Operációs rendszerek 1. – Alapok

Soós Sándor

Nyugat-magyarországi Egyetem
Simonyi Károly Műszaki, Faanyagtudományi és
Művészeti Kar
Informatikai és Gazdasági Intézet

E-mail: soossandor@inf.nyme.hu

Tartalomjegyzék.

Tartalomjegyzék

1. Ismétlés

A számítógép réteges felépítése.

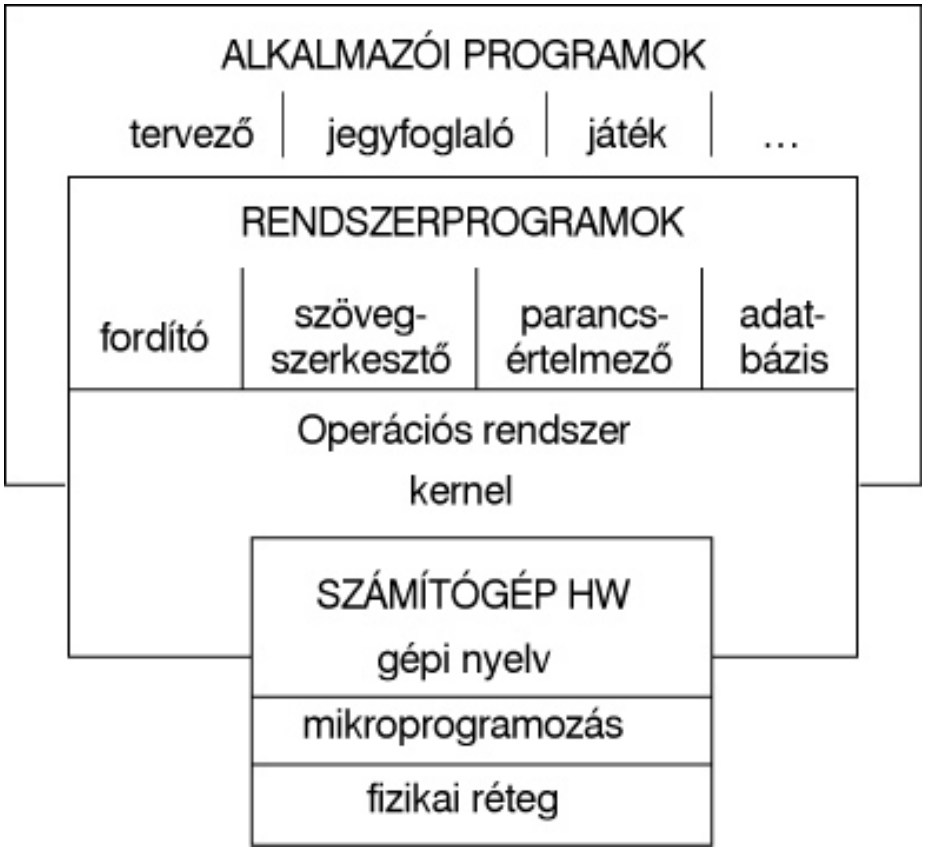
Az operációs rendszer feladata.

Az operációs rendszer alapvetően két feladatot lát el:

1. A felhasználó számára kényelmes és hatékony munkavégzést biztosít a számítógépen
2. Biztosítja a számítógép erőforrásainak hatékony, biztonságos és igazságos használatát (irányítás, koordinálás)

Az operációs rendszerek története.

- Őskor, 0. generáció
- 1. generáció (1945-1955): vákuumcsövek és kapcsolótáblák
- 2. generáció (1955-1965): tranzisztorok és kötegelt rendszerek
- 3. generáció (1965-1980): integrált áramkörök
- 4. generáció (1980-napjainkig): személyi számítógépek
- 5. generáció (napjainkban): Internet

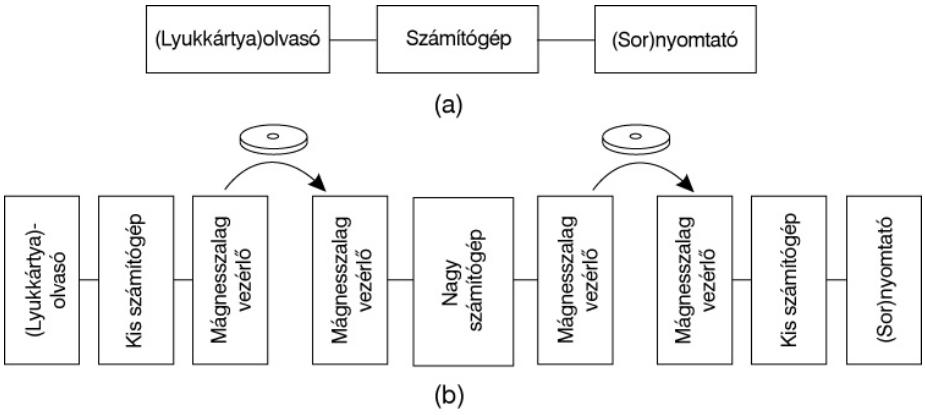


1. ábra. A számítógép réteges felépítése

Hatékonyságnövelő megoldások.

