

Operációs rendszerek története és osztályozása, HW környezet

dr. Kovácsházy Tamás

1. anyagrész, Bevezető



Méréstechnika és
Információs Rendszerek
Tanszék

Operációs rendszer

■ Definíció (Wikipedia)

- An operating system (OS) is an interface between hardware and user which is responsible for the management and coordination of activities and the sharing of the resources of a computer, that acts as a host for computing applications run on the machine.
- Operating systems offer a number of services to application programs and users. Applications access these services through application programming interfaces (APIs) or system calls.

■ Jó a definíció?

- Kliens operációs rendszerre többé-kevésbé.
 - Jó definíciót senki sem tud: Microsoft v. USA perek során sem
- Van benne jó néhány új fogalom. (később tisztázzuk őket)
- Milyen operációs rendszerek vannak?
 - Nagyon sok fajta, de inkább kezdjük a történelemmel...

Összefoglalva...

- Hardware és alkalmazói szoftver közötti interfész
- Erőforrás használat és aktivitások menedzsmentje és koordinálása
- Lehetővé teszi az alkalmazások futtatását
- Egy programozási felületet (API) nyújt az alkalmazások számára, hogy azok el tudják érni az OS szolgáltatásait
- Hogy ezeket a funkciókat meg tudja valósítani „erősebbnek” kell lennie, mint az alkalmazói programok...

Korai operációs rendszerek

■ HW fejlődése

- Korai számítógépek huzalozott programmal rendelkeztek
 - Egy feladatot tudtak végrehajtani egy időben
 - A feladatok váltása rendkívül időigényes volt
- Ekkor is felmerül az erőforrásokkal történő optimálisabb gazdálkodás, de ez emberi erővel történik
 - Feladatok és azok sorrendjének kiválasztása (fontosság, stb. alapján)
 - Emberi, gépi, és egyéb (idő) erőforrások allokációja feladathoz
 - Végrehajtási kísérlet
 - Eredmények kiértékelése
 - Újrahuzalozás...

Korai BATCH rendszerek

- Batch rendszer (kötegelt), hogyan történik:
 - Program papíron készül (korai Fortran pl.)
 - A program lyukkártyára írása
 - Lyukkártyák leadása az operátornál
 - Futtatás
 - Eredmények kinyomtatása
 - Teljesen on-line, szekvenciális periféria működés
- Hasonló erőforrás igényű munkák csoportosítása, sok különálló, ismétlődő lépésre nincs szükség
- On-line perifériás műveletekről áttérés off-line műveletekre, I/O processzor, nő a komplexitás
- Egyszerű monitor (resident monitor) a munkák ütemezésére
 - Egyiknek vége, automatikusan kezdődik a másik

Átlapolt feldolgozás

- Az I/O processzorok elfedik a periféria specialitásait
 - Szabványos, absztrakt interfész
 - Logikai B/K (I/O) perifériák megjelenése
- Pufferelés (buffering)
 - Az I/O perifériák és a központi egység közötti kapcsolat megoldására/optimalizálására
 - Input → CPU → Output átlapolódik
 - A hibakeresés még mindig nehézkes
 - A programozó nem fér hozzá a rendszerhez on-line
 - A hibakimenetet a program futása után megkapja

Spooling

- Nagyobb kapacitású, gyors, véletlen hozzáférésű memóriák megjelenése
 - Több feladat is egy időben a gépben
 - Egy fő program plusz az I/O-val kapcsolatos feladatok
 - Ezek egy időben futhatnak
 - Spooling (Simultaneous peripheral operation on-line)
 - A feladatok jobban átlapolódhatnak
 - Eredmény: elmozdulás a multiprogramozás felé

Multiprogramozás

- Még nagyobb kapacitású, még gyorsabb memóriák
 - A feladatok nem csak beérkezésük sorrendjében dolgozhatók fel (egy időben több befér a memóriába)
 - Optimalizáció lehetősége/igénye futási időben
 - Job pool (lehetséges feladatok készlete)
 - Cél a közel 100%-os CPU kihasználtság (nem lehetséges)
 - Megjelenik az ütemezés (scheduling)
 - A lehetséges feladatok közül melyik fusson?
 - Erőforrás (CPU, memória, tár, perifériák) gazdálkodás is feladat
 - On-line kapcsolat a felhasználóval lehetőségessé válik
 - Fő probléma a válaszidő az on-line felhasználók számára
 - A feladatok addig futnak, míg valamire várniuk nem kell

1960-as évek vége

- Miniszámítógépek (pl. PDP) megjelenése
 - Kisebb csoportok számára elérhető, olcsóbb gépek
 - Sokkal többen férnek hozzá (sok gép)
 - Egy gépre kevesebb ember jut (idő a kísérletezésre)
 - Programozók a géphez juthatnak
 - MULTICS, majd UNIX
 - C és egyéb modern nyelvek
 - Az első lépések az Internet felé (ARPANET, információ megosztása)
 - Gyors fejlődés
 - Első kísérletek fizikai folyamatok számítógépes irányítására
 - A beágyazott rendszerek ebből alakulnak ki majd

Időosztásos rendszerek

- Time sharing vagy más néven multitasking
- Az on-line felhasználók számára fontos a válaszidő
 - Több ember tudja egy időben használni a gépet ($n \cdot 10$ -100 embernek van egy gépe)
 - Gépnek valamit és választ várnak rá
 - A gép ne legyen üresjáratban (használjuk ki)
 - A válaszidő emberi léptékkel elfogadható legyen ($n \cdot 10$ ms)
 - A feladatok virtuálisan egy időben (párhuzamosan), szeletekben futnak
 - Egymás után, egy óra által periodikusan időzítve/váltva
 - Az óra megszakítja az éppen futó feladatot, és erre az OS vált egy másikra
 - A háttérben egy batch rendszer is fut
 - A megmaradó időszeletek kihasználására (erre is van igény)
- Pl. klasszikus UNIX erre vállalkozik

