Operációs Rendszerek

Az előadások anyag magyarázatokkal, kiegészítésekkel – félkész állapot.

A jegyzetet Umann Kristóf és Csonka Szilvia írta. A jegyzet első számú forrása az előadásdiák, valamint nagyban a következő pdf struktúráját követi: `http://people.inf.elte.hu/memsaai/000` . Azon túli kiegészítések forrása az internet (zömmel Wikipédia).

1. Előadás

A tárgy célja

- Az operációs rendszerek szerepének megvilágítása, fontosságának kiemelése.
- Az operációs rendszerek tervezése sprán felmerülő kérdések, problémák, megoldások ismertetése.
- Az igényeknek leginkább megfelelő operációs rendszer megválasztása.
- Rendszer közeli hívások megismerése.

Előismeretek

- Számítógépes alapismeretek: unix shell script programozás.
- Programozási alapismeretek: C, C++ nyelv, alapvető algoritmusok ismerete.
- Programozási nyelvek I. (C++)

Féléves tematika

- Bevezetés, előzmények (számítógépes alapismeretek, számítógépek felépítése, API)
- Operációs rendszer fogalma, fejlődése, felhasználói felület.
- Fájlok, könyvtárak, lemezkezelés.
- Folyamatok, ütemezések.
- Bevitel - kivitel, erőforrások, holtpontok kezelése.
- Memóriakezelés.
- Real-Time Operációs rendszer jellemzők.
- Suse Linux Enterprise szerver esettanulmány.
- Windows Core, Azure, Felhő.

Fordítás

- usr/bin/gcc-4.8
  - Alapértelmezett C, C++ fordító.
  - .c esetén C mód.
  - .cpp esetén C++ mód.
- cc, gcc néven köünnyen elérhetők
- `Man gcc` → információk, segítség
- Fordítás:
  - `cc alma.c` → eredmény: `a.out`
  - `cc -o alma -Wall alma.c` → eredmény: alma, összes warning
- `/usr/bin/g++-4.8`
  - Alapértelmezett C++ fordító.
  - gpp névvel is elérhető

2. Előadás - Bevezetés

2.1. Visszatekintés

- Ahol a számítógépes alapismeretek befejeződött...
- Script programok: Shell script, PowerShell.
- Kliens-szerver mint gép: hardwares (HW) különbségek
- Kliens - szerver mint szolgáltatás: adminisztráció, softwares (SW) különbségek.

2.2. Számítógépek felépítése: hardveres (HW) oldal

Hardver (angolul: hardware) alatt a számítógép fizikailag megfogható részeinek összességét értjük. A számítógép működéséhez alapvetően hardver és szoftver szükséges, a kettő közötti kapcsolatot a firmware hozza létre, ami a hardverekbe a gyártók által „beépített" szoftvernek tekinthető.

Bevezetés

- Tárolt program, utasítások, adatok azonos módon (binárisan) helyezkednek el a memóriában.
- A vezérlő egység (CPU), aritmetikai-logikai egység (ALU - Arithmetic Logic Unit) az utasítások végrehajtását, alapvető aritmetikai műveleteket felügyelik.
  - *A CPU* (Central Processor Unit) más néven processzor ill. mikroprocesszor, a számítógép „agya", azon egysége, amely az utasítások értelmezését és végrehajtását vezérli.
  - *Az ALU* (Arithmetic Logic Unit) aritmetikai és logikai műveleteket elvégző digitális áramkör, a processzor (CPU) része.
    - Alapvető eleme a számítógép központi vezérlőegységének.
    - A processzorok, a CPU-k és a GPU-k a számítógép belsejében található, belülről nagyon erős és bonyolult ALU-k.
    - Egy egyszerű alkatrész is számos ALU-t tartalmazhat.
- Szükség van be- és kimenetek (I/O: Input/Output) kezelésére, mely a gép és a külvilág kapcsolatát biztosítja. (Pl.: kimenet: monitor, nyomtató; bemenet: billentyűzet, szkenner.)
- Ezen jellemzőket gyakran a Neumann elv elemeiként is ismerjük.

Alapvető elemek

- Processzor (CPU): a számítógép „agya", azon egysége, mely a programutasítások értelmezését és végrehajtását vezérli.
- Memória: A számítógépek legfontosabb erőforrása a processzor mellett a memória. A tárolóban található a végrehajtás alatt lévő program és a feldolgozásban használt adatok is.
- Priférіák:
  - A periféria egy olyan számítógépes hardver, amivel egy gazda számítógép képességeit bővíthetjük. A fogalom szűkebb értelemben használva azon eszközökre értendő, amelyek opcionális természetűek, szemben azokkal, melyekre vagy minden esetben igény van, vagy elengedhetetlen fogalmi alapkövetelmény jelenlétük.
  - A fogalmat eredetileg azokra az eszközökre alkalmazták, melyek külsőleg csatlakoztak a gazdagéphez, tipikusan egy számítógépes buszon keresztül, mint például az USB. Tipikus példa a joystick, nyomtató, és lapolvasó. Manapság ezeket kissé tautologikusan külső perifériának nevezik. Az olyan eszköz, mint például monitor és a lemezmeghajtó manapság azért nem számít perifériának, mert igazából nem opcionálisak, a videodigitalizáló kártya pedig azért nem, mert belső eszköz.
  - Az eszközök, perifériák működtetéséhez a számítógépnek speciális, úgynevezett eszközmeghajtó (driver) programra van szüksége, amely biztosítja az eszköz operációs rendszerhez való illesztését, ezen keresztül a lehetőségeknek megfelelően szabványos kezelését.

– Háttértár:
A háttértár olyan számítógépes hardverelem, mely nagy mennyiségű adatot képes tárolni, és azokat a számítógép kikapcsolása után is megőrzi. Erre azért van szükség, mert a számítógép műveleti memóriájában csak ideiglenesen lehet adatot tárolni, ennek tartalma a számítógép kikapcsolása után törlődik.

– Összekötő kapocs: Busz (sín, adat, cím , vezérlő):
A busz vagy sín (hosszabb elnevezéssel buszrendszer vagy sínrendszer) a számítógép-architektúrákban a számítógép olyan, jól definiált része, alrendszere, amely lehetővé teszi adatok vagy tápfeszültségek továbbítását a számítógépen belül vagy számítógépek, illetve a számítógép és a perifériák között. Eltérően a pont-pont kapcsolattól, a busz logikailag összekapcsol több perifériát ugyanazt a vezetékrendszert használva. Minden buszhoz számos csatlakozó tartozik, amelyek lehetővé teszik a kártyák, egységek vagy kábelek elektromos csatlakoztatását.

Processzor főbb részei

– *ALU*: (Arithmetic and Logical Unit – Aritmetikai és Logikai Egység). A processzor alapvető alkotórésze, ami alapvető matematikai és logikai műveleteket hajt végre.

– *AGU*: (Address Generation Unit). A címszámító egység, feladata a programutasításokban található címek leképezése a főtár fizikai címeire és a tárolóvédelmi hibák felismerése.

– *CU*: (Control Unit a. m. vezérlőegység vagy vezérlőáramkör). Ez szervezi, ütemezi a processzor egész munkáját. Például lehívja a memóriából a soron következő utasítást, értelmezi és végrehajtatja azt, majd meghatározza a következő utasítás címét.

– *Regiszter* (Register): A regiszter a processzorba beépített nagyon gyors elérésű, kis méretű memória.

– *Buszvezérlő*: A regisztert és más adattárolókat összekötő buszrendszert irányítja. A busz továbbítja az adatokat.

– *Cache*: A modern processzorok fontos része a cache (gyorsítótár). A cache a processzorba, vagy a processzor környezetébe integrált memória, ami a viszonylag lassú rendszermemória-elérést hivatott kiváltani azoknak a programrészeknek és adatoknak előzetes beolvasásával, amikre a végrehajtásnak közvetlenül szüksége lehet. A mai PC processzorok általában két gyorsítótárat használnak, egy kisebb (és gyorsabb) elsőszintű (L1) és egy nagyobb másodszintű (L2) cache-t. A gyorsítótár mérete ma már megabyte-os nagyságrendű.

Processzor utasítások

– Gyakorlatilag a rendszer minden eleme intelligens, de a kulcsszereplő a processzor.

– *Regiszterek*
A regiszter a processzorba beépített nagyon gyors elérésű, kis méretű memória. A regiszterek addig (ideiglenesen) tárolják az információkat, utasításokat, amíg a processzor dolgozik velük. A mai gépekben 32/64 bit méretű regiszterek vannak. A processzor adatbuszai mindig akkorák, amekkora a regiszterének a mérete, így egyszerre tudja az adatot betölteni ide. Például egy 32 bites regisztert egy 32 bites busz kapcsol össze a RAM-mal. A regiszterek között nem csak adattároló elemek vannak (bár végső soron mindegyik az), hanem a processzor működéséhez elengedhetetlenül szükséges számlálók, és jelzők is.

– *Utasításcsoportok*
  - Adatmozgató utasítások (regiszter - memória)
    - Az adatmozgató utasítások egy, vagy több byte átvitelére alkalmasak. RISC processzoroknál külön byte-os és külön szavas adatmozgató utasítások vannak, míg összetett utasításkészletû processzorok esetében, ahol az utasításhossz nem állandó, magában az utasításban kell megadni az átvinni kívánt adat méretét. Az átvitt adat az eredeti helyén is megmarad, tehát ez valójában csak adatmásolás.

    - A memória-regiszter, regiszter-memória közötti adatmozgató utasításokat tároló hivatkozású adatátviteli utasításnak hívják. Memória-memória közötti adatátvitelt közvetlenül nem használnak a processzorok. Az adatmozgató utasítások közé tartoznak az úgynevezett periféria utasítások is.
  - Ugró utasítások, abszolút - relatív
  - I/O port kezelés
  - Megszakításkezelés, stb.

Processzor védelmi szintek

- *Intel 80286* - minden utasítás egyenlő
  - Az Intel által kifejlesztett 16 bites CPU az x86 architektúrából. 1982. február 1-jén adták ki. A 8086/8088 után az első komolyabb lépés volt az Intel részéről (a 80186 típus sok újdonságot nem hozott).
  - Két üzemmódban működtethető:
    - valós mód (real mode)
    - védett mód (protected mode)
  - Először jelent meg a modern operációs rendszerek támogatása hardver szinten, nevezetesen a védett mód (protected mode).
- *Intel 80386* - 4 védelmi szint, ebből 2-őt használ, kernel mód (védett, protected mód) és felhasználói mód
  - Az Intel 80386, amely másként i386 vagy egyszerűen csak 386 néven is ismert, az Intel által kifejlesztett x86-os processzorcsalád első 32 bites tagja. 1985-ben mutatták be.
  - A valós és védett üzemmód mellett megjelent egy harmadik is, a virtuális üzemmód. *(Igen, itt a diák és a Wikipédia mást mondanak...)*
  - Védett üzemmódja megőrizte a 80286 által bevezetett üzemmódot, amit 16 bites protected mód néven találunk meg, de kiterjesztette azt 32 bitesre is, ahol lehetőség van akár 4 gigabájt méretű szegmensek és címtér használatára is.
- Például tipikusan védett módú utasítások a következők
  - Megszakítás kezelés
  - I/O port kezelés
  - Bizonyos memóriakezelés

Processzor utasítások használata

- Adatok, utasítások a memóriában, ezeket a CPU végrehajtja
- Adatmozgató utasítások:
  - Mov al, 'F'
  - Mov ah, 'T'
  - Mov bl, 'C'
  - Stb.
- *"Hol van itt az élvezet?"*
  - Hát ott, ha látom is az eredményt (FTC)...
  - Ha egy perifériát (pl. képernyő) elérek és azon megjelenítem az adatokat. *(Igen, a Wikipédia szerint manapság már a képernyő nem számít perifériának, annyira alapvető tartozéka a gépnek...)*

Megszakítások

A megszakítások nagyon fontos elemei a számítógépek működésének. Amikor a mikroprocesszornak egy eszközt, vagy folyamatot ki kell szolgálni, annak eredeti tevékenységét felfüggesztve, megszakítások lépnek életbe. *Létezik szoftveres, és hardveres megszakítás.*

- ***Hardveres megszakítás*** akkor következik be, amikor *egy eszköznek szüksége van az operációs rendszer figyelmére.* Ilyenkor a processzor erőforrását (processzoridőt) igénylő eszköz megszakítás-

kérést küld a processzornak. Amennyiben a megszakítás lehetséges, a kérést kezdeményező eszköz használhatja a processzort.
- ***Szoftveres megszakítás*** esetén *a főprogram futását egy alprogram szakítja meg.* Ebben az esetben a főprogram futásállapota elmentésre kerül, majd miután a megszakítást kérő program befejezte a műveletet a főprogram folytatja a futását a megszakítás előtti pozícióból. Példa erre, amikor a folyamat egy rendszerhívást tesz meg. Ekkor a futása félbeszakad, végbemegy a hívás, majd a folyamat futása folytatódik.
- Egy harmadik megszakítási típus lehet a ***csapdák (traps)***, melyek *hibás szoftverműködés esetén* lépnek fel (pl. nullával történő osztás).

Megszakítás maszkolása
Maszkolással egyes megszakításokat figyelmen kívül tud hagyni az operációs rendszer. Vannak azonban nem maszkolható megszakítások is (NMI, Non-maskable interrupt), pl. azok melyek memóriahiba, vagy tápfeszültség kimaradás esetén keletkeznek.

2.3. Számítógépek felépítése: szoftveres (SW) oldal

A szoftver (angol: software) alatt a legszűkebb értelemben elektronikus adatfeldolgozó berendezések (például számítógépek) memóriájában elhelyezkedő, azokat működtető programokat értünk.

Végrehajtási, felépítési szintek

- Logikai áramkörök
- CPU, mikroprogram, mikroarchitektúra szint
- Számítógép, hardver elemek gépi kódja
- *Operációs rendszer*
- Rendszeralkalmazások
  - Alacsony szintű, gépi kódú programok, meghajtók
  - Magas szintű nyelvek, programok
- Alkalmazások: felhasználói programok, pl. pasziánsz, stb.

A szoftverek funkciójuk szerint
A programvezérelt gépek célszerű működését a szoftverek több rétege biztosítja. Aszerint, hogy egy szoftver specifikusan mennyire inkább a gép puszta működtetését, avagy az ember által igényelt feladatmegoldást segíti elő, a következő funkcionális csoportokat különböztetjük meg:

- *Alapszoftver* vagy indítóprogram: a felhasználó által a legkevésbé manipulálható, a gép üzemszerű működését beállító program(ok), ide tartozik a firmware is.

- Rendszerszoftver: a gép és perifériái kommunikációját lebonyolító programok, beleértve a felhasználó oly mértékű kiszolgálását, amely lehetővé teszi a számára más szoftverek elkészítését és üzembe helyezését is.
  - Operációs rendszerek
  - Meghajtóprogramok (driverek)
  - Segédprogramok: fájlkezelők, szövegszerkesztők (editorok), tömörítők.
  - Fejlesztési környezetek: fordítóprogramok (compilerek), értelmezők (interpreterek) és futtatókörnyezetek, nyomkövetők és hibakeresők (debuggerek) programszerkesztők (linkerek).

- *Alkalmazói szoftver* vagy alkalmazások: a felhasználót a számítógép használatán túl mutató céljainak elérésében támogató specifikus programok.
  - Irodai szoftverek: szervezőprogramok, prezentációkészítők, kiadványszerkesztők, táblázatkezelők.
  - Üzleti alkalmazások: számlázóprogramok, könyvelő programok, adatbázis-kezelők, vállalatirányítási rendszerek.
  - Tervezőrendszerek: CAD-rendszerek.
  - Grafikai szoftverek: rajzprogramok, képszerkesztők.

- Média szoftverek: médialejátszók, médiaszerkesztők.
- Kommunikációs szoftverek: levelező programok, csevegő programok, távbeszélő programok.
- Hálózati alkalmazások: webböngészők, fájlcserélők.
- Rosszindulatú alkalmazások: vírusok, férgek, kémprogramok.
- Biztonsági programok: vírusellenőrzők, kémprogram-felderítők, titkosító programok, tűzfalak.
- Játékszoftverek.

– *Felhasználó által készített szoftver*: valamilyen alkalmazáson belül, annak támogatását kihasználó, programozói ismeretek nélkül elkészíthető programok.

- Makrók
- Prezentációk
- Játékkészítő programok programjai
- Adatbáziskezelő-generátorok

2.4. Operációs rendszer

Fogalma:
Az operációs rendszer olyan program, ami egyszerű felhasználói felületet nyújt, eltakarva a számítógép (rendszer) eszközeit.

Mint kiterjesztett (virtuális) gép:
A virtuális számítógép *egy szimulált számítógépet jelent.* Az összetettebb változat a rendszer szintű virtualizáció, mellyel komplett számítógépek emulálhatók, operációs rendszerrel együtt. Előnyei közé tartozik, hogy *több operációs rendszer futtatható közvetlen az ún. gazdarendszerből*, valamint az elszigeteltségéből adódó biztonságos szoftvertesztelés. (Példaként említhető a különböző kártékony kódok, vírusok tesztelése) Főbb hátrányai közé sorolható a magas erőforrásigény és a viszonylagos megbízhatatlansága a közvetlenül hardverekkel való kommunikáció terén.

