás (egyébként 1 CPU látható)
- **I/O portok:**

  I/O portok (spec. I/O utasítások), memóriába leképzett I/O portok (memória írás/olvasás). Késleltetés és rendelkezésre álló sávszélesség.
- **DMA (Direct Memory Access):**

  A processzort megkerülve lehet adatot a periféria és a memória között mozgatni. Gyorsabb, a mozgatás nem igényel processzor műveleteket, de veszélyes lehet (CPU cache).
- **Megszakítás (Interrupt):**

  Különböző szinteken tiltható és engedélyezhető. Ha az adott megszakítás engedélyezve van, és a megszakítás beérkezik, akkor a CPU áttér a megszakításhoz tartozó IT rutin végrehajtására (a részletek erősen hardver függőek). A modern operációs rendszerek megszakítás vezéreltek.

- **Szoftver:**

  A rendszer (OS és alkalmazások) futása közben történő HW kivétel (laphiba, numerikus túlcsordulás, nullával történő osztás, vagy egyéb forrás). Szoftver esemény (pl. rendszerhívás), speciális utasítás végrehajtása.

- **Hardver:**

  Periféria használja értesítésre, időzítő, hálózati kártya, stb. Jelzi a kiszolgálási igényét, amit számos okból kérhet.
  Óra megszakítás (különösen fontos): ML1: Adott órajelű oszcillátor, amiben programozható számláló számlálja az impulzusokat és adott számú impulzus után kér egy HW megszakítást. A megszakítás futtatja az OS-t (ütemezőt). Ugyan ebből az óra megszakításból kerül származtatásra a rendszeróra. Periodikus vagy egyszeri várakozás: A felbontás az órajel periódusideje (tipikusan 20-1 ms között).

OS felépítése

- **Réteges szerkezet:**

  Strukturáltság és futási idejű hatékonyság optimumát kell megkeresni. A rétegek határán egy virtuális gép valósul meg (magas szintű funkciók, virtuális utasításkészlet). A legalacsonyabb szinten a valódi CPU és perifériák által megvalósított valódi gép található.

  - **Kernel:**

    Az operációs rendszer legalapvetőbb funkcióit tartalmazó rendszermag. Feladatok kezelése, tár (memória) kezelés, védelmi és biztonsági funkciók.

  - **Hardver közeli réteg:**

    Driverek, többnyire valamilyen hierarchiába rendezve. Alacsony szintű hardver kezelés, hálózat, keyboard, egér, képernyő, stb.

  - **Rendszerprogramok:**

    Rendszerfolyamatok és alrendszerek az egyéb funkciók megvalósítására. Filerendszer, magasabb szintű hálózatkezelés (TCP/IP), parancsértelmező, stb.

  - **Rendszerhívásokat fogadó réteg:**

    API megvalósítása különböző célnyelveken (target language). Az API leképzése rendszerhívásokra. Védelmi szintek váltása.

- **Tipikus kialakítások**

  - **Monolitikus:**

    Az összes szükséges funkció összefordítva egyetlen programba. Nem flexibilis, a lehetséges hardver elemeket be kell fordítani a kernelbe. Egy komponens hibája a teljes operációs rendszer hibáját okozza. Beágyazott rendszerekre jellemző (nem változik a HW).

  - **Moduláris kernel:**

    Minimális kernel, és utólagosan, igény szerint, dinamikusan betölthető kernel modulok, flexibilis. Egy komponens hibája a teljes operációs rendszer hibáját okozza. Pl: Újabb Linux kernelek, Windows.

  - **Mikorkernel:**

    Minimális funkciókkal rendelkező kernel, és ahhoz kliens-szerver architektúrában csatlakozó szolgáltatások. Nem hatékony, de a kernel védett még a hozzá csatlakozó szolgáltatásoktól is.

- **Rendszerhívások:**

  A rendszerhívás lényegében megszakítja a processzort alkalmazói program futtatása közben (szoftver megszakítás), és átadja a vezérlést a kernelnek (állapot mentés és visszaállítás, context switch). A kernel elvégzi a munkáját, majd visszaadja a futás jogát az alkalmazói programnak (állapot mentés és visszaállítás, context switch).
  Hogyan történik a rendszerhívás? Erősen implementáció függően.
  Mi a következménye a rendszerhívásnak? Nagy overhead-je van, sok CPU időt emészt fel. Rendszer állapotának mentése és visszaállítása 2 alkalommal. A CPU védelmi szintet is vált: user-kernel-user. Minimalizálni kell az alkalmazott rendszerhívások számát!

- **OS elindulása:**

  Bootstrap process (PC és Szerverek):
  - Init/RESET vector (CPU)
  - BIOS (firmware)
    - POST (Power on self test)
    - HW perifériák inicializálása
    - Boot média meghatározása
  - BOOT sector (HDD típusú tároló)
  - 2. szintű boot loader (GRUB, LILO, NTLDR)
  - OS betöltődik majd elindul
    - HW újraprogramozása (device driver BIOS helyett)
    - Védelmi szint váltás (kernel módra vált)

  Bootstrap process (Beágyazott rendszer, PC is BIOS/EFI szinten):
  - OS image ROM-ban (ROM v. flash, esetleg tömörítve)
  - ROM-ból futtatható
  - RAM-ba másolható majd onnan futtatható

Feladatok kezelése

Alapfogalmak

- **Feladat:**

  A feladat egy végrehajtás alatt álló program; aktív, nem passzív
  - **Feladat adatszerkezetei**

    Virtuális memória (OS + HW)
    Adat terület (globális adatok)
    Verem (stack)
    Halom (Heap)
  - **Feladat állapotai**
    - **Futásra kész**

      Az ilyen feladatnak minden erőforrás rendelkezésére áll, kivéve a CPU.
    - **Fut**

      CPU felszabadul, futásra készből futba megy, övé a CPU
    - **Eseményre várakozik**

      pl:rendszerhívást hajt végre, de nem kapja vissza a CPU-t, mert időre van szükség a kiszolgáláshoz, és addig a CPU-t más használja. A feladat "Eseményre várakozik", amit az OS vagy más feladatok fognak előállítani. A feladat passzívan várakozik, nem használ CPU időt. Ha a várt esemény bekövetkezik. A feladat ismét futásra kész állapotba kerülhet, hiszen minden feltétel megvan a futásához, kivéve a CPU.
  - **Feladat leíró (Task control block)**

    Adatstruktúra a feladattal kapcsolatos adatok
    operációs rendszeren belüli tárolására.
    Ez alapján végzi az OS a feladatok kezelését
    (taskid,állapot,cpu regiszterek, pc, mmu állpot, stb.)
- **Virtuális gép:**

  Alapesetben a feladatoknak nem szabad tudniuk egymásról, egy-egy külön virtuális gépen futnak, virtuálisan saját CPU és memória.
- **Esemény:**

  A rendszer életében lezajló változás vagy történés
  Belső esemény: Szoftver megszakítás vagy kivétel
  Külső esemény: Hardver megszakítás
  A modern operációs rendszerek megszakítás vezéreltek! „Eseményvezéreltnek” is hívják (event driven) ezért az OS-eket...

- **Program:**
  Utasítások sorozata, amelyeket a számítógép interpretálhat és végrehajthat. (A sequence of instructions that a computer can interpret and execute)
- **Kontextus váltás (context switch):**
  Amikor egy feladat futni kezd helyre kell állítani a környezetét, amiben korábban futott.

Feladatok ütemezése (taks scheduling)

Kiválasztani a futásra kész feladatok közül a futót. A feladatokat a többnyire feladat sorokban (task queue vagy job queue) tároljuk

- **Időskálája**
  - **Hosszú távú**
    Sokkal több feladatunk van, mint amennyit hatékonyan párhuzamosan végre tudunk hajtani. Percenként vagy ritkábban fut, ismernie kell a feladatot (az általa okozott terhelést).
  - **Közép távú**
    Swapping. A rendszerben lévő feladatok memóriájának egyes részei kiírhatóak háttértárra. Több feladat fér el a fizikai memóriába (virtuálisan). Egy feladatra több fizikai memória juthat (amikor fut).
  - **Rövid távú**
    A futásra kész sorból választ egy futó állapotba átmenő feladatot. Minimum 10-20 ms-enként végrehajtásra kerül (időzítő megszakítás), de inkább gyakrabban.
- **Fajtája**
  - **Preemptív ütemezés**
    A futó feladattól az OS elveheti a CPU-t. Bizonyos kernel feladatokra nem feltétlenül, azok nem megszakíthatóak, ennek vannak következményei (pl. valós idejű működés nehezen biztosítható)
  - **Nem preemptív**
    A futó feladatnak le kell mondania a CPU-ról vagy eseményre kell várnia, ahhoz hogy más feladat tudjon futni. Az egyes alkalmazásoktól függ a teljes rendszer működése. Ha a futó feladat hibás (végtelen ciklus), akkor a teljes rendszer működésképtelenné válik.
- **Mértékek**

Ezekkel tudjuk az algoritmusokat összehasonlítani.
Többnyire több mértéket együtt tekintve kell választanunk.
Mindegyiknek van mértékegysége (unit) is!