Személyi számítógépek

- 1970-es évek közepétől
 - Egy felhasználó egy gép összerendelés lehetséges a technikai fejlődés eredményeképpen
 - Az IBM PC és utódai
 - x86 CPU architektúra
 - memória + HDD
 - karakteres v. grafikus képernyő (X-windows, Windows)
 - billentyűzet és később egér
 - hálózat (LAN majd Internet)
- Lépések az elosztott rendszerek megjelenése felé
- Új követelmény: Felhasználóbarátság

Elosztott rendszerek

■ Decentralizálás

- Funkciók térbeli elosztása
- Előnyök - Hátrányok: Biztonság, skálázhatóság, megbízhatóság, fejlesztési kérdések
 - Nem egyértelmű, erősen implementáció függő
 - Mindenképpen ebbe az irányba mozgunk (cloudcomputing, stb.)

■ Ebbe már nem fogunk belemenni a tárgy során

■ Egy időben párhuzamos fejlődési lépés a mobil operációs rendszerek megjelenése, ezt felületesen érinteni fogjuk...

Többprocesszoros rendszer

- Homogén (egyforma) processzorok
 - Pl. több CPU magot, vagy több CPU-t, vagy több CPU-t és azon belül több magos CPU-kat tartalmazó rendszer (AMD Opteron, Intel Xeon)
- Heterogén (különböző) processzorok
 - Pl. több magos asztali gépben GPGPU (CUDA, OpenCL) vagy FPGA alapú gyorsító processzorok
 - Gyorsan terjed, ez a jövő (Intel, AMD, mobil SoC)
- Sokmagos CPU-k (pl. Intel kísérleti rendszerek, PS3 Cell)
- Hogyan lehet hatékony programokat írni ezekre a szörnyekre?
 - Erről sem lesz szó, de az oktatott információk alapján ezekkel a területekkel is lehet majd kezdeni ismerkedni
 - Ez a jövő kérdése, nagyon forró kutatási irányról van szó

Milyen operációs rendszerek vannak?

- Alkalmazás specifikus megközelítés
 - Kliens, szerver és mainframe operációs rendszerek (IT infrastruktúra)
 - Sokprocesszoros szerver, Grid, Cloud, Szuperszámítógépek
 - Beágyazott operációs rendszerek
 - Mobil operációs rendszerek
- Tulajdonság specifikus megközelítés
 - Valós idejű operációs rendszerek (válaszidő)
 - Nagy megbízhatóságú operációs rendszerek (rendelkezésre állás)
 - Konfigurálható operációs rendszerek (funkciók kiválasztása)
- Persze a tulajdonság és az alkalmazás szorosan összefügg...
 - Pl. A Linux minden szegmensben megtalálható
 - A Microsoft-nak is vannak termékei minden szegmensben
 - Rengeteg szűk területet megcélzó gyártó
 - Wikipedia 2012- ben: 45 kommersziális gyártó, és azon belül is több operációs rendszer, ezen kívül open source OS-ek
 - Hány Linux disztribúció van? Azok most különböző OS-ek? Nem akarom/tudom megválaszolni a kérdést...
 - Linux kernel + alkalmazáskészlet = Linux disztribúció

Kliens, szerver és mainframe OS-ek

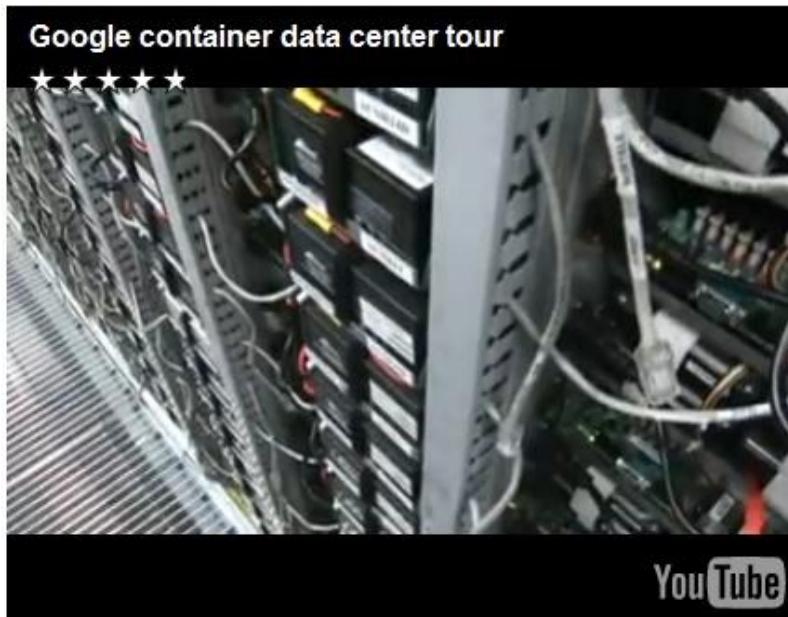
- Klienssel mindenki tisztában van...
(vagy legalábbis remélem)
- Sokprocesszoros szerver/mainframe
 - 8-64(256) CPU, $n \cdot 10/100$ Gbyte RAM
- Magas rendelkezésre állás **IBM System Z10**
 - Redundancia
 - Működés közben történő alkatrész csere
- Menet közben particionálható
- Virtualizáció HW támogatása



Sun/Oracle Fire X4600



Adatközpontok (grid, cloud, etc.)