Mint erőforrás menedzser:
Az operációs rendszer feladata, hogy a számítógépen futó, erőforrásokért versengő programok között *igazságosan felossza annak véges erőforrásait* (resource management). Például nyomtatási sor kezelő (időalapú) megosztása, memória (tér, címtér alapú) megosztása.

Kernel:
A rendszermag (angolul kernel) az operációs rendszer alapja (magja), amely felelős a hardver erőforrásainak kezeléséért (beleértve a memóriát és a processzort is). A többfeladatos rendszerekben – ahol egyszerre több program is futhat – a kernel felelős azért, hogy megszabja, hogy melyik program és mennyi ideig használhatja a hardver egy adott részét (ezen módszer neve a *multiplexálás*). A rendszermag nem „látható" program, hanem a háttérben futó, a legalapvetőbb feladatokat ellátó program.

Kernel mód és felhasználói mód:
Egy processzornak egy operációs rendszer alatt két módja (mode) van: felhasználói mód (user mode) és kernel mód (kernel mode). A processzor annak függvényében vált ezek között, hogy éppen milyen program fut a számítógépen. Az alkalmazások felhasználói módban futnak, míg az alapvető operációs rendszer feladatok kernel módban. Amíg egyes vezérlők kernel módban futnak, egyéb vezérők futhantnak felhasználói módban. A kernel mód egy felügyelt mód.
A rendszer- illetve az alkalmazói programok között egy szempont szerint tehetünk különbséget: az alkalmazások hasznos dolgokat (vagy egy játékot) valósítanak meg, míg a rendszerprogramok a rendszer működését biztosítják. Egy szövegszerkesztő alkalmazás, de a telnet egy rendszerprogram. A két kategória közti határ gyakran elmosódik, habár ez csak a megrögzött kategorizálók számára fontos.

Feladata:
Jól használható felhasználói felület biztosítása. Például a 0. generációs gépeknek sajátos kapcsolótáblás felülete volt, a korai rendszerek speciális terminálokkal rendelkeztek, míg az MS DOS-nak karakteres felülete volt.

Kommunikáció a perifériákkal:

– ***Lekérdezéses átvitel*** (polling): I/O port folyamatos lekérdezése. Sok helyen alkalmazott technika, gyakran szinkron szoftver hívásoknál is alkalmazzák.

– ***Megszakítás (Interrupt)***: Nem kérdezgetjük folyamatosan (az I/O portot), hanem az esemény bekövetkezésekor a megadott programrész kerül végrehajtásra. Pl.: aszinkron hívások esetén.

  • *Aszinkron hívások*: Olyan adatátviteli mód, amikor a két kommunikáló fél nem használ külön időzítő jelet, ellentétben a szinkron átvitellel. Éppen ezért szükséges az átvitt adatok közé olyan információ elhelyezése, amely megmondja a vevőnek, hogy hol kezdődnek az adatok.

– ***DMA (Direct Memory Access)***: közvetlen memória elérés (6. EA-ban van részletezve). Például közvetlen memóriacímzés: 0xb800:0.

Programkönyvtárak

– ***Programkönyvtár***

  • *A programkönyvtár* tulajdonképpen *egy gépi kódra lefordított fájl*, ami egy adott program különböző metódusait / procedúráit / függvényeit, adatait és erőforrásait tartalmazza. Ezeket az elemeket a program meg tudja hívni, és fel tudja használni a futása során.
  • Gépi kódra példák: intel x86, mov ax, 'F', mov eax, 'T', jmp cím.
  • *Normál, felhasználói programkönyvtárak (API, Application Programming Interface)*
    ◦ C64 ROM Basic
    ◦ DOS(IBM, MS), IO.sys, msdos.sys, interrupt tábla
    ◦ Windows 98, ... Windows7, Win32 API
    ◦ *Unix-Linux rendszerkönyvtárak, C nyelv.*
  • Script programozás (BASH, PoweShell)

– ***Felhasználói programkönyvtárak***

  • Jellemzően réteges szerkezetű
  • Alapvetően két rétegre oszthatjuk:
    ◦ Rendszer szintű hívás: kommunikáció a perifériákkal.
    ◦ Felhasználói hívás: széleskörű könyvtár biztosítás.
  • A könyvtárak számos programozási nyelvhez illeszkednek, például a C-hez, C++-hoz, Delphihez...
  • Kompatibilitás

Az API és a POSIX

– ***API*** *(Application Programming Interface)*
Egy program vagy rendszerprogram azon eljárásainak (szolgáltatásainak) és azok használatának *dokumentációj*a, amelyet más programok felhasználhatnak. Egy nyilvános API segítségével lehetséges egy programrendszer szolgáltatásait használni anélkül, hogy annak belső működését ismerni kellene. Általában nem kötődik programozási nyelvhez. Az egyik leggyakoribb esete az alkalmazásprogramozási felületnek az operációs rendszerek programozási felülete: annak dokumentációja, hogy a rendszeren futó programok milyen – jól definiált, szabványosított – felületen tudják a rendszer szolgáltatásait (pl. kiíratás) használni.

– ***POSIX*** *(Portable Operating System Interface for uniX)*
Valójában *egy minimális rendszerhívás (API) készlet, szabvány*, aminek témaköreibe tartozik pl.: fájl- és könyvtárműveletek, folyamatok kezelése, szignálok, szemaforok, stb. Szabvány ANSI C-vel azonos függvénykönyvtár. Ma gyakorlatilag minden OS POSIX kompatibilis.

– ***Függvénycsoport példák***

  • Matematikai függvények: sin, cos, tan, atan, log, exp, stb.
  • Állománykezelő függvények: create, open, fopen, close, read, write, unlink, stb.
  • Könyvtárkezelő függvények: opendir, closedir, mkdir, rmdir, readdir, stb.

- Karakterfüzér-kezelő függvények: strcpy, strlen, strcmp, strcat, strchar, strstr, stb.
- Memória-kezelők: malloc, free, memcpy, stb.
- Belső kommunikációs függvények: msgsnd, msgrcv, shmat, semop, signal, kill, pipe, stb.

– ***További POSIX függvénycsoportok***
- Csövek: pipe, mkfifo
- Üzenetsorok: msgsnd, stb.
- Szemaforok: semget, semxxx, sem_open, sem_yyyy.
- Bővebb információ: gyakorlaton, illetve manual.
- További Posix függvénycsoportok is léteznek, nem cél ezek elemzése.

– ***Fontosabb POSIX API témakörök***: fájl- és könyvtárműveletek, folyamatok kezelése, szignálok, csövek, standard C függvénykönyvtár, órák és időzítők, szemaforok, szinkron és aszinkron I/O, szálak kezelése, stb.

– ***Hogyan használjuk a gyakorlatban?***
- Operációs rendszer: Suse Linux Enterprise szerver pl. os.inf.elte.hu
- Szövegszerkesztő: vi, mcedit vagy helyi grafikus szerkesztés majd ftp.
- Segítség: man, pl. man exit, man strlen.
- Fordítás: cc -c elso elso.c //-c csak fordítás

– ***Operációs rendszer API-k***
- Ahány rendszer, annyi függvénykönyvtár.
- Ma is jellemző apik: Open VMS, OS/400, Win32 API, Mac OS API, Windows Mobile.
- Beágyazott API: Java, .NET.

Firmware - Middleware

– ***Firmware***
Hardverbe a gyártó által épített szoftver (merevlemezbe, billentyűzetbe, monitorba, memóriakártyába, de pl. távirányítóba vagy számológépbe épített is). Amelyek olyan rögzített, többnyire kis méretű programok és/vagy adatstruktúrák, melyek különböző elektronikai eszközök vezérlését végzik el.

– ***Middleware***
Operációs rendszer feletti réteg (pl. Java Virtual Machine, JVM).
Általánosan véve egy olyan számítógépes szoftver, amely az operációs rendszerek mögötti, azok számára nem elérhető szoftveralkalmazásokat biztosítja, de nem része egyértelműen az operációs rendszernek, nem adatkezelő rendszer és nem része a szoftveralkalmazásoknak sem.
Megkönnyíti a szoftverfejlesztők dolgát a kommunikációs és az input/output feladatok végrehajtásában, így a saját alkalmazásuk sajátos céljára tudnak összpontosítani.

2.5. Operációs rendszer generációk

– ***Történelmi generáció*** *(1792-1871)*:
- *elektromechanikus* számítógépek
- nincs oprendszer, operátoralkalmazás

– ***Első generáció****(1940-1955)*:
- a programot kapcsolótáblán kellett beállítani, *elektroncső*vel működött, programozása *kizárólag gépi nyelv*en történt, lukkártyák megjelenése
- *Neumann-elv* (Neumann János): kettes számrendszer alkalmazása, memória, programtárolás, utasításrendszer

– ***Második generáció****(1955-1965)*:
- már *tranzisztor*okat tartalmaztak (ami lecsökkentette a méretüket)

- memóriaként *mágnestár*at használnak (mágnesszalag, majd mágneslemez)
- *operációs rendszer* megjelenése
- magas szintű prognyelvek pl. fortan
- kötegelt rendszer megjelenése (5. EA-ban van részletezve)

– ***Harmadik generáció****(1965-1980)*:

- integrált áramkörök megjelenése
- azonos rendszerek, kompatibilitás megjelenése
- mutliprogramozás, multitask (több feladat a memóriában egyidejűleg) megjelenése
- időosztás megjelenése
- bonyolultabb operációs rendszerek

– ***Negyedik generáció****(1980-tól napjainkig)*:

- *személyi számítógépek*, MS Windows
- áramkörök, CPU (processzor) fejlődés
- hálózati, osztott rendszerek

MINIX 3: Kezdetben az UNIX forráskód az AT&T engedélye alapján felhasználható volt. A MINIX avagy MINI Unix már nyílt forráskódú volt, a Linux őse.

2.6. Rendszerhívások

– ***Rendszerhívások:*** azok a szolgáltatások amelyek az operációs rendszer és a felhasználói programok közötti kapcsolatot biztosítják. Két fő csoportba sorolhatók: folyamat/process kezelő csoport és fájlkezelő csoport.

– ***Process kezelés***

- *Processz:* egy végrehajtás alatt lévő program. Saját címtartománnyal rendelkezik, megszüntetés, felfüggesztés, és process-ek kommunikációja (szignálokkal) is lehetséges.
- *Processz táblázat:* Az operációs rendszer által nyilván tartott táblázat, melynek minden sora tartalmazza egy éppen futó process adatait: cím, regiszter, munkafájl adatok.
- Processz indítás, megszüntetés: shell, gyerekfolyamatok.
- Processz felfüggesztés: memória térkép + táblázat mentés.
- Processzek kommunikációja: szignálok.

– ***Fontosabb folyamatkezelő hívások***

- int pid = fork () – a folyamat tükrözése, szülő folyamatban
- int i = waitpid(pid, &status, opt) – adott (pid) gyermekfolyamat jelzésre vár, ha opt == null, akkor vége.
- Exec programcsalád
  - Execv – a paraméterek egy tömbben vannak.
  - Execl – a paraméterek felsorolva szerepelenek, null-al lezárva.
- Getpid(), getppid(), stb.

– ***Fontosabb szignálkezelők***

- signal(SIGKILL, handler) – jelzés, jelzéskezelő beállítás
- kill(pid, SIGKILL) – jelzés küldése
  - Pid = -1 – mindenkinek aki él, ha joga is van hozzá
  - Pid = 0 – a külcső processzcsoportnak, ha van joga.
- int i = pause() – várakozás (alvás) egy jelzésre.
- int i = sigaction(pid, &act, &saveact)
- int i = sigqueue(pid, SIGKILL, value) – signal küldése egy adattal
- Továbbiak: man

– ***Fájlkezelés***

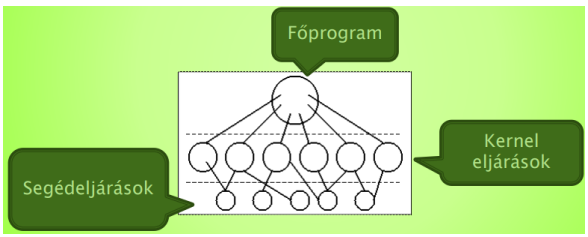
- A korai operációs rendszerek általában csak egy, nekik készült fájlkezelő rendszert támogattak, ami többnyire névtelen volt; például a CP/M csak a saját fájlkezelő rendszerét támogatta, amit „CP/M file system” néven ismerünk, de így szinte senki sem nevezte.
- A fentiek miatt szükségessé vált, hogy legyen egy interfész az operációs rendszer, a fájlkezelő rendszer és a felhasználó között. Ez az interfész lehet szöveges (amit egy parancssoros felhasználói felület biztosít, mint például a Unix shell, vagy OpenVMS DCL) vagy grafikus (mint amit egy grafikus felhasználói felület biztosít, mint például a fájlkezelők). Ha grafikus, akkor megfelel valamilyen gyakran használt mappaábrázolásnak, ami dokumentumokat és egyéb fájlokat, illetve beágyazott mappákat is tartalmazhat.
- Szerkezet: egy főkönyvtár, fastruktóra, kétféle bejegyzés: fájl és könyvtár.
- Műveletek: másolás, létrehozás, törlés, megnyitás, olvasás, írás
- Jogosultságok: rwx (read, write, mindkettő) - adott jog hiánya
- Speciális fájlok: karakter, blokk fájlok, /dev könyvtár, adatcső, pipe

– ***Fontosabb fájlkezelők***

- Bináris kontra szöveges fájl.
- Bináris:
  - Int
    fájl megnyitásda: fd=Open(név,O_WRONLY|O_CREAT,S_IRUSR|S_IWU SR);
  - beolvasás: Int db=Read(fd, &hova, mennyit)
  - kiírás: Int db=write(fd, &honnan, mennyit)
  - Close(fd);
  - offset beállítás: Int i=lseek(pid,offset,SEEK_SET)
    - SEEK_CUR – beállítás, az aktuális pozícióhoz képest
    - SEEK_END – beállítás fájlméret + offset
- Szöveges:
  - FILE* f=fopen(név, mód) – mód=r, r+,w,stb
  - int i=fclose(f); - fájl zárása
  - Kiírás: fprintf()...
  - Beolvasás:
    - fscanf(f,format,...)
    - char* s=fgets(hova,db,f);
    - int c=fgetc(f);
  - fseek(f,offset,SEEK_SET) – pozíció módosítás
    - void rewind(f) – fájl pozíció az elejére áll
  - Továbbiak: manual

2.7. Operációs rendszer struktúrák:

Monolitikus rendszerek:
Nincs különösebb struktúrája, a rendszerkönyvtár egyetlen rendszerből áll, így mindenki mindenkit láthat. Információelrejtést nem igazán valósítja meg. Létezik modul, modulcsoportos tervezés. Rendszerhívás során gyakran felügyelt (kernel) módba kerül a CPU. Paraméterek a regiszterekben, valamint a csapdázás (trap) is jellemző.



1. ábra. Monolitikus szerkezeti modell

Rétegelt szerkezet:

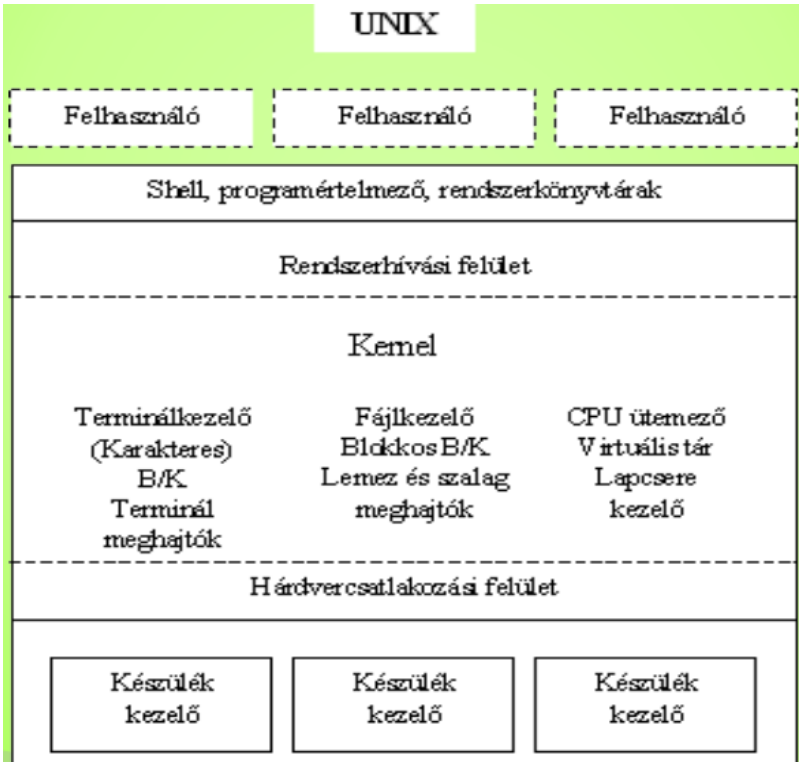
6. Gépkezelő
5. Felhasználói programok
4. Bemeneti / Kimenet kezelése
3. Gépkezelő-folyamat
2. Memória és dobkezelés
1. Processzorhozzárendelés és multiprogramozás

5.	A gépkezelő
4.	Felhasználói programok
3	Bemenet/Kimenet kezelése
2	Gépkezelő-folyamat
1	Memória és dobkezelés
0	Processzorhozzárendelés és multiprogramozás

2. ábra. Réteges szerkezeti modell

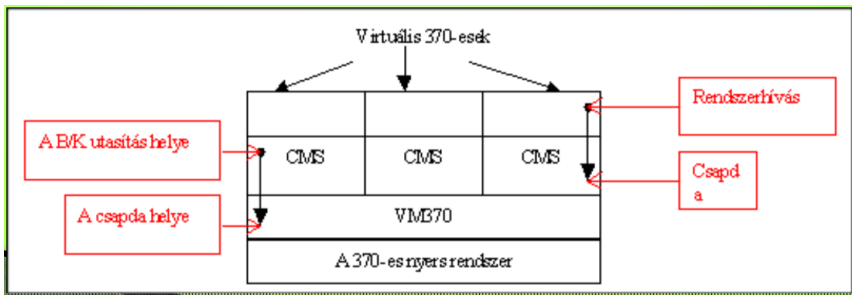
- E.W. Dijkstra tervezte, neve: THE(1968)
- A MULTICS-ban tovább általánosították. Gyűrűs rendszer.