  - **CPU kihasználtság**
    A hasznos munkával töltött idő aránya az összes időhöz képest.
  - **Átbocsátó képesség**
    Adott időegység alatt elvégzett feladatok száma.
    A rendszerfeladatokat nem számoljuk.
  - **Várakozási idő**
    Az összes idő, amit a feladat várakozással töltött.
  - **Körbefordulási idő**
    Egy feladatra vonatkozóan a rendszerbe helyezéstől a teljesítésig eltelt idő.
  - **Válaszidő**
    On-line, interaktív feladatok esetén. A feladat megkezdésétől az első kimenetek produkálásáig eltelt idő.
  - **Felhasznált energia**
    Mértékegység: pl. Ws/task Mennyi energia szükséges egy feladat elvégzéséhez

- **Kvalitatív jellemzők**
  Elvárt tulajdonságok (nem számszerűsíthetők)
  - **Korrektség**
  - **Kiéhezés elkerülése**
  - **Jósolható viselkedés**
  - **Alacsony administratív veszteség**
  - **Maximális átbocsátó képesség**
  - **Erőforrás-használat figyelembe vétele**
- **Egyéb tulajdonságok**
  - **Valós idejű ütemezés**
    Valós idejű ütemezés (hard or soft real-time). Lehessen feladatokat garantált körbefordulási idővel ütemezni, ha azok végrehajtási idejére adható felső korlát.
  - **Prioritás**
    A feladatokhoz fontosságot (prioritást) rendelünk.
  - **Fokozatos leromlás (Graceful degradation):**
    Ha a rendszer terhelése eléri az u.n. könyökkapacitást, akkor utána viselkedése megváltozik, a tovább növekvő terhelésre már egyre rosszabb működéssel reagál (overhead). Elvárható, hogy ezt fokozatosan tegye (ne omoljon össze)
  - **Statikus vagy dinamikus ütemezés**
    - Statikus:
      Tervezési időben teljesen meghatározott, hogy milyen feladatok és mikor futnak. Legrosszabb esetre tervezés. Speciális, többnyire biztonságkritikus beágyazott rendszerekben alkalmazzák.
    - Dinamikus:
      Futási időben dől melyik feladat és mikor fut. A gyakorlatban használt algoritmusok ilyenek. Dinamikus erőforrás kihasználás. Tervezési időben nehezen vizsgálhatók.
- **Ütemezési algoritmusok (egy CPU)**
  egy cpu, egyszerre egy feladat fut, feladatok cpu és i/o löketekből állnak, futásra kész feladatokat várakozási sorban tároljuk
  - **FIFO, FCFS**
    Amelyik folyamat előbb érkezett az kerül futó állapotba. Nem preemptív. Kis adminisztrációs overhead. Convoy hatás (hosszú folyamat feltartja a rövideket).
  - **Round-robin**
    Adott időnként garantáltan vált, függetlenül a feladattól. Időosztásos rendszerek számára találták ki. Preemptive.
    - **quantum, time slice**
      Időszelet, ennyi időközönként vált, ha nem fejeződött be előbb a folyamat. Ökölszabály: A CPU löketek 80% kisebb az időszeletnél.
  - **Prioritásos ütemezők**
    Ütemező család. A feladatokhoz prioritást(fontosságot) rendelünk. A gyakorlatban kombinálhatók.
    - **Külső, belső prioritás**
      Külső prioritás: Operátor vagy a feladat maga állítja be. Belső prioritás: A rendszer adja.
    - **Statikus, dinamikus prioritás**
      Statikus prioritás: Tervezési időben dől el, állandó a futás során. Dinamikus prioritás: Futási időben dől el, változik a rendszerben lezajló változások hatására.

- ### Shortest Job First SJF

  Nem preemptív, a legrövidebb (becsült) löketidejűt választja futásra.

- ### Shortest Remaining Time First SRTF

  Az SJF preemptív változata. Ha új folyamat válik futásra késsze, akkor vizsgálja meg, hogy melyik folyamat löketideje a kisebb. A legrövidebbet indítja el. A kontextus váltást is figyelembe kell venni (megéri-e a váltás).

- ### Highest Response Ratio HRR

  A kiéheztetés próbálja megoldani. Az SJF-ből indul ki. A prioritás képzésébe a várakozási idő is beleszámít. Minél többet vár, annál valószínűbb, hogy ütemezve lesz. A K-t meg kell választani, mennyire vegyük figyelembe a feladat korát: (löketidő + K*várakozási idő) / löketidő.

- ### Többszintű sorok

  Multilevel queue: Minden egyes prioritási szinthez "futásra kész" várakozási sor.

  - #### Prioritásonként futási sorok

    Többnyire a prioritás a feladat jellegéhez kapcsolódik.
    Batch feladatok alacsony prioritással.
    On-line (interaktív) feladatok közepes prioritással.
    Rendszer feladatok magas prioritással.
    Valós idejű feladatok legmagasabb prioritással.
    Tipikusan RR (Időosztás) minden prioritási szinten.

  - #### Weighted fair queuing

    Prioritásonkénti futási sor, kiegészítve:
    Időosztás prioritási szintek között (korrektség biztosítására, kiéheztetés elkerülésére). Adott % CPU idő egy adott prioritási szinthez. Ha az adott prioritási szinten vannak futásra kész feladatatok, azok az adott prioritási szinten elérhető CPU időt megkapják, de többet csak akkor kapnak, ha nincs alacsonyabb prioritású futásra kész feladat.

  - #### Visszacsatolt többszintű sorok

    A feladatok mozgatása a sorok között a ténylegesen végrehajtott CPU löketek alapján. A rövid CPU löketű feladatokat részesítjük előnyben, azok maradnak a jelenlegi sorukban. A hosszabb CPU löketű feladatokat alacsonyabb prioritású, de nagyobb időszeletű sorokba helyezzük. Ezeket dinamikusan felülvizsgáljuk, ha csökken a CPU löket, akkor magasabb prioritású, de rövidebb időszeletű sorba kerülnek. A régen várakozók prioritása is emelkedhet (ageing).

Többprocesszoros ütemezés:

Multiple-processor scheduling: Lehet SMP vagy NUMA architektúra a memória szempontjából (vagy azok keveréke). Egy adott I/O periféria egy adott processzorhoz van rendelve (arra csatlakozik). Homogén többprocesszoros rendszerekkel foglalkoztunk.

- ### Fajtái

  - #### Master and slaves

    Egy CPU osztja ki a feladatokat

  - #### Self-scheduling

    Minden CPU ütemez

- ### Processzor affinitás

  A feladatot ugyan azon a végrehajtó egységen tartani.
  Bővebben:
  A cache a futó feladat utasítás és adat tartalmának egy
  részét tartalmazza. A processzorok vagy processzor magok cache tartalma különböző lehet. A feladat más processzorra, vagy processzor magra kerülése csökkenti a végrehajtás sebességét, mert a lokális cache-ben nincs benne a kód és/vagy adat és a CPU cache tartalmat ismét fel kell építeni.

  - #### Laza affinitás

    Nincs garancia, de törekszik rá az OS (többnyire alapeset)

        - **Kemény affinitás**

          Biztosan ugyan azon a CPU-n marad (rendszerhívással)
    - **Terhelés megosztás (load balancing)**

      Egy globális futásra kész sor vagy processzoronkénti futásra kész sor. Ha orocesszoronkénti futásra kész sor van akkor:
      Push: OS kernel folyamat mozgatja a sorok között a feladatokat.
      Pull: Az idle állapotban (idle feladatot végrehajtó) CPU próbál a többi sorából feladatot kapni.
- **Konkrét ütemezési algoritmusok**
    - **Klasszikus UNIX ütemezés**

      Egyszerű, Prioritásos, futási sorok, Round robin 2.4 verzió: O(n) ütemező
    - **Linux ütemezők**

      2.6-os kernel
      Gond volt, ha sok feladat volt
      Következő feladat megtalálása: O(1)
      140 várakozási sor
      Külön „active” és „expired” sorok
      Jobb SMP támogatás
      CPU-nkénti runqueue, külön zárakkal
      Terheléselosztás CPU sorok között
        - **O(1)**

          Heurisztikákat használt, nehéz volt számolni
          Késleltetés mértékét nehéz volt garantálni O(1) azt jelenti független a folyamatok számától, nem azt hogy nagyon gyors!
        - **CFS**

          Completly Fair Scheduler
          Cél: fair működés garantálása
          Mindenki kapjon CPU időt
          Futási sorok helyett egy közös keresőfa (red-black tree)
          Prioritás a KF-on (decay) keresztül jut érvényre
    - **Windows ütemezés**

      Preemptív ütemező (kernel és user módban is!)
      32 prioritási szint
      Legmagasabb prioritású szál fut mindig
      Azonos prioritásúak között Round Robin
      A szálak adott ideig futnak (quantum)
      Szálak prioritása változhat a futás során
        - ***Prioritásos RR ütemező***

          Az van a szinteken gg
        - ***Quantum hossza***

          Időegység, amíg egy szál fut
          Óra megszakításban mérik (clock interval, clock tick) 1 clock tick = ~ 10-15 ms (HAL-tól függ) ide mit még?? Feladatok együttműködése

Feladat fogalom megvalósítása:

Feladat fogalom eredetileg folyamat (process) értelemben került használatra. A folyamat a végrehajtás alatt álló program. Ugyan abból a programból több folyamat is létrehozható. Saját kód, adat, halom (heap) és verem memória területtel rendelkezik. Védettek a többi folyamattól.