- kötegelt feldolgozás
- mágnesszalag használata
- online helyett offline működés
- egyszerű monitor program (resident monitor) (az első operációs rendszer)
- jobvezérlő program
- memória felosztása: monitor memóriaterület – felhasználói terület

On-line és off-line perifériás műveletek.



2. ábra. On-line és off-line perifériás műveletek

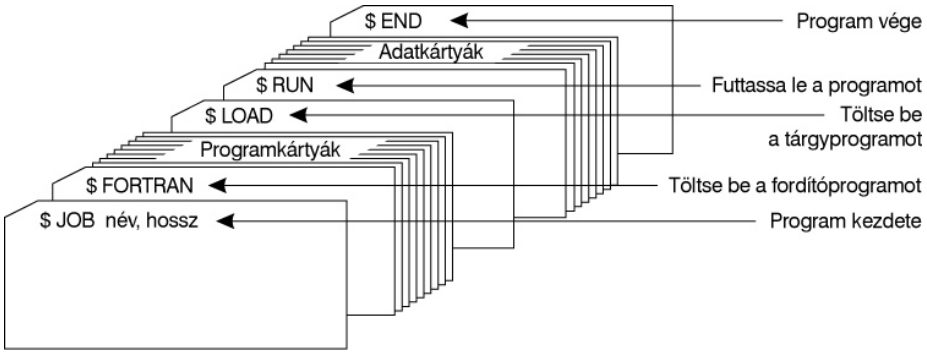
(a) On-line perifériák

(b) Off-line perifériák

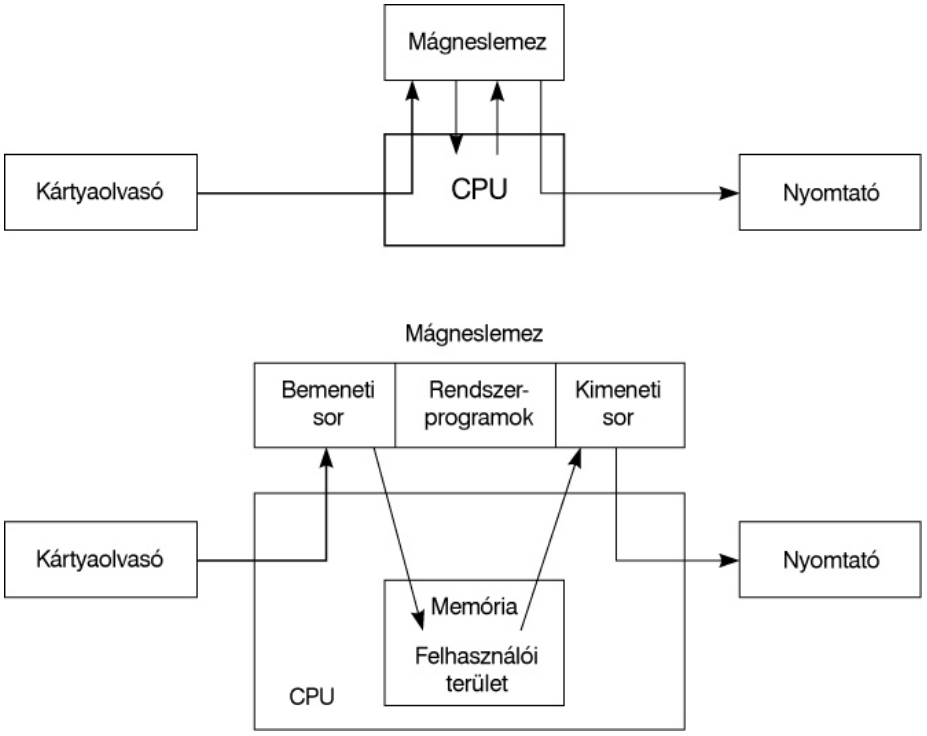
Munka vezérlő program – OS Job Control.

Spooling technika.

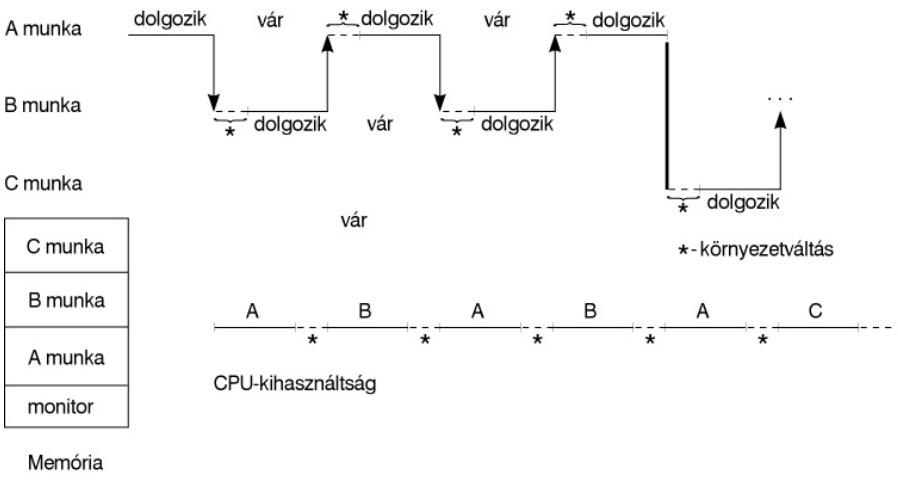
A multiprogramozás alapelve.