- $n \cdot 10.000$ szerver
- $n \cdot 100$ TB memória
- Hatalmas tárolóterület
- Masszívan elosztott feladatok (pl. keresés)
- Műveletek hossza 1-2 sec vagy akár napok
- Google, Microsoft, Facebook, YouTube
 - Szerver OS alapon
- Optimalizált HW
 - <http://www.opencompute.org/>

Beágyazott rendszer (embedded system) 1.

- Olyan **speciális számítógépes rendszerek**, amelyeket egy **jól meghatározott feladatra** találtak ki
 - Ezen feladat ellátása érdekében **a külvilággal intenzív információs kapcsolatban állnak**
 - Érzékelik annak bizonyos paramétereit, és gyakran be is avatkoznak abba
 - Gép – beágyazó fizikai környezet interfész: Szenzorok, beavatkozók, kommunikációs felület
 - Felhasználói felület (humán operátor)
- Ez nem zárja azt ki, hogy pl. PC-t használjunk beágyazott rendszerekben
 - De akkor annak legalább részben dedikált feladata lesz (alkalmazástól függ)!
 - Akár standard Windows vagy Linux operációs rendszerrel is!
 - Bár nem erre lettek kitalálva.
 - Korlátozott alkalmazási körben.

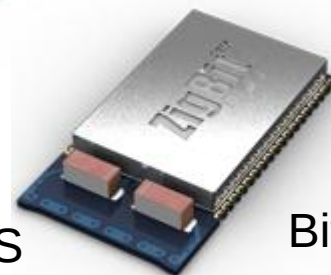
Beágyazott rendszer (embedded system) 2.

- Sokszor biztonságkritikus a környezet, a meghibásodás eredményeképpen:
 - Emberek sérülhetnek meg súlyosan vagy halhatnak meg
 - Nem elviselhető kockázat
 - El kell kerülni, bizonyítani kell, hogy a technológia adott szintjén mindent megtettünk az elkerülésére
 - Nincs 100%-os biztonság
 - Milyen szerepet játszik ebben az OS???

Beágyazott alkalmazások 1.

Speciális minősítések

- Gépjármű
 - Vonat
 - Légi járművek
 - Katonai
 - Humán egészségügy
 - Atomerőmű
 - stb.
- Valós idejű működés
 - Megbízhatóság, rendelkezésre állás
 - Egy OS nem képes ezeket a nagyon eltérő igényeket kielégíteni



uC/OS

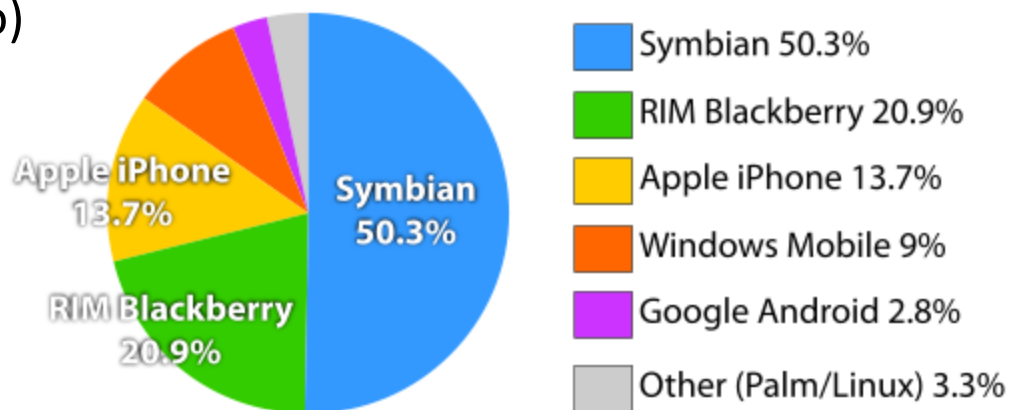
BitCloud

Mobile OS (2010, először tartom az OPRE-t)

- Mobil beágyazott rendszerek
- Napjainkban összemosódik a kliens operációs rendszerekkel
- Követelmények
 - Speciális GUI, multitouch, szenzorintegráció, stb.
 - Telep/akkumulátor élettartam maximalizálása
 - Limitált erőforrások
 - Részben valós idejű funkciók (alacsony szintű kommunikáció)
 - Heterogén architektúra
 - User CPU
 - kommunikációs DSP
 - Videó/3D feldolgozás



Global Smartphone Sales, Q2 2009



Mobile OS (5 éve tartom az OPRE-t)