- **Folyamat (process)**
  - **Folyamat vs. program**
  - **Részei**
    - ***Kód***
    - ***Adat***
    - ***Halom***
    - ***Verem***
  - **Folyamatok szeparációja**
    - **Virtuális CPU**

      Nem férhetnek hozzá a többi folyamat és az operációs rendszer futása során előálló processzor állapothoz. Kontextus váltás történik, ha más folyamat kerül futásra.
    - **Virtuális memória**

      Nem férhetnek hozzá más folyamatok virtuális memóriájához vagy direkt módon a fizikai memóriához. A processzor MMU-ja oldja ezt meg.
    - ***Folyamatok kommunikációja csak rendszerhívásokon keresztül***
  - **Folyamatok létrehozása, befejezése**
    - Létrehozás:
      OS specifikus rendszerhívás (pl. CreateProcess(), fork(), stb.). Szülő/gyermek viszony a létrehozó és a létrehozott között. -> Process fa (process tree), a szülő erőforrásaihoz hozzáférés többnyire konfigurálható, szülő megvárhatja a gyermek terminálódását, vagy futhat vele párhuzamosan. Paraméterezhető a gyermek (command line).
    - Befejezés:
      OS specifikus rendszerhívás (pl. TerminateProcess(), exit(), stb.). Nyitott, használatban lévő erőforrásokat le kell zárni. A szülő megkapja a visszatérési értéket, többnyire egy egész értékű változó (integer) formájában. Ha a szülő folyamat befejeződik, de a gyermek nem: OS függő (Alapértelmezett szülő folyamat, A gyermek automatikus befejezése)
- **Szál (thread)**
  - **CPU használat egysége**

    A szál a CPU használat alapértelmezett egysége, magában szekvenciális kód. Saját virtuális CPU-ja van, és saját verem áll rendelkezésre.
  - **Adott folyamat kontextusában**

    A kód, adat, és halom, és egyéb erőforrások (pl. fájl) tekintetében osztozik azokkal a további szálakkal, amelyekkel azonos folyamat kontextusában fut.
  - ***Szálak támogatása***
  - ***Felhasználói módú szálak***
  - **Szálak létrehozása**

    Win32: CreateThread() bonyolult paraméterezéssel. Pthreads: POSIX threads pl. Linux és más UNIX variánsok kernel vagy akár user szinten (csak viselkedést ad meg). JAVA (VM a folyamat, VM-en belül szál): Thread osztályból származtatva, Runnable interface megvalósítása

- ***Szálak alkalmazásának előnyei***
- ***Szálak alkalmazásának következményei***
- ***HW támogatás (ábra)***

- **Coroutine és fiber**
  - Kooperatív multitasking: Folyamaton vagy szálon belül.
    - OS támogatással vagy felhasználó megvalósítás
    - Az OS szinten a coroutine-eket vagy fiber-eket tartalmazó folyamat vagy szál kerül ütemezésre
    - Az ütemezési algoritmus a felhasználó kezében van, neki kell implementálnia (kooperatív ütemezés).
  - Teljesen az alkalmazás programozó kezében van.
  - Coroutine:
    programnyelvi szintű elem, Subroutine általánosítása
    - Subroutine:
      - LIFO (Last In/called, First Out/returns).
      - Egy belépési pont, és több kilépési pont (return/exit)
      - A vermet használja a paraméterek és a visszatérési érték átadására.
    - Coroutine:
      - Az első belépési pont azonos a subroutine-nal.
      - Utána viszont a legutolsó kilépési pontra tér vissza!
      - Átlépés a „yield to Coroutine_id” utasítással lehet.
      - Nem használhat vermet, hiszen az megtelne (ide-oda lépked, igazából nem tér vissza).
  - Fiber:
    Rendszerszintű eszköz
- ***Coroutine és fiber értékelése***

Feladatok együttműködése

- **Alapfogalmak**
  - **Erőforrás**

    Minden olyan eszköz, amire a párhuzamos programnak futása közben szüksége van. A legfontosabb erőforrás a végrehajtó egység. Memória és annak tartalma (tárolt adatstruktúrák). Perifériák.
  - **Közös erőforrás**

    Egy időintervallumban több, párhuzamosan futó feladatnak lehet rá szüksége. Az erőforráson osztoznak a feladatok. Többnyire egy időben egy vagy maximum megadott számú feladat tudja helyesen használni (írás és olvasás).
    - **Egy felhasználó**

      Printer, Asszinkron soros port (UART), összetett adattípusokból létrehozott változók (string, tömb, struktúra, objektum).
    - **Több párhuzamos felhasználó**

      SCSI vagy SATA NCQ HDD (N parancs optimalizált párhuzamos végrehajtására képesek).
  - **Kölcsönös kizárás (mutual exlusion)**

    Annak biztosítása, hogy a közös erőforrást egy időben csak annyi magában szekvenciális feladat használja, amely mellett a helyes működése garantálható.
    A kölcsönös kizárást meg kell oldanunk a programban
    Többnyire a használt erőforrást lock-oljuk (elzárjuk). (Nem engedjük hozzáférni a többi részfeladatot. A kérdés az, hogy azt hogyan tudjuk megoldani, és milyen részletességgel kell megoldanunk azt.)
  - **Kritikus szakasz (critical section)**

    A magában szekvenciális feladatok azon kódrészletei, amely során a kölcsönös kizárást egy bizonyos közös erőforrásra biztosítjuk.

A kritikus szakasz a kérdéses közös erőforráshoz tartozik.
A kritikus szakaszt a hozzá tartozó erőforrásra atomi műveletként (nem megszakítható módon) kell végrehajtanunk.

- **Atomi művelet (atomic operation)**

  Nem megszakítható művelet, amelyet a processzor egyetlen utasításként hajt végre.
  Egyprocesszoros rendszerben bármilyen műveletsor atomivá tehető a műveletsor elején az IT teljes tiltásával, majd a műveletsor végén annak engedélyezésével.
  TAS, RMW, speciális CPU utasítások az IT tiltás/engedélyezés elkerülésére.
  - Test and Set, Read-Modify-Write, stb.
  - Elemi adattípusra (8/16/32/64 bit).
  - A modern processzoroknak vannak ilyen utasításai.

  A közös erőforrások lock-olást, a kritikus szakasz megvalósítását atomi műveletekre vezetjük vissza.
- **Újrahívhatóság (reentrancy)**

  A közös erőforrás problémájának egyfajta kiterjesztett esete egy függvényen/objektumon belül is felléphet, amennyiben ezt a függvényt (metódust) egyszerre többen is meghívhatják. Pl. Ugyanazt a függvényt hívjuk egy taszkból is és egy megszakítás rutinból is. Az ütemezés preemptív, és ugyanazt a függvényt hívjuk két taszkból is.
- ***Lock-olás és az ütemező, szabad erőforrás***
- **Zárolás részletessége (fine or course grained locking)**

  A kölcsönös kizárás megvalósítása erőforrás használattal jár (CPU), minimalizálni kell a használatát -> Túl sok rendszerhívás jelentős overhead-del jár.
  Viszont a túl nagy egységekben végzett kölcsönös kizárás is erőforrás pazarlással jár. -> A rendszerben „nehezebb” futásra kész részfeladatot találni.

- **Hibák**
  - **Versenyhelyzet (race condition)**

    A párhuzamos program lefutása során a közös erőforrás helytelen használata miatt a közös erőforrás nem megfelelő állapotba kerül.
    A hibás állapot részletei erősen függenek azt használó szekvenciális részfeladatok lefutási sorrendjétől.
  - **Kiéheztetés (starvation)**

    Ha a párhuzamos rendszer hibás működése miatt egy feladat soha nem jut hozzá a működéséhez szükséges erőforrásokhoz, akkor ki van éheztetve (nem tud futni).
    Nem csak a CPU-ra, de más közös erőforrásokra is felmerülhet
  - **Holtpont (deadlock)**
    - **Holtpont definíciója**

      A közös erőforrások hibás beállítása vagy használata miatt a rendszerben a részfeladatok egymásra várnak.
      Nincs futásra kész folyamat.
      Nem jöhet létre belső esemény.
      A rendszer nem tud előrelépni.
      Egy rendszer feladatainak egy H részhalmaza holtponton van, ha a H halmazba tartozó valamennyi feladat olyan eseményre vár, amelyet csak egy másik, H halmazbeli feladat tudna előállítani.
    - **Szükséges feltételek**
      - **Kölcsönös kizárás (Mutual Exclusion)**

        Vannak olyan erőforrások, amelyeket csak kizárólagosan lehet használni, és azokat több feladat használja.
      - **Foglalva várakozás (Hold and Wait)**

        Legyen olyan feladat, amelyik lefoglalva tart erőforrásokat, miközben más erőforrásokra várakozik.

- **Nincs erőszakos erőforrás elvétel (No resource preemption)**

  A feladatok csak önszántukból szabadítják fel az erőforrásokat (kivéve az ütemezést, az lehet preepmtív).
- **Körkörös várakozás (Circular Wait)**

  A rendszerben lévő feladatok között létezik egy olyan {P0, P1, ... Pn) sorozat, amelyben P0 egy P1 által lefoglalva tartott erőforrásra vár, Pi egy Pi+1-ra épül, és Pn pedig P0-ra ilyen szempontból.

- **Holtpont kezelése**
  - ***Strucc algoritmus***
  - **Holtpont észlelés és feloldás (detection and recovery)**
    - **Észlelés: Wait-for gráfok**

      Egypéldányos eset
    - **Feloldás: radikális, kíméletes**

      Többféle megoldás.
      - Radikális: Az összes holtpontban lévő feladat felszámolása.
      - Kíméletes: Egyes feladatok felszámolása, ekkor dönteni kell, és a siker sem biztosított.

      Feladatok:
      - Áldozatok kiválasztása (Selecting a victim).
      - Feladatok visszaállítása (Rollback), közbenső visszaállítási pontokkal.

      El kell kerülni, hogy ne mindig ugyan azt a feladatot állítsuk vissza.
      - Kiéheztetés, esetleg rossz megoldás (victim) elkerülése.
  - **Holtpont megelőzés (deadlock prevention)**

    Tervezési idejű módszer.
    Olyan rendszert hozunk létre, ami holtpontmentes, vagyis a holtpont szükséges feltételeiből legalább egy nem teljesül.
    Nincs futási idejű erőforrás használat.
    Foglalva várakozás kizárása
    - Az erőforrást birtokló feladat kér újabb erőforrást.
    - Minden szükséges erőforrást egyben kell lefoglalni, egyetlen rendszerhívással.
    - Alkalmazástól függ a használhatósága.
    - Erőforrás-kihasználás romlik.
  - **Holtpont elkerülése futási időben (deadlock avoidance)**

    Az erőforrás igény kielégítése előtt mérlegelésre az erőforrás igény kielégítésével nem kerülhet-e holtpontba a rendszer, fennmarad-e a biztonságos kerül, hogy: állapot?
    Bankár algoritmus
    - Adott mennyiségű erőforrás kihelyezésnek az ütemezése úgy, hogy mindenki végig finanszírozható legyen (és végül a hitel visszafizethető legyen).
    - N feladat, M erőforrás
    - M erőforrás többpéldányos
    - A feladatok előzetesen bejelentik, hogy az egyes erőforrásokból maximálisan mennyit használnak futásuk során:
      - MAX NxM-es mátrix,
    - A feladatok által lefoglalt erőforrások száma
      - FOGLAL NxM-es mátrix
      - Megadják egy adott időpontban, vagy a beérkező kérések alapján előállítható.
    - Szabad erőforrások száma:
    - MAXr az egyes erőforrásokból rendelkezésre álló maximális példányszám
    - FOGLALr az egyes erőforrásokból foglalt példányszám
    - Egy feladat által még maximálisan bejelenthető kérések száma:
      - MÉG = MAX-FOGLAL NxM-es mátrix
    - A feladatok várakozó kérései:
    - KÉR NxM-es mátrix

- **Livelock**

  Példa: „Két kedves ember összetalálkozik az ajtóban.”
  Többnyire a hibás holtpont feloldás eredménye.
  A rendszer folyamatosan dolgozik, de nem lép előre.