3. ábra. Munka vezérlő program – OS Job Control



4. ábra. Spooling technika



5. ábra. A multiprogramozás alapelve

Időosztás (timesharing).

- Minden felhasználó kap egy on-line terminált (képernyő + billentyűzet), amivel közvetlenül kapcsolódik a számítógéphez
- Parancsokat tud begépelni a billentyűzeten, a választ a képernyőn látja
- Amikor a processzor felszabadul az éppen ellátott feladat alól, akkor kiválasztja a soron következőt és végrehajtja, az eredményt visszaküldi a felhasználó képernyőjére
- Amikor egy felhasználó gondolkodik, vagy éppen gépeli a parancsot, akkor nem terheli a számítógépet, így egyetlen gép sok felhasználót tud interaktívan kiszolgálni, miközben egy nagy kötegelt feladaton is dolgozik
- Egy felhasználó viszont képes hosszú időre lefoglalni a teljes gépet egy sokáig futó parancs kiadásával
- Ezek a feladatok tovább bonyolították az operációs rendszert
- A legismertebb ilyen operációs rendszer a MULTICS (MULTiplexed Information and Computing Service)
- Az utolsó nagy MULTICS rendszert 2000. októberében állították le a Kanadai Védelmi Minisztériumban

Napjainkban.

- Személyi számítógépek

- Felhasználóbarátság
- Számítógép hálózatok
- Hálózati operációs rendszerek
- Elosztott operációs rendszerek
- Valós idejű rendszerek
- Internet
- Szabványos rendszerek
- Nyílt rendszerek
- Virtualizáció

.

Ismétlés vége

2. Rendszermodell és rendszerarchitektúra

Rendszermodell és rendszerarchitektúra.

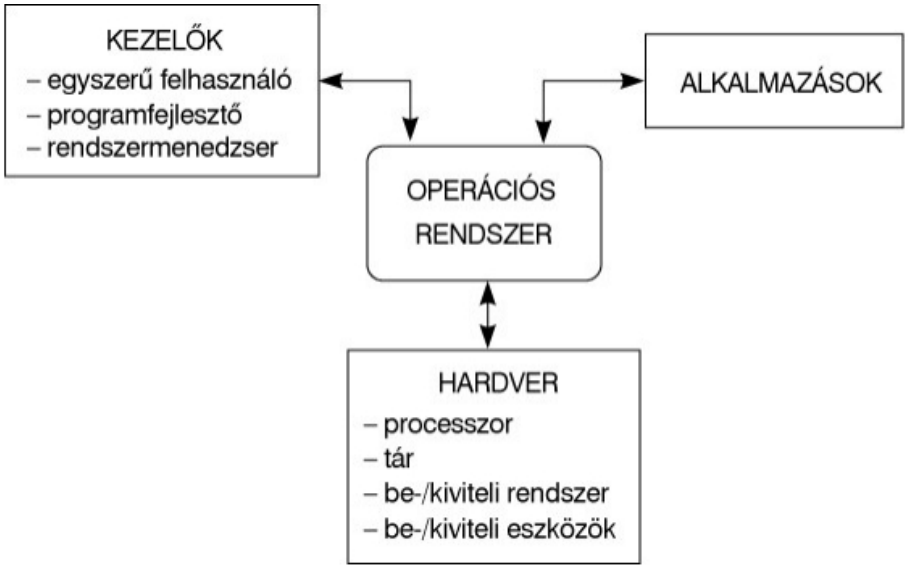
- Vizsgáljuk meg az operációs rendszereket úgy, ahogyan a szoftvereket vizsgáljuk általában:
  1. vizsgáljuk meg a rendszer kapcsolatait a külvilággal
  2. hogyan viselkedik a környezet szereplőivel
  3. milyen a rendszer belső szerkezete

2.1. Az operációs rendszerek felhasználói felülete

Az operációs rendszer környezete.

- Milyen „rendszerekkel” áll kapcsolatban egy operációs rendszer?
  - Hardver
  - Kezelők, operátorok, különböző felhasználók
  - Alkalmazások, szoftverek

Az operációs rendszer környezeti (context) diagramja.



6. ábra. Az operációs rendszer környezeti (context) diagramja

Kezelői felület – Ember-gép kapcsolat.

A részletesebb vizsgálatot kezdjük a Kezelőkkel!

- Feladatok:
  - Folyamatosan tájékoztatni kell a felhasználót a rendszer működéséről
  - Lehetővé kell tenni a beavatkozást
  - Kezelni kell az ehhez szükséges perifériákat: képernyő, billentyűzet, egér, esetleg hangszóró, mikrofon, stb.
- Különböző típusú kezelők (felhasználók):
  1. Egyszerű felhasználók
  2. Alkalmazásfejlesztők (programozók)
  3. Rendszermenedzserek (rendszergazdák)
- Ki hogyan használja az operációs rendszert?

Egyszerű felhasználók.