■ Mobil beágyazott rendszerek

■ Napjainkban

kliens

■ Következő

○ Sp

gy

○ Te

m

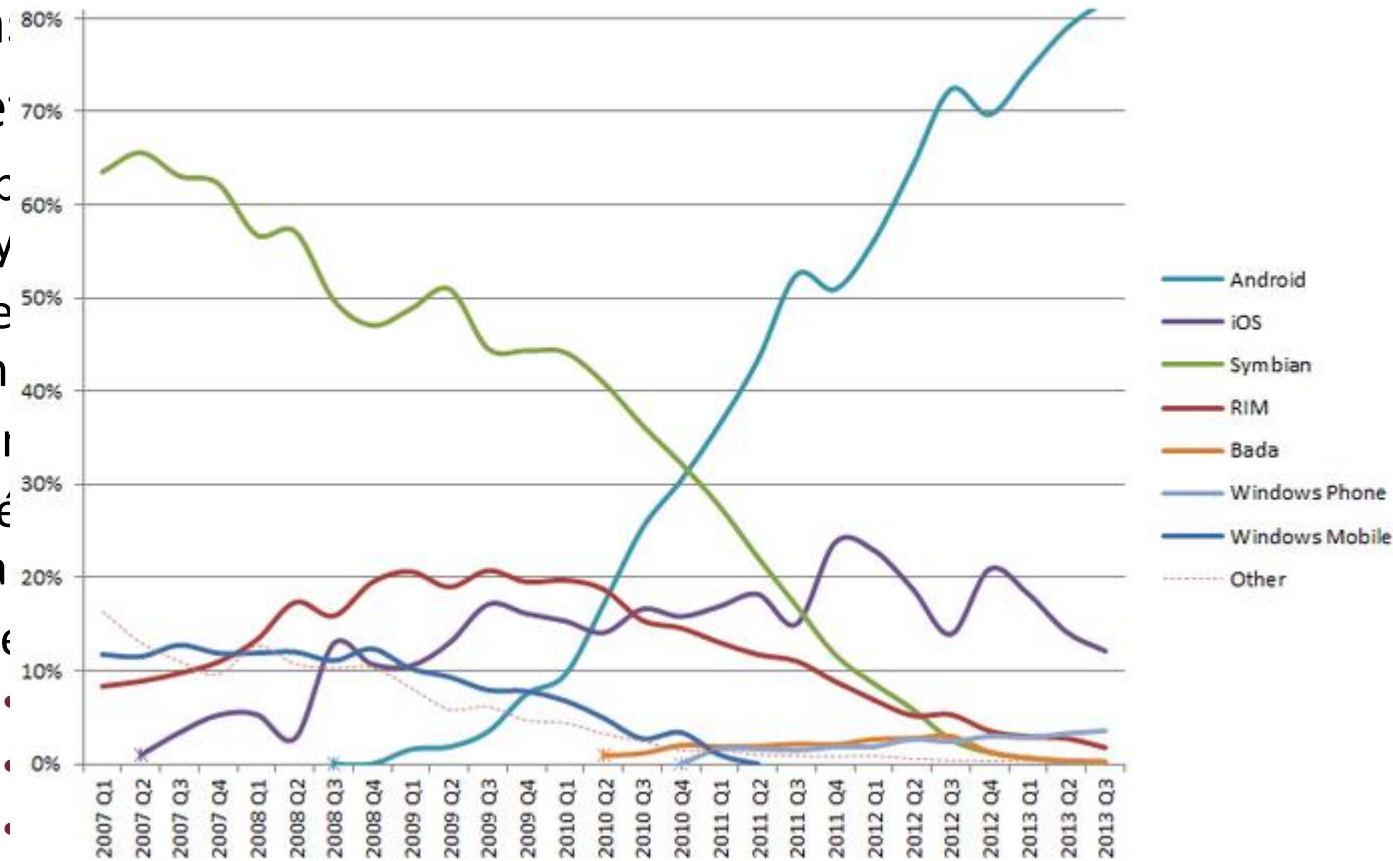
○ Li

○ Ré

(a

○ He

World-Wide Smartphone Sales (%)



Valós idejű (real-time) rendszer

- A rendszer adott eseményekre adott időn belül adott valószínűséggel válaszol (egyébként hibás a válasz, hiába funkcionálisan jó a válasz)
 - Pl. Vetélkedő a TV-ben
- Fajtái:
 - **Lágy valós idejű (soft real-time): A valószínűség < 1 , de elég közel van egyhez.**
 - Akkor használjuk, ha a határidők nem teljesítésének nincsenek katasztrofális következményei, azok bizonyos szintig elviselhetőek
 - A rendszer „időnként” késhet
 - Service Level Agreement (a szolgáltatás minősége)
 - NEM feltétlenül prioritásos a működése, de leggyakrabban az
 - **Kemény valós idejű (hard real-time): A valószínűség $= 1$.**
 - Ha nem válaszol időben, a válasz rossz...
 - A határidők nem teljesítésének katasztrofális következményei vannak
 - „**A rendszer NEM késhet!**”
- Hogyan bizonyítjuk?

Valós idejű (real-time) rendszer 2.

- A válaszra megadott időintervallum nagyságára a definíció nem mond semmit
 - Az erősen függ az alkalmazástól
 - Lassú kémiai folyamat esetén akár órák, jármű esetén jóval kisebb (ms alatt)
- Valós idejű operációs rendszer:
 - Felépítéséből következően megadott szolgáltatásai képesek valós időben működni.
 - Például a HW megszakítás után a megszakításhoz tartozó kód futása adott időintervallumon belül (többnyire néhány us) megtörténik
 - A futtatott alkalmazásnak is valós idejűnek kell lennie, a valós idejű OS csak lehetővé teszi a valós idejű működést
 - A Linux és a Windows nem ilyen, nem adható ilyen garancia
 - Mindkettőhöz létezik viszont ilyen garanciák megadását lehetővé tévő kiterjesztés (RTLinux, Windows: pl. Ardence RTX)