Tipikus rétegrendszer
A Multics utód UNIX jellemző réteges, gyűrűs szerkezete.



3. ábra. Tipikus szerkezeti modell

Virtuális gépek

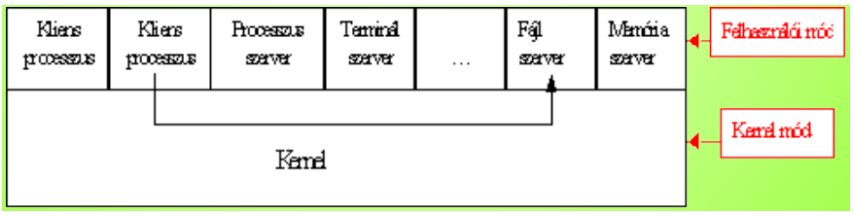


4. ábra. Virtuális gép ábra

- ***A virtuális számítógép*** egy szimulált számítógépet jelent. A virtuális számítógép fizikailag nem létezik: a felépítése csupán egy szimuláció, egy olyan számítógépes program, ami egy létező fizikai számítógépet, vagy egy fizikailag nem felépített számítógép működését szimulálja. Ez valójában egy "teljes számítógép egy másik számítógépen belül".
- Virtuális gép monitor: a hardvert pontosan másolja.
- Ezt tetszőleges példányban képes volt sokszorozni.
- ***Exokernel***: virtuális gép számára az erőforrások biztosítása (CPU, memória, HDD)
- ***Mai virtuális gépek***
  - VMWare - Unix - Linux platform (fut Windows-on is)
  - MS Virtual Server, Virtual PC
  - Más rendszerű virtuális gépek: JVM, .NET
- ***Virtualizációs fogalmak***
  - Host rendszer – vendég rendszer
  - Fő kérdés: processzor privilegizált, problémás utasításait hogyan kell végrehajtani?
  - Paravirtualizáció – vendég rendszerben módosítják a kritikus utasításokat, ma már nem igazán használt
  - Szoftveres virtualizáció – vendég rendszer változatlan, host rendszer problémás utasításnál emulál
  - Hardveres virtualizáció – processzor ad segítséget a kritikus utasításokhoz. Ma gyakorlatilag ez használt.
- ***Multiboot – Virtualizáció összegzés***
  - Igény több rendszerre:
    - Partíciónként más-más operációs rendszer, melynek hátránya, hogy újra kell indítani a gépet rendszerváltásnál.
    - Virtualizáció: ma gyakorlatilag teljes a hardveres támogatás, van elég memória, van elég háttértár, általános támogatás minden rendszerben. Hátránya, hogy a teljes hardver emuláció "túl sok" erőforrást igényel.
  - Újabb igény: Alkalmazás biztonság igénye.
    - Konténertechnológia, nem kell virtuális gép!
    - Elég a kernel támogatta "alkalmazás izoláció"

Kliens - Szerver modell

- A kliens-szerver szoftverarchitektúra egy sokoldalú, üzenetalapú és moduláris infrastruktúra amely azért alakult ki, hogy a használhatóságot, rugalmasságot, együttműködési lehetőségeket és bővíthetőséget megnövelje a centralizált, nagyszámítógépes, időosztásos rendszerekhez képest.
- A kliens-szerver szoftverarchitektúra egy sokoldalú, üzenetalapú és moduláris infrastruktúra amely azért alakult ki, hogy a használhatóságot, rugalmasságot, együttműködési lehetőségeket és bővíthetőséget megnövelje a centralizált, nagyszámítógépes, időosztásos rendszerekhez képest.



5. ábra. Kliens-szerver modell

- Felhasználói program: kliens program.
- Kiszolgáló program: szerver program.
- Mindegyik felhasználói módban fut.
- Egyre kevesebb funkció marad a kernelben.

Megjegyzés:

- *Kliens:* A kliens (angolul client) olyan számítógép amely hozzáfér egy (távoli) szolgáltatáshoz, amelyet egy számítógép hálózathoz tartozó másik gép nyújt.
  Jellemzői:
  - Kéréseket, lekérdezéseket küld a szervernek A választ a szervertől fogadja.
  - Egyszerre általában csak kisszámú szerverhez kapcsolódik
  - Közvetlenül kommunikál a felhasználóval, általában egy GUI-n (Graphical User Interface = Grafikus felhasználói felület) keresztül
- *Kiszolgáló*: A kiszolgáló vagy szerver (angolul server) olyan (általában nagy teljesítményű) számítógépet, illetve szoftvert jelent, ami más gépek számára a rajta tárolt vagy előállított adatok felhasználását, a kiszolgáló hardver erőforrásainak (például nyomtató, háttértárolók, processzor) kihasználását, illetve más szolgáltatások elérését teszi lehetővé.
  Jellemzői:
  - Passzív, a kliensektől várja a kéréseket
  - A kéréseket, lekérdezéseket feldolgozza, majd visszaküldi a választ
  - Általában nagyszámú klienshez kapcsolódik egyszerre
  - Általában nem áll közvetlen kapcsolatban a felhasználóval

2.8. Operációs rendszer elvárások

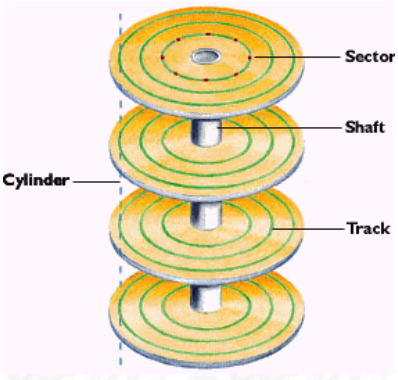
- ***Hatékonyság*** (Efficiency): a meglévő erőforrásokat a leghatékonyabban továbbítsa a felhasználók felé.
- ***Megbízhatóság*** (Reliability): a hibátlan működés biztosítása. Adatok megőrzése, rendelkezésre állás, megbízhatóság kiterjesztése: hibatűrés.
- ***Biztonság*** (Security): Külső rendszerekkel szembeni biztonság és adatbiztonság.
- ***Kompatibilitás*** (Compatibility): Hordozhatóság, két rendszer közti adat és programcsere lehetősége, lényeges a szebványok szerepe (POSIX).
- ***Alacsony energiafelhasználás*** (Nem csak mobil gépek esetén.)
- ***Rugalmasság*** (Flexibility): Más néven skálázhatóság. Erőforrások rugalmas kiosztása (memória, processzor).
- ***Kezelhetőség*** (Manageability): ütemezési, felhasználói szinten.
- ***Megjegyzés***: kérdéses, hogy ez mind megvalósítható-e egyszerre.

3. Előadás

3.1. Háttértárak

Mágneses elvű

- ***Mágnesszalagok:*** sorrendi, lineáris felépítéssel rendelkezik, keretek rekordokba szerveződnek. Rekordok között rekord elválasztó (record gap), fájlok között fájl elválasztó (file gap) található. Gigabyte-ra levetítve a legolcsóbb tárolási módszer. Biztonsági mentésekre, nagy mennyiségű adat tárolására használatos.
- ***Mágnesszalagok fizikai felépítése***
  - Mágnesszalagok: sorrendi, lineáris felépítés
    - 9 bites keret (8 bit + paritás)
    - Keretek rekordokba szerveződnek
    - Rekordok között: rekord elválasztó (record gap)
    - Egymás utáni rekordok után, fájl elválasztó (file gap)
    - Szalag elején a könyvtárszerkezet
  - Jellemző használat
    - Biztonsági mentés
    - Nagy mennyiséfű adattárolásra
  - Nem igazán olcsó
  - Jellemző méret: DLT (Digital Linear Tape), LTO (Linear Tape-Open) 4 Ultrium 800/1600 GB, LTO5 1.5TB/3TB
- ***Mágneslemezek: FDD*** (*Floppy* Disk Drive) és ***HDD*** (Hard Disk Drive - *Merevlemez*): Az FDD (általában) egy, a HDD (általában) több kör alakú lemezből áll. Ezek sávokra (tracks), a sávok blokkokra (blocks), a blokkok szektorokra (sectors) vannak felosztva. A több lemezen elhelyezkedő, egymás alatt lévő (azaz azonos sugarú) sávok összeségét cilindernek (cylinder) nevezzük (ld. 6. ábra). A HDD rendelkezik egy író/olvasó fejjel is, mely e lemezek fölött ill. között mozog, e segítségével lehet a különféle műveleteket végrehajtani.

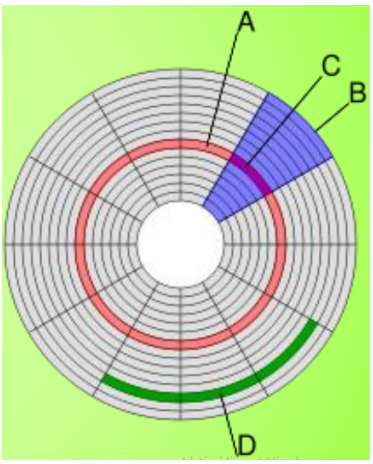


6. ábra. HDD részei

- ***A mágneslemezek felépítése***
  - FDD (Floppy Disk Drive): jellemzően egy lemez.
  - HDD (Hard Disk Drive): jellemzően több lemez.
  - A lemezek felosztása
    - Az adatok sávokban (track-ekben), azon belül szektorokan (sector-okban) helyezkednek el.
    - sávok: A lemez mágneses felületén elhelyezkedő koncentrikus körök, melyek száma 40 vagy 80.

    - Szektorok: A sáv szelete (körcikk), melyek száma 8 vagy 18 között van. Egy szektoron belül 512 byte adat tárolható.
    - Önállóan írható és olvasható.
    - Klaszter (cluster): (több szektor összefoglaló neve) melyet az operációs rendszer együttesen kezel.
  - Több lemez – egymás alatti sávok: cilinder
  - Logikailag egy folytonos blokksorozat.
  - A fizikai működést a meghajtó (firmware) eltakarja.

- ***A mágneslemez felépítése***
  - A: sáv.
  - B: szektor.
  - C: blokk, 512 byte.
  - D: klaszter, a fájlrendszer által megválasztott logikai tárolási egység. $D = n * C$, ahol $n = = 1...128$.
  - Cilinder: Az egymás alatti sávok (A).



7. ábra. A mágneslemez felépítése

- ***Mágneslemez felépítésére példa***
  - CHS címzés (Cilinder - Head - Sector)
    - Példa: 1.44 MB FD
    - Sávok száma: 80 (0-79)
    - Fejek (cilinder) száma: 2 (0-1)
    - Szektorok száma egy sávon: 18 (1-18)
    - Össz. méret: 80*2*18 = 2880 szektor * 512 byte
  - LBA címzés (Logical Block Adressing)
    - Korábban 28 bites, kb. 137 GB-ig jó.
    - Jelenleg 48 bites, 144 PB (Petabájt), (144 000 000 GB),.
    - $A = (c * N_{heads} * N_{sectors}) + (h * N_{sectors}) + s - 1$

- ***Mágneslemez formázása:***
  - Sávos-szektoros rendszer kialakítása.
  - Jellemzően egy szektor 512 byte.
  - Gyárilag a lemezek "elő vannak készítve".
  - Quick format – Normal format: normal format hibás szektorokat (bad sector) keres.
  - Logikai formázás: a partíciók kialakítása – max 4 logikai rész alakítható ki.
  - Szektor = Szektorfej + adatblokk + lábléc
    - Szektorfej: sáv száma, fej száma, szektor száma.
    - Lábléc: hibajavító blokk.
  - Alacsony szintű formázás: szektorok kialakítása (ez gyártóknál elérhető).

Optikai elvű

- **Optikai tárolók:**
  - Fényvisszaverődés idő különbség alapján működnek. Belső résztől spirális "hegyek - völgyek" (pit-land) sorozata. Írható lemezeknél az írás a lemezfelület mágnesességét, fény törésmutatóját változtatja meg, így más lesz a visszaverődő fény terjedési sebessége. Például: CD, DVD, blue-ray, stb.
  - Tipikusan 8 vagy 12 cm átmérőjű optikai lemezek
    - CD (Compact Disc), DVD (Digital Versatile Disc)
    - Méret: 650 MB - 17 GB között
    - Sebesség: 1x = 150 KB/sec
  - Működési elv: fényvisszaverődés idő különbség alapján.
    - Belső résztől spirális "hegyek-völgyek" (pit-land) sorozata
    - Írható lemezek: írás a lemezfelület mágnesességét, fény törésmutatóját változtatja meg, így más lesz a fény terjedési sebessége.
- **Eszközmeghajtó** (Device driver): Az a program, amely a közvetlen kommunikációt végzi az eszközzel. Ez a kernelnek (operációs rendszer magjának) a része. Lemezek írása, olvasása során DMA-t használnak (nagy adatmennyiség). (DMA: 6. EA-ban van kifejtve).
  - Megszakítás üzenet: tipikusan azt jelzi, ha befejeződött az írás-olvasás művelet.
  - I/O portokon az írás, olvasás paraméterek beállítását végzik.

  Réteges felépítésű.

Logikai tároló (logical disk): Olyan tároló, mely az operációs rendszer számára egybefüggő memóriaterületnek tűnik. A „logikai" szó arra utal, hogy gyakran ezek nem ténylegesen így tárolódnak, hanem akár több lemezen keresztül is, azonban a HDD firmware programja gondoskodik arról, hogy számunkra a teljes terület egybefüggőnek tűnjön.

Logikai formázás

- ***Partíciók kialakítása:*** egy lemezen a PC-s rendszerben maximum 4 logikai lemezrész kialakítható.
- ***0. Szektor – MBR (Master Boot Record):*** Partíciós szektor a merevlemez legelső szektorának (azaz az első lemezfelület első sávjának első szektorának) elnevezése. Csak a particionált merevlemezeknek van MBR-jük. A MBR a merevlemez legelején, az első partíció előtt található meg. Gyakorlatilag a merevlemez partíciók elhelyezkedési adatait tárolja. Elsődleges partíció: erről tölthető be operációs rendszer.
- A partíción a szükséges adatszerkezet (fájlrendszer) kialakítása.
- ***Címszámítás***: Blokkok sorszámainak meghatározása. Kell a fejek száma, szektorok száma. például tegyük fel, hogy adott 4 fej (2 vagy 4 lemez) és egy sáv legyen felosztva 7 szektorra. A lemezek forgási sebessége miatt a blokkok nem feltétlenül szomszédosak (interleave).
- ***Lemez elérés fizikai jellemzői***: Forgási sebesség – egy sávon (cilinderen) belül mekkorát kell fordulni.
- ***Fej mozgási sebesség***: egy cilinderen belül nem kell mozgatni a fejet. (Pl. 15 000 fordulat percenként megad egy lehetséges sebességet.)
- ***Az írás-olvasás ütemezés*** feladata a megfelelő (gyors, hatékony) kiszolgálási sorrend megválasztása. A hozzáférési idő csökkentése és az átviteli sávszélesség növelése.

3.2. Írás-olvasás műveletek

Boot folyamat: ROM-BIOS megvizsgálja, lehet-e operációs rendszert betölteni, ha igen, betölti a lemez MBR (lásd. fent) programját a `7c00h` címre (konvenció szerint ez mindig erre a címre történik). Ez után az MBR programja vizsgálja meg mi az elsődleges partíció, majd azt betölti a memóriába.

– Alacsony hívás során a következő adatok szükségesek: beolvasandó (kiírandó) blokk(ok) sorszáma, memóriaterület címe - ahova be kell olvasni, bájtok száma.
– Több folyamat használja: melyiket hajtsuk végre először?

BIOS

– A BIOS az angol Basic Input Output System rövidítése, ami magyarul alapvető bemeneti–kimeneti rendszert jelent, és a számítógép szoftveres és hardveres része közötti interfész megvalósítására szolgál. Fizikailag az alaplapon lévő BIOS, az egyes bővítőkártyákon található BIOS és ezek eszközmeghajtói alkotják a számítógép BIOS-át. Ezek közül az alaplap BIOS-a a BIOS legfontosabb része, mert ez tartalmazza az alapvető konfigurációs beállításokat és hajtja végre a diagnosztikai ellenőrzéseket.