- Leggyakoribb tevékenység: alkalmazások futtatása
- Idejük nagy részében az alkalmazások felhasználói felületét használják, az operációs rendszerrel ritkán kerülnek kapcsolatba
- Amikor viszont igen, akkor nem akarnak sokat tanulni ehhez, nem akarnak fejből tudni parancsokat
- Ezért van szükség könnyen kezelhető, intuitív felhasználói felületekre:
  - grafikus felület
  - ablaktechnika
  - egér
  - menük
  - ikonok
- Az operációs rendszer szolgáltatásai közül elsősorban a fájlrendszert használják
  - fájlok létrehozása, másolása, törlése
- Szeretnének több alkalmazást futtatni párhuzamosan
- A felhasználók elvárnak megbízhatóságot, védelmet, biztonságot a rendszertől, ezért hajlandók elviselni bizonyos kényelmetlenségeket, rendszabályokat (bejelentkezés, jogosultsági rendszer), de ezt nem meggyőződésből teszik, ha lehetőségük van rá, hajlamosak áthágni a szabályokat tudatlanul, vagy szándékosan
- Az egyszerű felhasználó számára az operációs rendszer egy olyan virtuális gép, ami lehetőséget nyújt:
  - adat- és programfájlok védett és rendezett tárolására
  - alkalmazások futtatására
- Számukra a legfontosabb operációs rendszer szolgáltatások:
  - bejelentkezés
  - a rendelkezésre álló alkalmazások áttekintése
  - alkalmazások indítása
  - alkalmazások együttműködése és leállítása
  - fájlműveletek (másolás, mozgatás, törlés, tartalomfüggő feldolgozás)

Alkalmazásfejlesztők, programozók.

- Leggyakoribb tevékenység: programírás, tesztelés, elemzés
- Ismerik az operációs rendszer programozói felületét, bizonyos mélységig a rendszer belső működését is
- Egyes operációs rendszerek nyújtanak segédeszközöket a programfejlesztéshez (pl. debug módú futtatás), de legtöbbször a speciális szoftverfejlesztő eszközöket használják (Pl. Microsoft Visual Studio, Netbeans, Eclipse)
- Az alapos munkához mindenképpen szükség van az operációs rendszer speciális eszközeire:
  - erőforrás-használat figyelő, naplózó eszközök
  - ezeket az információkat akkor is az operációs rendszer szolgáltatja, ha a programozó az integrált fejlesztőkörnyezettől kapja meg
- Az alkalmazásfejlesztő számára az operációs rendszer olyan virtuális gép, ami meghívható eljárásokat biztosít, amivel
  - elfedi a hardver felépítésének és működésének részleteit
  - felmenti a programozót ezek ismerete alól
  - lehetőséget ad arra, hogy egymással együttműködő programokat írjon anélkül, hogy ismerné a másik programot
  - ellenőrzött eszközöket ad a programok együttműködéséhez
  - megszervezi a programok együttfutását, megosztott erőforrás-használatát
  - megfigyelhetővé teszi a programok futása közben kialakuló rendszerállapotokat

Rendszermenedzserek (rendszergazdák).

- A rendszermenedzser feladata az operációs rendszer üzemeltetése. Ennek elemei:
  1. Rendszergenerálás:
     - a rendszer telepítése a rendelkezésre álló hardverhez és az ellátandó feladatokhoz illesztve
  2. Adminisztrációs feladatok:
     - felhasználók nyilvántartása
     - alkalmazások nyilvántartása
     - a felhasználók jogosultságainak kiosztása és nyilvántartása

   - a rendszer üzemeltetési, biztonsági szabályainak meghatározása, azok betartásának felügyelete

3. Hangolási feladatok:
   - a rendszer paramétereinek beállítása a rendszer hatékonyságának növelése érdekében (pufferméretek, ütemezési és kiosztási algoritmusok finomhangolása)

4. Rendszerfelügyelet:
   - a zavartalan munka folyamatos biztosítása
   - rendellenességek észlelése, elhárítása
   - időszakos karbantartási feladatok elvégzése (tesztek, javítások, biztonsági mentések)

- A rendszermenedzser részletes és alapos ismeretekkel rendelkezik az operációs rendszerről és a környezetéről
- Olyan beavatkozási lehetőségekre van szüksége, amelyek mások számára nem elérhetők (rendszergazdai jogosultság)
- Gyakran külön fizikai eszköz (rendszerkonzol) is rendelkezésére áll ehhez

Kezelői felületek különböző fajtái.

- A kezelői felület kétféle lehet: szöveges, vagy grafikus

1. grafikus:
   - könnyen áttekinthető
   - felhasználóbarát
   - viszont nem mindig ezek a legfontosabb szempontok

2. szöveges:
   - távolról, kis sávszélességű kapcsolaton keresztül is használható
   - készülékfüggetlen, azaz bármilyen kompatibilis eszközön egyformán használható
   - lehet parancsnyelvű, vagy menürendszerű
   - a menürendszer nem jelent egyben grafikusat is!
   - a menürendszer mentesít a parancskészlet és a paraméterezés megtanulása alól
   - a parancsnyelv előnye, hogy még inkább készülékfüggetlen lehet, és lehetőség van a parancsok összefűzésére (batch üzemmód) és a parancsfájlok előre elkészítésére

- Napjainkban a kezelőfelületek szinte mindig interaktív működésűek, azaz a kezelő parancsát a rendszer azonnali reakciója követi

  - szinkron üzemmód: a kezelő kiad egy parancsot, a rendszer válaszol, ezután következik az újabb parancs
  - aszinkron üzemmód: a kezelő kiad egy parancsot, a rendszer elkezdi feldolgozni, de közben a kezelő kiadja a következő parancsot
  - Hogyan működik ez Windows, Linux és Mac OS alatt?