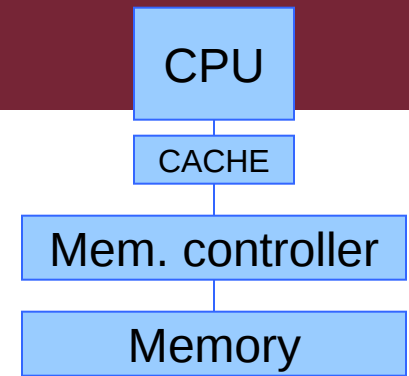
HW architektúrák

- Sokféle van, még pl. x86 PC esetén is...
- Szerencsére a különbségeket elfedheti az operációs rendszer:
 - Pl. egy jól megírt Linux alkalmazás képes futni pl. egy régi P3-as PC-én is, és a legújabb sokmagos Core i7 PC-n is.
 - Legfeljebb nem használja ki a több végrehajtó egységeget, csak az egyiket.
 - Sőt, újrafordítva megy egy ARM architektúrájú gépen is.
 - HW közeli rétegek cseréjével persze...
 - Pl. Linux on Raspberry PI-n vagy Beagleboard-on
 - Ugyanakkor tisztában kell lennünk, mi zajlik a háttérben
 - Sőt, egyre gyakrabban kell belenézünk az OS belsejébe is

Számítógép architektúrák

- Az OS belső működése függ attól, hogy:
 - Hány és hogyan összekapcsolt processzort tartalmazó számítógépen fut?
 - Hogyan szervezzük a memóriát?
 - Hogyan kommunikálunk a perifériákkal?
- Homogén többprocesszoros rendszerben gondolkodunk
 - Azonos processzorok...
- Csoportosítás
 - Single CPU (Uniprocessor)
 - Symmetric multiprocessing
 - Non-Uniform Memory Access

Single CPU (Uniprocessor)

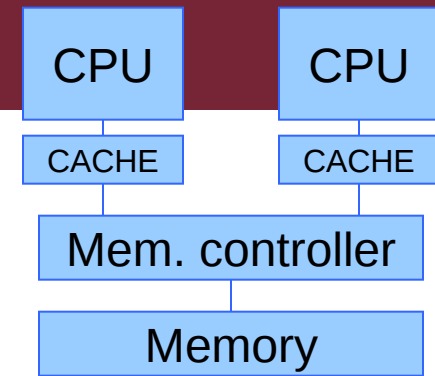


- Egy végrehajtó egység (single CPU)
 - Néhány évvel korábban ez volt a jellemző
 - Beágyazott rendszerekben ma is ez a jellemző!
 - A MCU-k (mikrovezérlő) jellemzően egymagosak
 - Teljesítményben skálázhatóak (architektúra+órajel) egyelőre
- DMA, ha van, párhuzamosan kezeli a memóriát a CPU-val
 - Versenyhelyzet a DMA vezérlő és a CPU között
 - Input: DMA transzfer $\Rightarrow$ IT $\Rightarrow$ CPU kezeli az adatokat
 - A CACHE koherencia sérülhet!
 - Egész CACHE érvénytelenítése/tiltása (a transzfer alatt)
 - Egyszerű, de katasztrofális a hatása a teljesítményre
 - DMA-val kezelt memóriaterületre tiltani kell a CACHE-t (pl. MMU, később)
 - CACHE koherens DMA (HW támogatás)

SMP

■ Symmetric multiprocessing

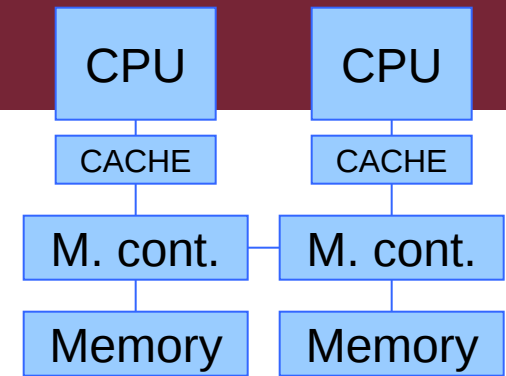
- Több, azonos végrehajtó egység
 - Több CPU, vagy CPU mag
 - Például: AMD Phenom vagy Athlon X2, Intel C2D/C2Q
- Esetleg saját CACHE hierarchiával
 - CACHE koherencia biztosításával
- Közös buszon/vezérlőn a memória
 - A memóriát minden végrehajtó egység azonos tulajdonságokkal (sebesség, késleltetés) éri el
- Muticore MCU-k megjelenése
 - ARM15, ARM11 MPCore, ARM Cortex-A9 MPCore
 - A CPU mag olcsó (relatív kis felületet használ a félvezetőn)
 - A beágyazott területen is szerepet kap az SMP
- OS támogatás kell hozzá (egyébként 1 CPU látható)