– A BIOS-t egy, az alaplapon elhelyezkedő integrált áramkör tartalmazza (a régebbi típusokban ROM-ba égetve, később EEPROM, manapság Flash RAM-ban). A BIOS chipjének a kapacitását megabitekben (Mb) mérjük, egy chip általában 1-4 Mb memóriát tartalmaz. Két része van: fix rész, variábilis rész.

– Annak érdekében, hogy biztosítsuk a számítógép helyes működését, a BIOS-nak ismernie kell a gép paramétereit, valamint a jelenlegi konfigurációt. Ez az információ egyrészt magába a BIOS-ba van bekódolva (pl. ACPI információk, IRQ routing), másrészt a CMOS RAM (Complementary Metal Oxide Semiconductor RAM) tárolja. Ez egy speciális memória, melynek elektromos táplálását akkumulátor vagy elem segítségével oldják meg. Ez is az alaplapon található. Ennek a rendszernek köszönhető, hogy a CMOS adatai a gép kikapcsolása után sem tűnnek el.

UEFI
Unified Extensible Firmware Interface (UEFI), amely a BIOS utódja, azt váltotta fel.

UEFI vs. BIOS

– BIOS (Basic Input-Output System) probléma: max. 2 TB háttértár kezelés.
– UEFI (BIOS utód) - Unified Extesible Firmware Interface
  • Bios x86 módban fut mindenhol, UEFI natívban (x64)
  • 2 TB-nál nagyobb meghajtók, 128 partíció, nagyobb RAM
  • MBR nem használt, helyette GPT (GUID Partition Table)
    ◦ OS betöltő saját fájlrendszerben .efi kiterjesztés
    ◦ Csak 64 bites OS betöltő!
    ◦ Secure boot (csak digitálisan aláírt boot engedélyezés)

Címszámítás

– Blokkok sorszámainak meghatározása
  • Kell a fejek száma, a szektorok száma
  • Tegyük fel adott 4 fej (2 vagy 4 lemez)
  • Egy sáv legyen felosztva 7 szektorra
– Lemezek forgási sebessége miatt a blokkok nem feltétlenül szomszédosak (interleave)
  • 1:2 interleave, párosával "szomszédosak"

	1 szektor	2 szektor	3 szektor	4 szektor	5 szektor	6 szektor	7 szektor
1 fej.	1	17	5	21	9	25	13
2 fej.	2	18	6	22	10	26	14
3 fej.	3	19	7	23	11	27	15
4 fej.	4	20	8	24	12	28	16

8. ábra. Címszámítás

Írás-olvasás műveletek ütemezése
Az alacsony szintű (kernel) feladat paraméterei a következők: kérés típusa (írás-olvasás), a blokk kezdőcíme (sáv, szektor, fej száma), DMA memóriacím, mozgatandó bájtok száma.
A lemezt több folyamat is használná zömmel, kérdéses lehet, hogy melyiket szolgálja ki először (azaz az ütemezés), ekkor a fejmozgást is figyelembe veszi (olvasandó blokk adataiból következik).

Ías-olvasás műveletek ütemezése

- ***Alacsony szintű (kernel) feladat paraméterek***
  - Keresés típusa (írás-olvasás)
  - A blokk kezdőcíme (sáv, szektor, fej száma)
  - DMA memóriacím
  - Mozgatandó bájtok száma
- ***Több folyamat is használná a lemezt***
  - Kit szolgáljunk ki először?
  - Fejmozgás figyelembe vétele (beolvasandó blokk adataiból következik)
- ***Sorrendi ütemezés*** (FCFS – First Come First Service)
  - Ahogy jönnek a kérések, úgy *sorban szolgáljuk ki* azokat.
  - Biztosan minden kérés kiszolgálásra kerül, de nem törődik a fej aktuális helyzetével, kicsi az adatátviteli sebesség, és ezért nem igazán hatékony.
  - Átlagos kiszolgálási idő, kis szórással.
- ***„Leghamarabb először"* ütemezés** (SSTF – Shortest Seek Time First)
  - *A legkisebb fejmozgást részesíti előnyben.*
  - Átlagos várakozási idő kicsi, a várakozási idő szórása nagy.
  - Átviteli sávszélesség nagy.
  - Fennáll a kiéheztetés veszélye (azaz előfordulhat, hogy egy kérést sose teljesít, mert minden más kérés teljesítéséhez kevesebbet kéne mozognia a fejnek).
- ***Pásztázó ütemezés***: SCAN (LOOK) módszer
  - A *fej állandó mozgásban van*, és a *mozgás újtába eső kéréseket kielégíti.*
  - A fej mozgás megfordul ha a mozgás irányában nincs kérés, vagy a fej szélső pozíciót ér el.
  - Rossz ütemben érkező kérések kiszolgálása csak oda – vissza mozgás (írás-olvasás) után kerül kiszolgálásra. Várakozási idő közepes, szórás nagy.
  - Középső sávok elérésének szórása kicsi.
- ***Egyirányú pásztázás*** (Circural SCAN, C-SCAN)
  - A SCAN javított verziója, írás-olvasás, csak a fej egyik irányú mozgásakor történik.
  - *Csak egyirányú mozgás*, ezért *gyorsabb a fejmozgás.*
  - Nagyobb sávszélesség.
  - Átlagos *várakozási idő hasonló, mint a SCAN esetén*, viszont a *szórás kicsi.* Nem igazán fordulhat elő rossz ütemű kérés.

Ütemezés optimalizálása

- ***Ütemezés javítások:***
  - FCFS esetében ha az aktuális sorrendi kérés kiszolgálás helyén van egy másik kérés is akkor szolgáljuk ki azt is (Pick up).
  - Egy folyamat adatai jellemzően egymás után vannak, így egy kérés kiszolgálásánál "picit" várva, a folyamat az adatainak további részét is kéri. (Előlegző ütemezésnek is nevezzük.)
  - A lemez közepe általában hatékonyan elérhető.
- ***Ütemezés javítása memória használattal:***

- DMA maga is memória (6. EA)
- Memória puffer (átmeneti tár) használat
  - Kettős körszerű használat
  - Olvasás: ütemező tölti, felhasználói folyamat üríti.
  - Írás: felhasználói folyamat tölti, ütemező üríti.
- Disc cache (Lemez gyorsítótár) használatával
  - Előre dolgozik az ütemező, a memóriába tölti a kért adatok "környéki2 lemezterületet is.
  - Operációs rendszernek jelent plusz feladatot.
  - Pl.: Smartdrive.

– ***Milyen ütemezést válasszunk?***
  - A fenti algoritmusok csak a fejmozgás idejét vették figyelembe, az elfordulást nem.
  - A sorrendi ütemezés tipikusan egy felhasználós rendszernél használt.
  - SSTF, kiéheztetés veszélye nagy.
  - C-Scan: nagy IO átvitel, nincs kiéheztetés
  - Beépített ütemező: Pl. SCSI vezérlők.
  - OS ömlesztve adja a kéréseket.

SLE Block device (blokk kezeléses) ütemezés

– ***CFQ – Completely Fair Queuing***:
  - Minden folyamat saját I/O sort kap.
  - Ezen sorok között azonosan próbálja az ütemező elosztani a sávszélességet.
  - Ez az alapértelmezett ütemező.
– ***Létezik még***:

  - *NOOP* – ez felel meg a "Strucc" algoritmusnak. Egy sor van, amit a (RAID) vezérlők gyorsan teljesítenek.
    - *Strucc algoritmus:*
      Nem foglalkozunk a holtponttal (strucc-politika). Valószínűtlen eseménynek tekintjük ezt a helyzetet, a kezelése pedig költséges.
      Ha a holtpont kialakulásának valószínűsége kicsi, a rendszer újraindításának nincsenek kritikus következményei, akkor megérheti ezt a megoldást választani.
  - *Deadline* – egy kéréshez határidő tartozik, két sort használ. Egy blokksorrend alapján készített sort (SSTF) és egy határidő alapján készített sort. Alapból a blokksorrend a lényeges, de ha határidő van, akkor az kerül sorra!

Ütemezés kulcsfeladata
Gyorsan (minél gyorsabban) kiszolgálni a kéréseket. Ezt elősegíti:

– Az összetartozó elemek együtt való tárolása (töredezettségmentesség).
– A sávszélesség a lemez közepén a legnagyobb.
– Leggyorsabban a lemez közepe érhető el (virtuális memória).
– Lemez gyorsítótára a memóriában.
– Esetleg adattömörítés (nagyobb CPU, azaz processzor terhelés).
– Ezeket az OS is használja a kérések kiszolgálásának gyorsításához.

Lemezek megbízhatósága:

– Jelentése: A lemezek meg tudnak sérülni, mely adatvesztéshez vezet. Erre egy megoldás lehet, ha az adatokat redundánsan tároljuk úgy, hogy egy lemez sérülése esetén se történjen adatvesztés.
– Operációs rendszer szolgáltatás
  - Dinamikus kötet: több lemezre helyez egy logikai meghajtót. Méret összeadódik.
  - Tükrözés: két lemezre helyez egy meghajtót. Mérete az egyik (kisebb) lemez mérete lesz.
  - Nagy(obb) CPU igény.
– Hardware szolgáltatás
  - Intelligens meghajtó szolgáltatás
  - Az SCSI eszköz világban jelent meg először (RAID).

Megbízható lemezmeghajtók: pl. RAID (Redundant Array if Intensive Disks).
SCSI (Small Computer System Interface) lemezegységeknél jelent meg először. Ez a számítógépek és perifériák közti adatcsere egy ma is népszerű szabvány együttese. Leggyakrabban lemezek körében használt, szerver gépek használják (ták). Ennek egy újabb változata a SAS csatoló (Serial Attached SCSI).

Dinamikus kötet: Több lemezre helyez egy logikai meghajtót. Méret összeadódik.

3.3. RAID

- Régebben Redundant Array of Inexpensive Disks, mostani időkben inkább Redundant Array of Independent Disks. Disk
- Ha oprendszer nyújta akkor SoftRaid-nek is nevezzük, ha külső vezérlőegység akkor Hardver Raid.

A RAID egy tárolási elv, mely az adatok biztonságos tárolását írja le. Az alapelve az, hogy *egyszerre több lemezről ír és olvas (ezért redundáns)*. Az operációs rendszer számára ez a több disk egynek tűnik (egy tömbnek, angolul array). Ez az elv régen azzal a szlogennel élt, hogy „an array of inexpensive drives could beat the performance of the top disk drives of the time" (ezért olcsó, angolul inexpensive), de ez a mai árak szerint drágább, így az inexpensive szót leváltották independent-re.

A RAID-nek 7 szintje vagy verziója létezik:

- ***RAID 0:*** több lemez logikai összefűzésével egy meghajtót kapunk (ezért tulajdonképpen nem is redundáns), ezek összege adja az új meghajtó kapacitását. A logikai meghajtó blokkjait szétrakja a lemezekre, ezáltal egy fájl írása több lemezre kerül. Gyorsabb I/O műveletek (azaz messze ez a leggyorsabb RAID), de nincs meghibásodás elleni védelem. Nevével szemben nincs benne redundáns adattárolás.
- ***RAID 1:*** Két független lemezből készít egy logikai egységet, minden adatot párhuzamosan kiír mindkét lemezre. Tárolókapacitás a felére csökken, drága megoldás, csak mindkettő lemez egyszerre történő meghibásodása esetén okoz adatvesztést.
- ***RAID 2:*** Több lemezen is tárol adatot, és valamennyi külön erre dedikált lemezen csak ezen adatok hibavizsgáláshoz és helyreállításhoz szükséges információkat ment le. Ennek nincsen semmilyen előnye a RAID 3-hoz képest, és már nem is használják.
- ***RAID 3:*** Egyetlen disc-en tárol paritás információkat (melyek alapján eldönthető, hogy sérült-e a fájl). Mivel egyszerre csak egy kérést tud teljesíteni, hosszú beolvasások és kiíratások esetén nagyon hatékony, gyakori rövid műveletek esetén a legrosszabb.
- ***RAID 4:*** A RAID 0 kiegészítése paritásdiszkkel. Ennek köszönhetően egyszerre több olvasást is végre tud hajtani. Semmilyen előnye nincs a RAID 5-tel szemben.
- ***RAID 5:*** Nincs paritásdiszk, az oda tartozó információ el van osztva az összes disc-re. Adatok is elosztva tárolódnak. Intenzív CPU igény, két lemez egyidejű meghibásodása esetén okoz adatvesztést. Azaz +1 diszket igényel.
- ***RAID 6:*** A RAID 5-höz hasonlóan tárolja a paritásinformációkat, de ezek mellett még hibajavító kódokat is tárol. Ez közel kétszer annyi területet foglal backup-ra mint a RAID 5, azonban két disc egyszeri meghibásodása se jár adatvesztéssel. Meglehetősen drága.
- ***Megjegyzés:*** leggyakrabban az 1, 5 verziókat használják, a 6-os vezérlők az utóbbi 1-2 évben jelentek meg.

3.4. Adattárolás összefoglalása

- Adatok biztonságos tárolását biztosítja.

– Több szintű:

1. Fizikai lemezek (HDD)
2. Harver RAID
3. Partíciók
4. Szoftver RAID
5. Volume Manager az operációs rendszeren

– Nem minden ellen véd
  - Pl.: Tápellátás elhal, emberi tévedés, stb.
  - Szotveres támadások, vírusok.

4. Előadás – Fájlok, könyvtárak, fájlrendszerek

4.1. Fájl

Fájl

– ***Adatok egy logikai csoportja***, névvel és egyéb paraméterekkel ellátva.
  - A fájl az információtárolás egysége.
  - Névvel hivatkozunk rá.
  - Jellemzően egy lemezen helyezkedik el, de általánosan a adathalmaz, adatfolyam akár képernyőhöz, billentyűzethez is köthető.
  - A lemezen általában három féle fájl, állomány található:
    - rendes felhasználói állomány,
    - ideiglenes állomány,
    - adminisztratív állomány (ez a működéshez szükséges, általában rejtett).

– ***Jellemzői***
  - *A fájlnév* egy karaktersorozat, az operációs rendszertől függ, hogy milyen a szerkezete (hossza, megengedett karakterek, kis-nagybetű különbség).
  - *Egyéb attribútumok* a fájl mérete, tulajdonosa, utolsó módosításának ideje, hogy rejtett fájlról van-e szó (hidden), vagy rendszer fájlról, milyen hozzáférési jogosítványok vannak kiadva hozzá, stb.
  - *Fizikai elhelyezkedése*: valódi fájl, link (hard), link (soft).

Könyvtár: fájlok (könyvtárak) logikai csoportosítása.

Fájlrendszer: módszer, a fizikai lemezünkön, kötetünkön a fájlok és könyvtárak elhelyezésrendszerének kialakítására.

Fájlok elhelyezése

4.2. Könyvtárak

Könyvtár: Fájlok (könyvtárak) logikai csoportosítása.

Könyvtárak: Valójában egy speciális bejegyzésű állomány, tartalma a fájlok nevét tartalmazó rekordok listája.

Könyvtár szerkezetek

- Katalógus nélküli rendszer, szalagos egység.
- Egyszintű, kétszintű katalógus rendszer (nem igazán használt).
- Többszintű, hierarchikus katalógus rendszer:
  - fa struktúra
  - hatékony keresés
  - ma ez a tipikusan használt.

Abszolút, relatív hivatkozás: PATH környezetbeli változó.

Hozzáférési jogok

- Nincs általános jogosítványrendszer.
- Jellemző jogosítványok:
  - Olvasás
  - Írás, létrehozás, törlés
  - Végrehajtás
  - Módosítás
  - Full control
- Jogok nyilvántartása
  - Attribútumként
  - ACL: NFS-AFS különbözőség, hasonló elve, különböző implementáció.

4.3. Fájlrendszerek

Fájlrendszer
Módszer a fizikai lemezünkön, kötetünkön a fájlok és könyvtárak elhelyezés rendszerének kialakítására.

Fájlok elhelyezése

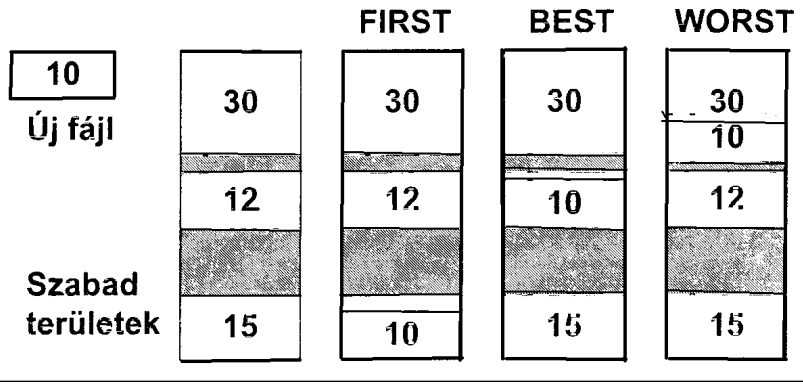
- A partíció elején, az ún. Szuperblokk (pl. FAT esetén a 0. blokk) leírja a rendszer jellemzőit.
- Általában következik a helynyilvántartás (FAT, láncolt listás nyilvántartás).
- Ezutána a könyvtárszerkezet (inode), a könyvtár bejegyzésekkel, fájl adatokkal. (FAT16-nál a könyvtár előbb van, majd utána a fájl adatok.)
- Hova kerüljön az új fájl?
- Milyen módszert válasszunk?