- Parancssorozatok
  - a legtöbb operációs rendszer lehetővé teszi, hogy különböző parancsokat összefűzzünk egyetlen paranccsá
  - egy szövegfájlban egymás után felsoroljuk az egyes parancsokat
  - az így összeállított parancsfájlt egyetlen parancsként hajthatjuk végre
  - általában használhatunk különböző programszerkezeteket (elágazások, ciklusok), így a parancsfájlok által az operációs rendszer hasonlóan programozható, mint ahogyan a processzort programozzák a szoftverfejlesztők
  - Windows alatt batch fájlnak, Linuxban shell scriptnek nevezzük ezt a lehetőséget
  - ezek segítségével egy hozzáértő felhasználó elkészíthet összetett programsorozatokat, amelyeket a kevésbé hozzáértő felhasználók is végrehajthatnak akár egy egyszerű ikonra kattintással

- A legtöbb operációs rendszernek többféle kezelői feladata van, a felhasználó eldöntheti, hogy milyen legyen az alapértelmezett kezelői felülete, de működés közben is választhat, akár párhuzamosan is használhat különböző felületeket

- Például Windowsban Command Prompt (cmd parancs), Linuxban X Window alatt Terminal window

- Milyen felületet használnak a különböző típusú felhasználók?

Kezelői felületek különböző fajtái.

- Milyen felületet használnak a különböző típusú felhasználók?
  - Egyszerű felhasználó: grafikus felület
  - Rendszermenedzser: szöveges felület, parancsfájlokkal, esetleg menürendszerrel támogatva
  - Programozó: attól függ, hogy milyen programot kell fejlesztenie, milyen eszközökkel

- Persze a rendszermenedzser és a programozó is időnként egyszerű felhasználóként dolgozik, és az egyszerű felhasználó is időnként rendszermenedzserré, vagy programozóvá válik

2.2. Alkalmazás programozói felület

Alkalmazás programozói felület, API.

- Különböző alternatív elnevezések:
  - Application Interface
  - Program Interface
  - Application Programing Interface
- Ezen keresztül kommunikálnak az alkalmazások az operációs rendszerrel
- Hogy ezt megértsük, vizsgáljuk meg, hogy mi történik, amikor programot fejlesztünk valamilyen operációs rendszer alatt?

Mi történik, amikor programot fejlesztünk?.

1. Ha a gépen nem lenne operációs rendszer
   - A processzor gépi kódban programozható
   - Néhány száz utasítást és beépített eljárást ismer
   - A programozónak ezekből kell felépítenie a programjait
   - Gépi kódban lehetetlen lenne egy igazi alkalmazást megírni
   - A magasszintű programozási nyelvek (pl. BASIC, Pascal, C/C++, Java) és az integrált fejlesztő környezetek (IDE) (pl. Visual Studio, Netbeans) megkönnyítik a programozó dolgát, de a legegyszerűbb feladatokat is le kellene programoznia:
     - például ha egy szöveget ki akarunk írni a képernyőre, akkor ismernünk kell az összes képernyőtípus alacsonyszintű programozását
     - a kiírás előtt meg kell vizsgálnunk, hogy milyen típusú képernyő csatlakozik a számítógéphez és az annak megfelelő parancsokkal kell elvégezni a kiírást
     - ha a felhasználó vásárol egy új monitort, amit a programunk írásakor még nem ismertünk, akkor új változatot kell készítenünk a programból
     - ez minden egyes program esetében így történne
   - A Windows megjelenése és elterjedése előtt pontosan ez volt a helyzet, ha grafikus programot akartunk írni, ugyanis a DOS operációs rendszer semmilyen támogatást nem adott a grafikus programok írásához

- A piacon verseny van az operációs rendszerek között, ezért az operációs rendszer gyártók igyekeznek minél jobb támogatást nyújtani a programozóknak, hogy szívesen írjanak alkalmazásokat az ő rendszerükre
- Ennek eredményeként a mai korszerű operációs rendszerek lényegesen több támogatást nyújtanak a programozóknak

2. Ha a gépen van operációs rendszer

   - Az operációs rendszer tartalmaz egy programkönyvtárat, ebben olyan eljárások találhatók, amelyek a legtöbb programban közösek, amire a legtöbb programozónak szüksége van
   - Ezeket az eljárásokat úgy írják meg az OS fejlesztői, hogy minden támogatott hardver eszközön egyaránt jól működjenek
   - A fejlesztők folyamatosan követik az új eszközök megjelenéseit, és rövid időn belül támogatják azokat is. Ez a hardvergyártók érdeke is, ezért együttműködnek az OS fejlesztő cégekkel
   - Ez a folyamatos követés az operációs rendszer rendszeres frissítéseivel valósul meg (patch, update, service pack, stb.), többek között ezért is fontos, hogy rendszeresen frissítsük az operációs rendszerünket kézzel, vagy automatikusan

Alkalmazás programozói felület, API, folytatás.

- Ezt a most megismert programkönyvtárat nevezzük API-nak
- Valójában nem egy API van, hanem sok, külön programkönyvtár a különböző funkciócsoportokhoz, például
  - be-/kiviteli műveletek (input/output)
  - memóriakezelés
  - dátum- és időkezelés
  - a programok együttműködését biztosító műveletek
  - grafikus funkciók
  - hálózatkezelés
  - ...
- A mi vizsgálatunk szempontjából mindegy, hogy ezeket egy egységes, vagy több különböző API-nak tekintjük
- Annak sincs jelentősége, hogy ezeket mind egy cég készítette (az OS gyártója), vagy egyes részeit a hardvergyártók írták meg és integrálták az OS-be

- Az API segítségével az operációs rendszer létrehoz egy virtuális számítógépet a programozó számára
  - A programozónak csak azt kell tudnia, hogy most egy Windows 7-et, egy Linuxot, vagy egy Mac OS X-et futtató számítógépre írja a programját
  - Azt nem kell tudnia, hogy abban a számítógépben milyen típusú és teljesítményű processzor, milyen és mennyi memória modul, milyen videókártya, egér, billentyűzet, hangkártya, stb. található
- Ez a megoldás lényegesen leegyszerűsítette és hatékonyabbá tette a programozók munkáját
- Ezt a gondolatmenetet továbbfejlesztve ma már olyan fejlesztő eszközök is léteznek, amelyeket használva már azt sem kell tudnia a programozónak, hogy milyen operációs rendszer alatt fog futni a programja
- Például a Java rendszer egy olyan virtuális számítógépet hoz létre az operációs rendszerek fölött, ami lehetővé teszi, hogy az egyszer megírt program változtatás nélkül futhasson bármilyen Windows, Linux, vagy Mac OS alatt, de akár megfelelő mobiltelefonon, vagy „kávéautomatán" is
- A második félévben a Programozás alapjai 2 kurzuson ezzel a rendszerrel fognak megismerkedni, akkor majd gondoljanak vissza erre a gondolatmenetre
- Visszatérve az operációs rendszerekhez, az API tehát kiegészíti a processzor utasításkészletét, általa egy virtuális számítógép jön létre, amire az alkalmazói program készül
- Az API eljárásait úgynevezett rendszerhívások formájában tudjuk végrehajtani
- Az API eljárásai és a normál alkalmazások közötti fontos különbség, hogy az API eljárásai egy speciális, védett, vagy rendszer (system) üzemmódban futnak szemben az alkalmazások felhasználói (user) üzemmódjával
  - védett, vagy rendszer üzemmódban olyan kritikus műveletek is végrehajthatók, amelyek felhasználói módban tiltottak
- A rendszerhívásokat legtöbbször úgynevezett megszakítások segítségével valósítják meg:
  - Az API dokumentációja tartalmazza, hogy az egyes API funkciók végrehajtásához milyen sorszámú megszakítást kell kérnie a programozónak és előtte hogyan kell megadnia a szükséges adatokat, illetve hogyan kapja vissza az eredményt (regiszterek)

  - Így a hívó programnak semmi többet nem kell tudnia a rendszerhívás részleteiről, ezért nem okoz gondot egy verzióváltás, vagy rendszerfrissítés, amíg az operációs rendszer kompatibilis marad a korábbi változattal, addig minden működni fog
  - A megszakításkor felfüggesztésre kerül az alkalmazói program, a rendszer átvált védett üzemmódba, végrehajtja a rendszerhívást, ezután visszavált felhasználói üzemmódba és folytatja az alkalmazói program futtatását a következő utasítástól
  - Hogy az egyes megszakítások hívásakor melyik rendszerhívás hajtódjon végre, az az operációs rendszer elindulásakor dől el, így tud az alkalmazói program minden számítógépen megfelelőképpen működni még akkor is, ha a program megírásakor az a számítógép konfiguráció még nem is létezett, így a programozó nem is készülhetett fel rá és nem tesztelhette rajta a programját

- Ha szabványosítani tudjuk a rendszerhívásokat, akkor a programok nagymértékben hordozhatók lehetnek, azaz módosítás nélkül futtathatók a különböző rendszereken
- A POSIX®-szabvány
  - A legelterjedtebb ilyen szabvány a **POSIX®** (**P**ortable **O**perating **S**ystem **I**nterface for UNI**X**)
  - C nyelven definiálja az operációs rendszer alkalmazás programozói felületét
  - A C nyelvű definíció további előnye, hogy ennek révén a C nyelvű programok forráskód szinten hordozhatók a különböző processzorok között, ha betartják a POSIX® szabványt
  - Ugyanis a POSIX® szabványnak megfelelő API hívásokat a fordító program bármilyen processzor gépi kódjára le tudja fordítani
  - Mit jelent a forráskód szintű hordozhatóság?
    * Ha megvan a program forráskódja, és azt módosítás nélkül le tudjuk fordítani különböző rendszereken futó fordítóprogramokkal, akkor a program forráskód szinten hordozható

2.3. Hardverfelület

Hardverfelület.

- A hardver fejlődése nagyon gyors
- Csak akkor tudjuk kihasználni a hardver lehetőségeit, ha a szoftver megfelelőképpen támogatja azt
- Ehhez megfelelő operációs rendszerre és megfelelő technikákra van szükség

- Ezek alkotják a hardverfelületet
- Az operációs rendszer és a hardver kapcsolata három szinten valósul meg
- Az operációs rendszer és a hardver kapcsolatának 3 szintje:

  1. Az operációs rendszer maga is egy program, aminek az adott hardveren kell futnia
     - alkalmazkodnia kell a hardverhez
     - különböző processzorcsaládokhoz külön változat
     - telepítéskor választják ki a megfelelőt és illesztik a rendszerhez
     - modultecnnika: a hardverfüggő részeket külön modulokba teszik, hogy könnyen lehessen cserélni
  2. Az operációs rendszer kezeli a hardver erőforrásait és gazdálkodik velük
     - az alkalmazások között elosztja azokat
     - ehhez ismernie kell a hardver elemeit és azok paramétereit, nyilván kell tartania azokat
     - absztrakció: a rendelkezésre álló erőforrásokat az operációs rendszer virtuális eszközökbe szervezi, és azokat bocsátja az alkalmazások rendelkezésére
     - pl. fizikai háttértárak $\Rightarrow$ logikai meghajtók a:, b:, c:, d:, ...
  3. Az operációs rendszernek kezelnie kell a rendszerhez kapcsolódó be-/kiviteli eszközöket
     - ezek nagyon sokfélék lehetnek, és meg is változhatnak a rendszer működése során sokkal gyakrabban, mint a rendszer alapelemei: processzor, memória, ...
     - új, korábban ismeretlen eszközök kerülhetnek be a rendszerbe, és korábban használtak kikerülhetnek belőle
     - (például gondoljunk egy USB-n keresztül csatlakozó külső merevlemezre)
     - a rendszernek eközben is működőképesnek kell maradnia
     - a megoldás itt is az absztrakció: virtuális eszközök + eszközmeghajtó programok (device driver)
     - az operációs rendszer és az alkalmazói programok a szabványosított virtuális eszközt látják, az eszköztől függő speciális részleteket a device driver valósítja meg
     - pl. egy adatblokk tárolása egy külső lemezegységen a következőképpen történik:
     - az alkalmazás kiküld egy *„adatblokk írása”* parancsot a d: meghajtóra
     - az operációs rendszer továbbítja ezt az eszköznek

- az eszközkezelő program (device driver) tudja, hogyan kell kezelni az eszközt
- megnézi, hogy milyen fordulatszámmal forog a lemez, ha nem megfelelő, akkor felpörgeti és megvárja, amíg felpörög
- beállítja az író fejet a megfelelő pozícióba és megvárja, amíg odaér
- bemásolja az adatokat az eszköz pufferébe (köztes tároló) és kiadja az írás parancsot
- ha minden rendben lezajlott, akkor visszajelez az operációs rendszernek
- ha valami hiba történt, akkor ezt jelzi
- az operációs rendszer jelzi az alkalmazásnak a művelet eredményét

Munkamegosztás az operációs rendszer szállítók és a készülékgyártók között .

- Egyrészt az operációs rendszer szállítója mellékeli a rendszerhez az elterjedt hardvereszközöket támogató modulokat
- Másrészt a készülékgyártók elkészítik az eszközeikhez a meghajtó programokat a különböző operációs rendszerekhez
- Mindkét oldalon készülnek különböző verziók részben a fejlesztések miatt, részben az észlelt hibák kijavítására
- A rendszer csak akkor működőképes, ha léteznek megfelelő megállapodások és szabványok, és persze mindenki betartja ezeket
- Kettős szabványosításra van szükség:
  1. Szabványok a hardvereszközök csatlakoztatására (pl. USB, Firewire, korábban soros port, párhuzamos port)
  2. Szabványok az eszközök közötti kommunikációra, szoftvercsatlakoztatás
- Az operációs rendszer eszközöket nyújt a rendszermenedzserek számára a készülékek és a kezelőprogramok csatlakoztatására

2.4. Belső szerkezet

Az operációs rendszerek belső szerkezete.

- Ahogyan az eddigiekben láttuk az operációs rendszerek nagy és összetett rendszerek

- Hogyan épül fel egy ilyen rendszer?
- Hogyan lehet ilyen rendszereket létrehozni?
- Amit most megismerünk, az jól fog jönni akkor is, amikor majd mi írunk programokat (PROG1, PROG2, ...)
- Hogyan álljunk neki egy nagy program megírásának?
  - A Programozás alapjai órán megtanuljuk, hogyan kell megírni egy egyszerű, kisméretű programot?
  - Mit nevezünk kisméretű, egyszerű programnak?
  - Ha már meg tudunk írni kis programokat, hogyan írunk meg egy nagyméretű, összetett programot?
  - A megoldás: problémafelbontás

Problémafelbontás.

- Nem csak a programozás közben használhatjuk ezt a módszert
- Ha egy feladat nagyobb méretű, mint amit könnyen át tudnánk látni, akkor az első feladat, hogy felbontjuk kisebb részekre
- Ha egy rész már akkora méretű, amit átlátunk és meg tudunk oldani, akkor oldjuk meg
- Ha még ezt sem látjuk át, akkor osszuk tovább még kisebb részekre
- Ezt addig folytatjuk, amíg minden részfeladatot meg nem oldottunk
- Ezután a megoldott részmegoldásokból összeépítjük a teljes megoldást
- A problémafelbontást és a részmegoldások összeépítését sokféleképpen végezhetjük
- Ezt a módszert mindenféle feladat megoldásánál használhatjuk
- Ha programozási feladatról van szó, akkor programozási módszertanoknak nevezzük ezeket a felbontási módszereket
- Sok különböző programozási módszertan létezik, ezek megismerése segítségünkre lesz a programozás során
- Vizsgáljunk meg két felbontási módszert:
  1. Rétegekre bontás
  2. Modulokra bontás

Rétegekre bontás.

- A programot egymásra épülő rétegekre bontjuk
- Minden réteg csak az alatta lévő réteg szolgáltatásait használhatja, és a felette lévő réteg számára nyújt szolgáltatásokat
- Előnyei:
  - Tiszta szerkezet
  - Könnyen továbbfejleszthető, bővíthető, módosítható
- Hátrányai:
  - Nehéz eldönteni, hogy mi kerüljön az egyes rétegekbe
  - A rétegek sorrendje nem egyértelmű
  - Csökkenti a teljesítményt, ha több rétegen kell átnyúlni

Modulokra bontás.

- Nem követeljük meg a szabályos rétegszerkezetet
- Egymástól független modulokba osztjuk a programrészeket
- A modulokat úgy kell kialakítani, hogy a modulokon belül maximális legyen a kapcsolat, míg a modulok között minimális
- Ha jól alakítottuk ki a modulokat, akkor az egyes modulokat egymástól függetlenül készíthetjük el
- Ha elkövetünk egy hibát, akkor az csak a modulon belül okozhat problémát, nem befolyásolja a többi modul működését

Az operációs rendszerek belső szerkezete.

- Térjünk vissza az operációs rendszerekhez!
- Az operációs rendszerek nagy méretű szoftverrendszerek, vizsgálatuk sok programozási tanulsággal szolgál
- A korai operációs rendszerek (pl. OS/360) úgynevezett monolitikus szerkezetűek voltak, azaz nem rendelkeznek belső szerkezettel, nem bonthatók kisebb részekre
- Ez azt eredményezi, hogy bármilyen módosítás a programban hatással lehet a teljes programra

- Ennek köszönhető a sok hiba az OS/360 operációs rendszerben, és ez okozza azt is, hogy minden javítás sok újabb hibát eredményezett
- Mit lehet tenni ez ellen?
- Részekre kell bontani a rendszert
- Leggyakrabban a két látott módszert kombináltan használjuk
- Kialakítunk néhány réteget és több funkcionális modult
- Tipikus rétegek:
  - a hardverfüggő részek egy alsó, hardverközeli rétegbe kerülnek
  - az operációs rendszer alapfunkciói a rendszermagban (kernel) valósulnak meg
  - a feltétlen szükségest meghaladó kényelmi szolgáltatások
  - a felhasználói programokból érkező rendszerhívások fogadó felületét megvalósító réteg
- Tipikus modulok:
  - folyamatkezelés
  - tárkezelés
  - fájlkezelés
  - be-/kivitel kezelése
  - háttértár kezelése
  - hálózatkezelés
  - védelmi és biztonsági rendszer
  - parancsértelmezés
  - felhasználóbarát kezelői felületek
- A következő órákon részekre szedjük az operációs rendszert, és sorban megvizsgáljuk az egyes részeket, rétegeket és modulokat

3. Befejezés

3.1. Emlékeztető kérdések

Emlékeztető kérdések.

1. Milyen rendszerekkel áll kapcsolatban egy operációs rendszer?
2. Milyen felhasználó típusokat különböztetünk meg az operációs rendszerekkel kapcsolatban?

3. Mire használják leggyakrabban az operációs rendszert a különböző típusú felhasználók?
4. Milyen felhasználói felületet várnak el a különböző típusú felhasználók?
5. Milyen OS szolgáltatásokat használnak leggyakrabban a különböző típusú felhasználók?
6. Milyen különböző fajta kezelői felületei lehetnek az operációs rendszereknek?
7. Milyen kezelői felületet szeretnek használni a különböző típusú felhasználók? Miért?
8. Mit jelent az API rövidítés?
9. Miben tér el a programozás operációs rendszerrel és nélküle?
10. Mit jelent a védett üzemmód?
11. Mit jelent a megszakítás?
12. Mit jelent a POSIX®?
13. Mit jelent a forráskód szintű hordozhatóság?
14. Hogyan kezeli az operációs rendszer a különböző perifériákat?
15. Mi történik az operációs rendszerben, amikor egy külső eszközt használunk egy felhasználói programban? Hogyan valósul meg mindez?
16. Hogyan tudunk megoldani nagyméretű problémákat?
17. Mit jelent a rétegekre bontás?
18. Milyen előnyei és hátrányai vannak?
19. Mit értünk modulokra bontás alatt?
20. Miért lehetett olyan sok hiba az OS/360 operációs rendszerben?
21. Milyen tipikus rétegekből és modulokból épülnek fel az operációs rendszerek?

Befejezés.

Köszönöm a figyelmet!