- **Prioritás inverzió (priority inversion)**

  Prioritásos ütemezőkben fordulhat elő, de az erőforrás használattal is összefügg.
  A legegyszerűbb esetének előfordulásához kell: 3 feladat, különböző prioritással, Egy közös erőforrás, amelyet a 3 feladat közül a legmagasabb és a legalacsonyabb is használni kíván. A közepes prioritású feladatnak CPU intenzívnek kell lennie.

  - **Prioritás öröklés (priority inheritance)**

    Az alacsony prioritású feladat megörökli az általa kölcsönös kizárással feltartott feladat prioritását a kritikus szakaszából való kilépéséig.
    Csak részben oldja meg a problémát.

  - **Prioritás plafon (priority ceiling)**

    Majdnem ugyan az, de az adott közös erőforrást használó legnagyobb prioritású feladat prioritását örökli meg (ami lehet nagyobb mint az éppen feltartott feladat).
    Az adott erőforrást máskor használó többi feladat sem tud futni (ha esetleg azok is CPU intenzívvé „válnak”).
    Az alacsony prioritású feladat akadályoztatás nélkül le tud futni.

  - ***Prioritás Inverzió más szempontból***

- **Együttműködés megvalósítása**
  - **Közös memórián keresztül**
    - **Párhuzamos végrehajthatóság feltétele (Bernstein)**

      Pi és Pj két darabja egy programnak.
      Pi összes bemeneti változója Ii, és az összes kimeneti változója Oi, ugyan ez Pj-re Ij és Oj.
      A két program párhuzamosan végrehajtható (vagyis független):
      - $Ij \cap Oi = \emptyset$
      - $Ii \cap Oj = \emptyset$
      - $Oi \cap Oj = \emptyset$

    - **RAM és PRAM modell**

      RAM
      - Klasszikus Random Access Memory (egy végrehajtó egység).
      - Tárolórekeszekből áll,
      - Egy dimenzióban, rekeszenként címezhető, csak rekeszenként, írás és olvasás műveletekkel érhető el,
      - Az írás a teljes rekesztartalmat felülírja az előző tartalomtól független új értékkel,
      - Az olvasás nem változtatja meg a rekesz tartalmát, tehát tetszőleges számú, egymást követő olvasás az olvasásokat megelőzően utoljára be-írt értéket adja vissza.

      PRAM
      - Parallel Random Access Memory (sok végrehajtó egység).
      - Több végrehajtó egység írhatja és olvashatja párhuzamosan.
      - Változások a RAM modellhez képest:
        - Az olvasás-olvasás ütközésekor mindkét olvasás ugyanazt az eredményt adja, és ez megegyezik a rekesz tartalmával,
        - Az olvasás-írás ütközésekor a rekesz tartalma felülíródik a beírni szándékozott adattal, az olvasás eredménye vagy a rekesz régi, vagy az új tartalma lesz (versenyhelyzet), más érték nem lehet,
        - Az írás-írás ütközésekor valamelyik művelet hatása érvényesül, a két beírni szándékozott érték valamelyike írja felül a rekesz tartalmát (versenyhelyzet), harmadik érték nem alakulhat ki.

- **Zárolás megvalósítása**
  - **Passzív várakozás**
    - **Sleeplock, blocking call**

      Ütemező által karbantartott várakozási sorok
      Ha az erőforrás nem lock-olt: Megkapja az erőforrást a feladat lezárva és fut tovább.
      Ha az erőforrás lock-olt: 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 megkapja az erőforrást lezárva, és futásra kész állapotba kerül.
      Erőforrás takarékos, de van overhead.
      Utána csak futásra kész sorba kerül a részfeladat, pontos időzítés nehezen megoldható (a futás kezdete érdekes). Alsó korlátot ad.
      A processzor aludhat, ha nincs feladat
  - **Aktív várakozás**
    - **Spinlock, busy wait**

      Aktív várakozás az erőforrás felszabadulására és megszerzésére (CPU erőforrás pazarlás).
      - Ha aktívan vár egy részfeladat, akkor a többi részfeladat hogyan tudja „előidézni” a várakozást megszüntető változást (eseményt) a rendszerben?
      - Nem tudnak futni, az aktívan várakozó fut! Külső HW megszakítás és/vagy több CPU esetén ez nem probléma.

      Fogyasztás is nő, hiszen a CPU folyamatosan fut, nem tud aludni (ha nincs éppen feladat).
      Függ a CPU sebességétől (ha adott időt akarunk várni)
- **Kölcsönös kizárás eszközei**
  - **Lock bit**

    Legegyszerűbb forma.
    A védendő erőforráshoz tartozik egy logikai változó (Boolean).
    Lock bit jelentése:
    - Lock bit FALSE -> nem használt az erőforrás.
    - Lock bit TRUE -> használt az erőforrás.

    Belépés művelet:
    - Tesztelés, ha
      - Lock bit == FALSE
        - Lock bit = TRUE
        - Megyünk tovább (belépünk a kritikus szakaszba)
      - Lock bit == TRUE
        - Aktív várakozás, amíg nem lesz FALSE

    Kilépés művelet:
    - Lock bit = FALSE
  - **Szemafor**
    - **Bináris**

      egy feladat a kritikus szakaszban. Magas szintű lock bit.
    - **Számláló típusú (counter)**

      több feladat a kritikus szakaszban, vagy N darabos erőforrás készletből M darab lefoglalása. Belépés és kilépés lehet egy számmal paraméterezett (hány egység lép be vagy ki). Ha 1-nél több erőforrásra van szükségünk, akkor azokat vagy egyben mind megkapjuk, vagy a töredékeket nem foglaljuk le (más feladatnak szüksége lehet rájuk).
    - **Kritikus szakasz objektum**

      Lényegében bináris szemafor szerűen működik.
      Létre kell hozni a CriticalSection objektumot
      Enter() metódussal lépünk be a kritikus szakaszba

Leave() metódussal lépünk ki a kritikus szakaszba.
Ha szükséges a CriticalSection objektum megszüntethető.

- **Mutex**

  Lényegében bináris szemafor szerűen működnek
  Acquire()/WaitOne() függvény vagy metódus.
  Release() függvény vagy metódus.

- **Kölcsönös kizárás megoldása**
  - **Monitor**

    A lock-olás nem szétszórva történik a programban, hanem egyetlen, a közös erőforráshoz szorosan tartozó programrészletben.
    A megvalósítás lehet automatikus, például nyelvi szinten (pl. JAVA).
    Kézzel is készíthetünk hasonló konstrukciót, pl. egy védett objektumot hozhatunk létre, amely a nyilvános metódusaiban elvégzi a lock-olást, és elrejti a közös erőforrást.
  - **Randevú (rendezvous)**

    Két vagy több feladat összehangolt végrehajtása
    Ebben az esetben az operációs rendszer által nyújtott szolgáltatásokat használunk, és a feladatok passzívan várnakegymásra.
    A szemaforok alapesetben foglaltként vannak inicializálva ebben az esetben.
- **Lock-olás során elkövetett tipikus hibák**
- **Hoare és Mesa szemantika**
- **Üzenetekkel**

Lehet például:
- Rendszerhívás.
- TCP/IP kapcsolat (TCP) vagy üzenet (UDP) gépen belül (localhost) vagy akár gépek között.

Többnyire az alkalmazás kódjában OS API függvény/metódus hivésként jelenik meg.
Az operációs rendszer valósítja meg szolgáltatásaival.

- **Üzenettovábbítás tulajdonságai**

  A közös memóriához képest:
  - Nagyobb késleltetés.
  - Kisebb sávszélesség.
  - A csatorna nem megbízható elosztott rendszerek esetén.
- **Üzenetek címzése**
  - **Unicast Egy adott folyamat**
  - **Broadcast Minden folyamat**
  - **Multicast Folyamatok csoportja**
  - **Anycast Egy folyamat**
- **Üzenettovábbítás fajtái**
  - **Direkt kommunikáció**

    Szimmetrikus üzenet alapú kommunikáció.
    - send(P, message)
    - receive(Q, message)
    - P, Q folyamat azonosítók. A vevő megadja az adót.
    - A message egy adatstruktúra, ami az üzenetet tartalmazza.

    Aszimmetrikus üzenet alapú kommunikáció.
    - send(P, message)
    - receive(id, message)
    - P folyamat azonosító, id a küldő azonosítója. A vevő bárkitől fogad üzenetet!
    - A message egy adatstruktúra, ami az üzenetet tartalmazza.

- **Indirekt kommunikáció**

  Egy köztes szereplőn keresztül történik a kommunikáció
  Ez a szereplő pl. lehet: Postaláda (Mailbox), Üzenetsor (MesssageQueue), Port, stb.

- **Üzenettovábbítás módja**
  - **Szinkron (blokkoló)**

    Az eredmény és mellékhatások a visszatérés utánjelentkeznek (megtörtént).
    Visszatérési érték kezelés egyszerű...
  - **Aszinkron (nem blokkoló)**

    Az eredmények és mellékhatások a visszatéréskor még nem jelentkeznek (nem történtek meg).
    Csak a végrehajtás kezdődik el a hívásra.
    A visszatérési érték kezelése, és az eredmények és mellékhatások kezelése csak más értesítés után lehetséges:
    - ***Esemény, jelzés (signal), callback függvény***

- **Implementációk**
  - **Postaláda (mailbox)**

    Indirekt kommunkáció.
    Egy vagy több üzenet tárolása, többnyire véges számú.
    Operációs rendszer szintű támogatás.
  - **Üzenetsor (message queue)**

    Indirekt kommunkáció.
    Többnyire végtelen számú üzenet tárolására alkalmas.
    Természetesen a rendszer erőforrásai korlátozzák.
    Üzenet alapú middleware-ek
  - **TCP/IP**

    Direkt kommunikáció.
    Socket interface.
    Gépen belül a localhost-on (127.0.0.1/8).
    Alacsony szintű, számos más middleware alapul rajta:
    - Távoli eljárás hívás (Remote Procedure Call, RPC).
    - Távoli metódus hívás
    - Üzenet alapú middleware-ek (korábban volt szó).
  - **Folyamok (stream) és csővezetékek (pipe)**

    Többnyire direkt.
  - **System V Shared Memory**

    Direkt.
    Memória szerű interfész.
    !!!!!!!!!!!!!
  - **Távoli eljárás hívás (Remote Procedure Call)**

    Egy másik folyamat kód memóriaterületén elhelyezkedő függvény „meghívása” a hívó folyamatból.
    A hívó fél blokkolva vár a távoli hívás lefutására.
    A meghívott függvény az őt tartalmazó folyamat egyik vezérlési szálában fut le.

        - **Platform független adat reprezentáció**
        - **Csonk (stub) hívása**
- ***Együttműködés ellenőrzése***
    - ***Célok***
        - ***Algoritmusok leírása***
        - ***Rendszer szimulálása***
        - ***Követelmények megfogalmazása***
        - ***Követelmények ellenőrzése***
    - ***Eszközök***
        - ***Modellellenőrző (model checker)***
        - ***Statikus ellenőrző (static cheker)***

Memóriakezelés

Alapfogalmak

- **Tárolóeszközök hierarchiája**
    - **CPU regiszterek**

      Típus: jellegzetesen D tároló. 10-100 gépi szó. x86 architektúrában kis számú. Nem általános felhasználású egy része (PC, szegmens reg., stb.).
      Sebesség:
        - Az utasítás végrehajtása alatt akár többször elérhető.
    - **Átmeneti/gyorsító tár**

      Típusa:
        - jellegzetesen SRAM.
        - Többnyire többszintű.
            - 1. szint 64-128 Kbyte (I+D).
            - 2. szint 1-8 Mbyte.
            - 3. szint 4-32 Mbyte (ha van).

      Sebesség:
        - Sávszélesség: n*10 Gbyte/s. 1. szint Egy vagy néhány órajel ciklus. 2. és 3. szint 10 – n*10 órajel ciklus
    - **Központi memória**

      Típusa:
        - jellegzetesen DRAM.

      Méret:
        - n*10 Mbyte – n*100 Gbyte

      Sebesség:
        - Memory wall (és okai).
        - Max. és random access más.
        - Max.: n*1 Gbyte/s – 10 Gbyte/s.
        - Random: kevesebb.
        - Késleltetés: n*10 ns (worst case)
    - **Permanens tár**

      Típusa:
        - HDD (mágneses diszk).
        - Flash memória (pendrive, SSD, stb.).
        - Fájl alapú elérés (blokk elérésű eszköz, fájlként látjuk a tartalmat, kivéve NOR Flash).

      Méret:
        - n*1 Mbyte (Flash) – n*100 Tbyte.

Sebesség:
- Max. és random access más.
- Olvasás és írás eltérő.
- Max.: n*10 Mbyte/s (olcsó pendrive) – n*1 Gbyte/s (RAID).
- Random: lényegesen kevesebb.
- Késleltetés: n*10 ns (Flash) – kb. 10 ms

Külső tár

Biztonsági másolat

Címzés

Fajtái

Logikai cím

A központi egység (CPU) generálja a folyamat futása közben.
Logikai címtartomány (logical address space): Egy adott folyamathoz tartozó logikai címek összessége.

Virtuális cím

A logikai címet virtuális címnek (virtual address) hívjuk.
Ebben az esetben a futási idejű leképzést az MMU valósítja meg.

Fizikai cím

Egy adott memória elem címe, ahogy az a fizikai memória buszon megadásra kerül a memória vezérlő által.
Fizikai címtartomány (physical address space): Egy adott folyamathoz tartozó fizikai címek összessége.

Címképzés ideje

Fordításkor (compile/link time)

Ismert a betöltés helye.
- Abszolút címzés.
- A program fizikai címeket tartalmaz.
  - Pl. Egyszerű beágyazott rendszerek firmware-je, BIOS/EFI és OS kernel legalapvetőbb része, DOS *.com programok.

Betöltéskor (load time)

Áthelyezhető kód (relocatable code).
- Betöltés során oldódik fel a leképzés.
- A program fizikai címeket használ.

Futás közben (execution time)

A folyamat számára transzparens módon más fizikai címterületre kerülhet.
Speciális HW szükséges ehhez (MMU).
- Megfelelő sebességű végrehajtáshoz.

A folyamat nem látja (láthatja), hogy milyen fizikai címeken található a memória, amit elér.
- Mindig logikai/virtuális címeket használ.

Dinamikus betöltés (dynamic loading)

Bizonyos funkciók nem töltődnek be, amíg nem használják őket.
- Gyors program indulás.
- Alacsonyabb memória használat.
- A program feladata a dinamikus betöltés megvalósítása.
  - Az operációs rendszer nem tud róla, nem támogatja azt.

Dinamikus kapcsolatszerkesztés (dynamic linking)

A dinamikus betöltés operációs rendszer támogatással.
- Dynamically linked/loaded library (Windows *.dll).
- Shared Object (UNIX/Linux *.so).
- A dinamikusan beszerkesztett programkönyvtárak több program számára is elérhetőek (code sharing).

Megvalósítás:
- A program csak egy csonkot (stub) tartalmaz.
- A csonk feladata a dll/so megtalálása vagy betöltése az OS felhasználásával.
- Egy adott funkciójú dll/so több eltérő verzióban is jelen lehet a rendszerben.

- **Problémák**
  - ***Külső tördelődés (external fragmentation)(ábrán van magyarázva, kicsit necc leírni)***
  - ***Belső tördelődés (internal fragmentation)***

Memóriaszervezés módszerei

- **Változó méretű partíciók**
  - **Bázis + méret**
  - **Foglalási stratégiák**
    - **First fit**
      A tár elejéről indulva az első elégséges méretű területet allokálja.
    - **Next fit**
      Nem a tár elején kezdi a keresést, hanem a legutolsóra allokált terület után.
    - **Best fit**
      A legkisebb szabad hellyel járó helyre teszzük be.
      Minimális külső tördelődés.
    - **Worst fit**
      A legnagyobb szabad hellyel járó helyre tesszük be.
      Talán a maradék helyre még befér majd valami.
- **Tárcsere (swapping)**
  - **Teljes folyamat kiírása lemezre**
    Ha nincs futó állapotban és nincs folyamatban lévő I/O művelet (DMA sem a területre).
    - Kivéve az OS területén lefoglalt pufferekbe/pufferekből végzett I/O.

    Ha a címképzés futási időben történik:
    - Akár még más fizikai címre is kerülhet a visszatöltött folyamat.

    A tárcserével összekapcsolt kontextus váltás nagyon időigényes (a háttértár lassú a memóriához képest).
    - A teljes folyamatot ki kell írni majd visszaolvasni!
- **Szegmensszervezés**
  - **Különböző méretű szegmensek**
    A programozó: Nem egy összefüggő lineáris memória területben gondolkozik...
  - **Címzés: segment name + offset**
    A logikai címtartomány szegmensekre van osztva.
    A programozó egy szegmenst (segment name/ID) és azon belül egy szegmens ofszetet (segment offset) 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!
  - **Szegmens tábla**
    A szegmensek mérete adott, azt is tárolni kell!
    - Adott szegmensen kívüli címzés esetén „segment overflow fault” kivétel.

    A szegmensek folytonosan tárolhatók, belső tördelődés nincs (meg lehet csinálni tördelődés nélkül).
    A szegmenseket a fordító és linker állítja össze, és adja meg a programot betöltő loader-nek.

- **Lapszervezés**
  - **Felosztás**
    - **Fizikai memória: keret (frame)**
      A fizikai memóriát keretekre (frame) osztjuk.
    - **Virtuális memória: lap (page)**
      A logikai memóriát lapokra (page) osztjuk.
  - **Laptábla (page table)**
    A lap szám a laptábla indexelésére szolgál
    - **Címtranszformáció (ábra)**
    - **Többféle lapméret**
    - **Többszintű laptáblák**
      Keresés:
      - A lap- és kerettábla nagy lehet a modern OS-ekben:
        - 4Kbyte-os lapok esetén 32 bites címzésnél 220 bejegyzés van a laptáblában, ami 4MByte memóriát igényel minimum.
        - Többféle lapméret támogatása (a folyamat memória igényének megfelelően).
      - Megoldás:
        - Többszintű laptáblák (hierarchical paging).
        - Hash-elt laptáblák (hashed page table)
        - Inverted page table
      - A keresés a laptáblában lassú:
        - 2x annyi idő (laptábla indexelés és olvasás majd adatelérés).
        - Asszociatív lekérdezés (Translation look-aside buffer, TLB) kombinálva a laptábla használatával.
  - **Translation Lookaside Buffer (TLB)**
    - **Találati arány (hit ratio)**
      A TLB méretétől és a végrehajtott kódtól függ.
      Hány lapra és milyen sorrendben hivatkozunk a lapokra.
      A lapok milyen arányban férnek be a TLB-be, és milyen a TLB frissítési algoritmusa (új bejegyzés milyen régi bejegyzés helyére kerül).
      Tipikusan 80-90% körül van.
    - **TLB ürítése**
  - **Kiegészítő bitek**
    - **Read/read-write, execute bit**
    - **Valid/invalid**
      A HW ellenőrzi ezeket, ha a szabályokkal ellentétes használatot talál, akkor kivételt generál, amit az OS kezel.
  - **Osztott lapok**
    Közös kód lapok
- **Szegmens és lapszervezés együtt**

Egyes rendszerek (pl. PC) mind a kettőt támogatják.
A szegmensszervezést minimálisan használják a x86 HW-ét támogató OS-ek.
- Pl. Linux esetén 6 szegmens: kernel kód/adat, user kód/adat, Task state segment, default local descriptor table.