Fájl elhelyezkedési stratégiák:

- ***Folytonos tárkiosztás*** (azaz folyamatos elhelyezésű)
  - Legegyszerűbb, és a későbbi használat szempontjából is kedvező, ha a fájl blokkjai folytonosan helyezkednek el. Az operációs rendszer felderíti, hogy mely szabad területek alkalmasak az állomány befogadására. Gyakran több ilyen terület is van, dönteni kell.
    1. *First fit*
       - A legelső alkalmas helyre rakjuk, ahová befér.
       - A legelső alkalmas hely felhasználása a leggyorsabb ugyan, de nem mindig kedvező.
       - Lehet, hogy az éppen elhelyezendő fájl számára ha szűkebben is, de máshol is lett volna hely, és a következő állomány egyetlen lehetőségét vettük el.
    2. *Best Fit* – arra a helyre, ahol a legkevesebb szabad hely marad.
       - Megkereshetjük azt a szabad tartományt, melynek mérete csak minimálisan haladja meg az állomány méretét, azaz a fájl a legjobban illeszkedik (Best Fit).
       - A módszer hátránya, hogy számításigényes, végig kell nézni az összes szabad helyet, mielőtt a döntésre sor kerülne.

3. *Worst Fit* – Arra a helyre illesztjük, ahol a legtöbb szabad hely marad.
   - A létező legnagyobb helyre is gondolhatunk, melyben az elhelyezett állomány mellett a legtöbb szabad hely marad, azaz a legrosszabbul illeszkedik (Worst Fit).
   - E mellett az elhelyezés mellett a legnagyobb az esélye annak, hogy egy újabb fájl is befér majd a fennmaradó szabad blokkokba.

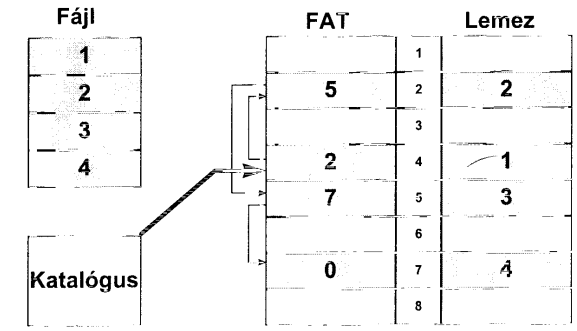
- Megjegyzés: mindegyik veszteséges.



9. ábra. Folytonos tárkiosztási stratégiák

– ***Láncolt elhelyezkedés*** (azaz láncolt tárolás)

- Diákról:
  - Nincs veszteség (csak a blokkméret).
  - A fájl adatai egy láncolt blokk listában vannak. (Így az utolsó blokk elérése lassú.)
  - Szabad-foglalt blokkok: File Allocation Table, FAT – Nagy méretű lehet, és a FAT mindig a memóriában van.



10. ábra. Láncolt elhelyezés

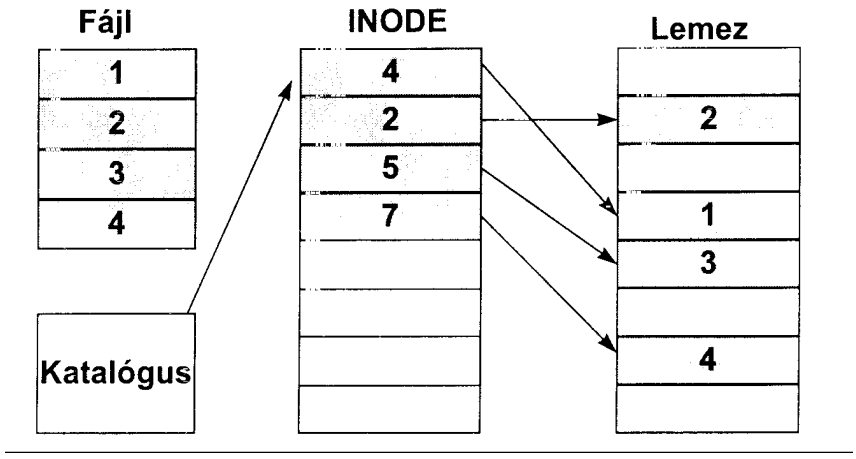
- Bővebben:
  - Ha lemondunk az egyszerű nyilvántartásról, és egy újabb táblázatot is igénybe veszünk, sokkal rugalmasabb módszerhez jutunk. A katalógusban itt is csak a fájl kezdő blokkjának címét kell megadni, az összes többi adatot a fájl elhelyezési tábla (File Allocation Table - FAT) tartalmazza. A táblázatnak ugyanannyi eleme van, mint ahány blokk a lemezen és minden rekesz tartalma a fájl következő blokkjára mutató sorszám, ha van következő blokk, 0, ha ez volt az utolsó blokk.
  - Az eljárás mentes a folytonos elhelyezésnél tapasztalható töredezés veszélyétől. A szabad helyek az utolsó blokkig kihasználhatók, a fájlok mérete a lemez fizikai határáig növekedhet. Az üres helyek keresésére sem kell időt és energiát szánni, az első szabad blokknál lehet kezdeni. Ennél a módszernél nem probléma a fájl méretének növelése, illetve csökkentése. Hátrány azonban, hogy inkább a szekvenciális hozzáférést támogatja, azaz, ha például egy

fájl 10. Blokkját akarjuk elérni, ahhoz végig kell követni az első 9 blokk helyét, ami lassú lehet.

- A FAT meglehetősen nagy táblázat lehet, és szerepe döntő. Ezen kívül nagyon sűrűn kell használni, ezért a memóriában kell tartani, ami még szűkösebb, mint a háttértár. A FAT sérülése esetén nagy valószínűséggel a kettészakadt fájlt visszaállítani nem lehet. A láncolási módszert alkalmazó operációs rendszerek, például a DOS, a Windows és a NetWare a biztonság kedvéért két ilyen táblázatot tartanak fenn.

– ***Indextáblás elhelyezés***

- Rugalmas elhelyezési lehetőségekhez jutunk, ha egy óriási táblázat helyett sok kicsit használunk, minden állományhoz külön. A katalógus tartalmazza a fájlhoz tartozó kicsi táblázat (un. *inode*) címét, a kicsi táblázat pedig a fájl blokkjainak a címét (ugyanis egy fájl gyakran több blokkban van.
- A könyvtár katalógus a file node-ok címét tartalmazza.
- Az *inode* cím mutat a fájl adatokra.
- Megjegyzés: A módszer előnye, hogy az elhelyezési információ gyorsan elérhető, kevésbé sérülékeny a tárolás is, hátránya, hogy valamiféle becslésre van szükség, hogy mekkorák lesznek a fájlok. Mekkora indextáblát kell fenntartani? Ha túlságosan nagy a tábla, pazarló, ha túlságosan kicsi, akkor ez határozná meg a maximális fájlméretet, ami megengedhetetlen. A Unix operációs rendszer kombinált módszer alkalmazásával oldotta meg.



11. ábra. Indexelt elhelyezés

Könyvtárak megvalósítása

– A fő funkciója, hogy a névből meghatározza az adatok helyét. (keresés: lineáris, hahs táblás, cacheelt).

– A könyvtárbejegyzés tartalmazhatja a:

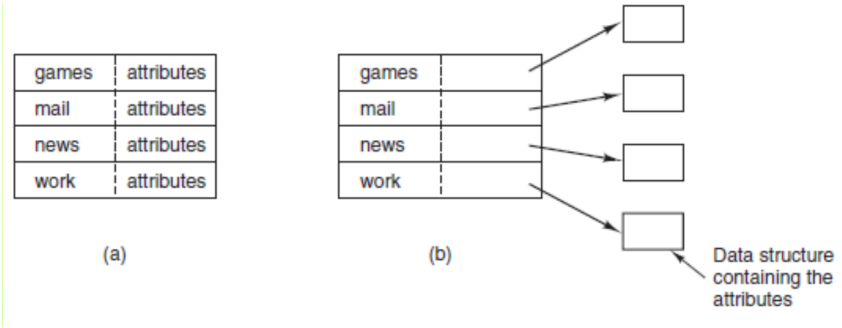
- A címét a teljes fájlnak (folyamatos elhelyezésnél).
- Az első blokk címét (láncolt listánál).
- Az i-node számot.

– Hogyan tárolja az attribútumokat?

- Bejegyzés hossza azonos, rögzített.
- I-node-okban.

Fájlnév tárolás

– Korábban: fix hosszúságú (8+3)



12. ábra.

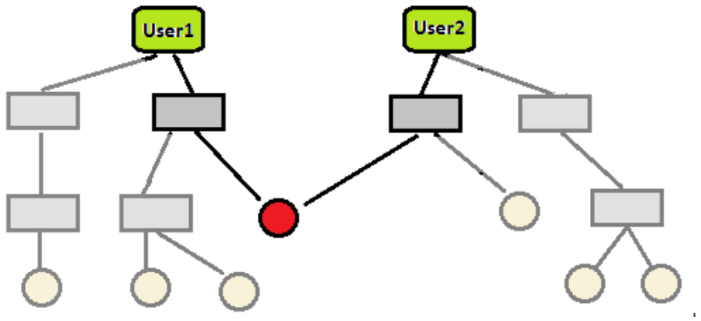
- Ma: általában max 255 karakter hosszú
  - Helytakarékos megoldások:
    - Különböző hosszágú bejegyzések. Az első helyen a bejegyzés hossza, majd az attribútumok, majd a név.
    - Egyforma hosszú bejegyzések, mutató a fájlnévre.

Diszk kvóták

- A felhasználói felnyitáskor
  - Megnyitja a fájl táblát a memóriában
  - A kvóta táblát a memóriában.
- Ha új blokkot foglal → változik a kvóta tábla
- Bejelentkezéskor ellenőrzés
  - Szoft limit átlépésekor bejelentkezhet (kivéve ha túllépte a lehetséges figyelmeztetések számát).
  - Hard limit átlépésnél be sem jelentkezhet.

Megosztott fájlok

- Megoldható, hogy mindenki számára minden változás látszódjon!
- Két módszer használatos:
  - A fájl blokkjai egy struktúrában találhatóak és erre mutat a könyvtárbejegyzés (nem a blokkok maguk vannak felsorolva a bejegyzésben). A user1 és user2 ugyanarra a struktúrára mutat. (Mi történik törlésnél?)
  - Új link fájl létrehozásával (symbolic link, lassú elérés).



13. ábra.

Mentések

- ***Fizikai mentés*** (Full backup): mindent lemásol.
  - Előny: simple, quick.
  - Hátrány: felesleges mentés, pl.: szabad blokkok.

- ***Logikai mentés***: csak a módosítottat menti.
  - Előny: igény esetén egy adott fájl vagy könyvtár is visszaállítható nem csak az egész.
  - Hátrány: bonyolult algoritmus.
  - Algoritmusa (Az UNIX rendszerekben gyakran használt algoritmus.)
    1. Minden módosított fájl és miniden könyvtár megjelölése.
    2. Eltávolítjuk azokat a könyvtár jelöléseket, amelyben (vagy amelyek alkönyvtáraiban) nem volt módosítás.
    3. Mentjük a megjelölt könyvtárakat attribútumokkal.
    4. Mentjük a megjelölt fájlokat attribútumokkal.
  - Kérdései
    - A szabad blokk lista nem fájl (nincs mentve). (Visszaállítható, hiszen a komplementere a foglaltnak.)
    - Linkek: minden könyvtárban, ahol a módosított fájl állományra volt link, azt vissza kell állítani.
    - A fájlok lyukakat tartalmazhatnak (pl. seek + write). Visszaállításnál a nem használt helyeket nem kell lefoglalni!
    - Nem valófi fájlokat pl. nevesített csővezetékeket nem kell menteni.
- ***Vegyes használat***
  - Időnként fizikai mentés
  - Sűrűbben logikai mentés
- ***Visszaállítás***: fizikai mentés, első logikai mentés, ..., utolsó logikai mentés.

Fájl, könyvtár műveletek

- Fájl
  - Megnyitás
  - Műveletek: írás, olvasás, hozzáfűzés
  - Lezárás
- Adatok
  - Bináris: bájt sorozat.
  - Szöveges: karakter sorozat
- Elérés módja: szekvenciális, random
- Könyvtár műveletek: létrehozás, tartalom listázása, állomány törlése.

Fájlrendszer típusok

- Merevlemezen alkalmazott fájlrendszer: FAT, NTFS, EXT2FS, XFS, stb.
- Szalagos rendszereken (első sorban backup) alkalmazott fájlrendszer: tartalomjegyzék, majd a tartalom szekvenciálisan.
- CD, DVD, Magneto-opto Disc fájlrendszere: CDFS, UDF (Universal Dick Format), kompatibilitás.
- RAM lemezek (ma már kevésbé használtak)
- FLASH memória meghajtó (FAT32)
- Hálózati megható: NFS.
- Egyéb pszeudo fájlrendszerek: Zip, tar.gz, ISO.

Naplózott fájlrendszer (journaling file system): A fájlrendszer nyilvántartást vezet a szándékozott változtatásokról (pl. fájl törlése). Sérülés, áramszünet, stb. esetén ha hiba következik be az ezáltal könnyen helyreállítható. Nagyobb erőforrás idényű de jobb a megbízhatósága.

Fájlrendszer támogatás

- Mai oprációs rendszerek "rengeteg" típust támogatnak, pl. Linux 2.6 kernel több mint 50-et.
- Fájlrendszer csatolása
  - Mount, eredményeként a fájlrendszer állományok elérhetők lesznek.

    - Automatikus csatolás (pl. USB drive)
    - Kézi csatolás (Linux, mount parancs)
- Külön névtérben való elérhetőség (Windows): A, B, C, ...
- Egységes névtér (UNIX)

Különböző fájlrendszerek együttes használata egy gépen: Lehetséges több különböző fájlrendszert használni ugyanazon a gépen. Pl.: Windows (NTFS, FAT-32, UDF(DVD? CD), stb.), UNIX (ext2, ext3, UDF, stb.).

Alkalmazás – Diszk kapcsolat

- Réteges felépítés
    - Alkalmazói szint
        - Az alkalmazás, fejlesztői könyvtárak segítségével megoldja a lemezen tárolt adatok írását-olvasását.
        - Szöveges, bináris fájlműveletek.
    - Operációs rendszer szint
        - Fájlrendszer megvalósítás: elérhetőség, jogosultságok.
        - Kötetkezelő (volume manager)
        - Eszközmeghajtó (device driver): BIOS-ra alapozva
    - Hadver eszköz szintje: I/O meghajtó, IDE, STA, stb.

Fájlrendszerek

- ***FAT*** (File Allocation Table): a FAT tábla a lemez foglalási térképe, annyi eleme van ahány blokk a lemezen. Lévén egy fájl jó eséllyel több blokkon helyezkedik el (akár nem is szekvenciálisan), így a FAT a katalógusában a fájl adatok (név stb) mellett csak a fájl első blokk sorszámát tárolja el. A FAT blokk azonosító mutatja a fájlhoz tartozó következő blokk címét, ha nincs ilyen akkor az értéke `FFF`. A fájl utolsó módosítási idejét is tárolja.

  Töredezettségmentesítés szükséges (azaz újra kell rendezni a blokkokat), amennyiben a fájlok blokkjai nagyon szétszórva helyezkednek el a disc-en. Ennek hiányában jelentős lelassulnak az I/O műveletek.

- ***NTFS*** (New Technology File System): A FAT-tel szemben számos újdonsággal/javulással rendelkezik: kifinomult biztonsági beállítások, POSIX támogatás, fájlok és mappák tömörítése, felhasználói kvóta kezelés (azaz egyes felhasználók csak a disc bizonyos részeihez férhessenek hozzá), ezen felül az NTFS csak klasztereket (más néven blokkokat) tart nyilván, szektort nem.

  Az NTFS partíció a Master File Table (MFT) táblázattal kezdődik, mely (hasonlóan a FAT-hez) az egy adott fájlhoz tartozó klasztereket tárolja. Egy adott fájlhoz 16 attribútumot rendel (pl. név). Egy attribútum max 1 kb lehet, ha ez nem elég, akkor egy attribtum mutat a folytatásra. Amennyiben a fájl maga kisebb mint 1 kb, akkor belefér egy attribútumba, ezáltal közvetlenül elérhető lesz az MFT-ből. A biztonság kedvéért két MFT-t is vezet az operációs rendszer, amennyiben az egyik tönkremegy, a másikat tudja használni. Töredezettségmentesítés szükséges. (ld. 14. ábra)

- ***UNIX könyvtárszerkezet***: Indextáblás megoldás, boot blokk (erről majd később bővebben) után a partíció un. szuperblokkja következik (mely leírja a rendszer tulajdonságiat, pl. a méretét, mekkora legyen egy blokk mérete, és még sok egyéb), ezt követi a szabad terület leíró rész (i-node tábla, majd gyökérkönyvtár bejegyzéssel). Moduláris elhelyezés, gyorsan elérhető az információ, sok kicsi táblázat, ez alkotja a katalógust. Egy fájlt egy i-node ír le.

14. ábra. NTFS partíció felépítése.

5. Előadás – Folyamatok (Processes)

5.1. Folyamatok (Processes)

Folyamatok modellje

- Folyamat (process): futó program a memóriában (kód + I/O adatok + állapot).
- Egyszerre hány folyamat működik?
  - Single task – Multi Task
  - Valódi Multi task?
- Szekvenciális modell
- Processzek közti kapcsolás: multiprogramozás.
- Egy időben csak egy folyamat aktív.