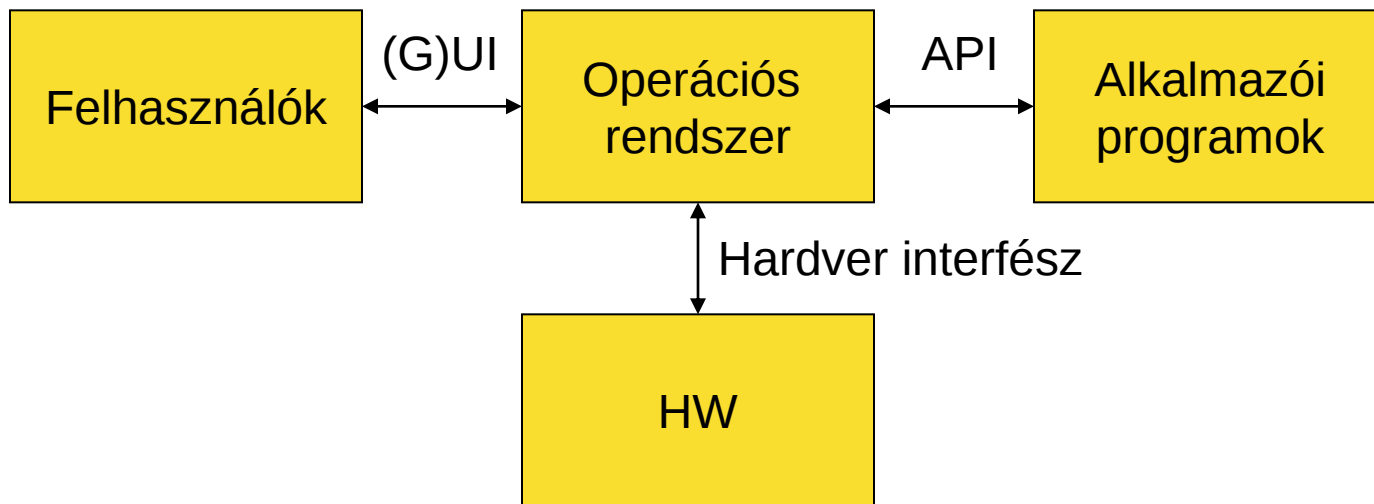
NUMA

Non-Uniform Memory Access

- A memória egyes része „közelebb” vannak egyes végrehajtó egységekhez mint másokhoz
- Összefüggő fizikai memória
- Cache koherencia
 - CACHE koherens (ccNUMA)
 - nem CACHE koherens
- A memóriavezérlők egy speciális kommunikációs felületen csatlakoznak
 - QPI az Intelnél, vagy Hypertransport az AMD-nél
- Pl. AMD Opteron vagy a Intel Core i7 alapú szerver processzorok több CPU esetén ccNUMA architektúrát is használhatnak (CPU-n belül SMP van)
- OS támogatás (egyébként 1 CPU látható)

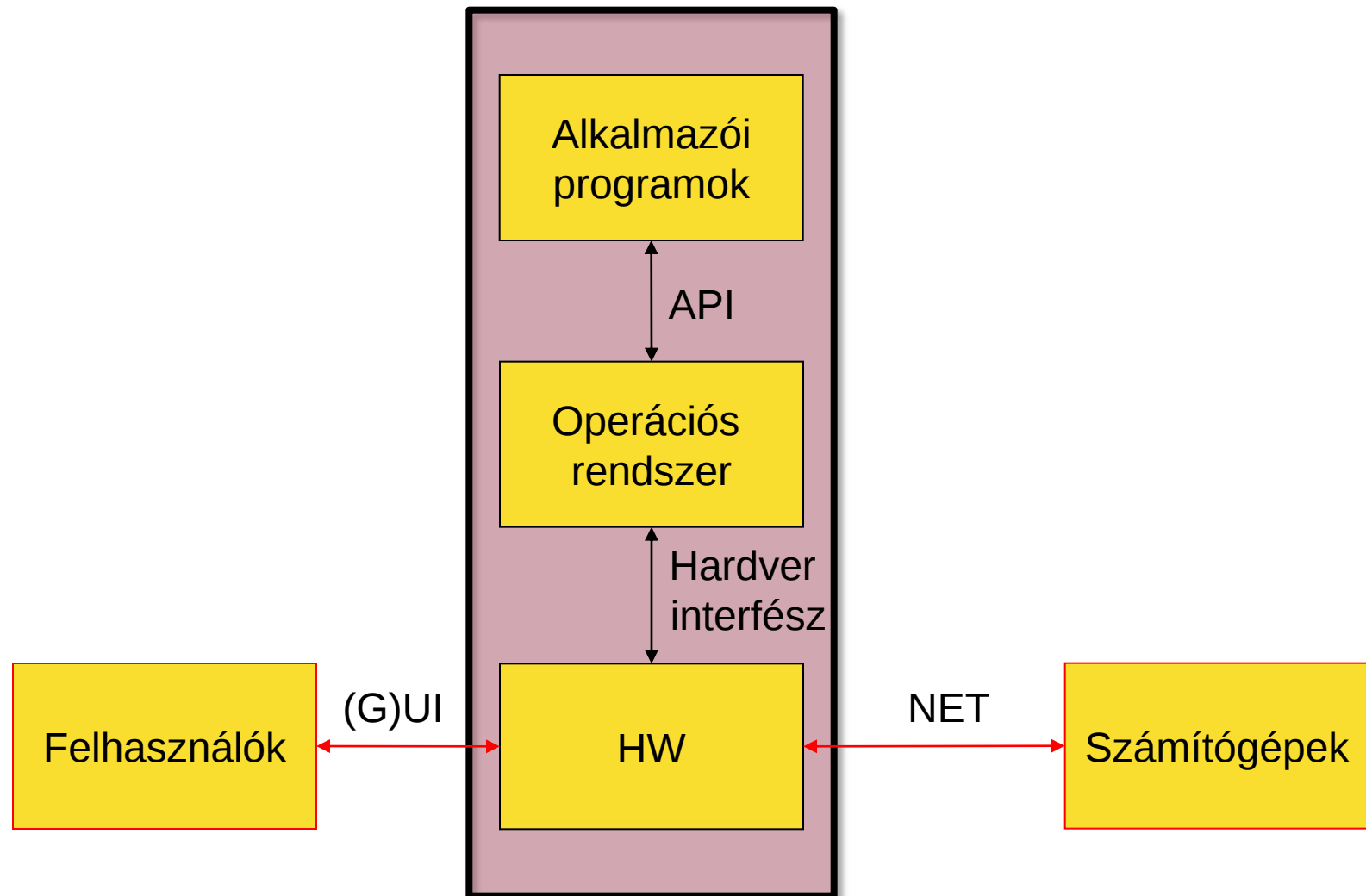


Az operációs rendszer és környezete



- Nagyon fontos, hogy a felhasználó és az alkalmazói programok, valamint a HW és az alkalmazói programom nincsenek direkt kapcsolatban egymással
- Minden az OS-en keresztül történik!
 - Teljesítmény okokból van kivétel az általános célú OS-ekben, pl. grafikus alrendszer...
- Egyes beágyazott operációs rendszerekben is vannak kivételek...