A lapozás viszont alapvető CPU szolgáltatás.
- x86 HW esetén 4KByte (2 szintű tábla) és 4Mbyte-os lapok (1 szintű tábla).
- A Linux 3 szintű laptáblát használ, ebből a középső üres x86 HW esetén.

- **logical, linear, physical address**

Virtuális tárkezelés (virtual memory)

- **Komplex megoldás (virtuális címek, lapszervezés...)**

  A korábban ismertetett memória menedzsment módszerek alapján kidolgozott komplex memória menedzsment megoldás a virtuális tárkezelés.
  A teljes folyamat nem szükséges annak a végrehajtásához, többnyire az aktuális utasítás számláló „környezete”, és az éppen használt adatszerkezetek elégségesek (lokalitás).
  A folyamatok kódrészleteinek nagy részét soha nem hajtjuk végre vagy nagyon ritkán hajtjuk végre (error handling, software/feature bloat).
  A folyamatok elindulásához nem szükséges a teljes program tényleges betöltése.
  Egyes kódrészletek, erőforrások megoszthatóak a folyamatok között (pl. ugyanazt a kódot, adatot használják).
  A párhuzamosan futó folyamatok egyes, már használt kódrészleteire sokáig vagy akár soha többé nincs szükségünk, míg más folyamatoknak nem elég a memória.
- **Fizikai memóriánál nagyobb címtér használata**

  Virtuális memória, a programozónak nem kell foglalkoznia a rendelkezésre álló memóriával.
  Vannak következményei: komplexitás és sebesség.
  Ha a folyamataink csak a tényleg szükséges fizikai memóriát tartják a fizikai memóriában, több folyamatot tudunk egy időben betölteni.
  A programok gyorsabban induljanak el, csak a szükséges kódrészleteket töltse be az operációs rendszer.
  Képesek legyenek osztozni a közös kód és adatszerkezeteken, erőforrásokon.
- **Lapozófájl (page vagy swap file)**

  Az alapja a lapozás.
  - A folytonos virtuális címteret egy tábla (memory map,page table, laptábla) képzi le.

  Nem direkt módon fizikai memóriára történik a leképzés, hanem:
  - Részben fizikai memóriára.
  - Részben a permanens táron (HDD, Flash tár) kialakított speciális területre.
    - Pagefile (Windows), vagy swap file (UNIX/Linux), a UNIX/Linux esetén a név történelmi örökség, nem swappingről van szó.

  A folyamat egy összefüggő virtuális címteret lát!
- **Kiegészítő bitek**
  - **Modified vagy dirty**

    Módosítás nyilvántartása (modified/dirty bit):
    - Minden laphoz tartozik egy HW által kezelt bit (pl. a laptáblában).
      - Betöltéskor törlik, módosításkor beállítják.
  - **Referenced vagy used**

    Hivatkozások nyilvántartása (referenced/used bit):
    - OS adott időnként és/vagy adott eseményekre törli.
    - Használat esetén beállítják.
  - **Valid/invalid bit**

    Ha egy éppen használt (pl. egy utasításban) virtuális memória lapon található cím bent van a fizikai memóriában, akkor a kód végrehajtható.
    Ha nincs benn (valid/invalid bit)?
    - **Laphiba („page fault”) kivételt generál az MMU.**

      Érvényes, de éppen nem fizikai memóriában lévő lap.
      Nem hiba, normális működés!
      Az operációs rendszer ezt kezeli, lehetőségek:
      - HDD-re ki van írva: be kell hozni a pagefile-ból.
      - Soha nem lett betöltve: be kell tölteni.
        - Kérdés: Hová, főleg ha tele van a fizikai memória?

      Az OS visszaadhatja a vezérlést a megszakított folyamatnak.
      A hozzárendelés során a folyamat passzívan várakozik.

- **Lapozási stratégia**
  - **Igény szerint (demand paging)**

    Csak laphiba esetén, és csak a laphibát megszüntető lapot hozza be a fizikai memóriába.
    Csak a szükséges lapok vannak a fizikai memóriában.
    Új lapra vonatkozó hivatkozás mindig hosszú várakozást eredményez (be kell hozni).
  - **Előretekintő lapozás (anticipatory paging)**

    Előre tekintve (becslés) az OS megpróbálja kitalálni, hogy mely lapokra lesz szükség, és azokat is behozza.
    Feltétel: szabad erőforrások (CPU, HDD, fizikai memória).
    Ha a behozott lapokra tényleg szükség lesz (sikeres a becslés), akkor csökken a laphibák száma.
    Ha nem, akkor feleslegesen használunk erőforrásokat.
- **Lapcsere**
  - **Stratégiák**
    - **Optimális**

      Előre néz, és teljes információval rendelkezik a jövőben használt lapokról.
      - Nem realizálható, de jó összehasonlítási alap.
    - **Legrégebbi lap (FIFO)**

      Egyszerű, a múlt alapján dönt, hátranéz.
      Azokat a lapokat is lecseréli, amelyeket a folyamatok gyakran használnak.
      - Nem nézi a tényleges használatot.
      - Csak a behozás sorrendjét.
      - **Bélády anomália**

        Növeljük a folyamatnak adott memória keretek számát (fizikai memória).
        Nő a laphibák gyakorisága.
        Anomális: Nem úgy működik, ahogy várnánk...
    - **Újabb esély (Second chance)**

      A sor elején lévő lapot csak akkor cserélik le, ha arra nem hivatkoztak (referenced/used bit).
      - Bonyolultabb, mint a FIFO, de alapvetően egyszerű algoritmus.
      - Hátranéz, és a behozás sorrendje és a használat alapján dönt.
      - A reference bit-et törli:
        - Ezért újabb esély a neve.
        - Egyébként végtelen ciklusba kerülhetne.
    - **Legrégebben nem használt (Least recently used, LRU)**

      Bonyolult, de jól közelíti az optimális algoritmust.
      - A lokalitás miatt jó a közelítés.

      Hátrafelé néz.
      Megvalósítás:
      - Számláló: Minden laphoz egy „last used” timestamp kerül.
      - Láncolt lista: A lista végére kerül a legutoljára használt.
      - Kétdimenziós tömb: NxN-es mátrix, ahol N a lapok száma.

      Sokszor a közelítéseit szokták használni.
    - **Legkevésbé használt (Least frequently used, LFU)**

      A közelmúltban gyakran használt lapokat a lokalitás miatt nagy valószínűséggel újra fogjuk
      használni.
      - A ritkán használtakat kis valószínűséggel fogjuk használni.
      - Az R bit értékét hozzáadja időnként egy laphoz tartozó számlálóhoz, és törli az R bitet.
      - A kisebb számláló értéket tartalmazó lapokat cseréljük le.
      - Az algoritmus nem felejt...
      - Az algoritmus a frissen behozott lapokat fogja lecserélni (0 vagy kicsi számláló érték).

        - Azokat be kell fagyasztani a memóriába egy időre.
    - **Utóbbi időben nem használt (Not recently used, NRU)**

      A hivatkozott és módosított biteket is használja.
      R törölhető, M-et viszont meg kell őrizni.
      Prioritást rendel a lapokhoz R és M alapján.
      - 0. prioritás: R=0, M=0 (legalacsonyabb)
      - 1. prioritás: R=0, M=1
      - 2. prioritás: R=1, M=0
      - 3. prioritás: R=1, M=1 (legmagasabb)

      Mindig a legkisebb prioritású csoportból választ, ahol még van lap.
  - **Kiegészítés**
    - **Lokális/globális**

      Globális lapcsere: A teljes fizikai memória potenciálisan lecserélhető.
      Lokális lapcsere: A folyamat által használt fizikai memória lapok között történik a csere.
    - **Lapok tárba fagyasztása (lock bit)**

      I/O műveletek hivatkoznak rá.
      - Ott fizikai címet kell használnunk!
      - Lehet kernel szinten pufferelni, és akkor ez is megkerülhető.

      LFU algoritmus frissen behozott lapjai (nem voltak használva, elsőrangú áldozatok).
- **Teljesítmény**
  - **Trashing**

    A gyakori laphibák által okozott rendszer teljesítmény csökkenést vergődésnek (trashing) nevezzük.
    - A laphiba kezelése során újabb laphiba jelenik meg.
    - A laphibák felgyűlnek a háttértár várakozási sorában.
    - A CPU a laphibák megoldására vár.
    - A hosszú távú ütemező (ha van), ezt I/O intenzív folyamatként is értelmezheti, újabb folyamatokat beengedve a rendszerbe...
  - **Page Falt Frequency**

    Cél: alacsony laphiba gyakoriság (Page Fault Frequency, PFF).
    Egy laphiba kezelése során ne jöjjön létre újabb laphiba:
    A háttértár várakozási sora csak csökkenhet...
  - **Lokalitás**

    Statisztika: Egy időintervallumban a folyamatok a címtartományuk csak egy kis részét használják.
    - Időbeli.
    - Térbéli.

    Vergődés:
    - Megfelelő számú fizikai memória keret allokálása.
      - Nincs vergődés.
      - Laphibák a lokalitás váltásakor.
    - Ennél kisebb: vergődés
    - A vergődés elkerülhető a multiprogramozás fokát közel optimálisan tartva.
    - Vergődés: magas PFF érték...
  - **Munkahalmaz (working set)**

    A lokalitáson alapul.
    A folyamat azon lapjainak halmaza, amelyekre egy időintervallumban (munkahalmaz-ablak) a folyamat hivatkozik.
    A munkahalmaz alapján megadható az adott folyamat munkahalmaz mérete (WSS).
    A futó folyamatok teljes fizikai memória keret igénye számítható (D).
    Az OS méri a WSS-t folyamatonként.
    - Ha van szabad memória keret:
      - Akkor az igények kielégíthetőek.
      - Új folyamatok engedhetőek be a rendszerbe.

- Ha nincs szabad memóriakeret:
  - Akkor ki kell választani egy „áldozat” folyamatot.
  - Azt fel kell függeszteni (suspend, tényleges swap out).
  - Az áldozat folyamat fizikai memória kereteit fel lehet használni.

PFF alapú optimalizáció

- Folyamatonként egyszerűen mérhető a PFF.
  - Alacsony: túl sok fizikai memória keret.
  - Magas: túl kevés memória keret.
- Felső és alsó PFF határérték megadása.
  - Felső határérték túllépés: kap egy fizikai memória keretet.
  - Alsó határérték túllépése: egy fizikai memória keret elvonása.
    - Esetleg csak akkor, ha nincs szabad fizikai memória keret a rendszerben.
    - Ha nincs fizikai memória: egy folyamat felfüggesztése

Permanens tár kezelése

Tulajdonságai:

- **Nagy**
- **Lassú: Adatátviteli sebesség, késleltetés**
- **Nem felejtő**
- **Blokk alapú szervezés:**

  Az OS ennél kisebb egységekben nem gondolkozik. Blokkokként olvasható, írható, törölhető. (Kivéve egyes beágyazott rendszereket.)

Absztrakciós szintek

- **Logikai fájlrendszer**
  - **OS specifikus**
  - **Metaadatok tárolása**
  - **Elemek**
    - **Fájl (file)**
    - **Könyvtár (directory/folder)**
      - **symbolic, hard link**
    - **Kötet (volume/drive)**
  - **Adatvesztés**
    - **Konzisztencia ellenőrzés**
    - **Tranzakció orientált fájlrendszer**
  - **Elterjedt fálrendszerek**
    - **FAT**
    - **NTFS**
    - **EXT2, EXT3, EXT4**

      EXT3 = EXT2 + Journaling
    - **CD-ROM/DVD fájlrendszerek**
- **Fájlrendszer leképezés**
  - **Logikai blokkok leképzése fizikai blokkokra (allocation)**

        - **Folytonos allokáció (contiguousallocation)**
        - **Láncolt listás allokáció (linked allocation)**
        - **Indexelt tárolás (indexed allocation)**
    - **Üres helyek menedzselése (free-space management)**
        - **Bit vektor (bit vector)**
        - **Láncolt lista (linked list)**
        - **Szabad helyek csoportjaink listája (Grouping)**
        - **Számlálás (Counting)**
        - **Egybefüggő szabad területek nyilvántartása (Spacemaps)**
- **Alacsony szintű fájlrendszer**
    - **Fizikai diszk blokk írása és olvasása**
    - **Cachelés**
        - **Puffer cache (buffer cache)**
        - **Egységes puffer cache (unified buffer cache)**
        - **Egységes virtuális memória (unified virtual memory)**
- **Eszközkezelő:**
  I/O control, eszköz meghajtó
- **Eszközök**
    - **HDD**
        - **Fej, tányér, sáv, szektor**
        - **Sebesség**
            - **Változó**
            - **Optimalizáció**
                - **Diszk ütemezés**
                - **HDD szintjén**
                - **OS szintjén**
                - **Prefetch**
            - **Többszintű cache**
    - **NAND flash**
        - **Olvasás gyors**
        - **Írás (törlés) problémás**
    - **Eszköz csatlakoztatása**
        - **Host-Attached Storage**
            - **Direkt: SATA, SCSI, SAS…**
            - **Indirekt**
                - **USB, FireWire**
                - **RAID**

- **Hálózati tárolók eszközök (Storage-Area Networks, SAN)**
  - **Blokk szintű megoldás**
  - **Fibre Channel, iSCSI**
- **Network-Attached Storage (NAS)**
  - **Fájl szintű megoldás**
  - **Protokoll**
    - **NFS**
    - **SMB/CIFS**

- **Redundant Array of Inexpensive Disks (RAID)**
  - **Több merevlemezt együtt kelezni**
    - **Gyorsabb és/vagy megbízhatóbb**
  - **Vezérlő**
    - **Hardver**
    - **Szoftver**
  - **RAID szintek**
    - **RAID 0 (striped disks):**
      Több diszk egybe, és csíkozás. Teljes tár a diszkek kapacitásának összege.
    - **RAID 1 (mirroring):**
      Tükrözés. Teljes tár a kisebb diszk mérete.
    - **RAID 5 (block interleaved distributed paritity):**
      Több diszk egybe, csíkozás + paritás. Legalább 3 diszk. A teljes tár egy diszk kivételével (paritás miatt) az összeg.
    - **RAID 6 (block interleaved dual distributed parity):**
      U.A. csak két paritás, így a teljes tár két diszk kivételével számít.
    - **Kombinált:**
      RAID 1+0 (tükrözésen csíkozás), RAID 0+1 (csíkozáson tükrözés) - 4 lemezes (2-2).

Felhasználó- és jogosultságkezelés

Biztonság (security)

- **Alapfogalmak**
  - **Bizalmasság (Confidentiality)**
  - **Sértetlenség (Integrity)**
  - **Rendelkezésre állás (Availability)**
- **Eszközök**
  - **Kriptográfia**
  - **Platform szintű behatolás elleni védelem**
  - **Hálózati behatolás elleni védelem**
  - **Redundancia, újrakonfigurálás**
  - **Hitelesítés, engedélyezés**

Hitelesítés (authentication)

Mindenféle szolgáltatás esetén szükség van rá
- Hálózati és egy operációs rendszeren belüliek között is

Hitelesítési protokollok kellenek
Hitelesítés három szinten kerülhet elő:
- Ember és gép közötti interakció
- Gép és gép között valamilyen hálózaton át
- Gépen belül futó alkalmazások valamint az OS között

- **Alapja**
  - **Amit tud (pl. jelszó)**
  - **Amije van (pl. belépőkártya)**
  - **Ami ő (pl. ujjlenyomat)**
- **Felhasználói fiók**
  - **Attribútumok (ID, login név, jelszó…)**
  - **UNIX/Linux felhasználó (UID, passwd, shadow…)**

Engedélyezés (authorization)

- **Általános séma**
  - **Szereplő (Actor)**

    A rendszerben a szereplőt egy adatszerkezet reprezentálja
    A szereplők műveleteket kezdeményeznek
  - **Biztonsági szabályzat (Policy)**
  - **Védett objektum (Protected object)**
- **Hozzáférés végrehajtása**
  - **Művelet + kontextus**

    A műveletek kontextusa tartalmazza a szereplő azonosítóját, a célobjektumot és az elvégzendő művelet fajtáját

- **Jogosultsági döntő (decision point)**
  A jogosultsági döntőkomponens kiértékeli kontextust és engedélyezi vagy megtiltja a műveletet
- **Jogosultság végrehajtó (enforcement point)**
  A jogosultsági végrehajtókomponens biztosítja, hogy a döntő által hozott döntés érvényre jusson

- **Jogosultságkezelés fajtái**
  - **Kötelezőség**
    - **Kötelező (Mandatory)**
      csak központi jogosultság osztás
      felhasználók nem módosíthatják a házirendet
    - **Belátás szerint (Discretionary)**
      megfelelő jogú felhasználó tovább oszthatja a jogokat
  - **Szint**
    - **Rendszer**
    - **Erőforrás**
  - **Típus**
    - **Integritási szintek**
      Objektumok címkézése
      - alacsony, közepes, magas… integritási szint

      Ellenőrzés:
      - alacsonyabb szintű felhasználó nem olvashat/írhat magasabb szintű objektumot
    - **Hozzáférési listák**
      - **Hozzáférési maszk (access mask)**
        A hozzáférési maszk (accessmask) tartalmazza, hogy pontosan milyen műveletekre vonatkozik az engedély
      - **Role-based Access Control (RBAC)**
        Szerep alapú hozzáférés-vezérlés
        A szerep fogalom hierarchikus szereplő csoportosítási lehetőséget ad.
      - **Hierarchikus objektumok (öröklés)**

- **Engedélyezés Linuxon**
  - **Felhasználó, csoport**
  - **Tulajdonos (owner)**
  - **Jogok: 3x3 bit**

Virtualizáció

Erőforrás tényleges fizikai tulajdonságainak elrejtése a felhasználója elől, pl. egy erőforrást több logikaiként felajánlani, több fizikai erőforrást összefogni egybe…

Virtualizáció fajtái

- **Számítógép**
  Platform virtualizáció: teljes számítógép virtualizálása, egy gépen több OS futtatása
  Elnevezés még: szerver, számítógép, hardver virtualizáció..
  Elemek:
  - Gazda gép (host machine) = fizikai gép
  - Vendég gép (guest machine) = virtuális gép

- Virtual Machine Monitor (VMM): a virtuális gépeket kezelő program

Miért jó a platform virtualizáció?

- Tesztrendszer kiépítése
- HW konszolidáció
- Régi rendszerek (legacy systems)
- On-demand architektúra
- Rendelkezésre állás, katasztrófa védelem
- Hordozható alkalmazások

Ezzel foglalkozunk elsősorban

- **Operációs rendszer szintű**

  elkülönített futási környezet kialakítása

- ***Alkalmazás***

  alkalmazások egy magukat látják és a fájlrendszert tudják olvasni, egyébiránt azt hiszik, hogy egy szűz operációs rendszeren vannak?
  Pl. Vmware ThinApp, PortableApps, etc?

- ***Megjelenítés***
  - több asztal, távoli asztali kapcsolat?
- **...**

Virtuális gépek fajtái

- **Folyamat VM**

  A process VM is a virtual platform that executes an individual process. This type of VM exists solely to support the process; it is created when the process is created and terminates when the process terminates.
  Magyarán egy program futását virtualizálja, ez a feladata, ha a program befejeződik, akkor a virtualizáció is

  - ***Multiprogramozott OS***

    Nem az egész OS-t virtualizáljuk, hanem az OS által nyújtott környezetet, így pl. Linux alatt is lehet Windowsos programot futtatni Winenal?

  - ***Futtatókörnyezetek (Java, .NET)***

    Maga a környezet, amiben a program íródott egy virtualizált környezet, így ha csak a futtatókörnyezet kínálta (bőséges) funkciókat használjuk, akkor a programunk platformfüggetlen és virtualizálható lesz?

- **Rendszer VM**

  „A system VM provides a complete, persistent system environment that supports an operating system along with its many user processes. It provides the guest operating system with access to virtual hardware resources, including networking, I/O, and perhaps a graphical user interface along with a processor and memory.
  Magyarán egy egész gépet levirtualizál, mintha 1 helyett N gépünk lenne.

  - **Platform/HW vitualizáció**
    - **Hosted**

      Fő komponense:
      VMM – Virtual Machine Monitor
      Jellemzően desktop megoldások: VMware Workstation, Server, Player, Sun VirtualBox, MS VirtualPC, KVM, UML
      Lényeg: VMM az oprendszerre települ, nem fér közvetlenül hozzá a hardverhez, csak az OSen keresztül

    - **Bare-metal**

      VMM – Virtual Machine Monitor ***Hypervisor***
      (Jellemzően) szerver megoldások: VMware ESX Server, Xen Enterprise, MS Hyper-V
      Lényeg: a Hypervisor egyből a hardvert használja, nincs köztes OS, minden egyéb OS guest OS

- **Emuláció**

Elméleti alapok

- **Követelmények**
  - **Azonosság**

    a virtuális gépen futtatott programok ugyanazt az eredményt adják
  - **Biztonságosság**

    a VMM kezeli az összes hardver erőforrást
    Vendég gépektől védeni kell a rendszert
    Pl.: HLT (Halt) utasítás kiadása
    Elvárt: csak a vendég álljon le
    Ha végrehajtanánk: mindenki leáll
    Megoldás: VMM felügyelje a vendég utasításait
    Privilegizált utasítások kezelése
  - **Hatékonyság**

    a vendég gép utasításainak nagy része beavatkozás nélkül fut
- **CPU virtualizáció**
  - **Tiszta emuláció**

    Teljes virtuális HW állapot eltárolása az emulátorban
    (regiszterek, flag-ek)
    Minden utasítást megvizsgál a VMM
    Alkalmazza a hatását az emulátorban, átalakítja a hívást, végrehajtja
    Előny: Más CPU is emulálható
    Hátrány: Lassú (NAGYON LASSÚ :))
  - **Trap and emulate**
    - **Trap**

      hardveres kivételkezelő rutin ami után a végrehajtás folytatódhat
    - **Deprivilegizálás**

      A nem privilegizált utasítások közvetlenül a valós CPU-n hajtódnak végre
      A privilegizált vagy érzékeny műveletek trap-et váltanak ki, és a VMM veszi át a végrehajtást
      HW támogatás:
      védelmi szintek (az x86-on ring, 4 db.)
      virtuális gép alacsony védelmi szinten fut
      privilegizált utasítások nem megfelelő szinten kiadva trap-et okoznak
  - **x86 megoldások**

    Egyes architektúrák könnyen virtualizálhatóak, az x86 nem ilyen. ~250 utasításból 17 megsérti a klasszikus feltételeket, pl. POPF utasítás: EFLAGS regisztert módosítja Ha nem ring 0-n adjuk ki, akkor nem ír felül bizonyos biteket, és nem is dob kivételt Privilegizált állapot kiolvasható: Virtuális gép a CS szegmens regisztert olvasva megtudhatja, hogy virtualizált
    - **Szoftveres: bináris átírás (binary translation)**

      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 módosítása. Problémás utasítások lecserélése
      - **Hypercall**

        VMM-et hívja közvetlen

- **Hardveres virtualizáció (VT-x, AMD-V)**

  ~2005: Intel Virtualization Technology (VT-x) és AMD AMD-V HW-es támogatás: root mode, VMCS Utasítások, pl.: VMCALL, VMLAUNCH

  - **Root mode**

    VMM Root Modeban fut és a valódi gépet használja, míg a Virtualizált OSek Non-Root modeban futnak és HW-esen támogatott, de nem valódi Ring-eket látnak (de így az OS pl. továbbra is a Ring0-ban hiheti magát)

- **Memória virtualizáció**
  - **Probléma: kétszeres címfordítás**

    1) Vendég: virtuális memória
    - 1 <-> 2 között : Vendég laptáblák

    2) Vendég: „fizikai” memória
    - 2 <-> 3 között : VM allokációs laptáblák

    3) Gazda: fizikai memória

    Megoldás:

    - **Árnyék laptáblák (shadows page table)**

      1 <-> 3 közötti közvetlen kapcsolat

  - **Megoldások**

    - **Szoftveres**

      GOND: szinkronizálás

    - **Paravirtualizáció**

      Vendég OS forrásának módosítása
      Ha a vendég módosítja a laptábláit, akkor értesítse a VMM-et is erről

    - **Hardveres (AMD RVI, Intel EPT)**

      HW támogatás az újabb CPU-kban
      (AMD Rapid Virtualization Indexing , Intel Extended Page Tables)
      Beágyazott laptábla (Nested page table)
      vendég fizikai -> gazda fizikai leképezés eltárolása
      cím leképezési rutin ezt is bejárja
      TLB bejegyzések azonosítóval ellátása
      Nagy teljesítménynövekedés

- **I/O eszközök virtualizációja**
  - **Szoftveres**

    - **Létező eszköz emulálása**

      A teljes, valós kommunikáció emulálása
      Lassú, minden kommunikáció a VMMen keresztül

  - **Paravirtualizáció**

    - **Paravirtualizált meghajtó csomag**

      Speciális csomag telepítése a vendégben
      (VMware Tools, Virtual PC Additions)
      Külön meghajtó program a virtualizált gépen, ami egyszerűsített hívások, adatstruktúrák megosztása lévén tud kommunikálni a fizika hardverrel

  - **Hardveres**

    Intel VT-d, AMD IOMMU
    PCI szabvány kiegészítése: I/O Virtualization (IOV)

    - **Eszközök megosztása**

      Több virtuális gép tudja egyszerre használni

    - **Eszköz közvetlen hozzárendelése**

      Csak egy virtuális gép kap rá kizárólagos jogot

uC/OS-II beágyazott OS

Története

uC/OS-II könyv (teljes működés leírása)

Open Source

- **Egyszerű, megismerhető, működése megérthető**
- **Oktatási célra ingyenes a licensz**
- **Sok HW platformon / portolható**

Tulajdonságok

- **Forráskódban rendelkezésre áll**
- **Hordozható (processzor függő részek külön)**
- **Skálázható**
- **Multi-tasking**
- **preemptív ütemezőP**
- **Determinisztikus futási idő**
- **Minden taszknak különböző méretű lehet a stack-je**
- **Rendszer szolgáltatások: mailbox, queue, semaphore**
- **Fix méretű memória partíció**
- **Idő kezelésére alkalmas szolgáltatások.**
- **Interrupt management (255 szintű egymásbaágyazhatóság)**
- **Robusztus és megbízható**
- **Kiegészítő csomagok**

Kiegészítők

- **TCP-IP (Protocol Stack)**
- **FS (Embedded File System)**
- **GUI (Embedded Graphical User Interface)**
- **USB Device (Universal Serial Bus Device Stack)**
- **USB Host (Universal Serial Bus Host Stack)**
- **FL (Flash Loader)**
- **Modbus (Embedded Modbus Stack)**
- **CAN (CAN Protocol Stack)**
- **BuildingBlocks (Embedded Software Components)**
- **Probe (Real-Time Monitoring)**

Felépítése

- **CPU specifikus (HAL)**
- **Processzor független kernel**
- **Alkalmazás specifikus kernel konfiguráció (C include file-ok)**
- **Alkalmazás**

uC/OS-II taszk állapotok

- **Állapotok**
  - Két speciális állapot: – DORMANT: „szunnyadó”, akkor van ebben az állapotban a taszk, amikor a memóriában ugyan megtalálható, de az ütemező hatáskörében nincs benne. – ISR: a taszkot megszakította egy interrupt rutin. Ha a rutin mellékhatásaként egy magasabb prioritású taszk válik futásra késszé, akkor a rutin végeztével az kerül a RUNNIG állapotba, és a megszakított taszk pedig a WAITING-be
- **Állapotátmenetek**
- **Függvények**

Ütemező működése

- **2D bitmap struktúra**

  az egyes bitek reprezentálják a taszkokat. Ha egy bit 1, a hozzá tartozó taszk futásra kész. A bitek prioritást is jelentenek.
  Gyors beszúrás, ami független a futásra kész taszkok számától.
- **Prioritásos, 1 taszk 1 prioritási szinten**

  A taszkoknak egyedi prioritással kell rendelkezniük (azaz a round-robin / time slice ütemezés nem lehetséges)
  Szükség van egy lookup táblázatra előre számított értékekkel a bitminta → legmagasabb prioritású futásra kész taszk megfeleltetéshez.
- **Kontextus váltás**

  1. az aktuális környezet mentése:
     - • regiszterek mentése
     - • veremmutató mentése
  2. az új környezet visszaállítása:
     - • veremmutató visszaállítása
     - • regiszterek visszaállítása