Valódi-e a Multi Task? - A Multi Task az, amikor több process párhuzamosan fut. Ezt egy darab processzormag nem tudja megvalósítani, ennek csak a látszatát tudja kelteni a processzek közötti kapcsolgatással. Egy időben csak egy folyamat aktív.

Egy feladat-végrehajtáshoz egy processzorra, külön rendszer memóriára és egy I/O eszközre van szükség.

Rendszer modell

- 1 processzor + 1 rendszer memória + 1 I/O eszköz = 1 feladat végrehajtás
- Interaktív (ablakos) rendszerek, több program, több processz fut:

- ***Környezetváltásos rendszer:*** Csak az előtérben lévő alkalmazás fut
- ***Kooperatív rendszer:*** Az aktuális process bizonyos időközönként, vagy időkritikus műveletnél önként lemond a CPUról (Win3.1).
- ***Preemptív rendszer:*** Az aktuális process-től a kernel bizonyos idő után elveszi a vezérlést, és a következő várakozó folyamatnak adja. (Ma tipikusan ilyen rendszereket használunk.)
- ***Real time rendszer:*** Igazából ez is preemtív rendszer (különbségek később).

Folyamatok létrehozása

- Ma tipikusan preemtív rendszereket használunk (igazából a valós idejű is az).
- Több folyamat él, aktív.
- ***Folyamatok létrehozásának okai:*** Például boot, folyamatot eredményező rendszerhívás(fork, execve), felhasználói kérés (parancs`&`), nagy rendszerek kötegelt feladata.
  - Rendszer inicializálás (boot)
  - Folyamatot eredményező rendszerhívás
    - Másolat az eredetíről (fork)
    - Az eredeti cseréje (execv)
  - Felhasználói kérés (parancs&)
  - Nagy rendszerek kötelgelt feladatai
- Előtérben futó folyamatok
- Háttérben futó folyamatok (démonok)

Folyamatok kapcsolata

- A folyamatok között szülő-gyerek kapcsolat lép fel.
- Az így felírható folyamatfában
  - Egy folyamatnak egy szülője van.
  - Egy folyamatnak több gyereke is lehet.
  - (Fork utasítás – vigyázni kell a használatával.)
- Linuxon, az első folyamat amelyik elindul az az `Init`, ez rendelkezik az 1. számú ID-vel. (C-ben egy process ID-jét lekérdezhetjük a `getpid()` (Get Process ID) függvénnyel)
- ***Reinkarnációs szerver:*** Meghajtó programok, kiszolgálók elindítója. Ha elhal az egyik, akkor azt újraszüli, reinkarnálja.

Folyamatok befejezése

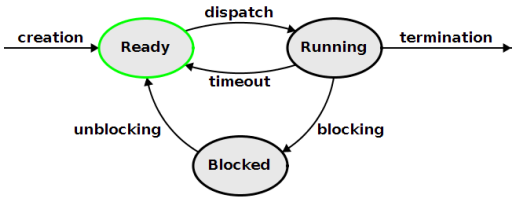
- Folyamat elindulása után a megadott időkerten belül végzi (elvégzi) a feladatát.
- A befejezés okai:
  - Önkéntes befejezés:
    - Szabályos kilépés (exit, return)
    - Korai terminálás a program által felfedezett hiba miatt, stb. (szintén pl. return utasítással).
  - Önkéntelen befejezés: Akkor következik be, ha a folyamat pl. illegális utasítást hajt végre (túlcímez), végzetes hibát vét (nullával való osztás). A befejezés végbemehet külső segítséggel, vagy a felhasználó által.

Folyamatok állapota

- ***Folyamat:*** önálló programegység, utasításszámlálóval, veremmel stb. Általában nem függetlenek, más folyamatok eredményétől függ a tevékenységük. Három állapotban lehet: futó, futásra kész, vagy blokkolt. (ld. 15. ábra)

– Általában nem függetlenek a folyamatok: egyik-másik eredményétől függ a tevékenység.

– Egy folyamatnak három állapota lehet:
  • Futó
  • Futásra kész: ideiglenesen leállították, arra vár, hogy az ütemező CPU időt adjon a folyamatnak.
  • Blokkolt: ha logikailag nem lehet folytatni a tevékenységet, mert pl. egy másik eredményére vár.



15. ábra. Folyamatok állapotai

Folyamatok megvalósítása

– A processzor "csak" végrehajtja az aktuális utasításokat.
– Egyszerre egy folyamat aktív.
– A processzor folyamatokról nem tud (csak utasításokat hajt végre)
  • Ezért mindent meg kell őrizni(utasításszámlálót, regisztereket, nyitot fájl infókat).
  • Ezeket az adatokat az úgynevezett folyamat leíró táblában tároljuk (processz tábla, processz vezérlő blokk).
– I/O megszakításvektor.

Folyamatok váltása

– A következők kezdeményeik: időzítő, megszakítás, esemény, rendszerhívás kezdeményezés.
– Ütemező elmenti az aktuális folyamat jellemzőket a folyamatleíró táblába.
– Betölti a következő folyamat állapotát, a processzor folytatja a munkát.
– Nem lehet menteni a gyorsító tárakat
  • A gyakori váltás miatt többlet erőforrást igényelne.
  • A folyamat váltási idő "jó" megadása nem egyértelmű.

Folyamatleíró táblázat (Process Control Block, PCB): A rendszer inicializálásakor jön létre. Már indításakor is tartalmaz 1 elemet, a rendszerleíró már bent a rendszer indulásakor bekerül. Tömbszerű szerkezete van (PID alapján). Egy sorában egy összetett processzus adatokat tartalmazó struktúra található. Egy folyamat fontosabb adatai: azonosítója, neve, tulajdonosa, csoportja stb.

Szálak (threads)

– Tipikus helyzet: egy folyamat – egy utasítássorozat – egy szál.
– Néha szükséges lehet, hogy egy folyamaton belül "több utasítássorozat" legyen
  • Szál: egy folyamaton belüli különálló utasítási sor.
  • Gyakran "leghtweight process"-nek nevezik.
– Szálak: egy folyamaton belül több egymástól "független" végrehajtási sor.
  • Egy folyamaton belül egy szál
  • Egy folyamaton belül több szál → ha egy szál blokkolódik, a folyamat is blokkolva lesz!
  • Száltáblázat
– A folyamatnak önálló címtartománya van, szálnak viszont nincs.
– Általában egy folyamat egy utasítássorozatból áll. Azonban néha szükség lehet egyszerre több, egymástól független utasítássorozatra is egy folyamaton belül. Ezeket az utasítássorozatokat szálaknak, thread-eknek vagy lightweight process-nek hívjuk.

– Minden folyamat egy szál, de nem minden szál folyamat. Csak egy folyamat rendelkezik címtartománnyal, globális változókkal, megnyitott file leírókkal, gyermek folyamatokkal. Mind a folyamatok, mind a szálak rendelkeznek utasításszámlálóval, regiszterrel, veremmel.

Szálproblémák:

– Fork: biztos, hogy a gyerekben kell több szál, ha a szülőben több van.
– Fájlkezelés: egy szál lezár egy fájlt, miközben még a másik még használná!
– Hibakezelés: globális hibajelző (`errno` C-ben egy globális változó, ha több szál akarja használni, könnyen galiba lehet).
– Memóriakezelés
– Lényeges: A rendszerhívásoknak kezelni kell tudni a szálakat (azokat hívásokat, melyek ezt tudják teljesíteni, thread-safe hívásoknak is nevezzük).

5.2. Folyamatok kommunikációja (Process communication)

Folyamatok kommunikációja

– IPC (Inter Process Comunication): Programozási interfészek egy halmaza, melyek lehetővé teszik a processzek kommunikációját. Lévén gyakran szükség van egy program futtatásakor sok másik programra is, az ezek közötti kommunikációt is biztosítani kell. Példa.: pipe, szemafor.

– Három területre kell megoldást találni:
  • Két vagy több folyamat ne keresztezze egymást kritikus műveleteknél.
  • Sorrend figyelembevétel (bevárás). Nyomtatás csak az adatok előállítása után lehetséges.
  • Hogy küldhet egy folyamat információt, üzenetet egy másiknak.

– Szálak esetén is mindhárom terület ugyanúgy érdekes, csak az információküldés az azonos címtartomány miatt egyszerű.

"Párhuzamos" rendszerek

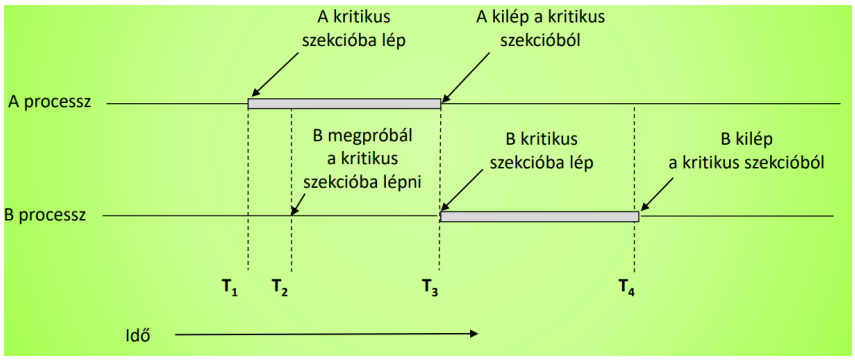
– Ütemező a folyamatok gyors váltogatásával "teremt' párhuzamos végrehajtás érzetet.
– Többprocesszoros rendszerek: több processzor egy gépben, nagyobb teljesítmény, megbízhatóságot általában nem növeli.
– Klaszterek (fürt, claster): megbízhatóság növelése elsősorban.
– Kulcskérdés: a közös erőforrások használata.

Közös erőforrások

– Avagy, amikor két vagy több folyamat ugyanazt a memóriát használja...
– ***Versenyhelyzet:*** két vagy több folyamat közös memóriát ír vagy olvas. (A végeredmény a futási időpillanattól függ!) (Nehezen felderíthető hibát okoz.)
– Megoldás: Módszer, ami biztosítja, hogy a közös adatokat egyszerre csak egy folyamat tudja használni.

Kölcsönös kizárás

– ***Kritikus programterület:*** Az a programterület, szekció, rész, ahol több process (folyamat) vagy szál ugyanazt az erőforrást (memóriát) használja.
– A jó kölcsönös kizárás az alábbi feltételeknek felel meg:
  • Nincs két folyamat egyszerre a kritikus szekciójában.
  • Nincs sebesség, CPU paraméterfüggőség.
  • Egyetlen kritikus szekción kívül levő folyamat sem blokkolhat másik folyamatot.
  • Egy folyamat sem vár örökké, hogy a kritikus szekcióba tudjon belépni.



16. ábra. A megkívánt kölcsönös kizárás viselkedése

Kölcsönös kizárás megvalósítása

- Megszakítások tiltása: Belépéskor az összes megszakítás letiltásra kerül, kilépéskor engedélyeződnek. A kernel pl. ezt (is) használja, azonban nem épp szerencsés megoldás.
- Osztott, un. zárolós változó használata: Egy több folyamat által is látható változó értéke adja meg, hogy a kritikus szekció foglalt-e. Pl. ha ez a változó 0, a kritikus szekcióban egyetlen folyamat se fut és 1, amennyiben egy program fut benne. Ennek az a hátránya, hogy ha két folyamat egyszerre akar belépni, akkor lehetséges hogy mindketten befférnek, mielőtt az a változó 1-re vált.
- Szigorú váltogatás: Több folyamatra is általánosítható. A kölcsönös kizárás feltételeit teljesíti, azonban felléphet az, hogy kritikus szekción kívül lévő folyamat blokkol egy másikat. Például, tekintsük az alábbi kódrészleteket, ahol `next` mind a két folyamat számára látható egész változó:

```
//process 0
while(1)
{
    while(next != 0) {}
    critical_section();
    next = 1;
    non_critical_section();
}
```

```
//process 1
while(1)
{
    while(next != 1) {}
    critical_section();
    next = 0;
    non_critical_section();
}
```

Amennyiben `process 1` lassú, nem kritikus szekcióban van, és a `process 0` gyorsan belép a kritikus szekcióba, majd befejezi a nem kritikus szekciót is, akkor a végtelen ciklus következő futásakor nem tud újra belépni a kritikus szekcióba, még akkor sem, ha `process 1` még mindig a nem kritikus szekcióban van. Ezáltal `process 0` saját magát blokkolja.

G.L. Peterson javítása (a szigorú kizáráson): A kritikus szekció előtt minden folyamat meghívja a belépés, majd utána kilépés fv-t.

```
int turn;
int wants_to_enter[2]; // Most csak két processre mutatja be az algoritmust.

// Legyen ez process 0.
```

```
while(1)
{
    enter_critical(0);
    critical_section();
    exit_critical(0);
    non_critical_section();
}

// process 1 hasonlóan
```

Az `enter_critial` és `exit_critical` implementációja:

```
void enter_critical(int entering_process_id)
{
    // Megállapítjuk, hogy ez process 0 vagy process 1-e.
    int other_process = 1 - entering_process_id;

    // Jelezzük, hogy a process be akar lépni a kritikus szekcióba
    wants_to_enter[entering_process_id] = 1;

    // Jelezzük, hogy a soron következő process a belépni kívánó process legyen
    turn = entering_process_id;

    while(turn == entering_process_id && wants_to_enter[other_process])
    {
        // Aktív várakozás.
        // Amíg a másik process is be akar lépni, és még nem jelezte hogy őlesz
        // a soron következő, vagy pedig a másik process nem fejezte be a futását
        // (a wants_to_enter[other_process_id] 0-ra állításával) addig ez a process
        // várakozik.
    }
}

void exit_critical(int exiting_process_id)
{
    // Jelezzük, hogy ez a process már nem akar belépni a kritikus szekcióba.
    wants_to_enter[exiting_process_id] = 0;
}
```

Megfigyelhető, hogy ha két sort felcserélünk az `enter_critical`, már hibás működéshez vezethez az algoritmus:

```
void enter_critical(int entering_process_id)
{
    int other_process = 1 - entering_process_id;
    turn = entering_process_id; // ez a sor
    wants_to_enter[entering_process_id] = 1; // és ez a sor meg van cserélve!
    while(turn == entering_process_id && wants_to_enter[other_process]) {}
}
```

Tegyük fel, hogy ismét `process 0` és `process 1` próbál belépni a kritikus szekcióba. Az ütemező `process 0` utasításait hajtja először végre, és eljut vele az `enter_critical` függvényen belül a `turn = entering_process_id;` sorhoz, így annak értékét 0-ra állítja.
Ismét tegyük fel, hogy az ütemező ekkor átvált `process 1`-re, mely átállítja `turn` értékét 1-re, nem várakozik (hisz `wants_to_enter[0]` értéke még mindig 0), és belép a kritikus szekcióba.
Ismételten tegyük fel, hogy ütemező átvált `process 0`-ra. A ciklusfeltétel itt sem teljesül, hisz `turn` át lett állítva 1-re. Ennek hatására mind `process 0`, mind `process 1` bekerült a kritikus szekcióba.

Tevékeny várakozás

- **TSL** (Test and Set Lock): megszakíthatatlan atomi művelet, pl. Peterson 1x-bb változata
- A korábbi Peterson megoldás is, a TSL használata is jó, csak ciklusban várakozunk.
- A korábbi megoldásokat, tevékeny várakozással (aktív várakozás) megoldottnak hívjuk, mert a CPU-t "üres" ciklusban járatjuk a várakozás során!
- A CPU időt pazarolja:(
- A CPU pazarlás helyett jobb lenne az, ha a kritikus szekcióba lépéskor blokkolna a folyamat, ha nem szabad belépnie!

Alvás-ébredés: Peterson megoldásában a folyamatok végtelen ciklusban várakoznak, mely pazarolja a processzor időt. Erre megoldás lehet, ha blokkoljuk a várakozó folyamatot, és felébresztjük amikor futhat. Ezt hívjuk alvás-ébredésnek.

- sleep-wakeup, down-up, stb.
- Különböző paramétermegadással is implementálhatók.
- Tipikus probléma: Gyártó-Fogyasztó probléma.

Gyártó-fogyasztó probléma

- Korlátos tároló problémaként is ismert.
- Pl.: Pék-pékség-vásárló háromszög.
  - A pék süti a kenyeret, amíg a pékség polcain van hely.
  - Vásárló tud venni, ha pékség polcain van kenyér.
  - Ha tele van kenyérrel a pékség, akkor "a pék elmegy pihenni".
  - Ha üres a pékség, akkor a vársárló várakozik a kenyérre.

Gyártó-Fogyasztó probléma egy megvalósítása

- Pék folyamat

```
#define N 100
int hely = 0;
void pék()
{
    int kenyér;
    while(1)
    {
        kenyér = új_kenyér();
        if(hely == N) alvás();
        polcra(kenyér);
        hely++;
        if(hely == 1) ébresztő(vásárló);
    }
}
```

- Vásárló folyamat

```
void vásárló()
{
    int kenyér;
    while(1)
    {
        if(hely == 0) alvás();
        kenyér = kenyeret();
```

```
            hely--;
            if(hely == N-1) ébresztő(pék);
            megesszük(kenyér);
        }
    }
```

Pék-Vásárló probléma

- A "hely" változó elérése nem korlátozott, így ez okozhat versenyhelyzetet.
  - Vásárló látja, hogy a hely 0 és ekkor az ütemező átadja a vezérlést a péknek, aki süt egy kenyeret.Majd látja, hogy a hely 1, ébresztőt küld a vásárlónak. Ez elveszik, mivel a vásárló még nem alszik.
  - Vásárló visszakapja az ütemezést, a helyet korábban beolvasta, az 1, megy aludni.
  - A pék az első után megsüti a maradék N-1 kenyeret és ő is aludni megy!
- Lehet ébresztőt bittel javítani, de több folyamatnál a probléma nem változik.