Az operációs rendszer és környezete 2.



OS-ek általános belső felépítése, rétegek 1.

■ Réteges szerkezet

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

OS-ek általános belső felépítése, rétegek 2.

■ Rétegek

- Kernel (az operációs rendszer legalapvetőbb funkcióit tartalmazó rendszermag)
 - Feladatok kezelése, tár (memória) kezelés, védelmi és biztonsági funkciók
- Hardver közeli réteg (driverok, többnyire valamilyen hierarchiába rendezve)
 - Alacsony szintű hardver kezelés, hálózat, keyboard, egér, képernyő, stb.
- Rendszerprogramok (rendszerfolyamatok és alrendszerek az egyéb funkciók megvalósítására)
 - Filerendszer, magasabb szintű hálózatkezelés (TCP/IP), parancsértelmező, stb.
- Felhasználói programokból érkező rendszerhívások fogadó felületét megvalósító réteg
 - API megvalósítása különböző célnyelveken (target language)
 - Az API leképezése rendszerhívásokra
 - Védelmi szintek váltása

Operációs rendszer kialakítások

■ Monolitikus kernel

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

■ Moduláris kernel

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

■ Mikrokernel

- Minimális funkciókkal rendelkező kernel, és ahhoz kliens-szerver architektúrában csatlakozó szolgáltatások
 - Többnyire 3 védelmi szintet használva
- Erőforrás igényesebb
- A kernel védett még a hozzá csatlakozó szolgáltatásoktól is
- Mac OS X

■ Exokernel (új kutatási terület)

- Direkt alkalmazás hozzáférés egyes perifériákhoz (hatákonyság)

Réteges szerkezet példák

■ Linux

- Alapesetben egy monolitikus kernel mag és modulárisan betölthető kernel modulok alkotják
 - Modul kezelő parancsok: modprobe, insmod, lsmod, rmmod
- Képes monolitikus kialakításban működni
 - Minden drivert és szolgáltatást belefordítunk a kernelbe
 - A modul kezelésre sincs szükség ekkor, kihagyható
 - Nem flexibilis, de sokszor erre nincs is szükség
 - Pl. Nem változik a HW
 - Ez beágyazott rendszerekben előnyös (kis méret, gyorsabb)

■ Windows

- Moduláris kernel

■ Apple OS X

- Darwin
 - Mach 3.0 mikrokernél + FreeBSD (Berkeley Software Distribution) UNIX
 - Erősen objektum-orientált keretrendszer

Hardver kezelés

- Speciális CPU regiszterek
- I/O portok (spec. I/O utasítások)
- Memóriába leképzett I/O portok (memória írás/olvasás)
 - Késleltetés és rendelkezésre álló sávszélesség
- DMA (Direct Memory Access)
 - DMA vezérlő (HW specifikus), fel kell programozni
 - A processzort megkerülve lehet adatot a periféria és a memória között mozgatni
 - Gyorsabb, a mozgatás nem igényel processzor műveleteket, de veszélyes lehet (CPU cache)
- Megszakítás (Interrupt)
 - Megszakítás vezérlő (HW specifikus)
 - Különböző szinteken tiltható és engedélyezhető
 - Ha az adott megszakítás engedélyezve van, és a megszakítás beérkezik, akkor a CPU áttér a megszakításhoz tartozó IT rutin végrehajtására (a részletek erősen hardver függőek)

Hogyan irányít az OS?

- Kontrollálnia kell a felhasználó alkalmazások működését, hogy azok egymásnak és magának az OS-nek ne tudjanak kárt okozni!
- A védelem HW szintű...

Védelmi szintek a CPU-ban

- Hierarchikus védelmi szintek (CPU privilege levels)
 - 1960 évek közepén jelenik meg (Multics, később UNIX támogatására)
 - Váltani kell közöttük: Erre szolgál a rendszerhívás (system call)
- A modern általános célú processzorok támogatják ezt a funkciót
 - Alacsony szintű CPU erőforrásokhoz történő hozzáférést szabályozza
 - Végrehajtható utasítások
 - CPU konfigurációs regiszterek
 - I/O végrehajthatósága
 - Memória területek elérése
 - Pl. x86 (286/386-től), ARM Cortex Ax sorozat, stb.
 - Mikrovezérlők (microcontroller) nem támogatják, pl. Atmel AVR, ARM Cortex Mx sorozat, stb.
- 2 vagy 4 szint
 - Jellegzetesen 2-őt használunk
 - User mode (real mode) – korlátozott elérés
 - Kernel mode (protected mode) – korlátlan elérés
 - Esetleg egy 3. is felhasználásra kerül
 - Driver és/vagy kernel service mode

Memory Management Unit (MMU)

- Speciális HW a CPU-ban
- Az MMU feladatai
 - Memória állapotának nyilvántartása
 - Tulajdonos folyamat azonosítója
 - Hozzáférési jogosultságok (ACL)
 - CACHE-elhetőség, ha van CACHE (pl. DMA)
 - Virtuális memória leképzése fizikai memóriára
 - Pl. Translation lookaside buffer (TLB)
 - Kontextus váltásnál ezt is kezelni kell (ha van)
 - Pagefile vagy SWAP (HDD)
 - Memória védelem
 - Tiltott memória hozzáférés megakadályozása vagy legalább jelzése (ACL alapján)
 - General Protection Fault (GPF) a Windows-ban
- Részletesen beszélünk róla később
- Linux, Windows, Windows CE, futtatásához szükséges
- Egyes mikrovezérlő Memory Protection Unitot (MPU) tartalmaznak
 - Csak védelem, nincs virtuális memória kezelés

Megszakítás

■ Típusai

- x86 terminológia, más architektúrák nem különböztetik meg ezeket ennyire, de nekünk ez a megközelítés szemléletes...
- Hardver: periféria használja értesítésre, időzítő, hálózati kártya, stb. jelzi a kiszolgálási igényét
 - Külső esemény hat a processzor működésére
 - Egy periféria számos okból kérhet kiszolgálást
- Kivétel (exception) : a rendszer (OS és alkalmazások) futása közben történő esemény (laphiba, numerikus túlcsordulás, nullával történő osztás, védelmi szintnek nem megfelelő működés, vagy egyéb forrás)
 - A processzor belsejéből származó esemény, hibát vagy valamilyen szoftveres kezelési igényt jelent
- Szoftver esemény (pl. rendszerhívás), speciális utasítás végrehajtása
 - Az éppen futó szoftver hat a CPU működésére, megváltoztatja azt

■ A modern operációs rendszerek megszakítás vezéreltek.

Hardver megszakítás

- Külső hardver elem kiszolgálási igényének a jelzésére
- Óra megszakítás (különösen fontos)
 - ML1 (Aki még emlékszik rá!)
 - Adott órajelű oszcillátor
 - Programozható számláló számlálja az impulzusokat
 - Adott számú impulzus után kér egy HW megszakítást
 - A megszakítás futtatja az OS-t (ütemezőt)
 - Ugyan ebből az óra megszakításból kerül származtatásra a rendszeróra
 - Periodikus vagy egyszeri várakozás
 - A felbontás az órajel periódusideje (tipikusan 1-20 ms között, tipikusan 10 ms)

Rendszerhívás

- Mi az a rendszerhívás?
 - Kiindulás: A processzor alkalmazói programot futtat
 - A rendszerhívás lényegében megszakítja azt (szoftver megszakítás), és átadja a vezérlést a kernelnek (állapot mentés és visszaállítás, context switch)
 - A kernel elvégzi a munkáját
 - Visszaadja a futás jogát az alkalmazói programnak (állapot mentés és visszaállítás, context switch)
- Hogyan történik a rendszerhívás?
 - Erősen implementáció függően...
- Mi a következménye a rendszerhívásnak
 - Nagy overhead-je van, sok CPU időt emészt fel
 - Rendszer állapotának mentése és visszaállítása 2 alkalommal
 - Minimalizálni kell az alkalmazott rendszerhívások számát
 - A CPU védelmi szintet is vált: user-kernel-user
- A rendszerhívás nem vermet használ a paraméterek átadására, teljesen máshogy működik, mint egy függvény vagy eljárás hívás!

Mi a része a kontextusnak?

- CPU regiszterek
 - PC, Státusz regiszter, munkaregiszterek, szegmens regiszterek, stb.
 - Pl. Linux esetén a kernel kontextusnak nem részei a lebegőpontos regiszterek
 - Következmény: A Linux kernelben nem használhatunk lebegőpontos aritmetikai műveleteket!
- MMU beállításai
 - Az adott alkalmazás memóriájának eléréséhez szükséges beállítások
- Egyéb alkalmazás specifikus HW beállítások
- Az egyik legkomolyabb feladat meghatározni, hogy mit kell, és mit érdemes elmenteni/visszaállítani.

Rendszerhívás a programozó szemében

- A programozó az API függvényt hívja meg a használt programozási nyelvenn
 - Pl. C nyelv esetén
 - Windows: windows.h
 - Linux: glibc
 - Az API rejti el a programozók elől a rendszerhívás részleteit
 - Lényegében egy magas szintű C wrapper a rendszerhívások körül
 - Az API megvalósítása az alkalmazásba fordul bele, vagyis felhasználói módban fut, és kiváltja a rendszerhívást
 - A rendszerhívások kezelésére az OS is tartalmaz egy komponenst, az már kernel módban fut
 - Később részletesen bemutatjuk példával
- Direktbe is meghívható a rendszerhívás, de extrém esetektől eltekintve nem ajánlott a direkt használata!

I/O műveletek

- Az alkalmazói programok nem hajthatnak végre alacsony szintű I/O műveleteket (user mode)
- Az I/O műveletek ezért rendszerhívásokon keresztül valósulnak meg
- A rendszerhívás hatására az azt kiváltó program vár az I/O művelet befejezésére
- Más program addig futhat
- A kernel hajtja végre az alacsony szintű I/O műveletet (kernel mode)
- Az I/O művelet lezajlását a periféria megszakítással jelzi
- A megszakítás hatására az OS fut, és dönthet a további teendőkről (pl. ismét az I/O műveletre váró feladatot futathatja)
- Az I/O művelet végén az alkalmazói program visszatér a rendszerhívásból

OS elindulás 1.

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

■ Bootstrap process

- Beágyazott rendszer (PC is BIOS/EFI szinten)
 - OS image ROM-ban (ROM v. flash, esetleg tömörítve)
 - ROM-ból futtatható (Harvard architektúra esetén nincs más lehetőség)
 - RAM-ba másolható (esetleg kitömörítés után) majd onnan futtatható

■ Leállás (teljes kikapcsolás vagy pl. hibernáció)

- Biztonságos leállás
 - Energiatakarékos működés részletei
 - HW leállítása vagy altatása (low power mode)
- Konzisztens állapotban hagyni a nem felejtő tárolókat (HDD)