Szemafor: egyfajta „kritikus szakasz védelem". A szemafor maga egy egész típusú változó, mely „tilosat mutat", ha értéke 0, és 0-nál nagyobb értéket, ha a folyamat beléphet a kritikus szekcióba. A szemafor változó módosítása csakis atomi (megszakíthatatlan) műveletekkel történhet, pl. rendszerhívással. Ez implikálja, hogy kizárólag felhasználói szinten nem lehet megvalósítani. (Mi a „baj" a szemaforokkal? - könnyen el lehet rontani a kódolás során).
Két művelet tartozik hozzá:

- Ha beléptünk, csökkentjük a szemafor értékét.(down)
- Ha kilépünk, növeljük a szemafor értékét (up).
- Ezeket Dijkstra P és V műveletnek nevezte.

Megjegyzés:

- Az implementáció során apró hibák is nehezen javítható hibát okozhatnak.
- Például: up-down felcserélése, vagy valamelyik elhagyása.

Elemi művelet: Megszakíthatatlan művelet, mellyel megakadályozható a versenyhelyzet kialakulása. Pl. ilyennek kell lennie a szemafor változó ellenőrzésének, módosításának. Ez garantálja, hogy ne alakuljon ki versenyhelyzet.

Szemafor megvalósítások

- Up, Down műveletknek atominak kell lenni. (Nem blokkolhatók!)
- Hogyan?
  - Op. Rendszerhívással, felhasználói szinten nem biztosítható.
  - Művelet elején például letiltunk minden megszakítást.
  - Ha több CPU van akkor az ilyen szemafort védeni tudjuk a TSL utasítással.
- Ezek a szemafor műveletek kernel szintű, rendszerhívás műveletek.
- A fejlesztői környezetek biztosítják. (Ha mégsem, gáz van...)
- Probléma: apró elírások nehezen felderíthető problémákhoz vezethetnek.

Gyártó.fogyasztó probléma megoldása szemaforokkal

- Gyártó (pék) függvénye

```
typedef int szemafor;
szemafor szabad = 1;  //Bináris szemafor, 1 mehet tovább, szabad a jelzés
szemafor üres = 0; tele = 1;  //üres a polc, ez szabad jelzést mutat
void pék()
{
    int kenyér;
    while(1)
    {
        kenyér = pék_süt();
        down(&üres);  //üres csökken, ha előtte > 0, mehet tovább
        down(&szabad)  //piszkálhatjuk-e a pékség polcát?
        kenyér_polcra(kenyér);  //igen, betesszük a kenyeret
        up(&szabad); //elengedjük a pékség polcát
        up(&tele);  //jelezzük a vásárlónak, hogy van kenyér
    }
}
```

– Fogyasztó (Vásárló) függvénye:

```
void vásárló()  //vásárló szemaforja a tele
{
    int kenyér;
    while(1)
    {
        down(&tele);  //tele csökken, ha előtte nagyobb mint 0, mehet tovább
        down(&szabad);   //piszkálhatjuk-e a pékség polcát?
        kenyér = kenyér_polcról();   //igen, levesszük a kenyeret
        up(&szabad);  //elengedjük a pélség polcát
        up(&üres); //jelezzük péknek, van hely, lehet sütni
        kenyér_elfogyasztása(kenyér);  //kiakasztja az álkapcsát és az egészet
            betömi egyben!!!
    }
}
```

Szemafor példa összegzés

– Szabad: kenyér polcot (boltot) védi, hogy egy időben csak egy folyamat tudja használni (vagy a pék, vagy a vásárló).
  • Kölcsönös kizárás
  • Elemi műveletek (up, down)
– Tele, üres szemafor: szinkronizációs szemaforok, a gyártó álljon meg, ha a tároló tele van, illetve a fogyasztó is várjon ha a tároló üres.

6. Előadás

6.1. Folyamatok kommunikációja

Monitor

– Hasonló a szemaforhoz, de itt eljárások, adatszerkezetek lehetnek. Egy időben csak egy folyamat lehet aktív a monitoron belül.

– Megvalósítása mutex (lásd: köv. fogalom) segítségével történik.

– Apró gond: mi van ha egy folyamat nem tud továbbmenni a monitoron belül? Erre jók az állapot változók (condition). Rajtuk két művelet végezhető – wait vagy signal.

– A monitoros megoldás egy vagy több VPU esetén is jó, de csak egy közös memória használatánál. Ha már önálló saját memóriájuk van a CPU-knak (dedikált memória) akkor ez a megoldás nem az igazi.

– Sokkal biztonságosabb, mint a szemafor használat.

– Egy vagy több CPU, de csak egy közös memóriahasználatnál jók!

– Ha a CPU-knak önálló saját memóriájuk van, akkor ez a megoldás nem az igazi...

– Példa:

```
Monitor veszélyes_zóna
    Integer polc[];
    Condition c;
    Procedure pék(x);
    ...
    End;
    Procedure vásárló(x);
    ...
    End;
End monitor;
```

– Monitotok tulajdonságai
  - Monitorban eljárások, adatszerkezetek lehetnek.
  - Egy időben csak egy folyamat lehet aktív a monitoron belül.
  - Ezt a fordítóprogram automatikusan biztosítja: Ha egy folyamat meghív egy monitor eljárást, akkor először ellenőrzi, hogy másik folyamat aktív-e
    - Ha igen, felfüggesztésre kerül.
    - Ha nem, akkor beléphet és végrehajthatja a kívánt monitor eljárást.

– Monitor megvalósítása
  - Mutex segítségével.
  - A felhasználónak nincs konkrét ismerete róla, de nem is kell.
  - Eredmény: sokkal biztonságosabb kölcsönös kizárás megvalósítás.
  - Apró gond: mi van, ha egy folyamat nem tud tovább menni a monitoron belül? (Pl.: a pék nem tud sütni, mert tele van a bolt?)
  - Megoldás: állapot változók (condition). (Rajtuk két művelet végezhető: wait, signal.)

Gyártó-fogyasztó probléma megvalósítása monitorral

– N elem.

– Monitor:

```
monitor Pék-Vásárló
condition tele, üres;
int darab;
kenyeret_plolcra_helyez(kenyér elem)
{
    if(darab == N) wait(tele);
    polcra(elem);
    darab++;
    if(darab == 1) signal(üres);
}
```

```
kenyér kenyeret_levesz_a_polcról()
{
    if(darab == 0) wait(üres);
    kenyér elem = kenyér_polcról();
    darab--;
    if(darab == N-1) signal(tele);
    return elem;
}
```

– Pék:

```
pék()
{
    while(1)
    {
        kenyér új;
        ú= kenyér_sütés();
        Pék-Vásárló.kenyeret_polra_helyez(új);
    }
}
```

– Vásárló:

```
vásárló()
{
    while(1)
    {
        kenyér új_kenyér;
        új_kenyér = Pék-Vásárló.kenyeret_a_polcról();
        lakoma(új_kenyér);
    }
}
```

Más megoldások

– Az előző úgynevezett Pidgin Pascal megoldás vázlat volt.
– C-ben nincs monitor. De C++-ben van: wait, notify.
– C#
  • Monitor osztály: Enter, TryEnter, Exit, Wait, Pulse (ez a notify megfelelője).
  • Lock nyelvi kulcsszó.
  • Példa: VC2008 párhuzamos slution, monitor projekt.

Mutex: A mutex egyik jelentése az angol mutual exclusion (kölcsönös kizárás) szóból ered. Programozástechnológiában párhuzamos folyamatok használatakor előfordulhat, hogy két folyamat ugyanazt az erőforrást (resource) egyszerre akarja használni. Ekkor jellemzően felléphet *versengés*. Ennek kiküszöbölésére a gyorsabb folyamat egy, az erőforráshoz tartozó mutexet zárol (ún. lock-ol). Amíg a mutex zárolva van (ezt csak a zároló folyamat tudja feloldani - kivéve speciális eseteket), addig más folyamat nem férhet hozzá a zárolt erőforráshoz. Így az biztonságosan használható. (Például nem lenne szerencsés, ha DVD-írónkat egyszerre két folyamat használná.)
Röviden *a mutex egy bináris szemafor.*

A szemafor és a mutex közti különbség: Az a különbség, hogy míg utóbbi csak kölcsönös kizárást tesz lehetővé, azaz egyszerre mindig pontosan csakis egyetlen feladat számára biztosít hozzáférést az osztott erőforráshoz, addig a szemafort olyan esetekben használják, ahol egynél több - de korlátos számú - feladat számára engedélyezett a párhuzamos hozzáférés.

Üzenetküldés: A folyamatok jellemzően két primitívet használnak: send (célfolyamat, üzenet) és receive(forrás, üzenet) – a forrás tetszőleges is lehet.

Nyugtázó üzenet: Ha a küldő és a fogadó nem azonos gépen van akkor szükséges egy úgynevezett nyugtázó üzenet. Ha ezt a küldő nem kapja meg (azaz úgy tűnik, hogy az üzenet nem érkezett meg), akkor ismét elküldi az üzenetet, ha a nyugta veszik el a küldő újra küld. Ismételt üzenetek megkülönböztetésére sorszámot használ.

Gyártó-fogyasztó probléma üzenetküldéssel

- A gyártó (pék) folyamata:

```
#define N 100  //a pékségben lévő helyek száma, a kenyeres polc mérete
void pék()  //pék folyamata
{
    int kenyér;  //"kenyér" elem tárolási hely
    message m;  //üzenet tároló helye
    while(1)  //folyamatosan sütünk
    {
        kenyér = kenyeret_sütünk();
        receive(vásárló, m);  //vásárlótól várunk egy üres üzenetet m-be
        m = üzenet_készítés(kenyér);
        send(vásárló, m); //elküldjük a kenyeret a vásárlónak
    }
}
```

- Fogyasztó-vásárló folyamata:

```
void vásárló() //vásárló folyamata
{
    int kenyér;  //"kenyér" elem tárolási hely
    message m; //üzenet tároló helye
    int I;
    for(i = 0; i < N; i++) send(pék, m); //N darab üres helyet küldünk a pé
        knek
    while(1) //a vásárlás is folyamatos
    {
        receive(pék, m);  //várunk a péktől egy kenyeret
        kenyér = üzenet_kicsomagolás(m);
        send(pék, m);  //visszaküldjük az üres kosarat
        kenyér_elfogyasztás(kenyér);
    }
}
```

Üzenetküldés összegzése

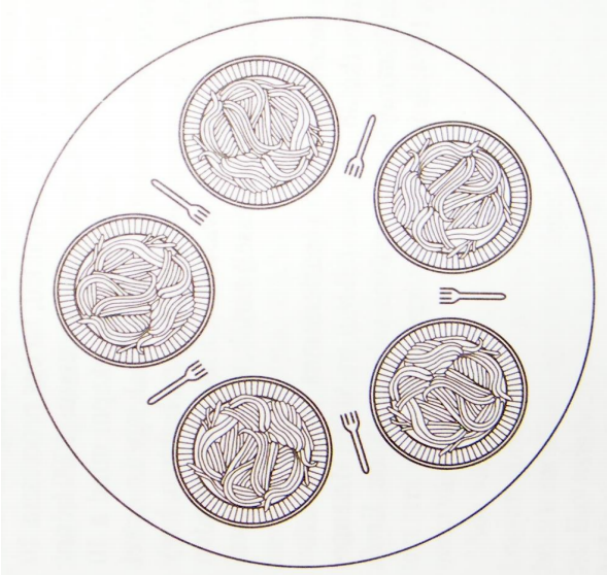
- Ideiglenes tároló helyek (levelesláda) létrehozása mindkét helyen.
- El lehet hagyni, ekkor ha send előtt van receive, a küldő blokkolódik, illetve fordítva.
  - Ezt hívják randevú stratégiának.
  - Minis 3 is randevút használ, rögzített méretű üzenetekkel.
  - Adatcső komunikáció hasonló, csak adatcsőben nincsenek üzenethatárok, ott csak bájtsorozat van.

Randevú stratégia: Üzenetküldésnél ideiglenes tárolók (más szóval temporális változók) is jönnek létre mindkét helyen (levelesláda). Ezt el lehet hagyni, ekkor a send előtt van receive, a küldő blokkolódik illetve fordítva. - ez a randevú stratégia.

6.2. Klasszikus IPC problémák

Étekő filozófusok esete

– 2 villa kell a spagetti evéséhez
– A tányér melelti villákra pályáznak.
– Esznek – gondolkoznak.
– Készítsünk programot, ami nem akad el!



17. ábra. Étkező filozófusok

1. Megoldás

– A megoldás hibája, hogy holtpont keletkezhet, ha pl. mindenki megszerzi a bal villát és vár a jobbra.
– Ha leteszi a bal villát és újra próbálkozik, még az se az igazi, hiszen folyamatosan felveszik a bal villát, majd leteszik. (Éhezés)

```
Void filozófus(int i)
{
    while(1)
    {
        gondolkodom();
        kell_villa(i);  //bal villa
        kell_villa((i+1)%N);  //jobb villa
        eszem();
        nemkell_villa(i);
        nemkell_villa((i+1)%N);
    }
}
```

– A holtpont problémára jó lehet megoldásként szemafort alkalmazni.

Olvasók-Írók probléma

– Adatbázist egyszerre többen olvashatják, de csak 1 folyamat írhatja.
– Erre a problémára is alkalmazható a szemafor megoldásként.

6.3. Ütemezés

Ütemező (scheduler): Ez dönti el, hogy melyik folyamat fusson (mikor jussanak processzoridőhöz, erőforrásokhoz) egy ütemezési algoritmus alapján.

Folyamat tevékenységei: Egy folyamat jellemzően két tevékenységet szokott végezni: vagy I/O igényt jelent be, azaz írni olvasni akar adott perifériára, vagy számításokat végez. Ez alapján megkülönböztetünk:

- Számításigényes feladat: hosszan dolgozik, keveset vár I/O-ra
- I/O igényes feladat: rövideket dolgozik, hosszan vár I/O-ra

Mikor váltsunk folyamatot?

- Biztosan van váltás:
  - Ha befejeződik egy folyamat.
  - Ha egy folymat blokkolt állapotba kerül (I/O vagy szemafor miatt).
- Általában van váltás:
  - Új folyamat jön létre
  - I/O megszakítás bekövetkezése: ezután jellemzően egy blokkolt folyamat, ami erre várt, folytathatja futását.
- Időzítő megszakítás
  - Nem megszakítható ütemezés
  - Megszakítható ütemezés

Ütemezések csoportosítása

- Minden rendszerre jellemzők:
  - Pártatlanság, mindenki hozzáférhet a CPU-hoz
  - Mindenkire ugyanazok az elvek érvényesek
  - Mindenki "azonos" terhelést kapjon
- Kötegelt rendszerek: áteresztőképesség, áthaladási idő, CPU kihasználtság.
- Interaktív rendszerek: válaszidő, megfelelés a felhasználói igényeknek.
- Valós idejű rendszerek: Határidők betartása, adatvesztés, minőségromlás elkerülése.

Az ideális ütemező tulajdonságai: Pártatlan, minden folyamat hozzáfér a CPU-hoz, minden folyamatra ugyanazok az elvek érvényesek, minden processzormag azonos terhelést kap. Az ütemezőket továbbá 3 kategóriákba sorolhatjuk, minden kategória az előbb említetteken kívül különböző tulajdonságokat próbál megvalósítani:

- Kötegelt rendszerek, ahol további szempont az áteresztőképesség, áthaladási idő és CPU kihasználtság.
- Interaktív rendszerek, ahol további szempont a válaszidő, megfelelés a felhasználói igényeknek.
- Valós idejű rendszerek, ahol további szempont a határidők betartása (erről majd később), adatvesztés, minőségromlás elkerülése.

Kötegelt rendszerek ütemezése (áteresztőképesség, áthaladási idő, CPU kihasználtság):

- ***Sorrendi ütemezés*** (First Come First Served, FCFS): Egy folyamat addig fut, amíg nem végez vagy nem blokkolódik. Egy pártatlan, egyszerű láncolt listában tartjuk a folyamatokat, ha egy folyamat blokkolódik, akkor a sor végére kerül.
  Nem szakítja meg a már futó, nem blokkolt folyamatokat.
- ***Legrövidebb feladat először*** (Shortest Job Next, SJN, Illés hibásan SJB-re rövidíti): Kell előre ismerni a futási időket, akkor optimális ha a kezdetben minden folyamat elérhető.
  Nem szakítja meg a már futó, nem blokkolt folyamatokat.

- ***Legrövidebb maradék futási idejű következzen***: Minden alkalommal, amikor egy újabb folyamat kerül be a sorba, újrarendezi a sort a hátralevő futási idő szerint. Amennyiben egy új process hátralévő futási ideje a legrövidebb a sorban, a sor elejére kerül.
  Megszakíthatja a már futó folyamatokat is.

- ***Háromszintű ütemezés:***
  - Bebocsátó ütemező: a feladatokat válogatva engedi be a memóriába.
  - Lemez ütemező: ha a bebocsátó sok folyamatot enged be és elfogy a memória, akkor lemezre kell írni valamennyit, meg vissza. - ez ritkán fut.
  - CPU ütemező: a korábban említett algoritmusok közül választhatunk.

Interaktív rendszerek ütemezése (válaszidő, megfelelés a felhasználói igényeknek):

- ***Körben járó ütemezés*** (Round Robin, RR): Minden folyamatnak ad egy időszeletet (valamekkora processzoridőt), aminek a végén, vagy blokkolás esetén jön a következő folyamat. Időszelet végén a lista végére kerül az aktuális folyamat, ami pártatlan és egyszerű. Gyakorlatilag egy listában tároljuk a folyamatokat és ezen megyünk körbe-körbe. A legnagyobb kérdés hogy mekkora legyen egy időszelet? Mivel a process átkapcsolás időigényes, ezért ha kicsi az időszelet sok CPU megy el a kapcsolgatásra, ha túl nagy akkor esetleg az interaktív felhasználóknak lassúnak tűnhet pl. a billentyűkezelés.

- ***Prioritásos ütemezés:*** Fontosság, prioritás bevezetése, a legmagasabb prioritású futhat. Prioritási osztályokat használ, egy osztályon belül az előbb említett Round Robin fut. Minden 100 időszeletnél újraértékeli a prioritásokat, jellemzően a magas prioritású folyamatok alacsonyabb prioritásra kerülnek, és viszont. Ennek hiányában nagy lenne a kiéheztetés veszélye.

- ***Többszörös sorok:*** Szintén prioritásos és Round Robinnal működik. A legmagasabb szinten minden folyamat 1 időszeletet kap, az alatta levő 2-t, az alatti 4-et, 16-et, 32-et, 64-et. Ha elhasználta a legmagasabb szintű folyamat az idejét egy szinttel lejjebb kerül.

- ***Legrövidebb folyamat előbb:*** Hasonlóan működik mint ahogy az a kötegelt rendszernél le volt írva, csak nem tudjuk előre a futási időt, így azt megbecsüljük, és az így kapott eredménnyel dolgozik az algoritmus.

- ***Garantált ütemezés:*** minden aktív folyamat arányos CPU időt kap, nyilván kell tartani, hogy egy folyamat már mennyi időt kapott, ha valaki arányosan kevesebbet akkor az kerül előre.

- ***Sorsjáték ütemezés:*** Mint az előző, csak a folyamatok között „sorsjegyeket" osztunk szét, az kapja a vezérlést akinél a kihúzott jegy van.

- ***Arányos ütemezés:*** Mint a garantált, csak felhasználókra vonatkoztatva.

Valós idejű rendszerek: (határidők betartása, adatvesztés,minőségromlás elkerülése)
Az idő a kulcsszereplő, garantálni kell adott határidőre a tevékenység, válasz megoldását. A programokat kisebb folyamatokra bontják.

- Hard Real Time (szigorú) abszolút nem módosítható határidők.
- Soft Real Time (toleráns) léteznek határidők, de ezek kis méretű elmulasztása tolerálható.

Szálütemezés:

- ***Felhasználói szintű szálak:*** A kernel nem tud róluk. A folyamat kap egy időszeletet, ezen belül a szálütemező döni el, melyik szál fusson. Gyorsan vált a szálak között. Lehetőség van alkalmazásfüggő szálütemezésre.
- ***Kernel szintű szálak:*** Kernel ismeri a szálakat, kernel dönt melyik folyamat szála következzen. Lassú váltás, két szál váltása között teljes környezetátkapcsolás kell.

7. Előadás

7.1. Beviteli/kiviteli (I/O) eszközök vezérlése

I/O eszközök:

- **Blokkos eszközök:** Adott méretű blokkokban tárolják az információt, egymástól függetlenül írhatók vagy olvashatók, illetve blokkonként címezhető. Ilyen pl. HDD, és a szalagos egység.
- **Karakteres eszközök:** Nem címezhető, karakterek (bájtok) egybefüggő sorozata.
- **Időzítő:** kivétel, nem blokkos és nem is karakteres.

I/O eszközök és a megszakítások: A megszakítások erősen kötődnek az I/O műveletekhez. Ha egy I/O eszköz „adatközlésre kész", ezt egy megszakításkéréssel jelzi. Alternatívaként, állapotfigyeléssel meg lehet ezt oldani.

- Állapotbittel: Általában az eszközöknek van állapotbitjük, jelezve, hogy az adat készen van. Ez nem egy hatékony megoldás. Tevékeny várakozás ez is, nem hatékony, ritkán használt.
- Megszakítás kezeléssel:
  - A hardver eszköz jelzi a megszakítás igényt az INTR hardver interrupt-al. Ez egy maszkolható interrupt (azaz figyelmen kívül hagyható).
  - CPU egy következő utasítás végrehajtás előtt, a tevékenységét megszakítja! (Precíz, imprecíz)
  - A kért sorszámú kiszolgáló végrehajtása. A kívánt adat beolvasása, a szorosan hozzátartozó tevékenység elvégzése.
  - Visszatérés a megszakítás előtti állapothoz.

Közvetlen memória elérés (DMA): Tartalmaz: memória cím regisztert, átviteli irány jelzésére, mennyiségre regisztert. Ezeket szabályos I/O portokon lehet elérni.

- Működésének lépései:

1. CPU beállítja a DMA vezérlőt (regisztereket). Ezután a CPU más számításokat végez, nem várja ki a teljes műveletet.
2. A DMA a lemezvezérlőt kéri a megadott műveletre.
3. Miután a lemezvezérlő beolvasta a pufferébe, a rendszersínen keresztül a memóriába(ból) írja (olvassa) az adatot.
4. Lemezvezérlő nyugtázza, hogy kész a kérés teljesítése.
5. DMA megszakítással jelzi, befejezte a műveletet.

I/O szoftverrendszer felépítése: Tipikusan 4 réteggel rendelkezik:

1. Megszakítást kezelő réteg: legalsó kernel szinten kezelt, szemafor blokkolással védve.
2. Eszközmeghajtó programok
3. Eszköz független operációs rendszer program
4. Felhasználói I/O eszközt használó program

Eszközmeghajtó programok (driver): Egy olyan eszközspecifikus kód, mely pontosan ismeri az eszköz jellemzőit, feladata a felette lévő szintről érkező absztrakt kérések kiszolgálása. Kezeli az eszközt I/O portokon, megszakítás kezelésen keresztül.

7.2. Erőforrás elérés problémái

Holtpont (deadlock): Egy folyamatokból álló halmaz holtpontban van, ha minden folyamat olyan másik eseményre vár, amit csak a halmaz egy másik folyamata okozhat. (Nem csak I/O eszközöknél, hanem jellemző pl. párhuzamos rendszereknél, adatbázisoknál stb.)

Holtpont feltételek: Coffman E.G. szerint 4 feltétel szükséges a holtpont kialakulásához:

1. Kölcsönös kizárás feltétel: minden erőforrás hozzá van rendelve 1 folyamathoz vagy szabad.
2. Birtoklás és várakozás feltétel: Korábban kapott erőforrást birtokló folyamat kérhet újabbat.
3. Megszakíthatatlanság feltétel: Nem lehet egy folyamattól elvenni az erőforrást, csak a folyamat engedheti el.
4. Ciklikus várakozás feltétel: Két vagy több folyamatlánc kialakulása, amiben minden folyamat olyan erőforrásra vár, amit egy másik tart fogva.

Holtpont stratégiák:

1. **A probléma figyelmen kívül hagyása:** Ezt a módszert gyakran strucc algoritmus néven is ismerjük. Kérdés, mit is jelent ez, és milyen gyakori probléma? Vizsgálatok szerint a holtpont probléma és az egyéb (fordító, oprendszer, hardver, szoftver) összeomlások aránya 1:250. A Unix, Windows világ is ezt a „módszert" használja.

2. **Felismerés és helyreállítás:** Engedjük a holtpontot megjelenni (kör), ezt észrevesszük és cselekszünk. Folyamatosan figyeljük az erőforrás igényeket, elengedéseket. Kezeljük az erőforrás gráfot folyamatosan. Ha kör keletkezik, akkor egy körbeli folyamatot megszüntetünk. Másik módszer, nem foglalkozunk az erőforrás gráffal, ha egy bizonyos ideje blokkolt egy folyamat, egyszerűen megszüntetjük.

3. **Megelőzés:** A 4 szükséges feltétel egyikének meghiúsítása. A Coffmanféle 4 feltétel valamelyikére mindig él egy megszorítás.

   – Kölcsönös kizárás. Ha egyetlen erőforrás soha nincs kizárólag 1 folyamathoz rendelve, akkor nincs holtpont se! De ez nehézkes, míg pl. nyomtató használatnál a nyomtató démon megoldja a problémát, de ugyanitt a nyomtató puffer egy lemezterület, itt már kialakulhat holtpont.
   – Ha nem lehet olyan helyzet, hogy erőforrásokat birtokló folyamat további erőforrásra várjon, akkor szintén nincs holtpont. Ezt kétféle módon érhetjük el. Előre kell tudni egy folyamat összes erőforrásigényét. Ha erőforrást akar egy folyamat, először engedje el az összes birtokoltat.
   – A Coffman féle harmadik feltétel a megszakíthatatlanság. Ennek elkerülése eléggé nehéz (pl. nyomtatás közben nem szerencsés a nyomtatót másnak adni).
   – Negyedik feltétel a ciklikus várakozás már könnyebben megszüntethető. Egy egyszerű módszer erre, ha minden folyamat egyszerre csak 1 erőforrást birtokolhat. Egy másik módszer: Sorszámozzuk az erőforrásokat, és a folyamatok csak ezen sorrendben kérhetik az erőforrásokat. Ez jó elkerülési mód, csak megfelelő sorrend nincs!

4. **Dinamikus elkerülés:** Erőforrások foglalása csak „óvatosan". Van olyan módszer amivel elkerülhetjük a holtpontot? Igen, ha bizonyos info (erőforrás) előre ismert. Bankár algoritmus (Dijkstra,1965) Mint a kisvárosi bankár hitelezési gyakorlata. Biztonságos állapotok, olyan helyzetek, melyekből létezik olyan kezdődő állapotsorozat, melynek eredményeként mindegyik folyamat megkapja a kívánt erőforrásokat és befejeződik.

Bankár algoritmus több erőforrás típus esetén:

Az 1 erőforrás elvet alkalmazzuk. Jelölések:

– $F(i, j)$ az i. folyamat j. erőforrás aktuális foglalása
– $M(i, j)$ az i. folyamat j. erőforrásra még fennálló igénye
– $E(j)$ a rendelkezésre álló összes erőforrás.
– $S(j)$ a rendelkezésre álló szabad erőforrás.

Az algoritmus:

1. Keressünk i. sort, hogy $M(i, j) \leq S(j)$, ha nincs ilyen akkor holtpont van, mert egy folyamat se tud végigfutni.
2. Az i. folyamat megkap mindent, lefut, majd az erőforrás foglalásait adjuk $S(j)$-hez
3. Ismételjük 1,2 pontokat míg vagy befejeződnek, vagy holtpontra jutnak.

8. Előadás

8.1. Memóriagazdálkodás

Memóriakezelő

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

Alapvető memóriakezelés: Kétféle algoritmus csoport létezik: Swap (szükséges a folyamatok mozgatása, cseréje a memória és a lemez között, amennyiben nincs elég memria, éa lemezt is igénybe kell venni) vagy nincs szükség, ha elegendő a memória.

Monoprogramozás: Egyszerre egy program fut. Nincs szükség programokat ütemező algoritmusra. Pl. bash-ben is ez történik.

Multi programozás
"Párhuzamosan" több program fut. A memóriát valahogy meg kell osztani a folyamatok között. Ez megvalósítható

1. ***Rögzített memória szeletekkel:*** Osszuk fel a memóriát n (nem egyenlő) szeletre (Fix szeletek). Ezt például rendszerindításnál meg lehet tenni. Vagy van egy közös várakozási sor, mely leosztja hogy melyik feladat melyik memóriacímre jusson, vagy minden szeletre külön-külön sor adott. Kötegelt rendszerek tipikus megoldása.
2. ***Memória csere használattal:*** időosztályos. Nincs rögzített memória partíció, mindegyik dinamikusan változik, ahogy az op. Rendszer oda-vissza rakosgatja a folyamatokat. Dinamikus, jobb memória kihasználtságú lesz a rendszer, de a sok csere lyukakat hoz létre! Memória tömörítést kell végezni! (Sok esetben ez az időveszteség nem megengedhető!). Grafikus felület esetén nem a legjobb megoldás.
3. ***Virtuális memória használatával***
4. ***Szegmentálással***

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

Dinamikus memória nyilvántartása: Allokációs egység definiálunk először: ennek mérete kérdéses: ha kicsi akkor kevésbé lyukasodik a memória, viszont nagy a nyilvántartási „erőforrás (memória) igény". Ha nagy, akkor túl sok lesz az egységen belüli maradékokból adódó memória veszteség. A nyilvántartás megvalósítása megtörténhet bittérkép használattal vagy láncolt listával.

Memóriafoglalási stratégiák:

- First Fit (első helyre, ahova befér, leggyorsabb, legegyszerűbb)
- Next Fit (nem az elejéről, hanem az előző befejezési pontjából indul a keresés, kevésbé hatékony mint a first fit)
- Best Fit (lassú, sok kis lyukat produkál)

- Worst Fit (nem lesz sok kis lyuk, de nem hatékony)
- Quick Fit (méretek szerinti lyuklista, a lyukak összevonása költséges)

Virtuális memória: Egy program használhat több memóriát mint a rendelkezésre álló fizikai méret. Az operációs rendszer csak a „szükséges részt" tartja a fizikai memóriában. (A definíció erősen kötődik a következőhöz (MMU)!)

MMU (Memória Menedzsment Unit)

- A virtuális címtér "lapokra" van osztva. (Ezt laptáblának nevezzük.)
- Jelenlét/hiány bit.
- Virtuális-fizikai lapok összerendelése.
- Ha az MMU látja, hogy egy lap nincs a memóriában, laphibát okoz, operációs rendszer kitesz egy lapkeretet, majd behozza a szükséges lapot.

Laptábla problémák

- 32 bites virtuális címtérből még megfelelő méretű laptábla megvalósítása elképzelhető.
- De 64 bites virtuális címtér esetén már nem.

TLB (asszociatív memória): MMU-ba kicsi HW egység, kevés bejegyzéssel, szoftveres TLB kezelés, 64 elem miatt a HW megoldás kispórolható

Invertált laptáblák: valós memória méret, laptáblában fizikai memóriából keletkező számú elem, TLB használata, ha hiba, akkor invertált laptáblában keresünk, TLB-be rakjuk.

Lapcserélési algoritmusok: ha nincs virtuális című lap a memóriában, egy lapot kidobni, másikat berakni. Optimális: címkézés, ahány CPU utasítás hajtódik végre hivatkozás előtt, legkisebb számú lap kidobni (megvalósíthatatlan)

- NRU: Modify and Reference bit használata, modify időnként 0-ra, 0-3.osztály: (nem,nem; nem,igen; igen,nem; igen,igen) megfelelő eredmény, nem hatékony
- FIFO: legrégibb eldob, ha új kell (előre jön, végéről megy). Javítása a Második lehetőség: ha hiv.bit 1->sor eleje bit=0
- Óra: Második Lehetőséghez hasonló, mutatóval járjunk körbe, legrégebbi lapra mutat, ha hiv.bit=1->0, továbblépünk
- LRU: legrégibb algoritmus. HW vagy SW megvalósítás.
- NFU: lapokhoz számláló, ehhez referencia bitet adunk, óramegszakításkor, legkisebb értékűt dobjuk el, nem felejt!

Munkahalmaz modell: előlapozás, használtak a fizikai memóriában tartva, lapnyilvántartás (Óra javítása: WSClock)

Lokális, globális helyfoglalás: laphibánál hogyan vizsgáljuk, méret szerinti lap-dinamikus, PFF alg. laphiba/mp (terhelosztás)

Helyes lapméret meghatározása: kicsi (lapveszteség kicsi, nagy laptábla), nagy (fordítva) n*512 bájt a lapméret

Szegmentálás: virtuális memó (1D címtér, 0-tól maxig), több progi dinamikus területtel, egymástól független címtér (szegmens), cím 2 része (szegmens szám, ezen belüli cím), egyszerű osztott könyvtárak, logikailag tagolható (adat és kód), védelmi szint egy szegmensre, fix lapméret, változó szegmensméret

Pentium processzor virtuális címkezelése: sok szegmens: 2^{32} bite (4 GB), 16000 LDT (folyamatonként) GDT(Globbal Descriptor Table 1db) fizikai cím: szelektor+offset művelet szegmens elérés: 16 bites szegmens szelektor Lineáris cím TLB a gyors lapkeret eléréshez védelmi szintek: 0-3 (Kernel,Rendszerhívások, Osztott könyvtárak, Felhasználói programok) alacsonyabbról adatelérés engedélyezett, fordítva tiltott eljárás hívása ellenőrzött módon felhasználói programok osztott könyvtárak adatait elérhetik, de nem módosíthatják. Fontosak még: