

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

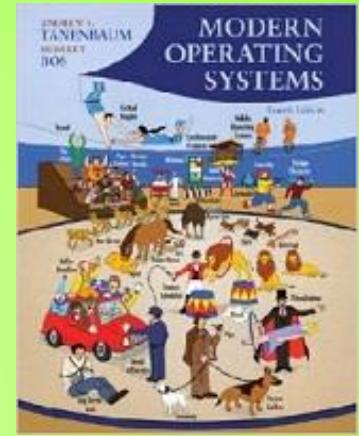
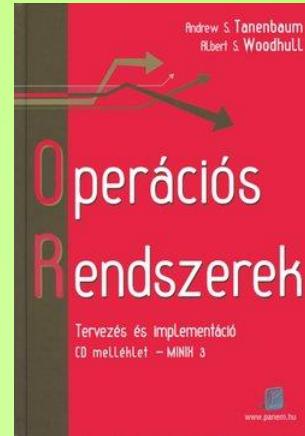
zoltan.illes@elte.hu

Elérhetőség, információ

- ▶ A „Operációs rendszerek” tárgy honlapja:
<http://os.inf.elte.hu>
- ▶ A tárgy órabeosztása: 2+1(+1)
- ▶ Gyakorlatok nem „párosával”
- ▶ Tárgy követelmény: összevont jegy
- ▶ Összevont jegy követelmények:
 - 2 beadandó feladat
 - 2 közös évfolyam zh.
 - 1. ZH időpontja: kb. 9.hét (előadás)
 - 2. ZH időpontja: utolsó hét (előadás+gyakorlat)

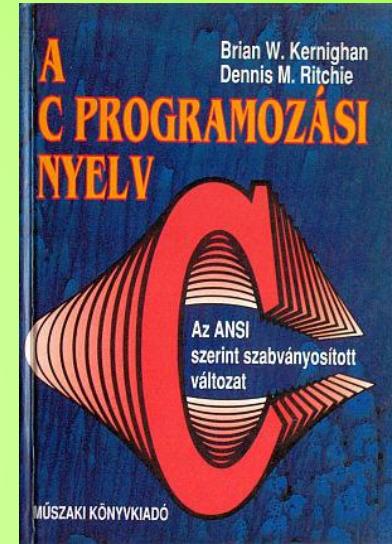
Irodalomjegyzék I.

- ▶ Andrew S. Tanenbaum, Albert S. Woodhull: Operációs rendszerek, Panem, 2. kiadás, 2007.
- ▶ Andrew S. Tanenbaum, Herbert Boss: Modern Operating Systems, 2014.
- ▶ Knapp Gábor, Adamis Gusztáv: Operációs rendszerek, LSI oktatóközpont, 2001



Irodalomjegyzék II.

- ▶ Brian W.Kernighan,
Dennis M. Ritchie:
A C programozási nyelv
 - <http://kr-c.freeweb.hu/>
- ▶ Brian W.Kernighan, Rob
Pike: A Unix operációs
rendszer



Irodalomjegyzék III.

- ▶ Büki András: Unix/Linux héjprogramozás (ez elsősorban gyakorlathoz)
- ▶ Illés Zoltán: C++ programozási nyelv, Mikrológia 23.
- ▶ <http://google.hu>
- ▶ <http://bing.com>
- ▶ Előadás anyagok
- ▶ Gyakorlat anyagok
- ▶ TÁMOP anyag



Az illusztrációk egy része az irodalomjegyzékben felsorolt művekből származik.

A tárgy célja

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

Előismeretek

- ▶ Számítógépes alapismeretek
 - Unix shell script programozás
- ▶ Programozási alapismeretek
 - C, C++ nyelv, alapvető algoritmusok ismerete.
 - C#
- ▶ 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

Gyakorlati előzetes

- ▶ Munkakörnyezet kialakítása.
 - Az os.inf.elte.hu kiszolgálót használjuk!
 - Belépő azonosító: Neptun kód
 - Jelszó, szóban. passwd parancs.
 - Szövegszerkesztő: vi, mcedit, joe
 - Lehet lokálisan is szerkeszteni
 - SSH alapú ftp-vel át kell mozgatni a fájlt.
 - A winscp editor használata.
 - Futtatáshoz a PATH módosítás.

Fordítás

- ▶ Az operációs rendszeren (SLE 12 RTE SP1) FSF C, C++ fordító van.
 - /usr/bin/gcc-4.8 Alapértelmezett C, C++ fordító, .c esetén C, .cpp esetén C++ mód.
 - cc, gcc néven könnyebben elérhetők
 - Man gcc
 - Fordítás: cc alma.c
 - Eredmény: a.out
 - Fordítás: cc -o alma -Wall alma.c
 - Eredmény: alma, összes warning
 - /usr/bin/g++-4.8 Alapértelmezett C++
 - gpp névvel is elérhető

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Bevezetés

- ▶ Visszatekintés (Számítógépes alapismeretek)
- ▶ Számítógépek felépítése I.(HW)
- ▶ Számítógépek felépítése II. (SW)
- ▶ Operációs rendszer fogalma
- ▶ Operációs rendszerek fejlődése, története
 - Múlt, Jelen, Jövő?
- ▶ Operációs rendszerek fogalmai
- ▶ Rendszerhívások
- ▶ Operációs rendszerek struktúrája

Visszatekintés

- ▶ Ahol a számítógépes alapismeretek befejeződött...
- ▶ Script programok
 - Rendszergazda legjobb barátja
 - Shell script
 - PowerShell
- ▶ Kliens-szerver – mint gép
 - HW különbségek
- ▶ Kliens-szerver – mint szolgáltatás
 - Adminisztráció
 - SW különbségek

Számítógépek felépítése (HW)

► Számítógépek felépítése

- Hardveres oldal

- Tárolt program, utasítások, adatok azonos módon (binárisan, miért?) a memóriában helyezkednek el.
- Vezérlő egység (CPU), aritmetikai–logikai egység (ALU) az utasítások végrehajtását, alapvető aritmetikai műveleteket felügyelik.
- Szükség van be/kimenetek (I/O) kezelésére, mely a gép és a külvilág kapcsolatát biztosítja.
- Ezen jellemzőket gyakran a Neumann elv elemeiként is ismerjük.

- **Alapvető elemek: Processzor, Memória, Perifériák, Háttértár**

- Összekötő kapocs: Busz (sín, adat, cím, vezérlő)

Processzor utasítások

- ▶ A rendszer gyakorlatilag minden eleme intelligens, de a kulcsszereplő: processzor
- ▶ Regiszterek: speciális memóriák, processzoron belül
 - Regiszter csoportok (általános, állapot jelző, stb)
- ▶ Utasításcsoportok
 - Adatmozgató utasítások (regiszter -memória)
 - Ugró utasítások, abszolút-relativ
 - I/O port kezelés,
 - Megszakítás kezelés stb.

Processzor védelmi szintek

- ▶ Intel 80286 minden utasítás egyenlő
- ▶ Intel 80386 nem az, 4 védelmi szint
 - Ebből 2-t használ, kernel mód (védett,protected mód) és felhasználói mód
- ▶ Tipikusan védett módú utasítások
 - Megszakítás kezelés
 - I/O port kezelés
 - Bizonyos memória kezelés
- ▶ Szoftveres megszakítás, csapda (trap) kezelése azonos a hardveres megszakítás kezeléssel
 - Különbség a megszakítás forrása!
- ▶ Megszakítások maszkolhatóak.
 - Kivéve az NMI .(Non Maskable Interrupt)

Processzor utasítások használata

- ▶ Adatok, utasítások a memóriában, ezeket a CPU végrehajtja
 - 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

Számítógépek felépítése (SW)

- ▶ **Végrehajtási, felépítési szintek**
 - Logikai áramkörök
 - CPU, mikroprogram, mikroarchitektúra szint
 - Számítógép, hardver elemek gépi kódja
 - **Operációs rendszer**
 - Rendszeralkalmazások
 - Alacsonyszintű, gépi kódú programok, meghajtók
 - Magas szintű nyelvek, programok
 - **Alkalmazások**
 - Felhasználói programok, Pasziánsz stb.

Operációs rendszer fogalma

- ▶ Operációs rendszer: Olyan program ami egyszerű felhasználói felületet nyújt, eltakarva a számítógép(rendszer) eszközeit.
- ▶ Op. Rendszer mint kiterjesztett (virtuális) gép
 - Nem érdekel hogyan, csak át akarok másolni egy képet.
- ▶ Op. Rendszer mint erőforrás menedzser
 - Nyomtatási sor kezelő (időalapú megosztás)
 - Memória (tér, címtér alapú megosztás)
- ▶ Kernel mód– Felügyelt mód
- ▶ Felhasználói mód
 - Gyakran op.rendszer feladatok is itt helyezkednek el.
- ▶ Speciális Felügyelt mód–Beágyazott rendszer

Operációs rendszer feladata

- ▶ Jól használható felhasználói felület biztosítása
 - 0. generációs felület: sajátos kapcsolótábla
 - Korai rendszerek felületei: Speciális terminálok
 - Már ekkor kialakul a mai rendszer szerkezete.
 - 80-as évek eleje: mikrogépek (ZX81 stb), Basic
 - PDP kompatibilis TPA1140, soros terminálok
 - MS DOS karakteres felület
 - Unix_X Window rendszer, Xerox, MacOS
 - Windows 3.1, 95,98,Mill,2000,XP, Win7
- ▶ Ezek mennyire jó felhasználói felületek?

Kommunikáció a perifériákkal

- ▶ **Lekérdezéses átvitel (polling)**
 - I/O port folyamatos lekérdezése.
 - Sok helyen alkalmazott technika, gyakran szinkron szoftver hívásoknál is alkalmazzák.
- ▶ **Megszakítás (Interrupt) használat**
 - Nem kérdezgetjük folyamatosan, hanem a kívánt esemény bekövetkezéskor a megadott programrész kerül végrehajtásra.
 - Aszinkron hívások (programesemények) megfelelő használata
- ▶ **DMA, közvetlen memória elérés**
 - Pl. közvetlen memória címzés: 0xb800:0

Programkönyvtárak

- ▶ Az iménti (gépi kódú, stb.) utasítások szintjei
 - Gépi kód
 - Pl: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,...Windows 7, Win32 API
 - Unix–Linux rendszerkönyvtárak, C nyelv
 - Script programozás (BASH, PowerShell)
 - Ezt láttuk, megismertük az I. félévben

Felhasználói programkönyvtárak

- ▶ Jellemzően réteges szerkezetű
- ▶ Alapvetően két rétre oszthatjuk:
 - Rendszer szintű hívás
 - Kommunikáció a perifériákkal
 - Felhasználói hívás
 - Széleskörű könyvtár biztosítás
- ▶ Milyen nyelvhez illeszkednek a könyvtárak?
- ▶ Hát a C nyelvhez! És még? A C++-hoz...😊
 - Persze más nyelvhez is, pl, Delphi-hez is van...
- ▶ Kompatibilitás

Mi a POSIX?

- ▶ **POSIX** = Portable Operating System Interface for uniX
- ▶ **Hivatalos neve:** IEEE 1003 – ISO 9945
- ▶ **A POSIX valójában egy minimális rendszerhívás (API) készlet, szabvány**
- ▶ **POSIX 1, 1a, 1b, 1c módosítások léteznek**
- ▶ **Szabvány ANSI C-vel azonos függvénykönyvtár**
- ▶ **Ma gyakorlatilag minden OS POSIX kompatibilis**
- ▶ **A Windows-nak is van POSIX felülete**
 - **Windows Services for Unix**

Fontosabb POSIX API témakörök

- ▶ Fájl, könyvtárműveletek
- ▶ Folyamatok kezelése
- ▶ Szignálok
- ▶ Csövek
- ▶ Standard C függvénykönyvtár
- ▶ Órák, időzítők
- ▶ Szemaforok
- ▶ Szinkron, aszinkron I/O
- ▶ Szálak kezelése
- ▶ Stb.

Függvénycsoport példák

- ▶ Matematikai függvények: pl. sin, cos, tan, atan, atan2, log, exp stb.
- ▶ Állománykezelő függvények: pl. creat, open, fopen, close, read, write, unlink stb.
- ▶ Könyvtárkezelő függvények: pl. 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.

Hogy használjuk a gyakorlatban?

- ▶ Operációs rendszer: Suse Linux Enterprise szerver
 - 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
 - Igyekezzünk a figyelmeztetéseket is orvosolni!

Operációs rendszer API-k

- ▶ Ahány rendszer, annyi függvénykönyvtár
- ▶ Ma is jellemző API-k:
 - Open VMS
 - OS/400
 - System V, BSD , közös rész: POSIX
 - Win32 API
 - Mac OS API
 - Windows Mobile, CE API
 - Palm OS
 - Nokia S40, S60, S80 API
 - Beágyazott API:
 - Java, .NET

Firmware – Middleware

- ▶ A két végletet láttuk: Hardware – Szoftvare
- ▶ Hardware alatt már egyáltalán nem csak a fizikai eszközt értjük.
 - Például: HDD, az operációs rendszer „logikai” kezelést végez, a valódi cilinderek elérése a HDD programjának feladata.
 - Például: BIOS,
- ▶ Firmware: Hardverbe a gyártó által épített szoftver
- ▶ Middleware: Op. Rendszer feletti réteg
 - PL: JVM

Operációs rendszer generációk I.

- ▶ Történelmi generáció: Charles Babbage (1792–1871)
 - Tisztán mechanikus, nincs op.rendszer
 - Operátor alkalmazás
 - Később mint programozót alkalmazta Ada Lovelace-t (Lord Byron lánya) (Ada nyelv)
- ▶ Első generáció, 1940–1955, kapcsolótábla, relé, vákumcső
 - Neumann János, Institute for Advanced Studies, Princeton
 - Egyedi gépek
 - Gépi kód, egyszerű matematikai számítások
 - Lyukkártyák megjelenése

Operációs rendszer generációk II.

- ▶ Második generáció 1955–1965,
tranzisztoros rendszerek
 - Megbízhatóvá váltak az elemek
 - Géptermek (mainframe) kialakulása
 - Tervezés, gyártás, programozás, üzemeltetés fázisának elkülönülése
 - Lyukkártyás, szalagos egységek, kötegelt rendszer megjelenése
 - Fortran nyelv
 - Op. Rendszer
 - FMS, Fortran monitor system
 - IBM 7094 hármasa, 1401 beolvasó – 7094 feldolgozó– 1401 megjelenítő

Operációs rendszer generációk III.

- ▶ Harmadik generáció, 1965–1980, integrált áramkörök megjelenése
 - IBM 1401 és 7094 egybeolvadása: System/360 gépcsalád
 - Azonos rendszerek, felépítések, kompatibilitás megjelenése
 - OS/360 megjelenése, ez minden gépre jó, eredmény nagy, bonyolult op. Rendszer.
 - Multiprogramozás, multitask megjelenése
 - Több feladat a memóriában egyidejűleg.
 - Spooling, időosztás megjelenése
 - Nincs közvetlen on-line munka

Operációs rendszer generációk III.

- ▶ Első időosztásos rendszer: M.I.T-en CTSS (Compatible Time Sharing System)
- ▶ MULTICS, Multiplexed Information and Computing System
 - AT&T Bell labs, General Electric támogatás
 - PL/1 nyelven készült
- ▶ Bell Labs, Ken Thompson, Multics lecsupaszítás, PDP 7->UNIX
- ▶ Két fő irány
 - Berkeley University – Berkeley Software Distribution
 - AT&T Bell Labs, System V Unix

Operációs rendszer generációk IV.

- ▶ 1980-tól napjainkig, személyi számítógépek, MS Windows
- ▶ LSI (large scale integration) áramkörök, CPU fejlődés
- ▶ Z80– CP/M (Control Program for Microcomputers)
 - ZX-81, ZX-Spectrum– Basic
- ▶ Intel x86 család, IBM PC– DOS, MS DOS
 - Parancssoros felület
- ▶ GUI– X Window, Mac OS X, MS Windows
- ▶ Hálózati, osztott rendszerek

MINIX 3

- ▶ Kezdetben a UNIX forráskód az AT&T engedélye alapján felhasználható volt.
- ▶ UNIX – nem nyílt a forráskód, AT&T 7. verziótól
- ▶ MINIX – MINI Unix, nyílt forráskód
 - A.Tanenbaum, Vrije Univ. Amszterdam
 - C nyelven készült,
- ▶ Linus Torvalds, Tanenbaum hallgatója
 - MINIX módosítás, 1994, LINUs uniX->LINUX
 - Nyílt forráskód
 - LAMP-Linux-Apache-MysqI-PhP

Rendszerhívások

- ▶ Rendszerhívásoknak nevezzük azokat a szolgáltatásokat, melyek az operációs rendszer és a felhasználói programok közti kapcsolatot biztosítják.
- ▶ Két fő csoportba sorolhatók:
 - Folyamat vagy processz kezelő csoport
 - Fájlkezelő csoport
- ▶ Programozó legjobb barátja: man, ...

Processz kezelés

- ▶ **Processz – egy végrehajtás alatt lévő program**
 - Saját címtartomány
 - Processz táblázat
 - Cím, regiszter, munkafájl adatok
 - **Processz indítás, megszüntetés**
 - Shell, gyerekfolyamatok
 - **Processz felfüggessztés**
 - memória térkép + táblázat mentés
 - **Processzek kommunikációja**
 - Szignálok

Fájlkezelés

- ▶ **Egy főkönyvtár, /**
 - Fastruktúra
 - Bejegyzés kétféle: fájl, könyvtár
- ▶ **Műveletek: másolás, létrehozás, törlés, megnyitás, olvasás, írás**
- ▶ **Jogosultságok: rwx, – adott jog hiánya**
 - SETUID, SETGID, Sticky bit
- ▶ **Fájlrendszer hozzácsatolása, mount, leválasztása, unmount**
- ▶ **Specifikus fájlok:**
 - Karakter, blokk fájlok, /dev könyvtár
- ▶ **Speciális fájl: Adatcső, pipe**

Fontosabb folyamatkezelő hívások

- ▶ **int pid=fork()** – folyamat tükrözése, szülő folyamatban $pid > 0$.
- ▶ **int i=waitpid(pid,&status,opt)** – adott (pid) gyermekfolyamat jelzésre vár, ha **opt==null**, akkor a végére.
 - **int i=wait(&status)** – egy gyerek befejezésre vár
- ▶ **Exec programcsalád**
 - **Execv** – a paraméterek egy tömbben vannak
 - **Execl** – a paraméterek felsorolva szerepelnek, null-al lezárva

Getpid(),getppid(), stb. manual !

Fontosabb szignálkezelők

- ▶ `signal(SIGKILL,handler)` – jelzés, jelzéskezelő beállítás
- ▶ `kill(pid,SIGKILL);` – jelzés küldése
 - `Pid=-1` – mindenkinnek aki él, ha jog is van hozzá!
 - `Pid=0` – a küldő processzcsoporthoz, ha van jog.
 - Default: egy parancs, alapból a pid-el azonos gpid-et is definiál (`ps j`), `getgpid()`, `setgpid(pid1,gpid)`
- ▶ `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

Fontosabb fájlkezelők

- ▶ Bináris kontra szöveges fájl
- ▶ Bináris:
 - Int fd=Open(név,O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR); – fájl nyitás
 - Int db=Read(fd, &hova, mennyit) – beolvasás
 - Int db=write(fd, &honnán, mennyit) – kiírás
 - Close(fd);
 - Int i=lseek(pid,offset,SEEK_SET) – offset beállítás
 - SEEK_CUR – beállítás, az aktuális pozícióhoz képest
 - SEEK_END – beállítás fájlméret + offset

Fájlkezelés – szöveges

- ▶ FILE* f=fopen(név, mód) – mód=r, r+,w,rb
- ▶ 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

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.

Operációs rendszer struktúrák

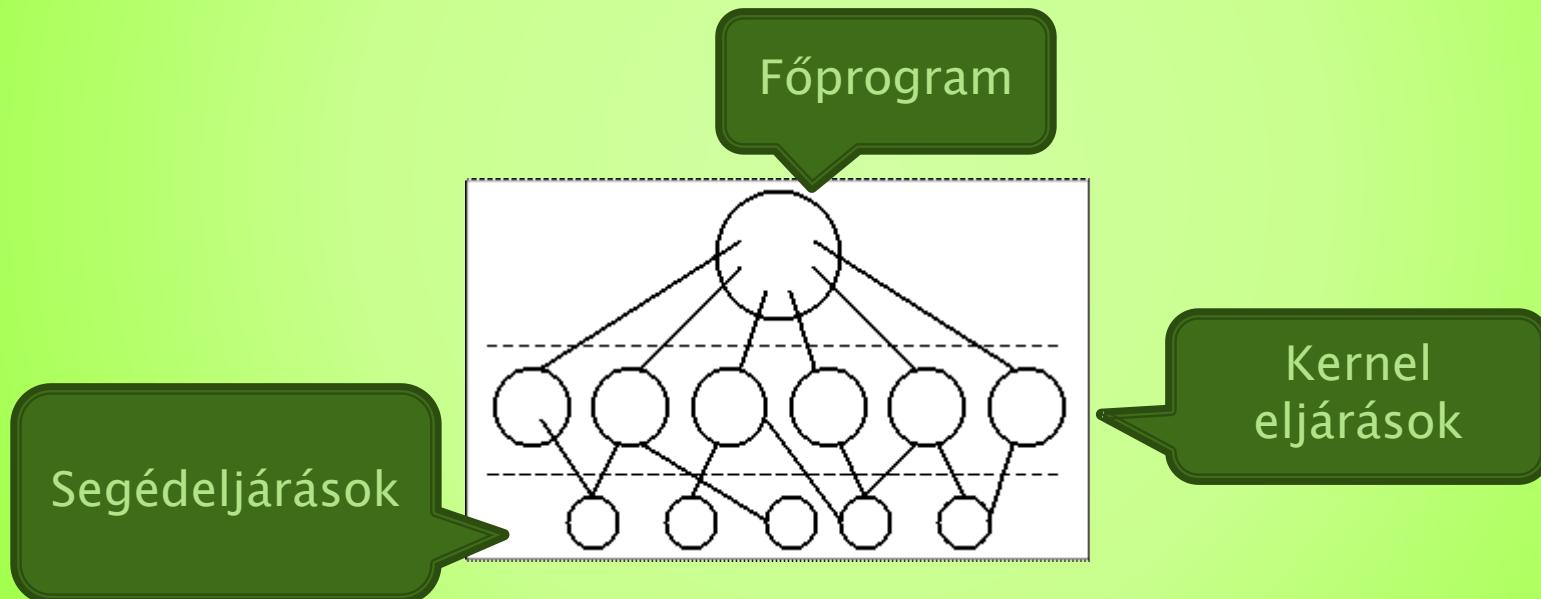
- ▶ Monolitikus rendszerek
- ▶ Rétegelt rendszerek
- ▶ Virtuális gépek
 - Exokernelek
- ▶ Kliens – Szerver modell

Monolitikus rendszerek

- ▶ Általában igaz: nincs különösebb struktúrája, de...
- ▶ Rendszerkönyvtár egyetlen rendszer, így mindenki mindenkit láthat.
 - Információelrejtés nem igazán van.
- ▶ Létezik modul, modulcsoporthoz tervezés
 - Csak az előre tervezett belépési pontok hívhatók
- ▶ Rendszerhívás során gyakran felügyelt módba (kernel mód) kapcsolja a CPU-t
 - Paraméterek jellemzően regiszterekben
 - Trap, csapdázás

Monolitikus szerkezeti modell

- ▶ Monolitikus rendszer: tipikusan 2 szintű támogatással



Rétegelt szerkezet

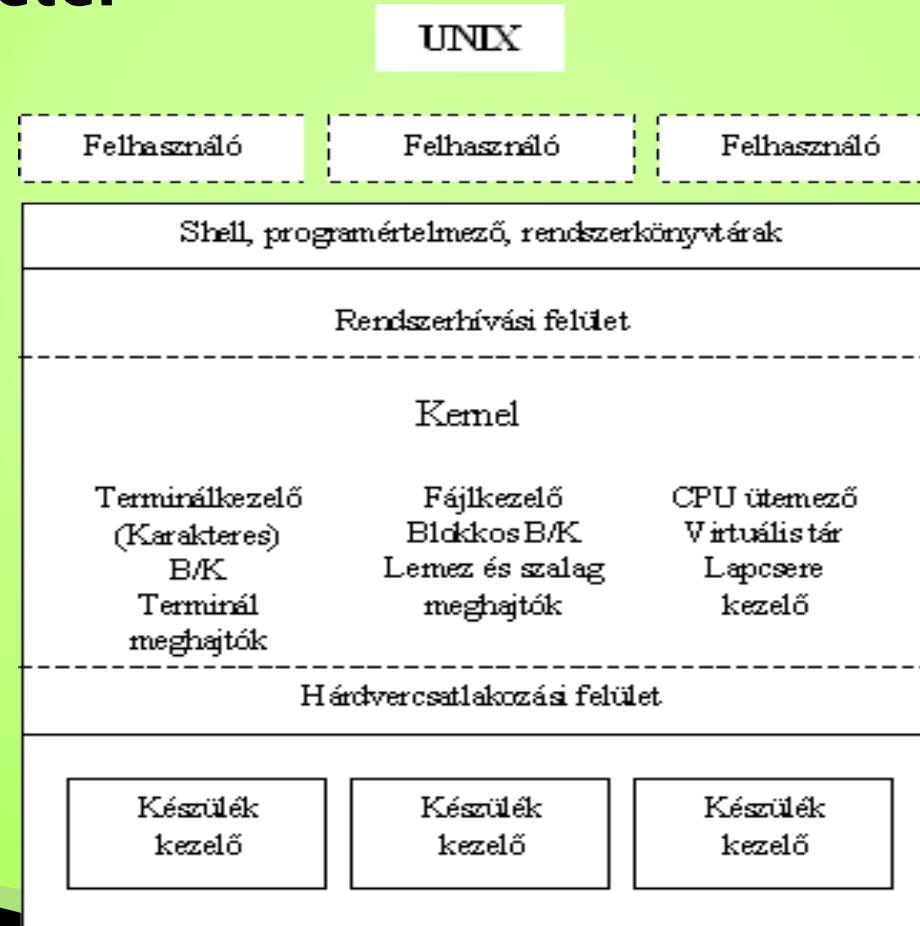
- ▶ E.W. Dijkstra tervezte, neve: THE (1968)

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

- ▶ A MULTICS-ban tovább általánosították
 - Gyűrűs rendszer

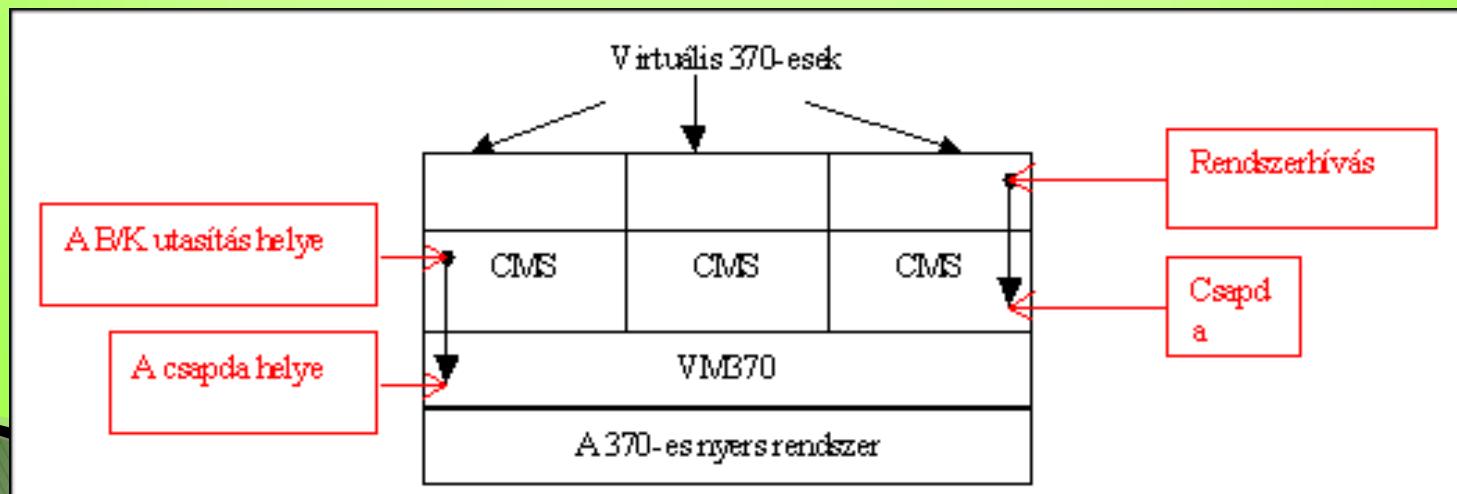
Tipikus rétegrendszer

- ▶ A Multics utód UNIX jellemző réteges, gyűrűs szerkezete.



Virtuális gépek

- ▶ Eredetileg az IBM-től származik az ötlet
- ▶ VM/370 rendszeren valósul meg először
- ▶ Virtuális gép monitor: a hardvert pontosan másolja
- ▶ Ezt tetszőleges példányban képes volt sokszorozni



Mai virtuális gépek

- ▶ **VMWare – Unix– Linux platformon**
 - Fut Windows-on is
- ▶ **MS Virtual Server, Virtual PC**
 - Létezik a Pentium utáni processzorokban 8086 virtuális üzemmód.
 - A Windows ebben futtatja a régi DOS programokat
 - Ez nem az igazi virtuális mód!!!
- ▶ **Hyper–V – XEN–KVM, VMWARE**
- ▶ **Exokernel: virtuális gép számára az erőforrások biztosítása(CPU,memória,HDD)**
- ▶ **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.
 - A Pentium 4 óta, kb. 2005, elérhető
 - Ring 0 alatt létrejön egy Root mode a host rendszernek

Multiboot – Virtualizáció összegzés

- ▶ Igény több rendszerre:
 - Partícionként más–más operációs rendszer
 - Hátrány: Újra kell indítani a gépet hamásik rendszert akarok!
 - Virtualizáció
 - Ma gyakorlatilag teljes a hardveres támogatás
 - Van elég memória
 - Van elég háttértár
 - Általános támogatás minden rendszerben.
 - Hátrány: Teljes hardver emuláció „túl sok” erőforrást igényel
- ▶ Újabb igény: Alkalmazás biztonság igénye
 - Konténertechnológia, nem kell virtuális gép!
 - Elég a kernel támogatta „alkalmazás izoláció”
 - Docker, stb

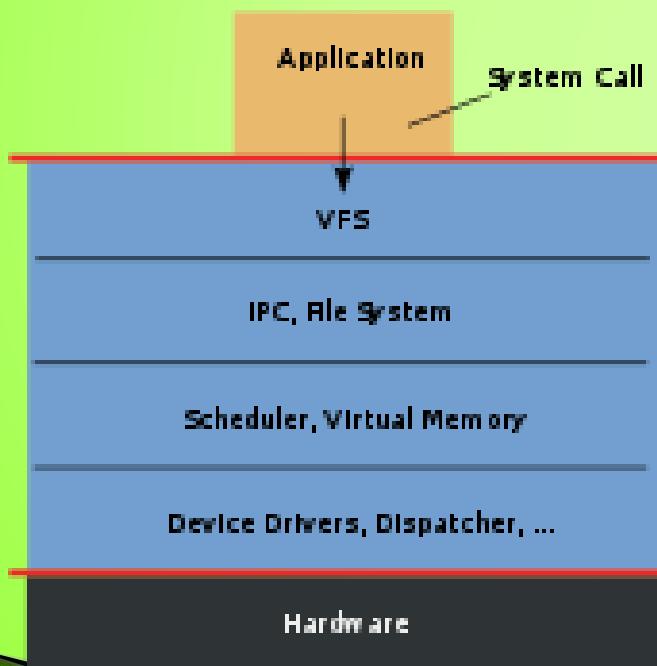
Kliens-Szerver modell

- ▶ A vm/370 ötlet továbbfejlesztése
 - Még jobban szét kell választani a feladatokat.
- ▶ Felhasználói program: kliens program
- ▶ Kiszolgáló program: szerver program
- ▶ Mindegyik felhasználói módban fut
- ▶ Egyre kevesebb funkció marad a kernelben

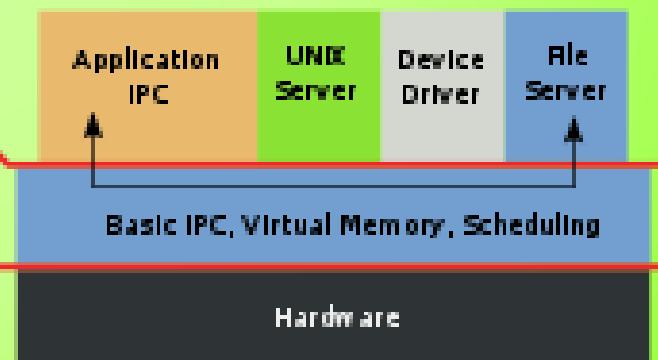


Monolitikus vs. Mikro kernel modell

Monolithic Kernel
based Operating System



Microkernel
based Operating System



Operációs rendszer elvárások I.

- ▶ Hatékonyság, a meglévő erőforrásokat a leghatékonyabban továbbítsa a felhasználók felé.
 - Efficiency
- ▶ Megbízhatóság, a hibátlan működés biztosítása.(Reliability)
 - Adatok megőrzése
 - Rendelkezésre állás (3–4 kilences...)
 - Megbízhatóság kiterjesztése: hibatűrés
 - Redundáns rendszerek (SW szinten is), Server Cluster

Operációs rendszer elvárások II.

- ▶ **Biztonság (Security)**
 - Külső rendszerekkel szemben
 - Adatbiztonság
- ▶ **Kompatibilitás, hordozhatóság (Compatibility)**
 - Két rendszer közti adat, programcsere lehetősége.
 - Szabványok szerepe (POSIX)
- ▶ **Alacsony energia felhasználás**
 - Nem csak mobil gépek esetén.

Operációs rendszer elvárások III.

- ▶ **Rugalmasság, skálázhatóság (Flexibility)**
 - Erőforrások rugalmas kiosztása (memória, processzor)
- ▶ **Kezelhetőség (Manageability)**
 - Üzemeltetési, felhasználói szinten
- ▶ **Megvalósítható minden egyszerre?**
 - A gyártók szerint igen....😊
- ▶ **A félév végén meg fogjuk látni!**

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Mi történt a múlt héten...

- ▶ Operációs rendszerek kialakulása
 - Sz.gép - Op.rendszer generációk
- ▶ Op. Rendszer fogalma
- ▶ Fogalmak:
 - Fájlok, könyvtárak, processzek
- ▶ Rendszerhívások
- ▶ Rendszer struktúrák
 - Ma: Vegyes, tipikus kliens-szerver modell, rétegelt jellemzőkkel

Mi következik ma...

- ▶ Háttértárak
- ▶ Fájlok
 - Fájltípusok
- ▶ Könyvtárak
 - Könyvtárszerkezetek
- ▶ Fájlrendszerrek
- ▶ Fájlrendszer kérés ütemezések
- ▶ Biztonsági kérdések
- ▶ ...

Háttértár típusok ma

- ▶ Mágneses elvű
 - Mágnesszalagok
 - Mágneslemezek
 - Merevlemez
 - Floppy
- ▶ Optikai elvű
 - CD, DVD, Blu-Ray, lézer elv, kb. 5xDVD a kapacitás
- ▶ Félvezető
 - USB, memóriakártya
 - SSD(Solid State Drive/Disk) diszk

Háttértár típusok holnap

- ▶ Holografikus
 - GE 2011 bejelentés, 500GB, hologramok a bitek
- ▶ Biológiai
- ▶ Nano felépítésű
- ▶
- ▶ Moore törvény, ..."1-2 évenként duplázódik az integrált áramkörök összetettsége ..", nem kifejezetten a lemezekre vonatkozik, de...

Mágnesszalagok fizikai felépítése

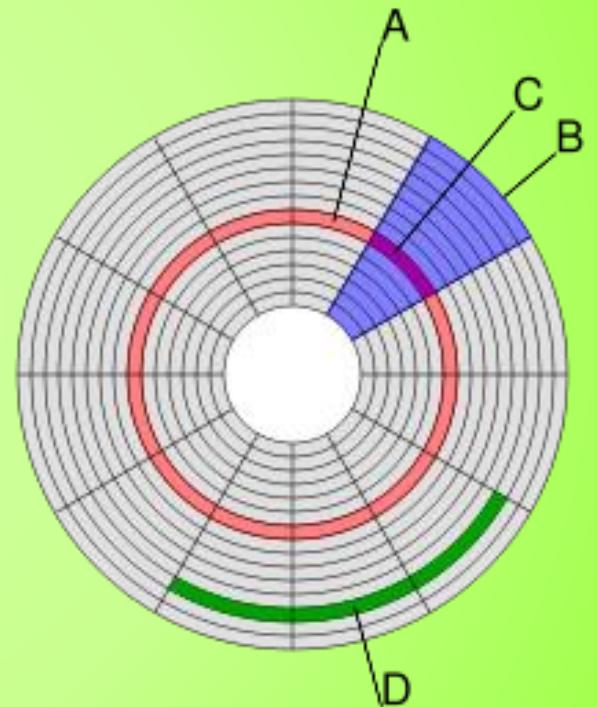
- ▶ Mágnesszalagok – sorrendi, lineáris felépítés
 - 9 bites keret (8 bit + paritás)
 - Keretek rekordokba szerveződnek
 - Rekordok között: rekord elválasztó (record gap)
 - Egymás utáni rekordok után, fájl elválasztó (file gap)
 - Szalag elején a könyvtárszerkezet
- ▶ Jellemző használat
 - Biztonsági mentés
 - Nagy mennyiségű adattárolásra
- ▶ Nem igazán olcsó
- ▶ Jellemző méret: DLT (Digital Linear Tape), LTO (Linear Tape–Open) 4 Ultrium 800/1600 GB, LTO5 1.5TB/3TB

Mágneslemezek felépítése I.

- ▶ FDD – Floppy Disk Drive
 - Jellemzően egy lemez
- ▶ HDD – Hard Disk Drive
 - Jellemzően több lemez
- ▶ Kör alakú lemez – sávos felosztás
- ▶ Sávok szektorokra oszthatók – blokk
 - Klaszter – több blokk
- ▶ Több lemez – egymás alatti sávok : cilinder
- ▶ Logikailag egy folytonos blokksorozat
- ▶ A fizikai működést a meghajtó (firmware) eltakarja.

Mágneslemez felépítése II.

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



Mágneslemez felépítése példa

- ▶ CHS címzés (Cylinder- 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: $80 \cdot 2 \cdot 18 = 2880$ szektor * 512byte
- ▶ LBA címzés (Logical Block Addressing)
 - Korábban 28 bites, kb 137GB-ig jó.
 - Jelenleg 48 bites, 144 PB (Petabájt), (144 000 000 GB)

$$A = (c \cdot N_{\text{heads}} \cdot N_{\text{sectors}}) + (h \cdot N_{\text{sectors}}) + s - 1$$

Optikai tárolók

- ▶ Tipikusan 8 vagy 12 cm átmérőjű optikai lemezek
 - CD – Compact Disc, DVD –Digital Versatile Disc
 - Méret: 650MB – 17 GB között
 - Sebesség: 1x = 150 KB/sec
- ▶ Működési elv: Fény visszaverődés 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.
- ▶ A kernelnek, az operációs rendszer magjának része.
- ▶ A lemezek írása–olvasása során jellemzően DMA-t használnak (nagy adatmennyiség).
 - Megszakítás üzenet, tipikusan azt jelzi ha befejeződött az írás–olvasás művelet.
 - I/O portokon az írás, olvasási paraméterek beállítását végzik.
- ▶ Réteges felépítés

Mágneslemez formázása

- ▶ Sávos-szektoros rendszer kialakítása
- ▶ Jellemzően egy szektor 512 byte
- ▶ Gyárilag a lemezek „elő vannak készítve”
- ▶ Quick format- Normal format
 - A normál hibás szektorokat (bad sector) keres
- ▶ Szektor= Szektorfej+adatblokk+lábléc
 - Szektorfej: sáv száma, fej száma, szektor száma
 - Lábléc: hibajavító blokk
- ▶ A szektorok kialakítását alacsonyszintű formázásnak nevezzük.

Logikai formázás

- ▶ Partíciók kialakítása
 - Egy lemezen PC-s rendszeren maximum 4 logikai lemezrész kialakítható.
- ▶ 0. szektor– MBR (Master Boot Record)
 - 2 részből áll, mérete: 512 bájt
 - Rendszerindító kód (bootloader, 446 bájt)
 - Max. 4 partíció adatai (4x 16 bájt=64 bájt)
 - 2 bájt, minden: 0x55 0xAA
 - Elsődleges partíció– erről tölthető be operációs rendszer
 - Kiterjesztett partíció– több logikai meghajtó lehet
 - Swap partíció
- ▶ A partícion a szükséges adatszerkezet (fájlrendszer) kialakítása

Az MBR szerkezete

MBR szerkezet				
Cím			Leírás	
Hex	Oct	Dec		
0000	0000	0	Betöltő programkód	440 (max. 446)
01B8	0670	440	Opcionális Disk kód	4
01BC	0674	444	Tipikusan: 0x0000	2
01BE	0676	446	Elsődleges partíciós tábla adatok (4 db 16-bájtos rész, IBM Partició Tábla séma)	64
01FE	0776	510	55h	MBR zárás: 0xAA55
01FF	0777	511	AAh	
MBR, teljes méret: 446 + 64 + 2 =				512

Partíciós tábla bejegyzés

- ▶ 1. bájt: Partíció státusa (80=aktív, 0=nem boot)
- ▶ 2–3–4. bájt : Partíció kezdőblokk CHS címe
 - 0–5. bit: fej száma
 - 6–15. bit: cilinder száma
 - 16–23. bit: szektor száma
- ▶ 5. bájt: Partíció típusa
- ▶ 6–7–8. bájt : Partíció befejező szektor CHS címe
- ▶ 9–10–11–12. bájt: Partíció kezdőszektor LBA címe
- ▶ 13–14–15–16. bájt: Szektorok száma
 - 4 bájt: $4 \text{ GB} * 512 = \underline{\text{2 TB}}$

Boot folyamat

- ▶ ROM-BIOS megvizsgálja, lehet-e operációs rendszert betölteni, ha igen betölti a lemez MBR programját a 7c00h címre.
- ▶ Egy elsődleges partíció lehet aktív, az MBR programja megvizsgálja melyik az.
- ▶ Az aktív partíció boot szektorát (1. szektor) betölti a memóriába.
- ▶ Ez már a partícióra installált operációs rendszer betöltő programja Pl. LILO, NTFS boot
- ▶ A boot program tudja, hogy a partíció melyik fájljait kell a memóriába tölteni, majd elindít egy „rendszerstartot”
 - Többszintű folyamat, rendszerfüggő.

UEFI vs. BIOS

- ▶ BIOS probléma: MAX. 2TB háttértár kezelés
 - Basic Input–Output System, IBM PC alap firmware
- ▶ UEFI (BIOS utód) – Unified Extensible Firmware Interface
 - BIOS x86 módban fut mindenhol, UEFI natívban (x64)
 - 2TB-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, 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

Lemez elérés fizikai jellemzői

- ▶ Forgási sebesség (ma tipikusan 5400,7200,10000 vagy 15000 percenként)
 - Egy sávon (cilinderen) belül mekkorát kell fordulni
- ▶ Fej mozgási sebesség
 - Egy cilinderen belül nem kell mozgatni a fejet.
- ▶ Az írás–olvasás ütemezés feladata a megfelelő (gyors, hatékony) kiszolgálási sorrend megválasztása
 - Hozzáférési idő csökkentése
 - Átviteli sávszélesség növelése

Írás–Olvasás műveletek

- ▶ Alacsonyszintű hívás során az alábbi adatok szükségesek:
 - Beolvasandó (kiírandó) blokk(ok) sorszáma
 - Memóriaterület címe, ahova be kell olvasni.
 - Bájtok száma
- ▶ Több folyamat használja
 - Melyiket hajtsuk végre először?

Írás–Olvasási műveletek ütemezése

- ▶ Alacsonyszintű (kernel) feladat paraméterek
 - Kérés típusa (írás–olvasás)
 - A blokk kezdőcíme, (sáv, szektor, fej száma)
 - DMA memóriacím
 - Mozgatandó bájtok száma
- ▶ Több folyamat is használná a lemezt
 - Kit szolgálunk ki először.
 - Fejmozgás figyelembevétele (olvasandó blokk adataiból következik)

Sorrendi ütemezés (FCFS)

- ▶ First Come – First Service
- ▶ Legegyszerűbb „stratégia”, ahogy jönnek a kérések, úgy sorban kiszolgáljuk azokat.
- ▶ Biztosan minden kérés kiszolgálásra kerül.
 - Nincs kiéheztetés.
- ▶ Nem törődik a fej aktuális helyzetével.
- ▶ Nem igazán hatékony.
- ▶ Kicsi az adatátviteli sávszélesség.
- ▶ Átlagos kiszolgálási idő, kis szórással.

SSTF ütemezés

- ▶ Shortest Seek Time First – SSTF, leghamarabb elérhetőt először
- ▶ A legkisebb fejmozgást részesíti előnyben.
- ▶ Átlagos várakozási idő kicsi.
 - A várakozási idő szórása nagy
- ▶ Átviteli sávszélesség nagy
- ▶ Fennáll a kiéheztetés veszélye

Pásztázó ütemezés

- ▶ SCAN (LOOK) módszer
- ▶ A fej állandó mozgásban van, és a mozgás útjába eső kéréseket kielégíti.
- ▶ A fej mozgás megfordul ha a mozgás irányában nincs kérés, vagy a fej szélső pozíciót ért el.
- ▶ Rossz ütemben érkező kérések kiszolgálása csak oda–vissza mozgás(írás–olvasás) után kerül kiszolgálásra.
 - Várakozási idő közepes,Szórás nagy
- ▶ Középső sávok elérés szórása kicsi

Egyirányú pásztázás

- ▶ Circural SCAN, C-SCAN
- ▶ A SCAN javítása, írás–olvasás, csak a fej egyik irányú mozgásakor történik.
- ▶ Gyorsabb fejmozgás
- ▶ Nagyobb sávszélesség
- ▶ Az átlagos várakozási idő hasonló mint a SCAN esetén, viszont a szórás kicsi.
 - Nem fordulhat elő igazán rossz ütemű kérés

Ütemezés javítások

- ▶ FCFS módszernél, ha az aktuális sorrendi kérés kiszolgálás helyén van egy másik kérés blokkja (mozgás nélkül elérhető), akkor szolgáljuk ki azt is. (Pick up)
- ▶ Egy folyamat adatai jellemzően egymás után vannak, így egy kérés kiszolgálásnál „pici” várva, a folyamat az adatainak további részét is kéri a folyamat.
 - Előlegező ütemezésnek is nevezzük
- ▶ A lemez közepe általában hatékonyan elérhető.

Ütemezés javítása memória használattal

- ▶ A DMA maga is memória
- ▶ Memória puffer (átmeneti tár) használat
 - Kettős körszerű használat
 - Olvasás: Ütemező tölti, felhasználói folyamat üríti
 - Írás: Felhasználó folyamat tölti, ütemező üríti
- ▶ Disc cache- Lemez gyorsítótár
 - Előre dolgozik az ütemező, a memóriába tölti a kért adatok „környéki” 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 ütemezést 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 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 teljesítnek.
 - 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 mi is (OS is) elősegíthetjük.
 - Összetartozó adatok együtt legyenek (töredezettség)
 - Sávszélesség a lemez közepén a legnagyobb.
 - Leggyorsabban a lemez közepét érjük el (virtuális memória)
 - Lemez gyorsító tár a memóriában.
 - Esetleg adattömörítés (nagyobb CPU terhelés)

Lemezek megbízhatósága

- ▶ Jelentése: Az adatok redundáns tárolása, hogy lemezsérülés esetén se legyen adatvesztés
- ▶ Operációs rendszer szolgáltatás
 - Dinamikus kötet– több lemezre helyez egy logikai meghajtót. Méret összeadódik.
 - Tükörözés– 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

- ▶ RAID – Redundant Array of Inexpensive Disks
- ▶ SCSI lemezegységeknél jelent meg először
 - Nem scsí...😊
 - Small Computer System Interface
 - Számítógépek és perifériák közti adatcsere egy ma is népszerű szabvány együttese.
 - Leggyakrabban lemezek körében használt, szerver gépek használják (ták)
 - Ennek egy újabb változata: SAS csatoló (Serial Attached SCSI)

RAID

- ▶ Ha operációs rendszer nyújtja, gyakran SoftRaid-nek nevezik.
- ▶ Ha intelligens (külső) vezérlőegység nyújtja, gyakran Hardver Raid-nek, vagy csak Raid diszkrendszernek nevezik.
- ▶ Bár névében olcsó (Inexpensive), valójában inkább nem az.
- ▶ Több lemezt fog össze, és egy logikai egységként látja az operációs rendszer.
- ▶ Többféle „összefogási” elv létezik: RAID 0-6

RAID 0(striping)

- ▶ Ez az a Raid, ami nem is redundáns...
- ▶ Több lemez logikai összefűzésével egy meghajtót kapunk.
- ▶ A lemezkapacitások összege adja az új meghajtó kapacitását.
- ▶ A logikai meghajtó blokkjait szétrakja a lemezekre (striping), ezáltal egy fájl írása több lemezre kerül.
- ▶ Gyorsabb I/O műveletek.
- ▶ Nincs meghibásodás elleni védelem.

RAID 1 (tükrözés)

- ▶ Két független lemezből készít egy logikai egységet.
- ▶ minden adatot párhuzamosan kiír minden két lemezre.(Tükrözés,mirror)
- ▶ Tárolókapacitás felére csökken.
- ▶ Drága megoldás.
- ▶ Jelentős hibatűrő képesség.
 - Mindkét lemez egyszerre történő meghibásodása okoz adatvesztést.

RAID 1+0, RAID 0+1

- ▶ RAID 1+0: Tükrös diszkekbtől vonunk össze többet.
- ▶ RAID 0+1: Raid 0 összevont lemezsorportból vegyük kettőt.
- ▶ A vezérlők gyakran nyújtják egyiket, másikat, mivel így is, úgy is tükrözés van, azaz drága, így ritkán használt.

RAID 2,3,4

- ▶ RAID 2: Adatbitek mellett hibajavító biteket is tartalmaz. (ECC–Error Correction Code)
Pl. 4 diszkhez 3 javító diszk
- ▶ RAID 3: Elég egy plusz „paritásdiszk”, $n+1$ diszk, Σn a kapacitás
- ▶ RAID 4: RAID0 kiegészítése paritásdiszkkel.
- ▶ Ma ezen megoldások nem gyakran használatosak.

RAID 5

- ▶ Nincs paritásdiszk, ez el van osztva a tömb összes elemére.(stripe set)
- ▶ Adatok is elosztva kerülnek tárolásra.
- ▶ Intenzív CPU igény (vezérlő CPU!!!)
- ▶ Redundáns tárolás, 1 lemez meghibásodása nem okoz adatvesztést.
 - 2 lemez egyidejű meghibásodása már igen
 - Hogy működik? (A paritásbitből meg a többiből az egy eltűnt kiszámítható!)
- ▶ N lemez RAID 5 tömbben($N \geq 3$), $n-1$ lemez méretű logikai meghajtót ad.

RAID 6

- ▶ A RAID 5 paritásblokkhoz, hibajavító kód kerül tárolásra.(+1 diszk)
- ▶ Még intenzívebb CPU igény.
- ▶ Két diszk egyidejű kiesése sem okoz adatvesztést!
- ▶ Relatív drága
- ▶ N diszk RAID 6-os tömbjének kapacitása, N-2 diszk kapacitással azonos.
- ▶ Elvileg általánosítható a módszer (3 diszk kiesése...)

RAID összegzés

- ▶ Ma leggyakrabban a RAID 1,5 verziókat használják.
- ▶ A RAID 6 vezérlők az utóbbi 1–2 évben jelentek meg.
 - Bár olcsó diszkekéről szól a RAID, de valójában ezek nem mindenkorán olcsók!
 - Itt már 2 lemez kiesik, így ez még inkább drága.
- ▶ Hot-Swap(forró csere) RAID vezérlő: működés közben a meghibásodott lemezt egyszerűen kicseréljük.

Adattárolás összefoglalása

- ▶ Adatok biztonságos tárolását biztosítja.
- ▶ Több szintű:
 1. Fizikai lemezek (HDD)
 2. Hardver RAID
 3. Partíciók
 4. Szoftver RAID
 5. Volume Manager az operációs rendszerben.
- ▶ Nem minden ellen véd
 - PL: Tápellátás elhal, emberi tévedés, stb.
 - Szoftveres támadások, vírusok.
- ▶ Hogy szerveződnek adataink a „volume”-on?

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Mi történt a múlt héten...

- ▶ Operációs rendszerek kialakulása
 - Sz.gép - Op.rendszer generációk
- ▶ Op. Rendszer fogalma
- ▶ Fogalmak:
 - Fájlok, könyvtárak, processzek
- ▶ Rendszerhívások, rendszer struktúrák
- ▶ Rendszer struktúrák
- ▶ Háttértárak
- ▶ RAID

Mi következik ma...

- ▶ **Fájlok**
 - Fájltípusok
- ▶ **Könyvtárak**
 - Könyvtárszerkezetek
- ▶ **Fájlrendszerek**
- ▶ **Fájlrendszer kérés ütemezések**
- ▶ **Biztonsági kérdések**
- ▶ ...

Fájlrendszer

- ▶ Fájl: adatok egy logikai csoportja, névvel egyéb paraméterekkel ellátva.
- ▶ Könyvtár: fájlok (könyvtárak) logikai csoporthoz kötött elhelyezés.
- ▶ Fájlrendszer: módszer, a fizikai lemezünkön, kötetünkön a fájlok és könyvtárak elhelyezés rendszerének kialakítására.

Fájlok

- ▶ A fájl az információtárolás egysége.
- ▶ Névvel hivatkozunk rá.
- ▶ Jellemzően egy lemezen helyezkedik el.
 - De általánosan az adathalmaz, adatfolyam akár képernyőhöz, billentyűzethez is köthető.
- ▶ A lemezen általában 3 féle fájl, állomány található:
 - Rendes felhasználói állomány.
 - Ideiglenes állomány
 - Adminisztratív állomány. Ez a működéshez szükséges, általában rejtett.

Fájl jellemzők

- ▶ Fájlnév: Karaktersorozat
 - Operációs rendszer függvénye, hogy milyen a szerkezete(hossza, megengedett karakterek, kis-nagybetű különbözőség)
- ▶ Egyéb attribútumok (információ)
 - Mérete, tulajdonosa, utolsó módosítás ideje, rejtett (hidden) fájl-e, rendszer fájl-e, hozzáférési jogosítványok, tulajdonos,...
- ▶ Fizikai elhelyezkedés
 - Valódi fájl, link (hard), link (soft)

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 környezeti változó

Hozzáférési jogok

- ▶ Nincs általános jogosítvány rendszer
- ▶ Jellemző jogosítványok:
 - Olvasás
 - Írás, létrehozás, törlés
 - Végrehajtás
 - Módosítás
 - Full control
- ▶ Jogok nyilvántartása
 - Attribútumként
 - ACL
 - NFS–AFS különbözőség, hasonló elv, különböző implementáció

Fájlrendszer

- ▶ Fájl: adatok egy logikai csoportja, névvel egyéb paraméterekkel ellátva.
- ▶ Könyvtár: fájlok (könyvtárak) logikai csoporthoz kötött elhelyezés.
- ▶ Fájlrendszer: módszer, a fizikai lemezünkön, kötetünkön a fájlok és könyvtárak elhelyezés rendszerének kialakítására.

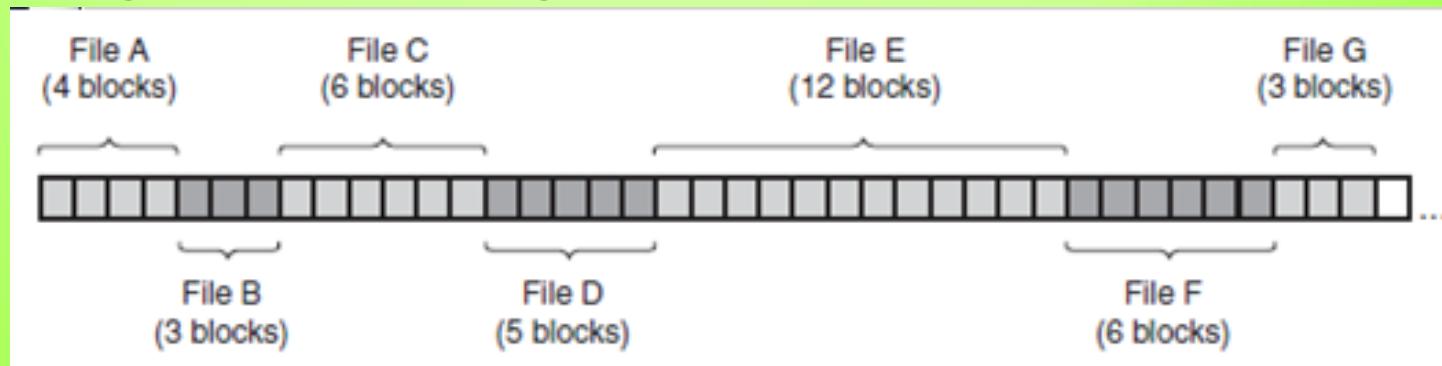
Fájlok elhelyezése

- ▶ A partíció elején, az un. Szuperblokk (pl. FAT esetén a 0. blokk) leírja a rendszer jellemzőit.
- ▶ Általában következik a helynyilvántartás (FAT, láncolt listás nyilvántartás)
- ▶ Ezután 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 elhelyezési stratégiák I.

▶ Folyamatos elhelyezésű

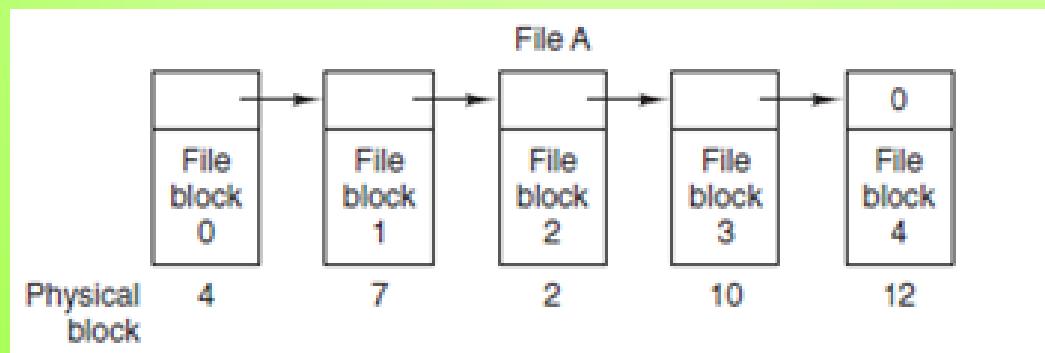
- First Fit (első szabad hely, ahová befér)
- Best Fit (arra a helyre, ahol a legkevesebb szabad hely marad)
- Worst Fit (Arra a helyre illesztjük, ahol a legtöbb szabad hely marad)
- Mindegyik veszteséges.



Fájl elhelyezési stratégiák II.

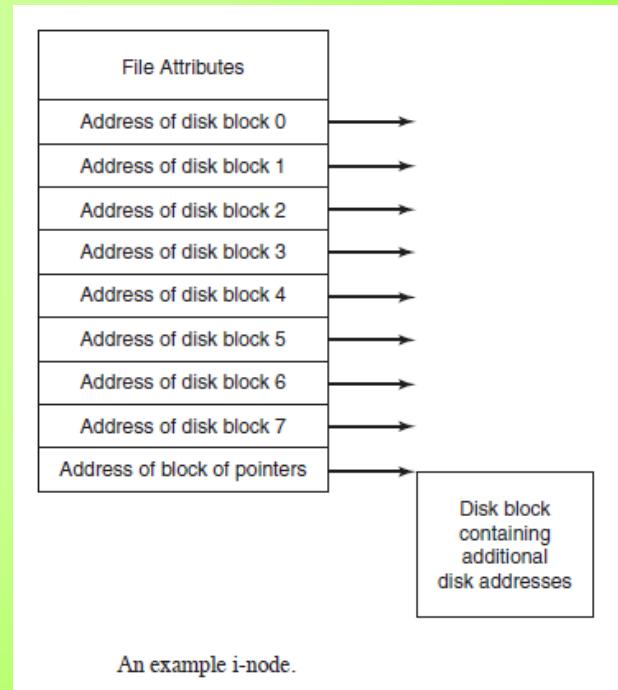
► Láncolt tárolás

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



Fájl elhelyezési stratégiák III.

- ▶ Indextáblás elhelyezés
 - A könyvtár katalógus a file node-ok címét tartalmazza
 - Az inode cím mutat a fájl adatokra.

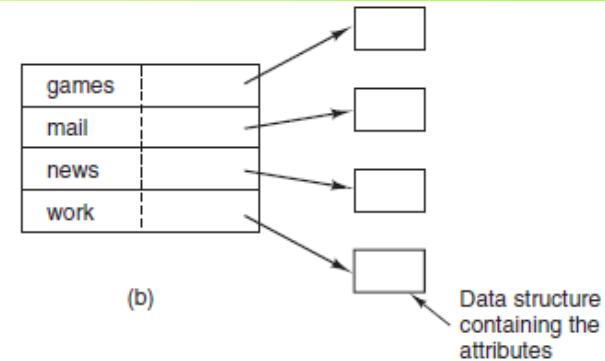


Könyvtárak megvalósítása

- ▶ A fő funkciója, hogy a névből meghatározza az adatok helyét. (keresés: lineáris, hash táblás, cache-elt)
- ▶ A könyvtárbejegyzés tartalmazhatja a:
 - A címét a teljes fájlnak (folyamatos elhelyezésnél)
 - Az első blokk címét (láncolt listánál)
 - Az i-node számot.
- ▶ Hogyan tárolja az attributumokat?
 - Bejegyzések hossza azonos, rögzített
 - I-node-okban

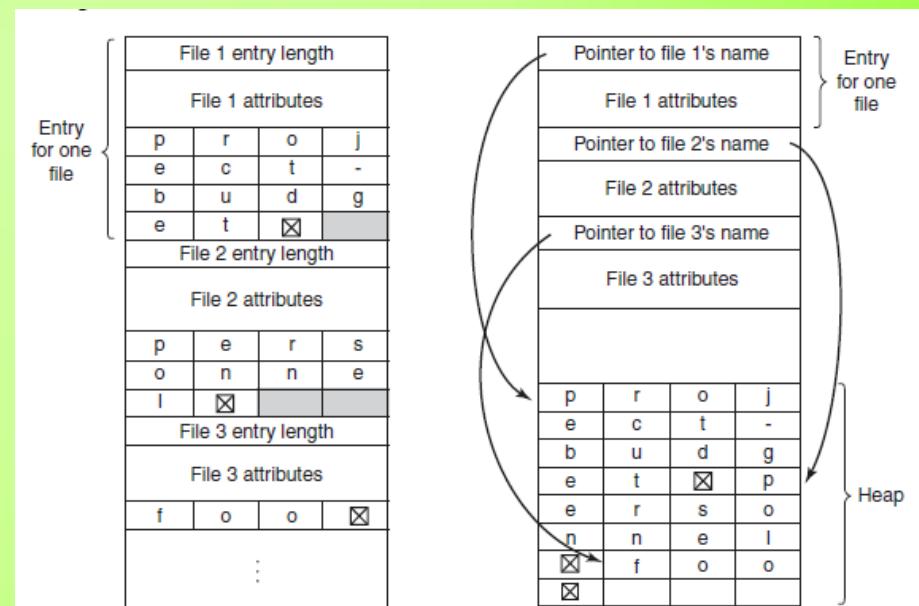
games	attributes
mail	attributes
news	attributes
work	attributes

(a)



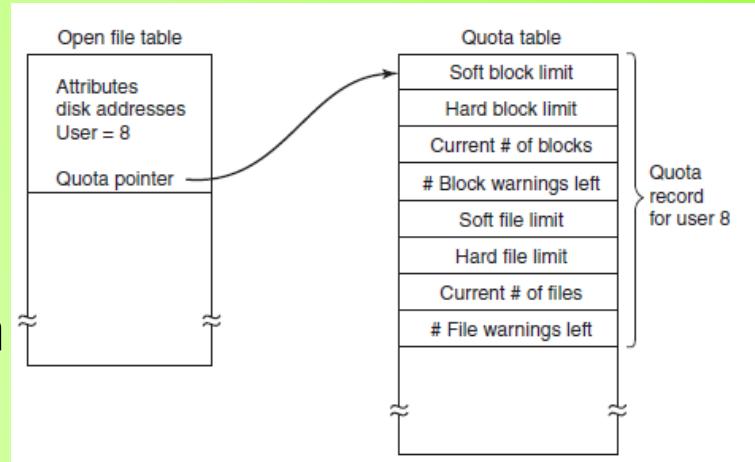
Fájlnév tárolás

- ▶ Korábban – fix hosszúságú (8+3)
- ▶ Ma – általában max 255 karakter hosszú
 - Helytakarékos megoldások:
 - Különböző hosszúságú bejegyzések.
Az első helyen a bejegyzés hossza, majd az attribútumok, majd a név.
 - Egyforma hosszú bejegyzések, mutató a fájlnévre.



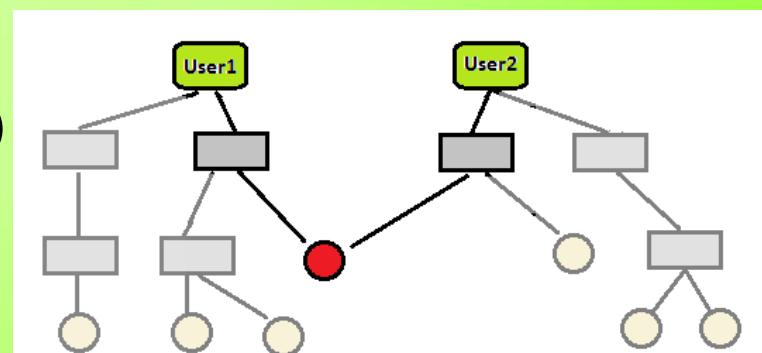
Diszk kvóták

- ▶ A felhasználói felnyitáskor
 - Megnyitja a fájl táblát a memóriában
 - A kvóta táblát a memóriában
- ▶ Ha új blokkot foglal -> változik a kvóta tábla
- ▶ Bejelenkezéskor ellenőrzés
 - Szoft limit átlépésekkel bejelentkezhet (kivéve túllépte a lehetséges figyelmeztetések számát)
 - Hard limit átlépésnél – be sem jelentkezhet



Megosztott fájlok

- ▶ Megoldandó, 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) (pl. UNIX, i-node) A user1 és user2 ugyanarra a struktúrára mutat. (Mi történik törlésnél?)
 - Új link fájl létrehozásával. (symbolic link) (lassú elérés.)

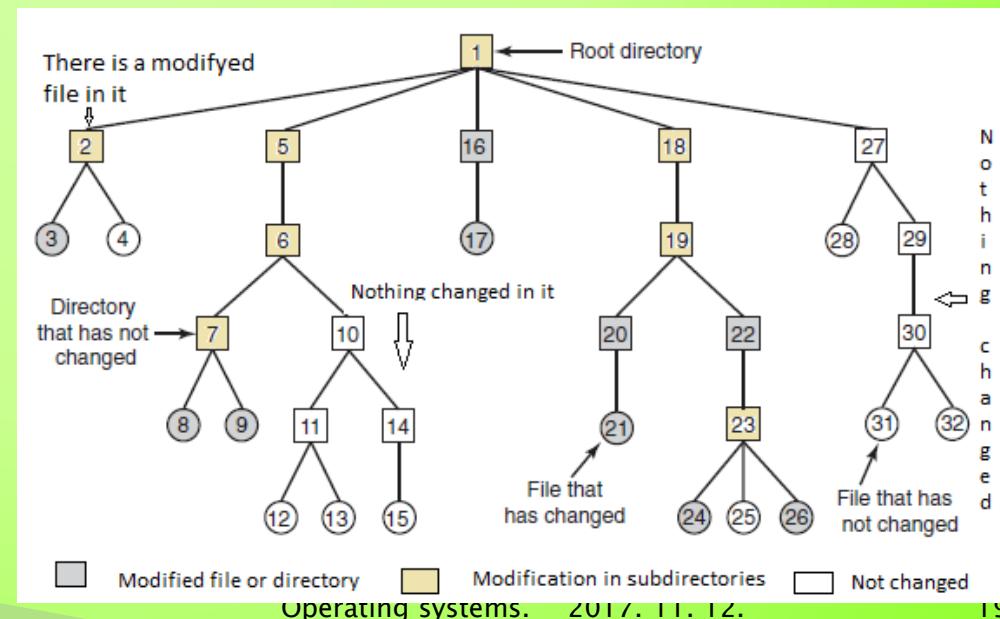


Mentések

- ▶ 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 nemcsak az egész)
 - Hátrány (bonyolult algoritmus)
- ▶ Vegyes használat
 - Időnként fizikai mentés
 - Sűrűbben logikai mentés
- ▶ Visszaállítás (fizikai mentés, első logikai mentés,, utolsó logikai mentés)

Logikai mentés – algoritmus

- ▶ A UNIX rendszerekben gyakran használt algoritmus
 - minden módosított fájl és minden könyvtár megjelölése
 - eltávolítjuk azokat a könyvtár jelöléseket, amelyekben (vagy amelyek alkönyvtáraiban) nem volt módosítás.
 - Mentjük a megjelölt könyvtárakat attribútumokkal).
 - Mentjük a megjelölt fájlokat attribútumokkal.



A logikai mentések kérdései

- ▶ A szabad blokk lista nem fájl (nincs mentve). (Visszaállítható, hiszen a komplementere a foglaltaknak.)
- ▶ Linkek – minden könyvtárban, ahol a módosított állományra volt link, azt vissza kell állítani.
- ▶ A fájlok lyukakat tartalmazhatnak (pl. seek+write) – Visszaállításnál a nem használt helyeket nem kell lefoglalni!
- ▶ Nem valódi fájlokat pl. nevesített csővezetékeket nem kell menteni.

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 törlés

Fájlrendszer típusok

- ▶ Merevlemezen alkalmazott fájlrendszer
 - FAT, NTFS, EXT2FS, XFS, stb.
- ▶ Szalagos rendszerekben (elsősorban backup) alkalmazott fájlrendszer
 - Tartalomjegyzék, majd a tartalom szekvenciálisan
- ▶ CD, DVD, Magneto-opto Disc fájlrendszere
 - CDFS, UDF (Universal Disc Format), kompatibilitás
- ▶ RAM lemezek (ma már kevésbé használtak)
- ▶ FLASH memória meghajtó (FAT32)
- ▶ Hálózati meghajtó
 - NFS
- ▶ Egyéb pszeudó fájlrendszerek
 - Zip, tar.gz, ISO

Naplózott fájlrendszerek

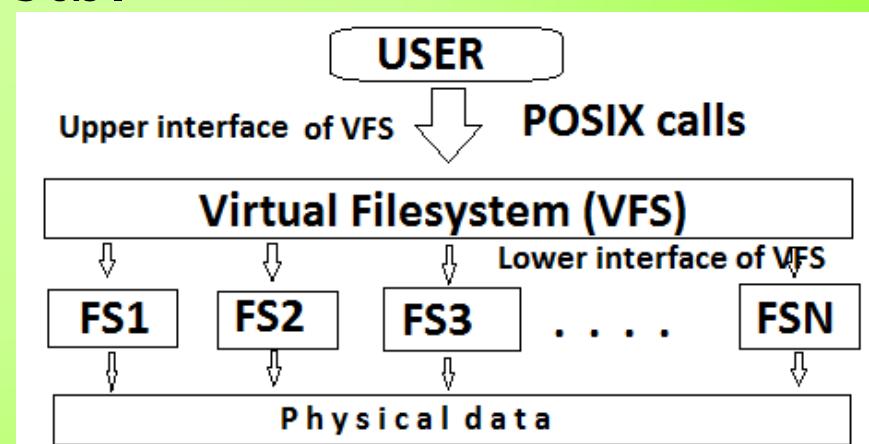
- ▶ Fájlrendszer sérülés, áramszünet stb. esetén inkonzisztens állapotba kerülhet.
- ▶ Gyakran nevezik: LFS-nek (Log-structured File System) vagy JFS-nek(Journaled)
- ▶ Adatbázis kezelők mintájára: művelet + log
 - Tranzakciós alap
 - Leállás, hiba esetén a log alapján helyre lehet állítani.
 - Célszerűen a log másik lemez (másik partíció)
- ▶ Nagyobb erőforrás igény– nagyobb megbízhatóság

Fájlrendszer támogatás

- ▶ Mai operációs rendszerek „rengeteg” típust támogatnak
 - PL: Linux 2.6 kernel több mint 50-et.
- ▶ Fájlrendszer csatolása
 - Mount, eredményeképpen 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.
 - A rendszer egymástól függetlenül használja őket. Az hogy éppen melyiket kell használni, a drive betű határozza meg. (c:, a:)
 - Pl.:UNIX: ext2,ext3, UDF, stb.
 - A modern UNIX rendszereke egységesen kezelik létrehozva egy egy közbülső réteget a virtuális fájlrendszert! (VFS)



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
- Hardver eszköz szintje
 - I/O meghajtó, IDE, SATA stb.

FAT

- ▶ **File Allocation Table**
 - Talán a legrégebbi, ma is élő fájlrendszer!
- ▶ **A FAT tábla a lemez foglaltsági térképe, annyi eleme van, ahány blokk a lemezen**
 - Pl: Fat12, FDD, Cluster méret 12 bites. Ha értéke 0, szabad, ha nem foglalt.
 - Biztonság kedvéért 2 tábla van.
- ▶ **Láncolt elhelyezés**
 - A katalógusban a file adatok (név stb.) mellett csak az első fájl blokk sorszáma van megadva.
 - A FAT blokk azonosító mutatja a következő blokk címét.
 - Ha nincs tovább, FFF az érték.
- ▶ **Rögzített bejegyzés méret, 32 bájt (max. 8.3 név)**
- ▶ **System,Hidden,Archive,Read only, könyvtár attribútumok**
- ▶ **A fájl utolsó módosítás ideje is tárolva van.**

FAT jellemzők

- ▶ FAT16, 16 bites cluster leíró, 4 bájt (2x2) írja le a fájl kezdőblokkját
 - Max. 4 GB partíciós méret (64kb blokk méretnél), jellemzően 2 GB.
 - Fájl méret maximum is a 4 (2) GB.
 - Külön könyvtári terület (FDD-n ez a 0. sáv)
 - FDD-n 512 könyvtári bejegyzés
 - HDD-n 32736 könyvtári bejegyzés (16 bit előjelesen)
- ▶ FAT32 (1996-tól elérhető)
 - 28 bites cluster leíró
 - 2 TB partíciós méret (alap szektor mérettel)
- ▶ 32MB-ig, 1 blokk = 1 szektor(512bájt)
 - 64 MB, 1 blokk=1KB (2 szektor), 128MB, 1 blokk=2KB
 - 1 blokk max. 64 KB lehet.
- ▶ Támogatták már a hosszú fájl neveket is
 - Többszörös 8.3 részre fenntartott bejegyzésekkel.
- ▶ Töredezettség mentesítés szükséges.

UNIX könyvtárszerkezet

- ▶ Indextáblás megoldás
- ▶ Boot blokk után a partíció szuperblokkja (fájlrendszer paraméterek)
- ▶ 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!
 - 15 rekeszből áll, első 12 a fájl blokkokra mutat.
 - Ha kevés,a 13. rekesz újabb i-node-ra, ami +15 rekesz.
 - Ha ez is kevés, a 14. rekesz újabb i-node-ra ami az első mintáját ismétli.

i-node láncolás példa

Gyökérkönyvtár		A /usr könyvtár i-csomópont száma 6	A 132. blokk a /usr könyvtár blokkja	A /usr/ast könyvtár i-csomópont száma 26	A 406. blokk a /usr/ast könyvtár blokkja
1	.				
1	..				
4	bin	Mód méret idő			
7	dev	132			
14	lib				
9	etc				
6	usr				
8	tmp				
A usr keresése a 6. i-csomópontot adja		A 6. i-csomópontban van, hogy a /usr a 132. blokkban található		A /usr/ast könyvtár i-csomópont száma 26	
26	.			Mode size times	
6	..			406	
64	grants				
92	books				
60	mbox				
81	minix				
17	src				
A /usr/ast/mbox fájl i-csomópont száma 60					

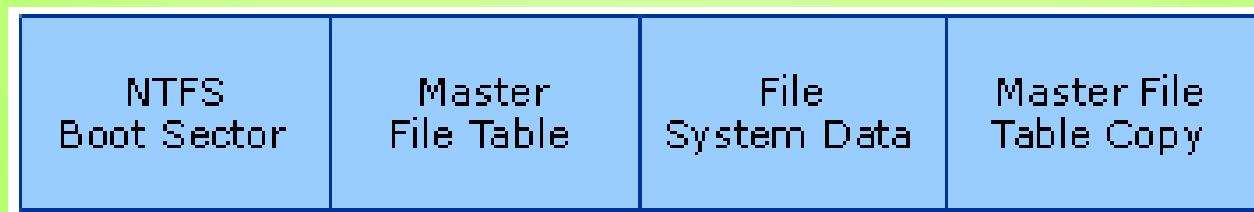
5.16. ábra. A /usr/ast/mbox keresésének lépései

NTFS

- ▶ New Technology File System
 - FAT–NTFS hatékonysági határ: kb. 400 MB.
- ▶ 255 karakteres fájl név, 8+3 másodlagos név
- ▶ Kifinomult biztonsági beállítások
- ▶ Ahogy a FAT esetén, itt is szükséges a töredezettség mentesítés.
- ▶ Titkosított fájlrendszer támogatása, naplázás
- ▶ POSIX támogatás
 - Hard link (fsutil hardlink parancs), időbelyegek, kis– nagybetűk különböznek
- ▶ Tömörített fájl, mappa, felhasználói kvóta kezelés
- ▶ Az NTFS csak klasztereket tart nyilván, szektort (512bájt) nem

NTFS partíció felépítése

- ▶ A Master File Table egy táblázat.
- ▶ A Files System Data szintén



NTFS partíció Boot szektor

- ▶ Boot sector
 - JMP +0x52 (EB 52)
 - OEMID (8 byte, MSWINx.y)
 - BPB (Bios Paraméter Blokk)
 - Bytes per sector (512)
 - Sectors per cluster (8)
 - Extended BPB
 - Total Sector number (8 byte-on tárolva)
 - LCN – Logical Cluster Number for MFT
 - Volume serial number
 - Betöltő kód (betölti az ntldr.dll-t, majd tovább az ntfs.sys,ntoskrnl.exe)
 - Sector end(0xAA55)

MFT

- ▶ NTFS partíció az MFT (Master File Table) táblázattal kezdődik
 - 16 attribútum ad egy fájl bejegyzést.
 - minden attribútum max. 1kb. Ha ez nem elég akkor egy attribútum mutat a folytatásra.
 - Az adat is egyfajta attribútum, így egy bejegyzés több adatsort tartalmazhat. (PL: Betekintő kép)
 - Elvi fájlméret 2^{64} bájt lehet
 - Ha a fájl $< 1\text{ kb}$, belefér az attribútumba, közvetlen fájl.
 - Nincs fájl méret maximum.

Az NTFS partíció felépítése

0	\$Mft – Master File Table
1	\$MftMirr – MFT Mirror
2	\$LogFile – Naplófájl
3	\$Volume – Kötetfájl
4	\$AttrDef – Attribútum definíciók
5	\ – Gyökérkönyvtár
6	\$BitMap – Cluster foglaltság
7	\$Boot – Bootszektor
8	\$BadClus – Hibás clusterek
9	\$Secure – Biztonsági leírók
10	\$UpCase – Unicode karaktertábla
11	\$Extend – Egyéb metadata
12	Nem használt
...	...
15	Nem használt
16	Felhasználói fájlok és mappák

Az NTFS metadata számára fenntartva

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK. BSC.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Miről beszéltünk korábban...

- ▶ Operációs rendszerek kialakulása
 - Sz.gép - Op.rendszer generációk
- ▶ Op. Rendszer fogalmak, struktúrák
 - Kliens–szerver modell, ...
 - Rendszerhívások
- ▶ Fájlok, könyvtárak, fájlrendszerek
 - Fizikai felépítés
 - Logikai felépítés
 - FAT, UNIX, NTFS,...

Mi következik ma...

- ▶ **Folyamatok– Processes**
 - Létrehozása, befejezése– Creating, ending
 - Folyamat állapotok– States of processes
- ▶ **Folyamatok kommunikációja– Process communication**
 - Versenyhelyzetek, kritikus szekciók– Race situation
 - Szemaforok, mutexek, monitorok
- ▶ **Klasszikus IPC problémák**

Folyamatok modellje

- ▶ Program – folyamat különbsége
- ▶ 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.

Rendszer modell

- ▶ 1 processzor + 1 rendszer memória + 1 I/O eszköz = 1 feladatvégrehajtás
- ▶ Interaktív (ablakos) rendszerek, több program, több processz fut
 - Környezetváltásos rendszer: csak az előtérben lévő alkalmazás fut
 - Kooperatív rendszer: az aktuális processz bizonyos időközönként, vagy időkritikus műveletnél önként lemond a CPU-ról (pl:Win 3.1)
 - Preemptív rendszer: az aktuális processztől a kernel bizonyos idő után elveszi a vezérlést, és a következő várakozó folyamatnak adja.
 - Real time rendszer

Folyamatok létrehozása

- ▶ Ma tipikusan preemptív rendszereket használunk (igazából a valós idejű is az)
- ▶ Több folyamat él, aktív.
- ▶ Folyamat létrehozás oka lehet:
 - Rendszer inicializálás
 - Folyamatot eredményező rendszerhívás
 - Másolat az eredetiről (fork)
 - Az eredeti cseréje (execve)
 - Felhasználói kérés (parancs&)
 - Nagy rendszerek kötegelt feladatai
- ▶ Előtérben futó folyamatok
- ▶ Háttérben futó folyamatok (démonok)

Folyamatok kapcsolata

- ▶ Szülő – gyermek kapcsolat
- ▶ Folyamatfa:
 - egy folyamatnak egy szülője van
 - Egy folyamatnak több gyermeké lehet
 - Összetartozó folyamatcsoport
 - Pl: Init, /etc/rc script végrehajtása
 - Az init id-je 1.
 - Fork utasítás...vigyázat a használatával
- ▶ Reinkarnációs szerver
 - Meghajtó programok, kiszolgálók elindítója.
 - Ha elhal az egyik, akkor azt újraszüli, reinkarnálja.

Folyamatok befejezése

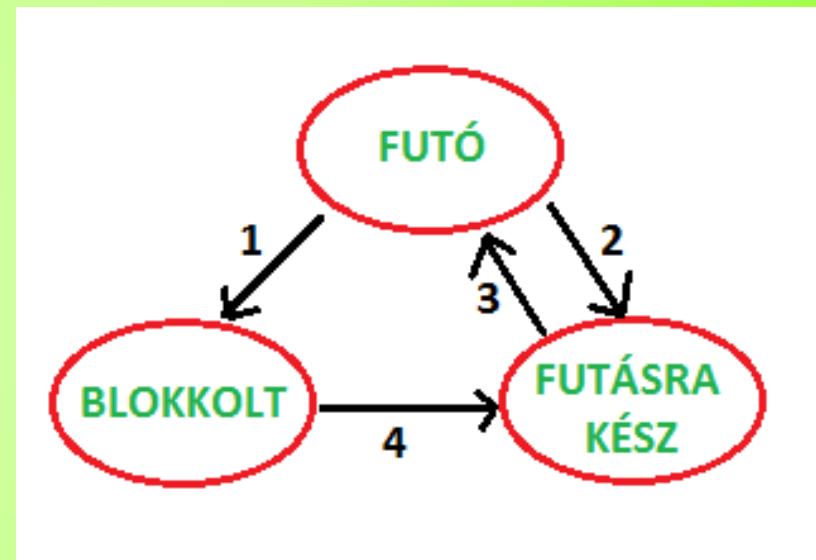
- ▶ Folyamat elindulása után a megadott időkeretben végzi (elvégzi) a feladatát.
- ▶ A befejezés okai:
- ▶ Önkéntes befejezések
 - Szabályos kilépés (exit, return stb.)
 - Kilépés valamilyen hiba miatt, amit a program felfedez (szintén pl. return utasítással)
- ▶ Önkéntelen befejezések
 - Illegális utasítás, végzetes hiba (0-val osztás, nem létező memória használat, stb)
 - Külső segítséggel. Másik processz, netán mi „lőjük” ki az adott folyamatot.

Folyamatok állapota

- ▶ Folyamat: önálló programegység, saját utasításszámlálóval, veremmel stb.
- ▶ Általában nem függetlenek a folyamatok
 - Egyik–másik eredményétől függ a tevékenység
- ▶ Egy folyamat három állapotban 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. (cat Fradi.txt|grep Fradi|sort, grep és sort blokkolt az elején...)

Állapotátmenetek

1. Futó -> Blokkolt
 - Várni kell valamire
2. Futó ->Futásra kész
3. Futásra kész ->Futó
 - Ezekről az ütemező dönt, a folyamatok nem nagyon tudnak róla.
4. Blokkolt->Futásra kész
 - A várt adat megérkezett



Folyamatok megvalósítása

- ▶ A processzor „csak” végrehajtja az aktuális utasításokat (CS:IP)
- ▶ Egyszerre egy folyamat aktív.
- ▶ Folyamatokról nem tud.
 - Ha lecseréljük az aktív folyamatot a következőre, mit kell megőrizni, hogy visszatérhessünk a folytatáshoz?
 - Mindent....utasítás számlálót, regisztereket, lefoglalt memória állapotot, nyitott fájl infókat, stb.
 - Ezeket az adatokat az un. Folyamat leíró táblában tároljuk (processz tábla, processz vezérlő blokk)
- ▶ I/O megszakításvektor

Folyamatok váltása

- ▶ Mi kezdeményezi?
 - 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
 - Gyakori váltás – többlet erőforrást igényel
 - A folyamat váltási idő „jó” megadása nem egyértelmű.

Folyamatleíró táblázat – Process Control Block (PCB)

- ▶ A rendszer inicializáláskor létrejön
 - 1 elem, rendszerindító már bent van mikor az rendszer elindul.
- ▶ Tömbszerű szerkezet(PID alapon) – de egy-egy elem egy összetett processzus adatokat tartalmazó struktúra.
- ▶ Egy folyamat fontosabb adatai:
 - Azonosítója (ID), neve (programnév)
 - Tulajdonos, csoport azonosító
 - Memória, regiszter adatok
 - Stb.

Szálak

- ▶ Tipikus helyzet: Egy folyamat – egy utasítássorozat – egy szál
- ▶ Néha szükséges lehet, hogy egy folyamaton belül „több utasítássorozat” legyen
 - Szál: egy folyamaton belüli különálló utasítás sor
 - Gyakran „lightweight 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
- ▶ Folyamatnak önálló címtartománya van, szálnak nincs!

Folyamatok-Szálak jellemzők

- ▶ Csak folyamatnak van:
 - Címtartománya
 - Globális változók
 - Megnyitott fájl leírók
 - Gyermek folyamatok
 - Szignálkezelők, ébresztők
 - ...
- ▶ Szálnak is van:
 - Utasításszámlálók
 - Regiszterek, verem

Szálproblémák

- ▶ Fork– Biztos, hogy a gyerekben kell több szál, ha a szülőben több van! (Igen)
- ▶ Fájlkezelés– Egy szál lezár egy fájlt, miközben a másik még használná!
- ▶ Hibakezelés– errno globális értéke
- ▶ Memóriakezelés–...
- ▶ Lényeges: A rendszerhívásoknak kezelní kell tudni a szálakat (thread safe)

Folyamatok kommunikációja

- ▶ IPC – Inter Process Communication
- ▶ Hárrom 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.
- ▶ Szálakra minden hárrom terület ugyanúgy érdekes, csak az információküldés, az azonos címtartomány miatt egyszerű.

„Párhuzamos” rendszerek

- ▶ Ütemező a folyamatok gyors változatával „teremt” párhuzamos végrehajtás érzetet.
- ▶ Többprocesszoros rendszerek
 - Több processzor egy gében
 - Nagyobb teljesítmény
 - Megbízhatóságot általában nem növeli
- ▶ Klaszterek (Fürt, Cluster)
 - Megbízhatóság növelése elsősorban
- ▶ Kulcskérdés: a közös erőforrások használata

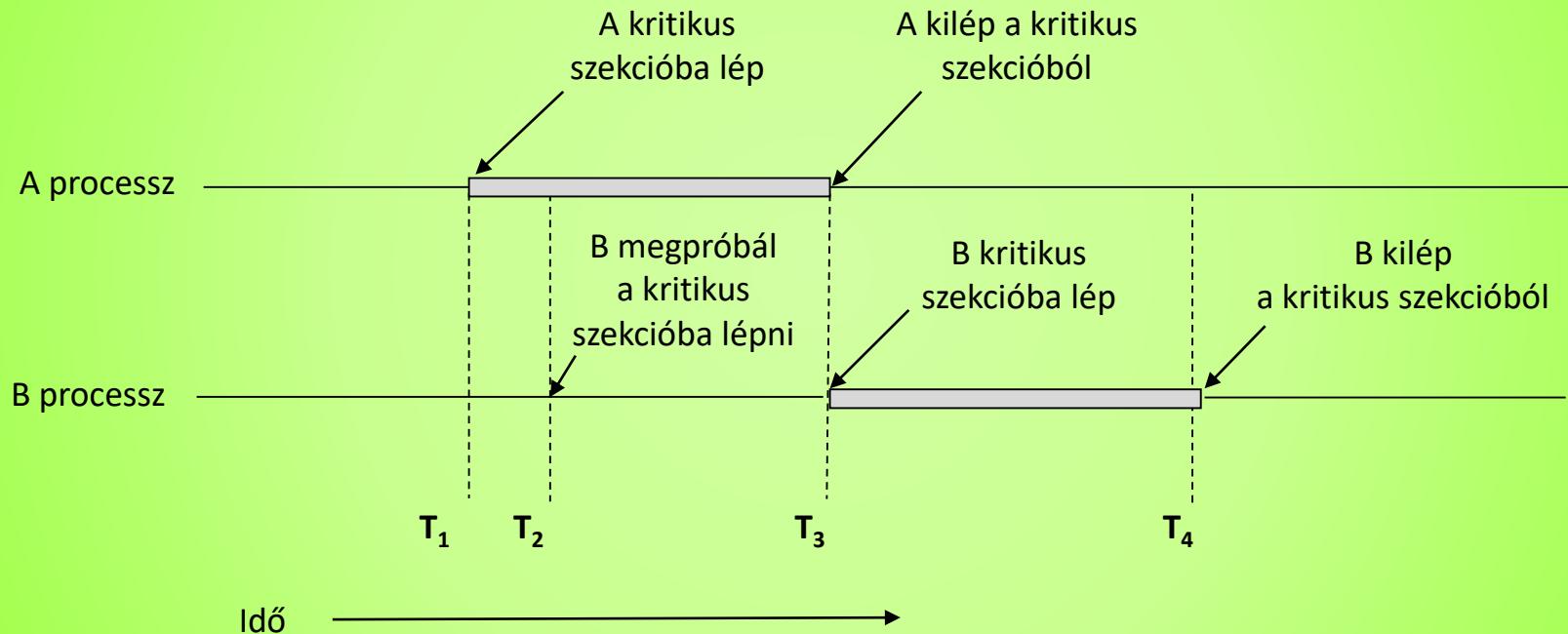
Közös erőforrások

- ▶ Avagy, amikor két folyamat ugyanazt a memóriát használja...
 - Közös ló ...
 - Pl: 2 folyamat nyomtatása, közös nyomtatósor
- ▶ 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 kizárási szekciók

- ▶ Kritikus programterület, szekció, az a rész mikor a közös erőforrást (memóriát) használjuk.
- ▶ A jó kölcsönös kizárást az alábbi feltételeknek felel meg:
 - Nincs két folyamat egyszerre a kritikus szekciójában.
 - Nincs sebesség, CPU paraméter függőség.
 - Egyetlen kritikus szekción kívül levő folyamat sem blokkolhat másik folyamatot.
 - Egy folyamat sem vár örökké, hogy a kritikus szekcióba tudjon belépni.

A megkívánt kölcsönös kizárási viselkedése



Kölcsönös kizárási megvalósítások I.

- ▶ Megszakítások tiltása (összes)
 - Belépéskor az összes megszakítás tiltása
 - Kilépéskor azok engedélyezése
 - Ez nem igazán jó, mivel a felhasználói folyamatok kezében lenne a megszakítások tiltása...persze a kernel használja.
- ▶ Osztott, un. zárolás változó használata
 - 0 (senki) és 1 (valaki) kritikus szekcióban van
 - Két folyamat is kritikus szekcióba tud kerülni!
 - Egyik folyamat belép a kritikus szekcióba, de éppen az 1-re állítás előtt a másik folyamat kerül ütemezésre.

Kölcsönös kizárás megvalósítások II.

► Szigorú változat:

- Több folyamatra is általánosítható.
- A kölcsönös kizárás feltételeit teljesíti a 3 kivételével, ugyanis ha pl 1 folyamat a lassú, nem kritikus szekcióban van, és a 0 folyamat gyorsan belép a kritikus szekcióba, majd befejezi a nem kritikus szekciót is, akkor ez a folyamat blokkolódik mert a kovetkezo=1 lesz!(Saját magát blokkolja!)
- 0. folyamat

1.folyamat

```
while(1)
{
    while(kovetkezo!=0) ;
    kritikus_szekcio();
    kovetkezo=1;
    nem_kritikus_szekcio();
}
```

```
while(1)
{
    while(kovetkezo!=1) ;
    kritikus_szekcio();
    kovetkezo=0;
    nem_kritikus_szekcio();
}
```

G.L.Peterson javítása

- ▶ 1981, a szigorú változat javítása
- ▶ A kritikus szekció előtt minden folyamat meghívja a belépés, majd utána kilépés fv-t.

```
#define N 2
int kovetkezo;
int akarja[N];
/* a módosított folyamat*/
while(1)
{
    belepes(processz);
    kritikus_szekcio();
    kilepes(processz);
    nem_kritikus_szekcio();
}
```

```
void belepes(int proc)
{
    int masik;
    masik=1-proc; //mivel N=2...
    // masik=(proc+1) % N;
    akarja[proc]=1; //processz futni akar
    kovetkezo=proc;
    while( kovetkezo==proc &&
           akarja[masik]);
}

void kilepes(int proc)
{
    akarja[proc]=0; //hamis
}
```

Kis Peterson „javítás”– nagy hiba

- ▶ Tegyük fel $proc=0$!
- ▶ A jelölt ütemezés váltásnál a $proc=1$ belépése jön.
- ▶ Mivel akarja[0] értéke 0, ezért az 1-es process belép a kritikus szakaszba!
- ▶ Ekkor újra váltson az ütemező, akarja[1]=1, a következő értéke szintén 1, így a következo==proc hamis, azaz a 0. proc is belép a kritikus szakaszba!

```
void belepes(int proc)
{
    int masik;
    masik=1-proc; //mivel N=2...
    // masik=(proc+1) % N;
    kovetkezo=proc; //két sor csere
    /* itt van ütemező váltás
    akarja[proc]=1; //proc futni akar
    while( kovetkezo==proc &&
           akarja[masik]);
}
void kilepes(int proc)
{
    akarja[proc]=0; //hamis
}
```

Tevékeny várakozás gépi kódban

- ▶ TSL utasítás – Test and Set Lock
 - Atomi művelet (megszakíthatatlan)

belepes:

```
TSL regiszter, LOCK      ; LOCK a regiszterbe kerül
                           ; és LOCK=1
                           ; TSL alatt a CPU zárolja a
                           ; memóriasínt!!!
```

```
cmp regiszter,0
jne belepes    ; ha nem 0, ugrás
ret
```

;

Kilepes:

```
mov LOCK,0
ret
```

Tevékeny várakozás

- ▶ 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 pazarlása helyett jobb lenne az, ha a kritikus szekcióba lépéskor blokkolna a folyamat, ha nem szabad belépnie!

Alvás – ébredés

- ▶ Az aktív várakozás nem igazán hatékony
- ▶ Megoldás: blokkoljuk(alvás) várakozás helyett a folyamatot, majd ha megengedett ébresszük fel.
 - sleep -wakeup, down-up, stb.
 - Különböző paraméter megadással 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 a pékség polcain van kenyér.
 - Ha tele van kenyérrel a pékség, akkor „a pék elmegy pihenni”.
 - Ha üres a pékség, akkor a vásárló 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, mert még a vásárló nem alszik.
 - Vásárló visszakapja az ütemezést, a helyet korábban beolvasta, az 0, megy aludni.
 - A pék az első után megsüti a maradék $N-1$ kenyeret és ő is aludni megy!
- ▶ Lehet ébresztő bittel javítani, de több folyamatnál a probléma nem változik.

Szemaforok I.

- ▶ E.W. Dijkstra (1965) javasolta ezen új változótípus bevezetését.
- ▶ Ez valójában egy egész változó.
- ▶ A szemafor tilosat mutat, ha értéke 0.
 - A folyamat elalszik, megáll a tilos jelzés előtt.
- ▶ Ha a szemafor >0 , szabad a pálya, beléphetünk a kritikus szakaszra.
- ▶ Két művelet tartozik hozzá:
 - Ha beléptünk, csökkentjük szemafor értékét. (down)
 - Ha kilépünk, növeljük a szemafor értékét. (up)
 - Ezeket Dijkstra P és V műveletnek nevezte.

Szemaforok II.

- ▶ Elemi művelet: a szemafor változó ellenőrzése, módosítása, esetleges elalvás, oszthatatlan művelet, nem lehet megszakítani!
- ▶ Ez garantálja, hogy ne alakuljon ki versenyhelyzet.
- ▶ Ha a szemafor tipikus vasutas helyzetet jelöl, azaz 1 vonat mehet át csak a jelzőn, a szemafor értéke ekkor 0 vagy 1 lehet!
 - Bináris szemafor
 - Ezt MUTEX-nek (Mutual Exclusion) is hívjuk, kölcsönös kizáráásra használjuk.

Szemafor megvalósítások

- ▶ Up, Down műveleteknek 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...

Gyártó–fogyasztó probléma megoldása szemaforokkal I.

► Gyártó (pék) függvénye

```
typedef int szemafor;
szemafor szabad=1; /*Bináris szemafor,1 lehet tovább, szabad a jelzés*/
szemafor üres=N, tele=0; /* üres a polc, ez szabad jelzést mutat*/
void pék()           /* N értéke a „kenyerespolt” mérete */
{
    int kenyér;
    while (1)
    {
        kenyér=pék_süt();
        down(&üres);      /* üres csökken, ha előtte>0, lehet tovább*/
        down(&szabad);    /* Piszálhatjuk-e a pékség polcát? */
        kenyér_polcra(kenyér); /* Igen, betesszük a kenyeret. */
        up(&szabad);     /* Elengedjük a pékség polcát. */
        up(&tele);       /* Jelezzük vásárlónak, van kenyér. */
    }
}
```

Gyártó–fogyasztó probléma megoldása szemaforokkal II.

► Fogyasztó (Vásárló) függvénye.

```
void vásárló()          /* vásárló szemaforja a tele */
{
    int kenyér;
    while (1)
    {
        down(&tele);      /*tele csökken, ha előtte>0, lehet tovább*/
        down(&szabad);    /*Piszálhatjuk-e a pékség polcát? */
        kenyér=kenyér_polcról(); /* Igen, levesszük a kenyeret. */
        up(&szabad);      /* Elengedjük a pékség polcát. */
        up(&üres);        /* Jelizzük péknek, van hely, lehet sütni. */
        kenyér_elfogyasztása(kenyér);
    }
}
```

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.

Szemafor példa

► Unix környezetben:

- semget: szemafor létrehozása(System V)
- semctl: szemafor kontrol, kiolvasás, beállítás
- semop: szemafor operáció (up,down művelet)
- sembuf struktúra
- Gyakorlaton részletesen szerepel
- sem_open,sem_wait,sem_post,sem_unlink (Posix)

► Most nézzünk egy C# példát szemaforokra.

- VS 2008.
- Szemafor–pék–vásárló példa.

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Miről beszéltünk korábban...

- ▶ Operációs rendszerek kialakulása
- ▶ Op. Rendszer fogalmak, struktúrák
- ▶ Fájlok, könyvtárak, fájlrendszerek
 - Fizikai felépítés
 - Logikai felépítés
- ▶ Folyamatok
 - Létrehozásuk, állapotuk
- ▶ Folyamatok kommunikációja
 - Kritikus szekciók, szemaforok.

Mi következik ma...

- ▶ **Folyamatok kommunikációja**
 - Monitorok
 - üzenetküldés
- ▶ **Klasszikus IPC problémák**
 - Étkező filozófusok esete
- ▶ **Folyamatok ütemezése**
 - Elvek, megvalósítások
 - Szálütemezés

Mi a baj a szemaforokkal?

- ▶ Semmi...viszont apró elírások nehezen felderíthető programhibákhoz vezetnek.
 - Pl. ha felcseréljük a 2 sort, akkor mikor a bolt tele van, a péket az üres szemafor blokkolja, a vásárlót emiatt pedig a szabad blokkolja, ezért minden két folyamat blokkol, egymásra várnak! (holtpont)

```
void pék() /*eredeti recept*/
{
    int kenyér;
    while (1)
    {
        kenyér=pék_süt();
        down(&üres);
        down(&szabad);
        kenyér_polcra(kenyér);
        up(&szabad);
        up(&tele);
    }
}
```

```
void pék() /*cserélt recept*/
{
    int kenyér;
    while (1)
    {
        kenyér=pék_süt();
        down(&szabad); /*itt a csere*/
        down(&üres);
        kenyér_polcra(kenyér);
        up(&szabad);
        up(&tele);
    }
}
```

Van baj a szemaforokkal?

- ▶ Alapvetően nincs, de ahogy az előbbi csere során láttuk, kicsi tévesztés, nagy nehézséget tud okozni.
- ▶ Ugyanígy bárhol az up, down utasítások felcserélése hasonló eredményt ad.
- ▶ Valamelyik elhagyása, hiba...
- ▶ Lehet valami jobb?
 - Kernel (gépi kód) szinten nem igazán.

Monitorok

- ▶ Brinch Hansen (1973), Charles Anthony Richard Hoare (1974) magasabb szintű nyelvű konstrukciót javasoltak.
- ▶ Ezt nevezték el monitornak.
 - Kicsit a mai osztálydefinícióra hasonlít.

Monitor veszélyes_zóna

```
Integer polc[];  
Condition c;  
Procedure pék(x);  
...
```

```
End;  
Procedure vásárló(x);  
...
```

```
End;  
End monitor;
```

Monitorok 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 beléphet, végrehajthatja a kívánt monitor eljárást.

Monitor megvalósítása

- ▶ Mutex segítségével
- ▶ A felhasználónak nincs konkrét ismerete róla, de nem is kell.
- ▶ Eredmény: sokkal biztonságosabb kölcsönös kizárási megvalósítás
- ▶ Apró gond: mi van ha egy folyamat nem tud továbbmenni 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. I.

► N elem

```
monitor Pék–Vásárló
    condition tele, üres;
    int darab;
    kenyér polcra helyez(kenyér elem)
    {
        if (darab==N) wait(tele);
        polcra(elem);
        darab++;
        if (darab==1) signal(üres);
    }
    kenyér kenyér levez_a_polcról()
    {
        if (darab==0) wait(üres);
        kenyér elem=kenyér polcról();
        darab--;
        if (darab==N-1) signal(tele);
        return elem;
    }
end monitor
```

Pék–Vásárló folyamata.

```
pék()
{
    while(1)
    {
        kenyér új;
        új=kenyér_sütés();
        Pék–Vásárló.kenyeret_polcra_helyez(új);
    }
}
vásárló()
{
    while(1)
    {
        kenyér új_kenyér;
        új_kenyér=Pék–Vásárló.kenyeret_a_polcról();
        lakoma(új_kenyér);
    }
}
```

Más megoldások

- ▶ Az előző az un. Pidgin Pascal megoldás vázlat volt.
- ▶ C-ben nincs monitor
 - C++-ban igen, wait, notify
- ▶ Java:
 - Synchronized metódusok
 - Nincs állapotváltozó, de van wait, notify
- ▶ C#
 - Monitor osztály
 - Enter,TryEnter,Exit,Wait,Pulse (ez a notify megfelelője)
 - Lock nyelvi kulcsszó
 - Példa: VS2008 párhuzamos solution, monitor projekt.

Mi a baj a monitorokkal?

- ▶ Hm,...semmi.
- ▶ Sokkal biztonságosabb mint a szemafor használat.
- ▶ Egy vagy több CPU, de csak egy közös memória használatnál jók!
- ▶ Ha a CPU-knak önálló saját memóriájuk van, akkor ez a megoldás nem az igazi...

Üzenetküldés

- ▶ A folyamatok jellemzően két primitívet használnak:
 - Send(célfolyamat, üzenet)
 - Receive(forrás, üzenet)
 - Forrás tetszőleges is lehet!
- ▶ Rendszerhívások, nem nyelvi konstrukciók
- ▶ Ha küldő–fogadó nem azonos gépen van, szükséges un. nyugtázó üzenet.
 - Ha küldő nem kapja meg a nyugtát, ismét elküldi az üzenetet.
 - Ha a nyugta veszik el, a küldő újra küld.
 - Ismételt üzenetek megkülönböztetése, sorszám segítségével.

Gyártó–fogyasztó probléma üzenetküldéssel I.

► 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 -ben
        m=üzenet_készítés(kenyér);
        send(vásárló,m); // elküldjük a kenyeret a vásárlónak
    }
}
```

Gyártó–fogyasztó probléma üzenetküldéssel II.

► Fogyasztó–vásárló folyamata:

```
void vásárló()          // vásárló folyamata
{
    int kenyér;      // „kenyér” elem tárolási hely
    message m;       // üzenet tároló helye
    int l;
    for(i=0;i<N;i++) send(pék,m); // N darab üres helyet
                                    // küldünk a péknek
    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 minden két helyen.
- ▶ El lehet hagyni, ekkor ha send előtt van receive, a küldő blokkolódik, illetve fordítva.
 - Ezt hívják randevú stratégiának.
 - Minix 3 is randevút használ, rögzített méretű üzenetekkel.
 - Adatcső kommunikáció hasonló, csak az adatcsőben nincsenek üzenethatárok, ott csak bájtsorozat van.
- ▶ Üzenetküldés a párhuzamos rendszerek általános technikája. Pl. MPI

Klasszikus IPC problémák I.

► Étkező filozófusok esete:

- 2 villa kell a spaghetti evéshez
- A tányér melletti villákra pályáznak.
- Esznek-gondolkoznak
- Készítsünk programot, ami nem akad el!



Megoldás I.

- ▶ A megoldásnak apró hibája, hogy pl. holtpont lehet, ha egyszerre megszerzik a bal villát és minden várnak 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 filozofus(int i)
{
    while(1)
    {
        gondolkodom();
        kell_villa(i); // bal villa
        kell_villa((i+1)%N); //jobb
        eszem();
        nemkell_villa(i);
        nemkell_villa((i+1)%N);
    }
}
```

Megoldás II.

```
Int s[5];          // eszik, éhes, gondolkodom értékei lehetnek
Szemafor safe_s=1; //jelző az s tömb használatához
Szemafor filo[5]={0,0,0,0,0}; //1 szemafor minden filozófushoz, 0=tilos
Void filozófus(int i)
{
    while(1) {
        gondolkodom();
        down(safe_s); //csak én módosítom s[]-t
        s[i]=éhes;   //
        if (s[bal]!=eszik && s[jobb]!=eszik) //vajon szabad a 2 szomszed villa?
            { s[i]=eszik; up(filo[i]); }; //i eszik, filo[i] szabad jelzést mutat
        up(safe_s); // s[]-t más is elérheti
        down(filo[i]); // blokkol, ha nincs 2 villa, ha nem eszik az i. filozófus
        spaghetti_evés();
        down(safe_s); // evést befejeztem,újra védem s[]-t, mert módosítom
        s[i]=gondolkodom;
        if (s[bal]==éhes && s[bal2]!=eszik) { s[bal]=eszik;up(filo[bal]);}
        if (s[jobb]==éhes && s[jobb2]!=eszik) { s[jobb]=eszik;up(filo[jobb]);}
        up(safe_s);
    }
}
```

Megoldás III.

- ▶ Legyen 5 villa szemaforunk az egyes villákra.
- ▶ Max szemafor
- ▶ Ez korlátozott erőforrás megszerzésre példa.

```
Int N=5;  
Szemafor villa[]={1,1,1,1,1}; //mind  
szabad  
Szemafor max=4; //max 4 villa használt  
//egyszerre  
Void filozófus(int i)  
{  
    while(1)  
    {  
        gondolkodom();  
        down(max);  
        down(villa[i]); // bal villa  
        down(villa[(i+1)%N]); //jobb  
        eszem();  
        up(villa[i]);  
        up(villa[(i+1)%N]);  
        up(max);  
    }  
}
```

Olvasók–Írók probléma

- ▶ Adatbázist egyszerre többen olvashatják, de csak 1 folyamat írhatja:

```
// író folyamat
Szemafor database=1;
Szemafor mutex=1;
int rc=0;
Void író()
{
    while(1)
    {
        csinál_valamit();
        down(database); // kritikus
        írunk_adatbázisba();
        up(database);
    }
}
```

```
Void olvasó()
{
    while(1)
    {
        down(mutex);
        rc++;
        if (rc==1) down(database);
        up(mutex);
        olvas_adatbázisból();
        down(mutex); // kritikus
        rc--;
        if (rc==0) up(database);
        up(mutex);
        adatot_feldolgozunk();
    }
}
```

Ütemezés

- ▶ Korábbiakban láttuk több folyamat képes „párhuzamosan” futni.
- ▶ Egyszerre csak 1 tud futni.
- ▶ Melyik fusson?
- ▶ Aki a döntést meghozza: Ütemező
- ▶ Ami alapján eldönti, hogy ki fusson: ütemezési algoritmus

Folyamatok I/O igénye

- ▶ Egy folyamat jellemzően kétféle tevékenységet végez:
 - Számolat magában
 - I/O igény, írni, olvasni akar adatot perifériára
- ▶ Számításigényes folyamat
 - Hosszan dolgozik, keveset várakozik I/O-ra
- ▶ I/O igényes folyamat
 - Rövideket dolgozik, sokszor várakozik I/O-ra

Mikor váltsunk folyamatot?

- ▶ Biztosan van váltás:
 - Ha befejeződik egy folyamat
 - Ha egy folyamat blokkolt állapotba kerül (I/O vagy szemafor miatt)
- ▶ Általában van váltás:
 - Új folyamat jön létre
 - I/O megszakítás bekövetkezés
 - I/O megszakítás után 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
 - mindenki 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

Ütemezés kötegelt rendszerekben

I.

- ▶ Sorrendi ütemezés, nem megszakítható
 - First Come First Served – (FCFS)
 - Egy folyamat addig fut, amíg nem végez vagy nem blokkolódik.
 - Ha blokkolódik, a sor végére kerül.
 - Pártatlan, egyszerű, láncolt listában tartjuk a folyamatokat.
 - Hátránya: I/O igényes folyamatok nagyon lassan végeznek.
- ▶ Legrévidebb feladat először, nem megszakítható ez se, (shortest job first–SJB)
 - Kell előre ismerni a futási időket
 - Akkor optimális, ha a kezdetben mindenki elérhető

Ütemezés kötegelt rendszerekben

II.

- ▶ Legrövidebb maradék futási idejű következzen
 - Megszakítható, minden új belépéskor vizsgálat.
- ▶ 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.

Ütemezés interaktív rendszerben I.

► Körben járó ütemezés–Round Robin

- mindenkinél időszelet, aminek végén, vagy blokkolás esetén jön a következő folyamat
- Időszak végén a körkörös listában következő lesz az aktuális folyamat
- Pártatlan, egyszerű
- Egy listában tárolhatjuk a folyamatokat (jellemzőit), és ezen megyünk körbe-körbe.
- Egy kérdés van: Mekkora legyen az időszak?
 - Processz átkapcsolás időigényes
 - Kicsi az idő -> sok CPU megy el a kapcsolatokra
 - Túl nagy -> interaktív felhasználóknak lassúnak tűnhet pl a billentyűkezelés

Ütemezés interaktív rendszerben II.

▶ Prioritásos ütemezés

- Fontosság, prioritás bevezetése
 - Unix: 0–49 → nem megszakítható (kernel) prioritás
 - 50–127 → user prioritás
- Legmagasabb prioritású futhat
 - Dinamikus prioritás módosítás, különben éhenhalás
- Prioritási osztályok használata
 - Egy osztályon belül Round Robin
 - Ki kell igazítani a folyamatok prioritását, különben az alacsonyak nagyon ritkán jutnak CPU-hoz.
 - Tipikusan minden 100 időszaknál a prioritásokat újraértékeli
 - Jellemzően a magas prioritások alacsonyabbra kerülnek, majd ezen a soron megy RR. A végén újra felállnak az eredeti osztályok.

Ütemezés interaktív rendszerben III.

- ▶ Többszörös sorok
 - Szintén prioritásos és RR
 - Legmagasabb szinten minden folyamat 1 időszámlálót kap
 - Következő 2-t, majd 4-et, 8, 16, 32, 64-et.
 - Ha elhasználta a legmagasabb szintű folyamat az idejét egy szinttel lejjebb kerül.
- ▶ Legrévidebb folyamat előbb
 - Bár nem tudjuk a hátralévő időt, de becsüljük meg az előzőekből!
 - Öregedés, súlyozott átlag az időszámlálóra.
 - $T_0, T_0/2+T_1/2, T_0/4+T_1/4+T_2/2,$
 $T_0/8+T_1/8+T_2/4+T_3/2$

Ütemezés interaktív rendszerben IV.

► Garantált ütemezés

- minden aktív folyamat arányos CPU időt kap.
- Nyilván kell tartani, hogy egy folyamat már mennyi időt kapott, ha valaki arányosan kevesebb időt kapott az kerül előbbre.

► Sorsjáték ütemezés

- Mint az előző, csak a folyamatok között „sorsjegyeket” osztunk szét, az kapja a vezérlést akinél a kihúzott jegy van
- Arányos CPU időt könnyű biztosítani, hasznos pl. video szervereknél

► Arányos ütemezés

- Vegyük figyelembe a felhasználókat is! Mint a garantált, csak a felhasználókra vonatkoztatva.

Ütemezés valós idejű rendszerben I.

- ▶ Mi az a valós idejű rendszer?
 - Az idő kulcsszereplő. Garantálni kell adott határidőre a tevékenység, válasz megadását.
 - Hard Real Time (szigorú), abszolut, nem módosítható határidők.
 - Soft Real Time (toleráns), léteznek a határidők, de ezek kis mértékű elmulasztása tolerálható.
 - A programokat több kisebb folyamatra bontják.
 - Külső esemény észlelésekor, adott határidőre válasz kell.
 - Ütemezhető: ha egységnyi időre eső n esemény CPU válaszidő összege ≤ 1 .
- ▶ Unix, Windows valós idejű?

Ütemezési elvek, megvalósítás

- ▶ Gyakori a gyermek folyamatok jelenléte a rendszerben.
- ▶ A szülőnek nem biztos, hogy minden gyermekével azonos prioritásra van szüksége.
- ▶ Tipikusan a kernel prioritásos ütemezést használ (+RR)
 - Biztosít egy rendszerhívást, amivel a szülő a gyermek prioritását adhatja meg
 - Kernel ütemez – felhasználói folyamat szabja meg az elvet, prioritást. (nice)

Szálütemezés

▶ Felhasználói szintű szálak

- Kernel nem tud róluk, a folyamat kap időszeletet, ezen belül a szálütemező dönt ki fusson
- Gyors váltás a szálak között
- Alkalmazásfüggő szálütemezés lehetséges

▶ Kernel szintű szálak

- Kernel ismeri a szálakat, kernel dönt melyik folyamat melyik szála következzen
- Lassú váltás, két szál váltása között teljes környezetátkapcsolás kell
- Ezt figyelembe is veszik.

Folyamatok, prioritások

- ▶ A Unix, Linux prioritás alapú folyamat ütemezést végez!
- ▶ POSIX4 – IEEE1003.1b (1993), Valós idejű kiterjesztés megjelenés
- ▶ Két prioritás lista
 - nice -20-19, ahogy láttuk
 - Valós idejű prioritási lista, 0-99 közti értékekkel.(100 prioritási szint)
 - Ebben a listában a nagyobb szám jelenti a nagyobb prioritást
 - A lista közös értelmezése a következő:
 - 99,98...1,-20,-19..0,1...,19, ezt gyakran 140-es prioritás intervallumnak neveznek, amiben különböző eltolások(mapping) lehetségesek.

Windows prioritás osztályok

- ▶ Windows operációs rendszer 32 prioritás szintet használ.
 - 0...31-ig, a 0. szintet az ún. zero page thread használja csak!
 - Feladata: kitörölni a memórialap tartalmát!
 - 1–31-ig használt a standard folyamatok számára
 - 1–15 -ig használják a normál folyamatok
 - 16–31-ig valós idejű osztályhoz tartozó folyamatok
- ▶ A prioritási szintek prioritási osztályokba vannak csoportosítva!

Linux ütemezés – O(1)

- ▶ Korábban láttuk, prioritásos, preemptív ütemezést használnak a mai interaktív rendszerek!
- ▶ O(1) ütemezés (ordó 1 – konstans keresési idő, kb 2003-tól, 2.6 kerneltől)
 - Molnár Ingo (ELTE, fizikus, Red-Hat kernel fejlesztő)
 - Processz számtól független, konstans idő alatti ütemezés.
 - SMP támogatás– runqueue
 - I/O, CPU igény szerinti heurisztikus szétválasztás. (bonyolult, nem ad biztos eredményt)

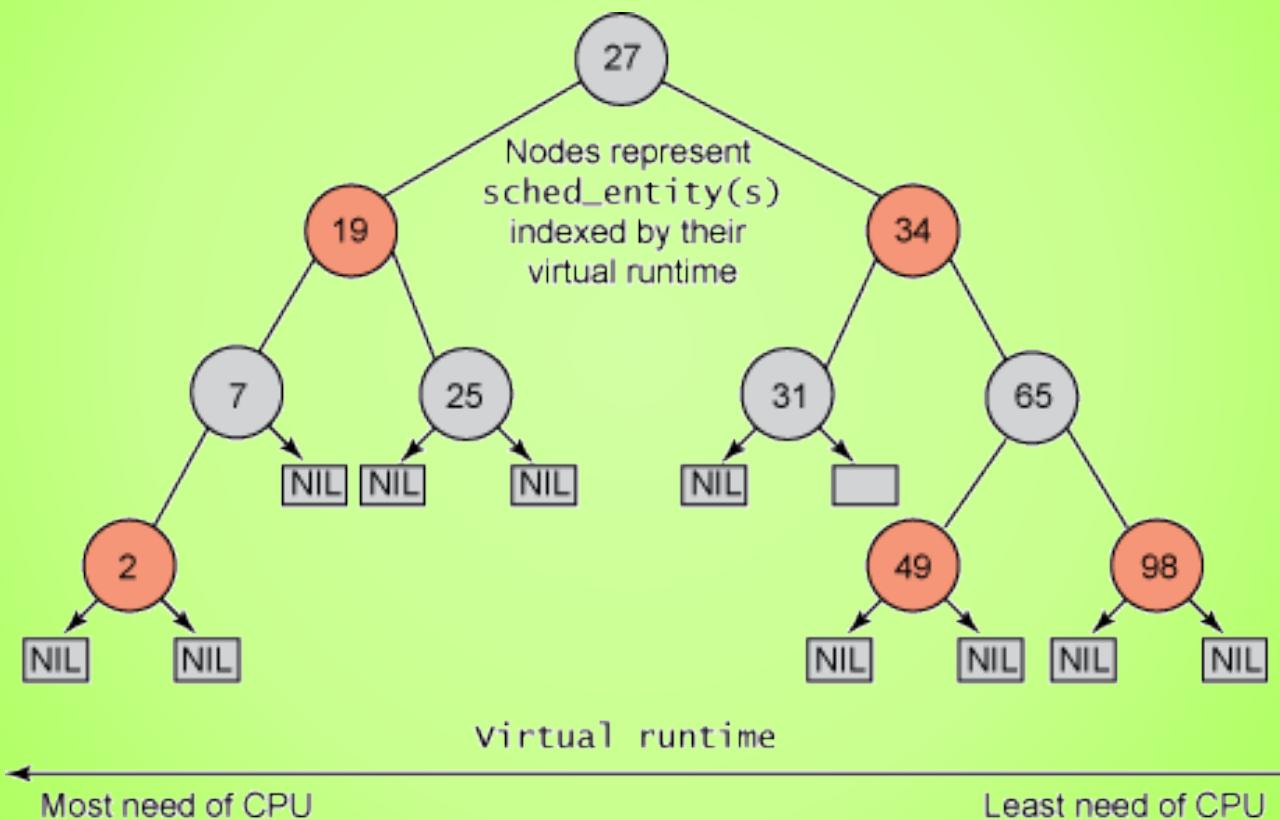
O(1) ütemezés

- ▶ Processzoronkénti futási sor (runqueue)
 - minden futási sor 140 elemű láncolt lista, minden elem egy „dupla” tömbmutató!
 - minden prioritási szinthez tartozó folyamatokból egy aktív és lejárt tömb mutatót tart nyilván.
 - végigmegy az aktív tömbön, ha egy folyamat időszelete lejár, átkerül a lejárt tömbbe!
 - ha kiürül az aktív tömb, akkor helyet cserélnek!
 - Ütemezés esetén

CFS- Completely Fair Scheduler

- ▶ O(1)- továbbfejlesztése (Molnár Ingo)
- ▶ 2.6.23 kerneltől kezdve
- ▶ Kon Colivas, Rotating Staircase Deadline Scheduler (RSDL) elemeket is használ.
- ▶ A CPU idő „fair” kiosztása, hasonlít a garantált ütemezésre!
- ▶ A CPU idők nyilvántartása egy fa struktúrában, balra kisebb, jobbra nagyobb idejű folyamatok (redblacktree)

CFS - Sched-entity fa



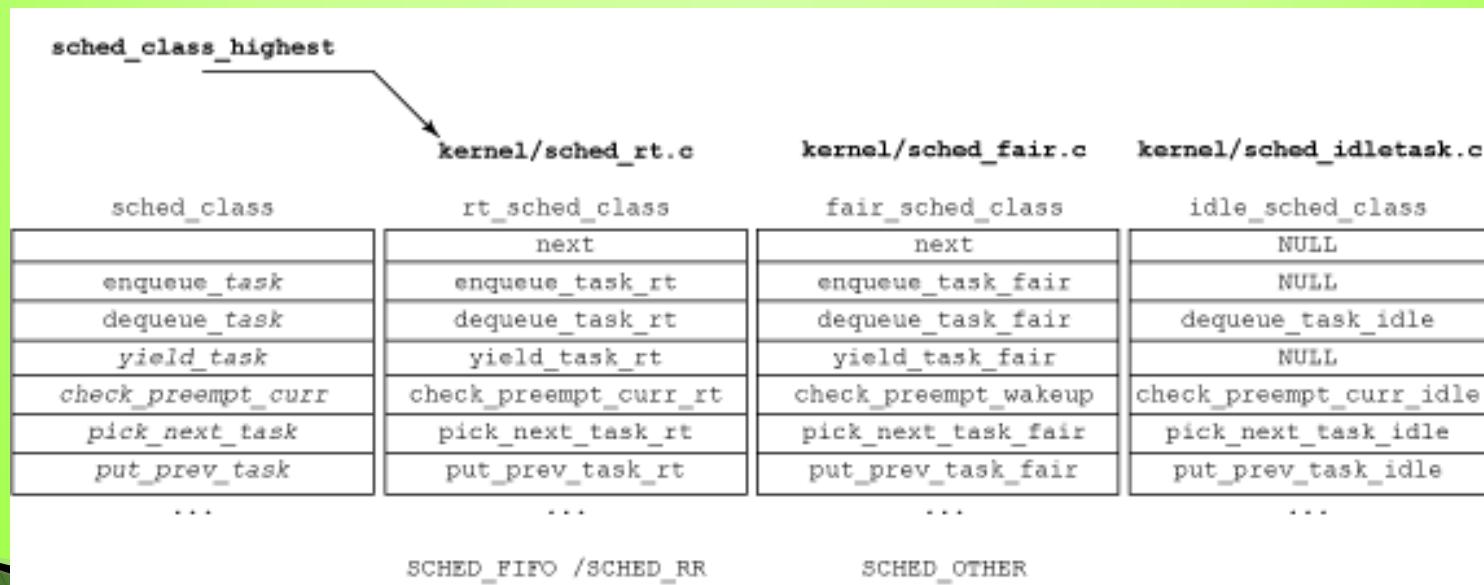
Prioritások CFS esetén

- ▶ Nincs direkt prioritás CFS-ben!
- ▶ mindenki azonos, fair módon részesül a CPU erőforrásokból, de:
 - A nagyobb prioritású folyamat kisebb idő csökkenést szenved el.
 - Az alacsonyabb prioritás nagyobbat!
- ▶ Így érvényesül a prioritási elv, nincs kiéheztetés!
- ▶ Nem kell prioritásonkénti folyamat nyilvántartás!

CFS - moduláris ütemezés

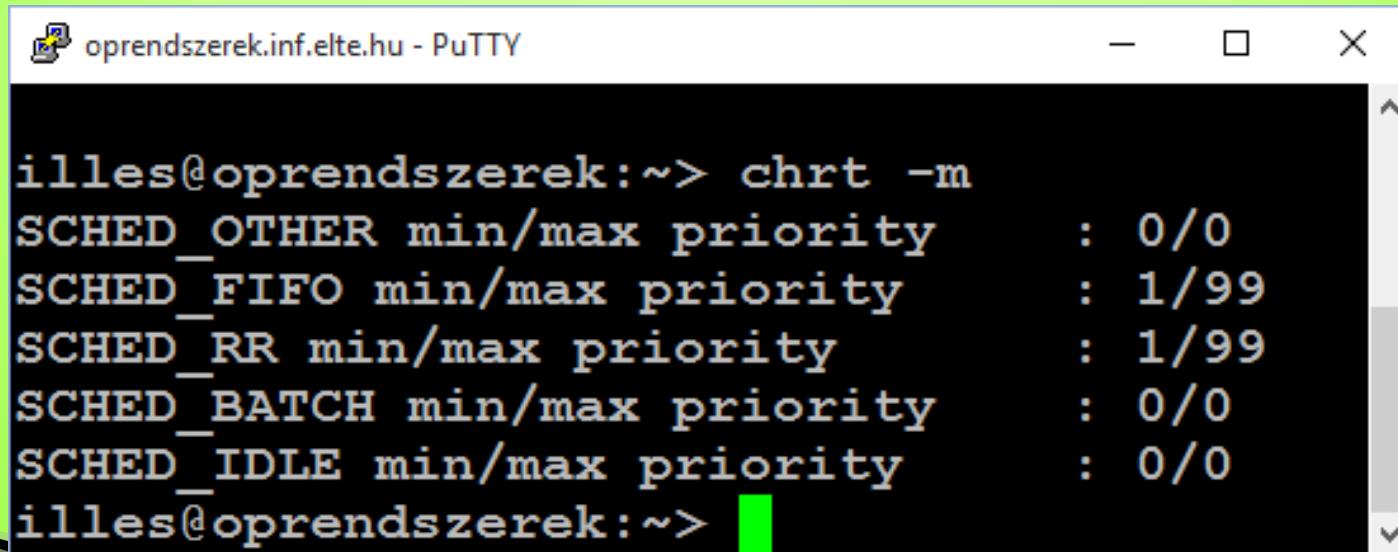
► Alaposztály: sched_class

- Ebből származik: rt_sched_class, fair_sched_class(sched_other), idle_sched_class, batch_sched_class



Ütemezési jellemzők - chrt

- ▶ Chrt -m
- ▶ SCHED_FIFO,SCHED_RR (Round Robin), klasszikus ütemezések RT folyamatokra!
- ▶ SCHED_OTHER- Alapértelmezett CFS!



A screenshot of a PuTTY terminal window titled "oprendszer.inf.elte.hu - PuTTY". The window displays the output of the command "chrt -m". The output shows the current priority settings for different scheduling policies:

```
illes@oprendszer.inf.elte.hu:~> chrt -m
SCHED_OTHER min/max priority      : 0/0
SCHED_FIFO min/max priority      : 1/99
SCHED_RR min/max priority        : 1/99
SCHED_BATCH min/max priority     : 0/0
SCHED_IDLE min/max priority      : 0/0
illes@oprendszer.inf.elte.hu:~>
```

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Miről beszéltünk korábban...

- ▶ Operációs rendszerek kialakulása
- ▶ Op. Rendszer fogalmak, struktúrák
- ▶ Fájlok, könyvtárak, fájlrendszerek
- ▶ Folyamatok
 - Folyamatok kommunikációja
 - Kritikus szekciók, szemaforok.
- ▶ Klasszikus IPC problémák
- ▶ Ütemezés

Mi következik ma...

- ▶ **Beviteli/ Kiviteli eszközök vezérlése**
 - I/O eszköztípusok
 - Eszközök elérése
 - Megszakítás
 - DMA
 - I/O portok
- ▶ **I/O programok alapelvei**
- ▶ **Erőforrás elérés problémái**
 - Holtpontok
 - Alapelvek
 - Felismerés, megelőzés, elkerülés

Input–Output eszközök

► Blokkos eszközök

- Adott méretű blokkban tároljuk az információt.
- Blokkméret 512 byte– 32768 byte között.
- Egymástól függetlenül írhatók vagy olvashatók.
- Blokkonként címezhető
- Ilyen eszköz: HDD, CD, szalagos egység, stb

► Karakteres eszközök

- Nem címezhető, csak jönnek–mennek sorban a „karakterek” (bájtok)

► Időzítő: kivétel, nem blokkos és nem karakteres

I/O eszközök modellje

- ▶ Bár minden rendszernek van kivétel (timer), az op.rendszer eszköz független szoftvermodellje erre a blokkos-karakteres modellre épül.
 - Pl. A fájlrendszer absztrakt blokkos eszközökkel foglalkozik.
 - Szalagegység is blokkos, az N. blokk olvasása parancsnál, előbb visszateker, majd előre.
 - Az eszközfüggő részt az eszközmeghajtók (device driver) jelentik. (DDK)

I/O eszközök sebessége

- ▶ Billentyűzet: 10 bájt/sec
- ▶ Egér: 100 bájt/sec
- ▶ 56k modem: 7kbájt/sec
- ▶ Szkenner: 400kbájt/sec
- ▶ 52xCD ROM: 8 MB/sec (1xCD, 150kb/sec)
- ▶ Firewire : 50 MB/sec
- ▶ USB2: 60 MB/sec
 - USB3 500 MB/sec (elméleti, 4.8GBPS)
- ▶ SATA: 200 MB/sec
- ▶ SCSI UW4: 320 MB/sec
- ▶ PCI sín: 528 MB/sec

Eszközvezérlők

- ▶ Az I/O eszközt a számítógéphez (rendszer busz-hoz) kapcsoló elem az eszközvezérlő, vagy adapter.
 - Video vezérlő
 - Soros-párhuzamos vezérlő
 - USB vezérlő
 - HDD vezérlő
 - IDE, SATA, SCSI
- ▶ Nagy gépek esetén az I/O speciális I/O gépekkel van megvalósítva.
- ▶ CPU-eszközvezérlő kommunikációja
 - Memórialeképezésű I/O, Megszakítás, DMA

I/O kapu,memórialeképezésű I/O

- ▶ A CPU külvilág felé kétféle adatcserét ismer
 - I/O kapuk írása olvasása
 - IN regiszter, port ; A port (8 vagy 16 bites szám) ; adatának regiszterbe olvasása
 - OUT port, regiszter ; A regiszter kiírása a portra.
 - Az I/O eszköz regiszterei, adatterülete a memória egy részén helyezkedik el.
- ▶ Létezhet olyan környezet, ahol
 - Csak I/O kapukat használnak. (IBM 360)
 - Csak memóriában vannak az I/O kapuk (PDP-11)
 - Memórialeképezésű I/O
 - Vegyes (Intel x86, Pentium)

Megszakítások I.

- ▶ Interrupt – fogalma az I/O eszközökhöz kötődik!
 - Ha egy I/O eszköz „adatközlésre kész”, ezt egy megszakításkéréssel jelzi!
- ▶ Szoftveres vs. Hardveres megszakítás
 - Szoftveres: gépi kódú utasítás (int x) végzi, multitask esetén „trap-nak” (csapda) hívják!
- ▶ Megszakításnak száma van!
 - Megszakítás vektor (valós mód, 0 címtől)
 - Megszakítás kiszolgáló rutin.
 - INTR, NMI

Megszakítások kezelése

- ▶ Általában az eszközöknek van állapotbitjük, jelezve, hogy az adat készen van.
 - Ezt lehet figyelni, nem az igazi.
 - Tevékeny várakozás ez is, nem hatékony, ritkán használt.
- ▶ Megszakítás kezelés folyamata (IRQ)
 - A HW eszköz jelzi a megszakítás igényt(INTR)
 - CPU egy következő utasítás végrehajtás előtt, a tevékenységét megszakítja! (Precíz, imprecíz)
 - A kért sorszámu kiszolgáló végrehajtása.
 - A kívánt adat beolvasása, a szorosan hozzátartozó tevékenység elvégzése.
 - Visszatérés a megszakítás előtti állapothoz.

Megszakítások prioritása

- ▶ INTR – maszkolható, NMI nem maszkolható
 - INTR megszakítás prioritások
 - Megszakítás közben érkező hasonló vagy alacsonyabb prioritású kérés várakozik!
 - NMI – csak egy kiszolgálás, legnagyobb prioritás
- ▶ PC világban 15 különböző interrupt
 - 2x8 csatornás vezérlő
 - Kézi adapter IRQ állítás korábban, jumper-rel, sw-el
 - BIOS automatikus megszakítás hozzárendelés (Plug&Play)

Közvetlen memória elérés (DMA)

- ▶ Direct Memory Access
 - Tartalmaz: Memória cím regisztert, átvitel irány jelzésre, mennyiségre, vezérlésre regisztert
 - Ezeket szabályos in, out portokon lehet elérni.
- ▶ Működés jellemző lépései:
 1. CPU beállítja a DMA vezérlőt. (Regisztereket.)
 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 szoftver célok I.

► Eszközfüggetlenség

- Ugyanaz a kód, parancs legyen képes pl. fájlt olvasni HDD-ről, CD-ről stb.
- Egységes névhasználat, a név mint paraméter jelenti (eltakarja) a valódi eszközt.
- Logikai csatolás (mount)
 - Unix: Floppy csatolva a /home/fdd könyvtárhoz
 - Windows: Floppy csatolva az a: névhez

► Hibakezelés

- Hardver közeli szinten kell(ene) kezelni
- Az esetek többségében itt elvégezhető, magasabb szintre csak „fatális” hiba esetén kerüljön a kezelés.

I/O szoftver célok II.

- ▶ Szinkron (blokkolásos), aszinkron (megszakításos) átviteli mód támogatása.
 - Szinkron módban könnyebb írni a felhasználói programokat.
 - Kell támogatnia az op. Rendszernek az aszinkron mód használatot is.
- ▶ Pufferezés
 - mindenki használja, billentyű puffer stb.
- ▶ Eszközök megosztott, vagy egyedi használat
 - Lemezt egyszerre több folyamat is tudja használni.
 - Egyedi (monopol) használatú pl. a szalagegység, CD író.
- ▶ DDK– Device Driver Kit

I/O szoftverrendszer felépítése

- ▶ Réteges szerkezet
 - Tipikusan 4 rétegbe van szervezve.
- ▶ Hardver eszköz
 1. Megszakítás kezelő réteg
 - Legalsó kernel szinten kezelt.
 - Szemafor blokkolással védve a kritikus (egész?) rész.
 2. Eszközmeghajtó programok
 3. Eszköz független operációs rendszer program
 4. Felhasználói I/O eszközt használó program

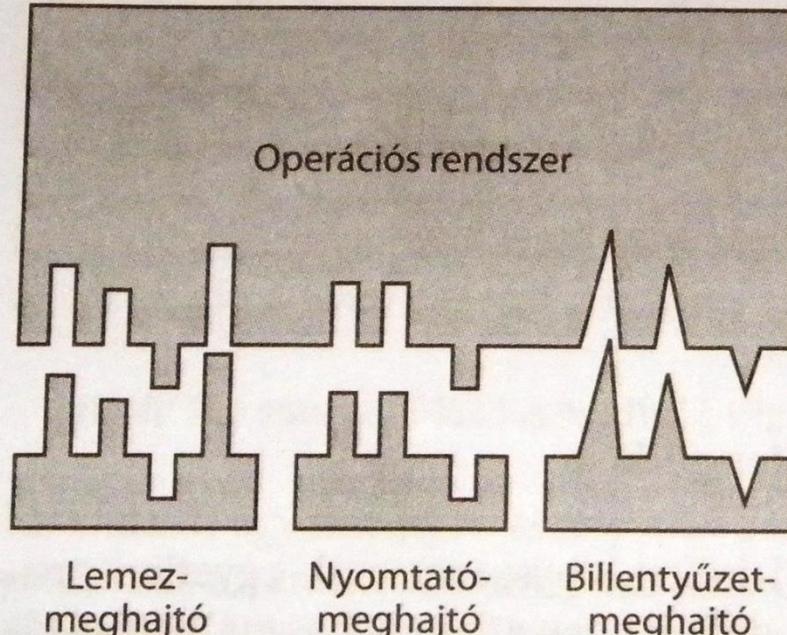
Eszközmeghajtó programok

- ▶ Eszköz specifikus kód – eszközmeghajtó program (driver)
- ▶ Ez pontosan ismeri az eszköz jellemzőit
 - Egérmegható megmondja az elmozdulást, melyik gombot nyomták meg.
 - Lemezmeghajtó ismeri a fejek mozgatását, beállítását adott sávhoz, szektorhoz.
 - Stb.
- ▶ Feladata a felette lévő szintről érkező absztrakt kérések kiszolgálása
 - Egyszerűbb eszközök, egy időben egy kérés
 - Intelligensebbek, kérések sorát fogadhatja (scsi)
- ▶ Kezeli az eszközt I/O portokon, megszakítás kezelésén keresztül.
- ▶ Blokkos – karakteres eszközök

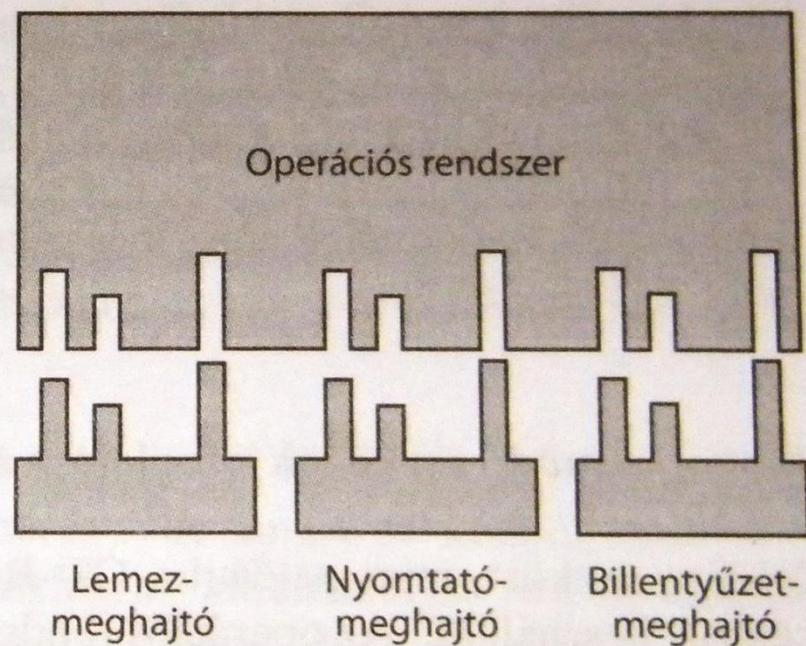
Eszköz független I/O program feladata I.

- ▶ Egységes, szabványos kapcsolódási felület biztosítása.
 - Azonos elnevezések, hívási konvenciók.
 - I/O eszközök szimbolikus neveinek valós meghajtóhoz kapcsolása.
 - Főeszköz szám : a meghajtóra utaló azonosító
 - Mellékeszköz szám: + paraméter, pl. írás–olvasás jelzésre
 - I/O eszköz védelme
 - Fájlrendszerszerű jogosultságok alkalmazása.

Egységes felület illusztráció.



(a)



(b)

3.7. ábra. (a) Szabványos meghajtó interfész nélkül. (b) Szabványos meghajtó interfésszel

Eszköz független I/O program feladatai II.

- ▶ Pufferezés, hasonlóan az adapterekhez, általános elv, a felhasználó független az eszköz független I/O programtól.
(Gyorsabban ír a pufferbe mint az „eszközbe”.)
- ▶ Hibakezelés
- ▶ Monopol módú eszközök lefoglalása, elengedése
- ▶ Eszköz független blokkméret kialakítása.
 - Lemezek logikai blokkmérete

Felhasználói I/O programok

- ▶ Könyvtári I/O eljárások 2 kategóriája
 - Továbbítja a paramétereket a rendszerhívás számára
 - `N=write(fd, buffer, db);`
 - Tényleges feladatot (`is`) végeznek, majd rendszerhívás
 - `printf(“%d almafa”, n);`
- ▶ Háttértárolás (spooling)
 - Monopol eszközök kezelési módja (nyomtató)
 - Speciális folyamatok kezelik a háttértár, spooling könyvtárakat. Démonok.

Monopol módú erőforrások használata

- ▶ Láttuk korábban, ezek az I/O eszközök egyik fontos csoportja.
- ▶ De ilyen a rendszer belső táblázat (pl folyamat tábla) kezelése is.
- ▶ Egy időben csak egy folyamat használhatja.
 - Pl: „Párhuzamosan” 2 fájlt nyomtatni nem az igazi.
- ▶ Nem csak monopol I/O eszköznél fordulhat elő versenyhelyzet
 - Egyszerű memória rekesznél is, de jellemzően a monopol I/O eszközökhöz kötődik.
- ▶ Tipikus helyzet: két folyamat ugyanarra vár.
 - Két udvarias ember a lift előtt...lift elmegy ők maradnak...

Holtpont (deadlock)

- ▶ Két vagy több folyamat egy erőforrás megszerzése során olyan helyzetbe kerül, hogy egymást blokkolják a további végrehajtásban.
 - Pontos definíció: Folyamatokból álló halmaz holtpontban van, ha minden folyamat olyan eseményre vár, amit csak a halmaz egy másik folyamata okozhat.
- ▶ Nem csak az I/O eszközökhöz kötődik
 - Párhuzamos rendszerek
 - Adatbázisok
 - Stb.

Holtpont feltételek

- ▶ Coffman E.G., szerint 4 feltétel szükséges a kialakuláshoz
 1. Kölcsönös kizárási 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.

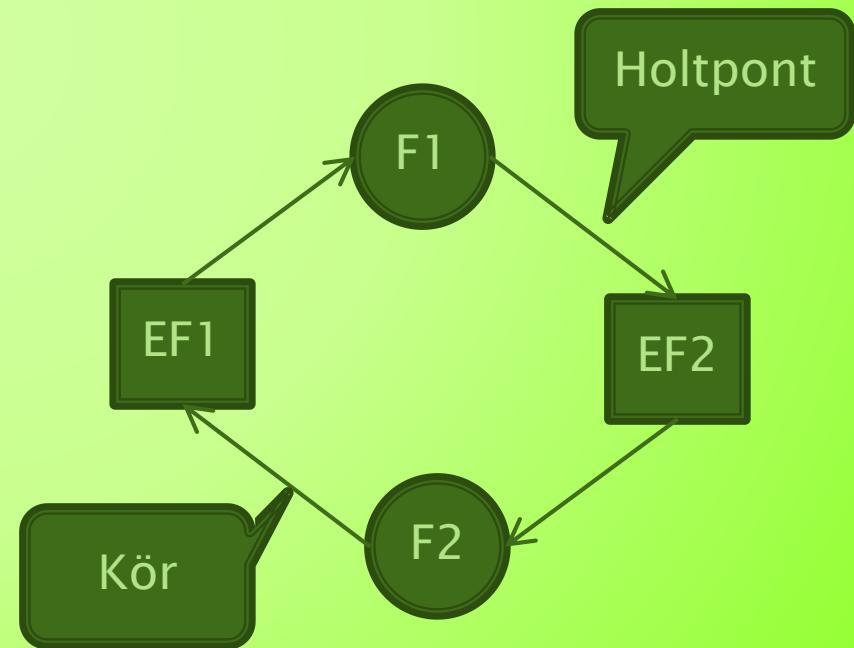
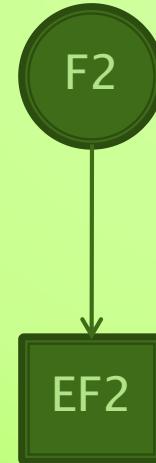
Holtpont gráfmodellje

- ▶ Holtpont feltételek modellezése irányított gráfokkal (Holt, 1972)
 - Folyamat– kör
 - Erőforrás– négyzet
 - Ha az erőforrások, folyamatok irányított gráfjában kört találunk, ez holtpontot jelent.

Erőforrás birtoklás



Erőforrás kérés



Holtpont kialakulása, példa

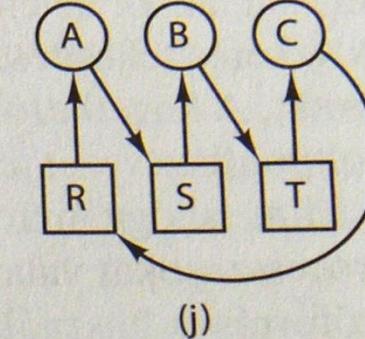
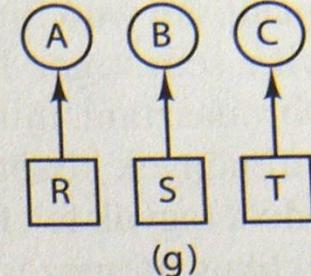
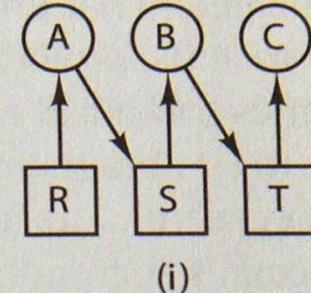
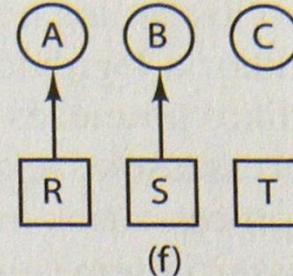
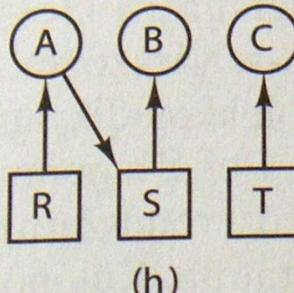
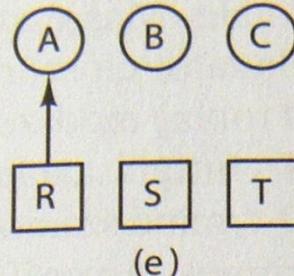
A
R kérése
S kérése
R elengedése
S elengedése
(a)

B
S kérése
T kérése
S elengedése
T elengedése
(b)

C
T kérése
R kérése
T elengedése
R elengedése
(c)

1. A kéri R-t
 2. B kéri S-t
 3. C kéri T-t
 4. A kéri S-t
 5. B kéri T-t
 6. C kéri R-t
- holtpont

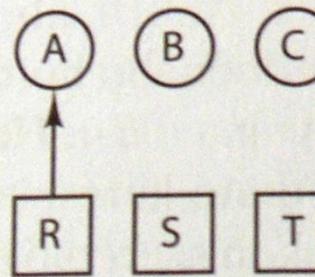
(d)



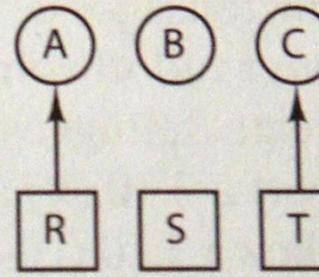
Holtpont elkerülése, példa

1. A kéri R-t
 2. C kéri T-t
 3. A kéri S-t
 4. C kéri R-t
 5. A elengedi R-t
 6. A elengedi S-t
- nincs holtpont

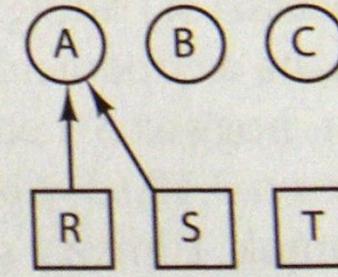
(k)



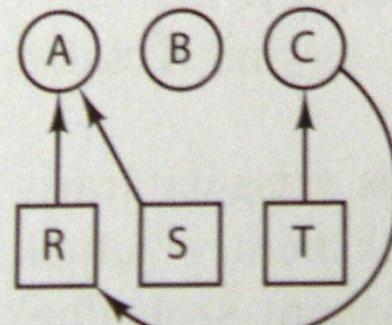
(l)



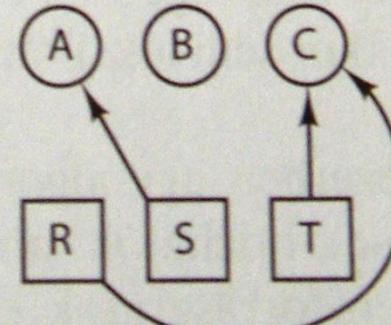
(m)



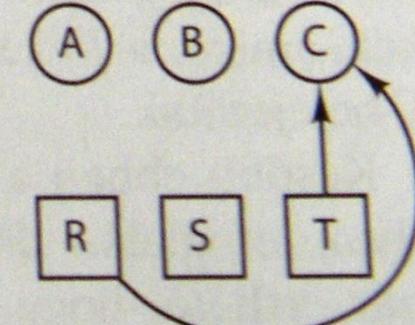
(n)



(o)



(p)



(q)

Holtpont stratégiák

1. A probléma figyelmen kívül hagyása.
 - Nem törődünk vele, nagy valószínűsséggel Ő sem talál meg bennünket, ha mégis ...
2. Felismerés és helyreállítás.
 - Engedjük a holtpontot megjelenni (kör), ezt észrevesszük és cselekszünk.
3. Megelőzés. A 4 szükséges feltétel egyikének meghiusítása.
4. Dinamikus elkerülés. Erőforrások foglalása csak „óvatosan”.

1. 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, op.rendszer, hw, sw hiba) összeomlások aránya 1:250.
- ▶ A Unix, Windows világ is ezt a „módszert” használja.
 - Túl nagy az ár a várható haszonért cserébe.

2. Felismerés, helyreállítás

- ▶ 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 x (fél óra?) ideje blokkolt egy folyamat, egyszerűen megszüntetjük.
 - Nagygépes rendszereknél ismert módszer.

3. Megelőzés

- ▶ A Coffman féle 4 feltétel valamelyikére mindenél egy megszorítás.
 - Kölcsönös kizárási feltétel. Ha egyetlen erőforrás soha nincs kizárolag 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.

3. Megelőzés (folyt.)

- ▶ A Coffman féle harmadik feltétel a megszakíthatatlanság. Ennek elkerülése eléggé nehéz.
 - Nyomtatás közben nem szerencsés a nyomtatót másnak adni.
- ▶ Negyedik feltétel a ciklikus várakozás már könnyebben megszüntethető.
 - Egyszerű mód: minden folyamat egyszerre csak 1 erőforrást birtokolhat.
 - 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

- ▶ 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!
- ▶ A bankár algoritmus minden kérés megjelenésekor azt nézi, hogy a kérés teljesítése biztonságos állapothoz vezet-e?
 - Ha igen jóváhagyja, ha nem a kérést elhalasztja.
 - Eredetileg 1 erőforrásra tervezett.

Bankár algoritmus mintállapotok (Egy erőforrásra)

	Birtokol	Maximum
A	0	6
B	0	5
C	0	4
D	0	7

Szabad: 10

(a)

	Birtokol	Maximum
A	1	6
B	1	5
C	2	4
D	4	7

Szabad: 2

(b)

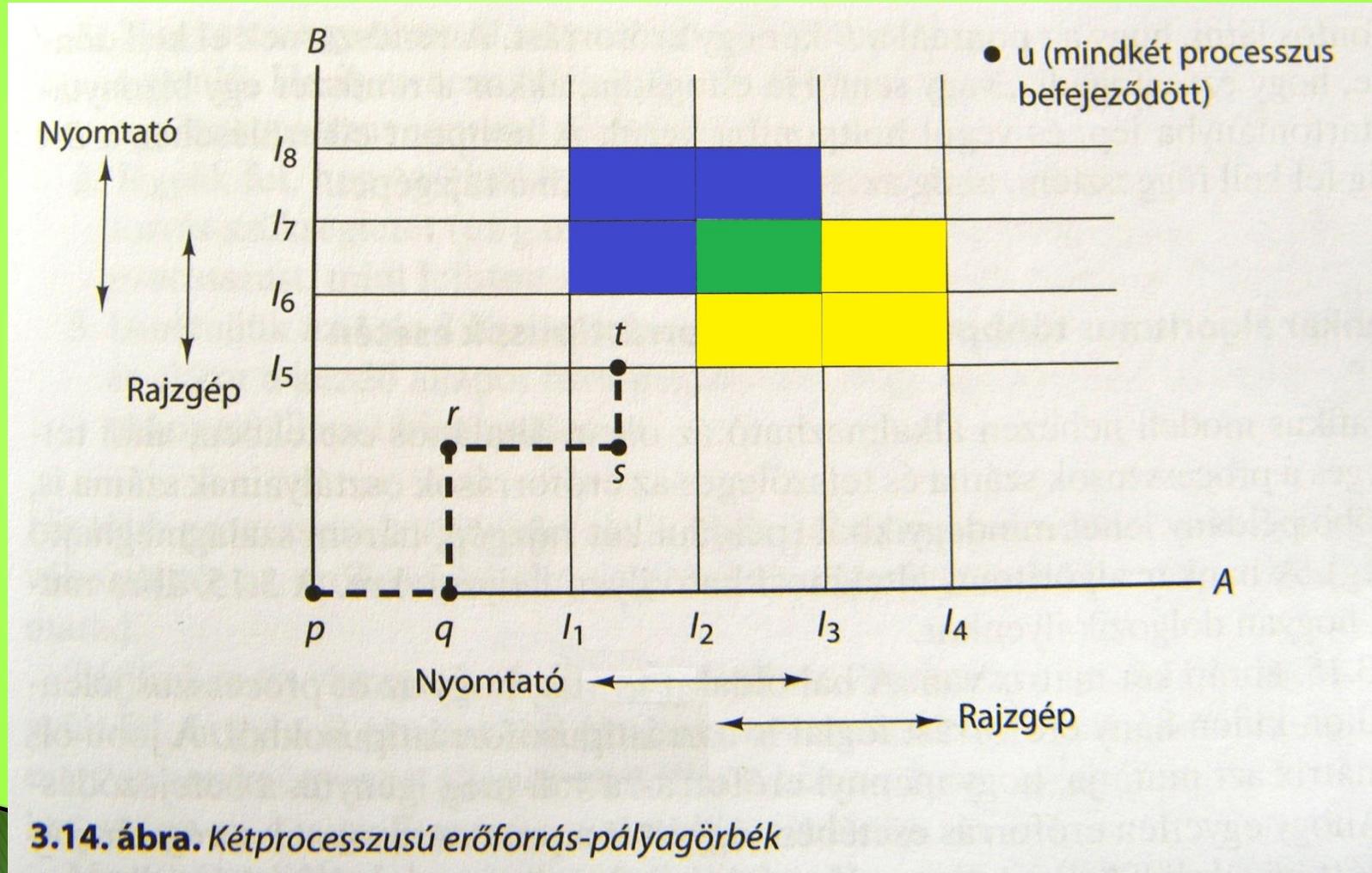
	Birtokol	Maximum
A	1	6
B	2	5
C	2	4
D	4	7

Szabad: 1

(c)

3.13. ábra. Három erőforrás-lefoglalási állapot. (a) Biztonságos. (b) Biztonságos. (c) Bizonytalan

Két folyamat- erőforrás pályagörbe



Bankár algoritmus több erőforrás típus esetén

- ▶ Az 1 erőforrás elvet alkalmazzuk:
 - Jelölés: $F(i,j)$ 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.
- 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 minden, lefut, majd az erőforrás foglalásait adjuk $S(j)$ -hez
- 3. Ismételjük 1,2 pontokat míg vagy befejeződnek, vagy holpontra jutnak.

Több erőforrásos bankár példa

	Processzus	Szalagmeghajtó egységek	Rajzgépek	Nyomtatók	CD-ROM-ok
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

	Processzus	Szalagmeghajtó egységek	Rajzgépek	Nyomtatók	CD-ROM-ok
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

$F(i,j)$: Lefoglalt erőforrások

$E(j) : (6342)$ összes erőforrás

$M(i,j)$: További erőforrásigények

$S(j) : (1020)$ a még szabad erőforrások

Kaphat B 1 nyomtatót, az még biztonságos állapot lesz. (D be tud fejeződni, utána A,E,C,B.)

Bankár algoritmus összegzés

- ▶ A korábbi megelőzés is, meg ez az elkerülés is olyan információt kér (az erőforrás pontos igényeket, a folyamatok számát előre), ami nehezen megadható.
 - Folyamatok dinamikusan jönnek létre, erőforrások dinamikusan módosulnak.
- ▶ Ezért a gyakorlatban kevesen alkalmazzák.
- ▶ ...

Köszönöm a figyelmet!

zoltan.illes@elte.hu

Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Miről beszéltünk korábban...

- ▶ Operációs rendszerek kialakulása
- ▶ Op. Rendszer fogalmak, struktúrák
- ▶ Fájlok, könyvtárak, fájlrendszerek
- ▶ Folyamatok
 - Folyamatok kommunikációja
 - Kritikus szekciók, szemaforok.
- ▶ Klasszikus IPC problémák
- ▶ Ütemezés
- ▶ I/O, holt pont probléma

Mi következik ma...

- ▶ **Memória gazdálkodás**
 - Alapvető memória kezelés
 - Csere
 - Virtuális memória
 - Lapcserélési algoritmusok
 - Lapozásos rendszerek tervezése
 - Szegmentálás

Memória kezelő

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

Alapvető memória kezelés

- ▶ Kétféle algoritmus csoport:
 - Szükséges a folyamatok mozgatása, cseréje a memória és a lemez között. (swap)
 - Nincs erre szükség
- ▶ Ha elegendő memória van, akkor nem kell csere (swap)!
- ▶ Van elegendő memória?
 - HT1080Z – 16 KB, C64 – 64 KB, IBM PC-640 KB
 - Ma: PC 2-4-8 GB, kis szerver 4-16 GB

Monoprogramozás

- ▶ Egyszerre egy program fut.
 - Nincs szükség „algoritmusra”. Parancs begépelése, annak végrehajtása, majd várjuk a következőt.
 - Tipikus helyzetek
 - Op.rendszer az alsó címeken, program felette (ma ritkán használt, régen nagygépes, minibeszemben használták).
 - Alsó címeken a program, a felső memória területén a ROM-ban az operációs rendszer (kézi számítógép, beágyazott rendszer)
 - Op.rendszer az alsó címeken, majd felhasználói program, felette a ROM-ban eszközmeghajtók. (MS-DOS, ROM-BIOS)

Multiprogramozás(Multitask)

- ▶ „Párhuzamosan” több program fut. A memóriát valahogy meg kell osztani a folyamatok között.
 1. Multiprogramozás megvalósítása rögzített memória szeletekkel.
 2. Multiprogramozás megvalósítása memória csere használattal.
 3. Multiprogramozás megvalósítása virtuális memória használatával.
 4. Multiprogramozás szegmentálással.

1. Multiprogramozás rögzített memória szeletekkel

- ▶ Monoprogramozás ma jellemzően beágyazott rendszerekben van jelen (PIC)
- ▶ Ma tipikusan preemptív időosztásos rendszereken több folyamat van a memóriába „végrehajtás alatt”.
- ▶ Osszuk fel a memóriát n (nem egyenlő) szeletre. (Fix szeletek)
 - Pl. rendszerindításnál ez megtehető
 - Egy közös várakozási sor
 - minden szeletre külön-külön várakozási sor.
 - Kötegelt rendszerek tipikus megoldása.

Memória felosztása

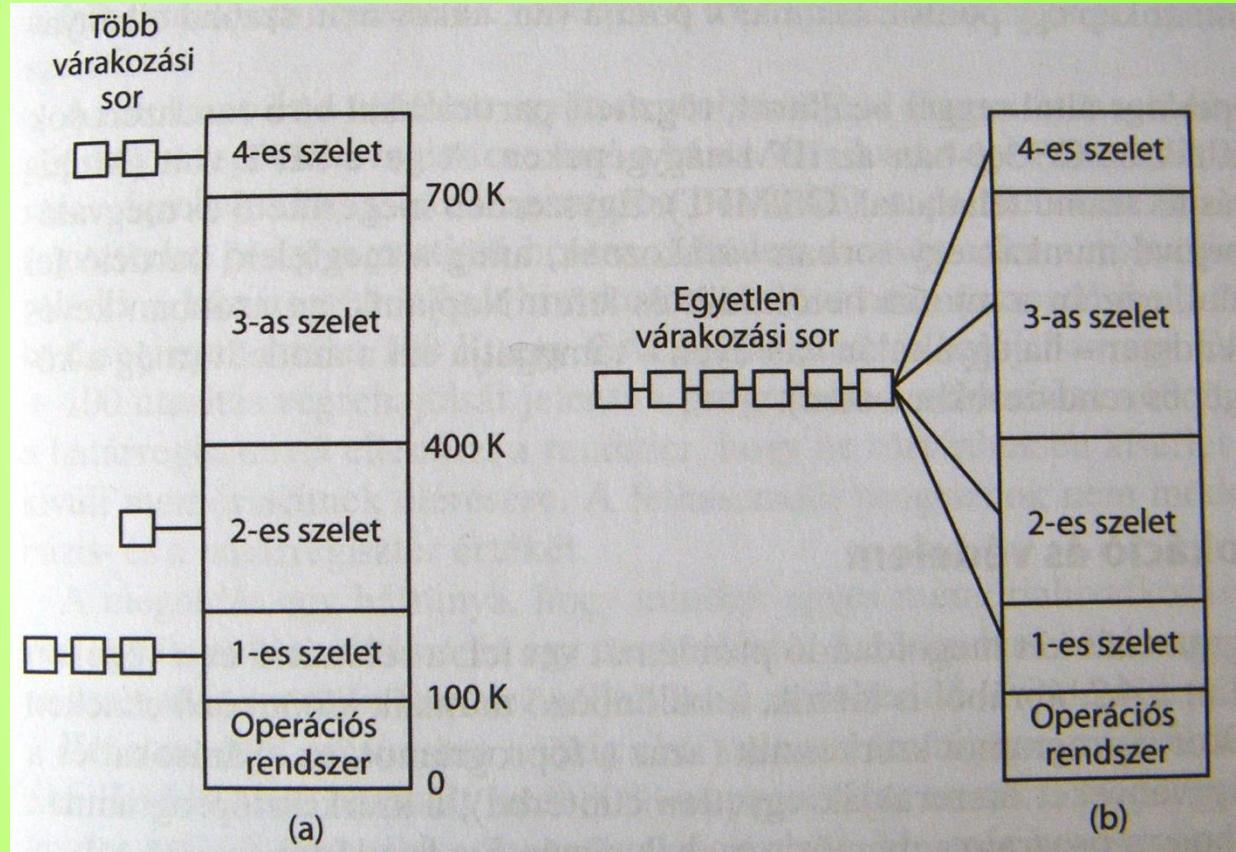
Több várakozási sor:

- Kihasználatlan partíciók

Egy sor:

- Ha egy partíció kiürül, a sorban első beleférő bekerül.

IBM használta az OS/360 rendszeren:
OS/MFT (Multiprogram Fix Task)



Relokáció – Védelem

- ▶ Nem tudjuk hova kerül egy folyamat, így a memória hivatkozások nem fordíthatók fix értékekre! (Relokáció)
 - OS/MFT program betöltéskor frissítette a „relokálandó” címeket.
- ▶ Nem kívánatos, ha egy program a másik memóriáját „éri el”! (Védelem)
 - OS/MFT PSW (program állapotszó, 4 bites védelmi kulcs)
- ▶ Másik megoldás: Bázis+határregiszter használata
 - Ezeket a programok nem módosíthatják.
 - minden címhivatkozásnál ellenőrzés: lassú.

2. Multiprogramozás memória csere használattal

- ▶ A korábbi kötegelt rendszerek tipikus megoldása a rögzített memória szeletek használata (IBM OS/MFT)
- ▶ Időosztásos, grafikus felületek esetén ez nem az igazi.
- ▶ Teljes folyamat mozgatása memória–lemez között.
- ▶ Nincs rögzített memória partíció, minden egyik 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 (foci nézés) ez az időveszteség nem megengedhető!)

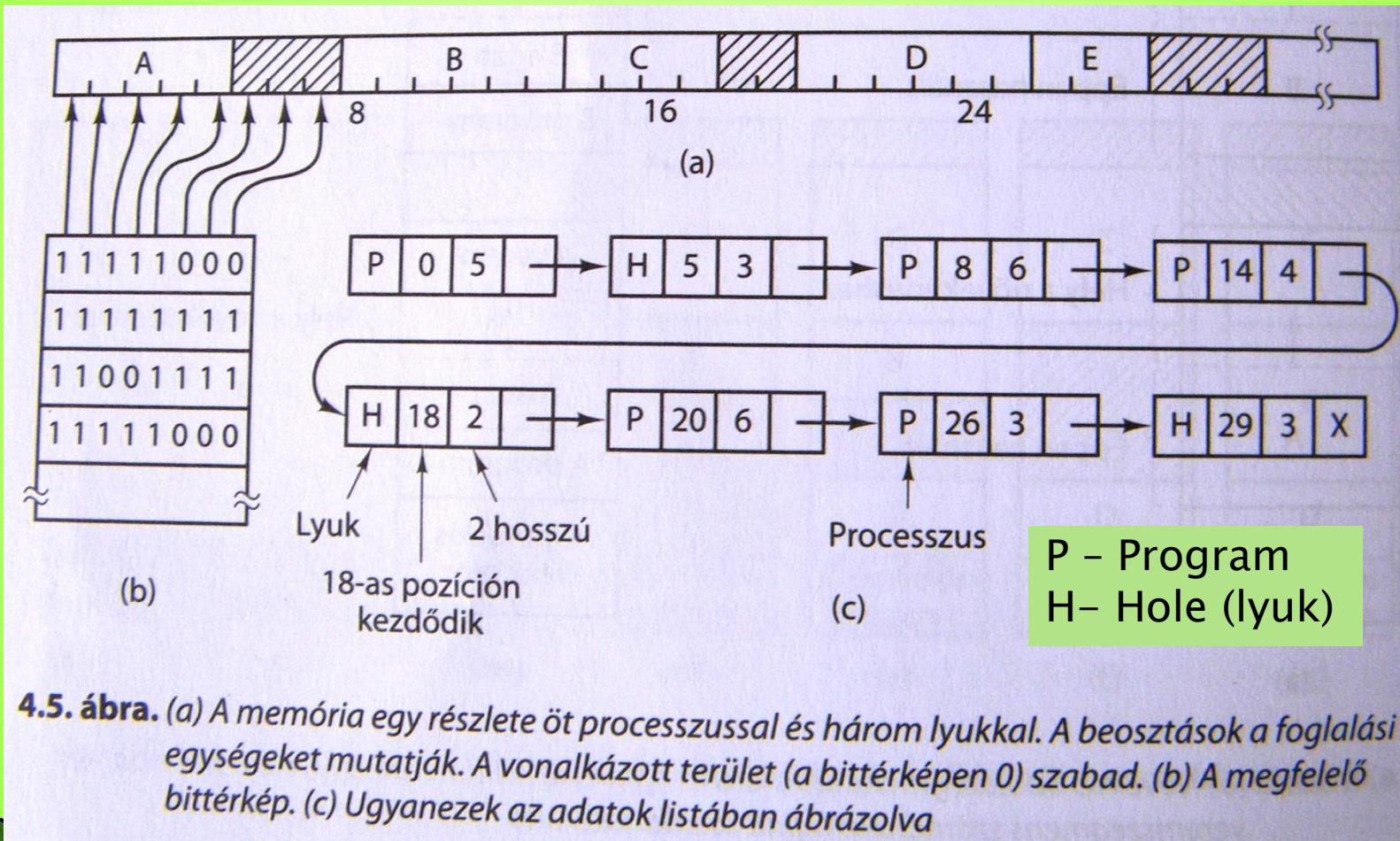
Dinamikus memória foglalás

- ▶ Általában nem ismert, hogy egy programnak mennyi dinamikus adatra, veremterületre van szüksége.
- ▶ A program „kód” része fix szeletet kap, míg az adat és verem 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.
- ▶ Hogy tudjuk nyilvántartani a „dinamikus” memóriát?

Dinamikus memória nyilvántartása

- ▶ Allokációs egység definiálása.
 - Ennek mérete kérdés. Ha kicsi akkor kevésbé lyukasodik a memória, viszont nagy a nyilvántartási „erőforrás (memória) igény”.
 - Ha nagy az egység túl sok lesz az egységen belüli maradékokból adódó memória veszteség.
- ▶ A nyilvántartás megvalósítása:
 - Bittérkép használattal
 - Láncolt lista használattal
 - Ha egy folyamat befejeződik, akkor szükség lehet az egymás melletti memória lyukak egyesítésére.
 - Külön lista a lyukak és folyamatok listája.

Bittérkép, Láncolt lista megvalósítása



Memória foglalási startégiák

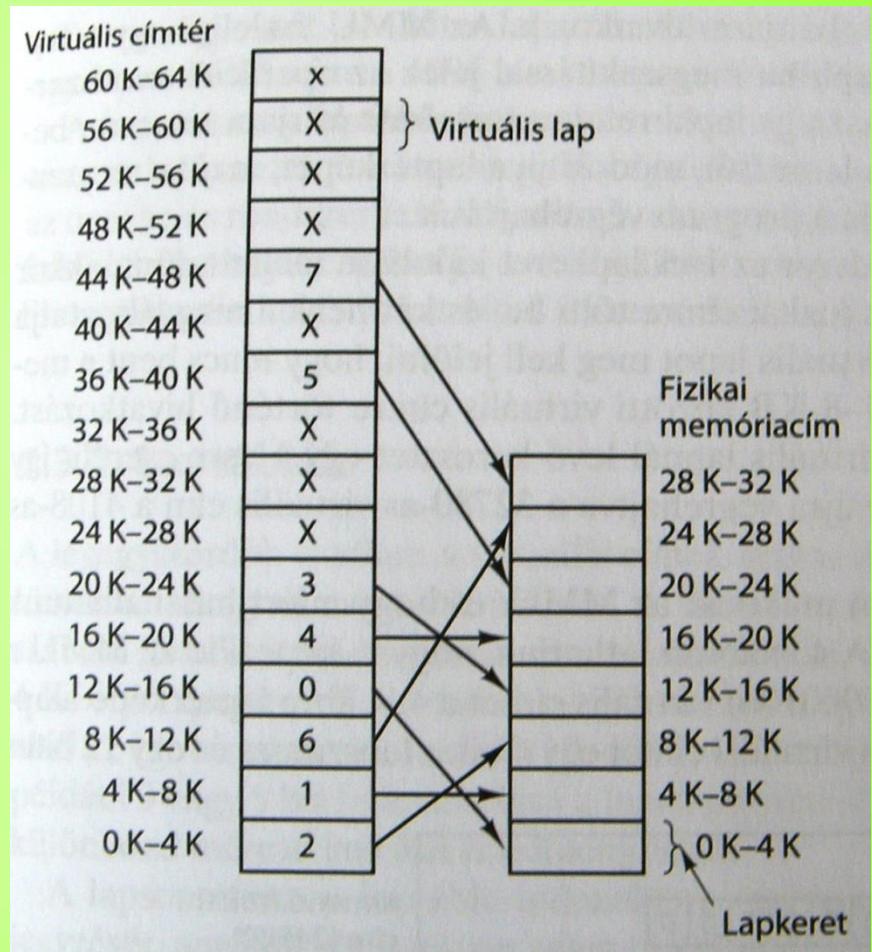
- ▶ Új vagy swap partícióról behozott folyamat számára, több memória elhelyezési algoritmus ismert (hasonlóak a lemezhez) :
 - First Fit (első helyre, ahova befér, leggyorsabb, legegyszerűbb)
 - Next Fit (nem az elejéről, hanem az előző befejezési pontjából indul a keresés, kevésbé hatékony mint a first 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)

3. Multiprogramozás virtuális memória használattal

- ▶ Egy program használhat több memóriát mint a rendelkezésre álló fizikai méret.
 - Az operációs rendszer csak a „szükséges részt” tartja a fizikai memoriában.
 - Egy program a „virtuális memória térben” tartózkodik.
 - John Fotheringham (1961, ACM)
 - Az elv akár a monoprogramozás környezetben is használható.
 - 1961–ben és utána a „single task” rendszerű (kis)gépek léteztek.

Memória Menedzsment Unit (MMU)

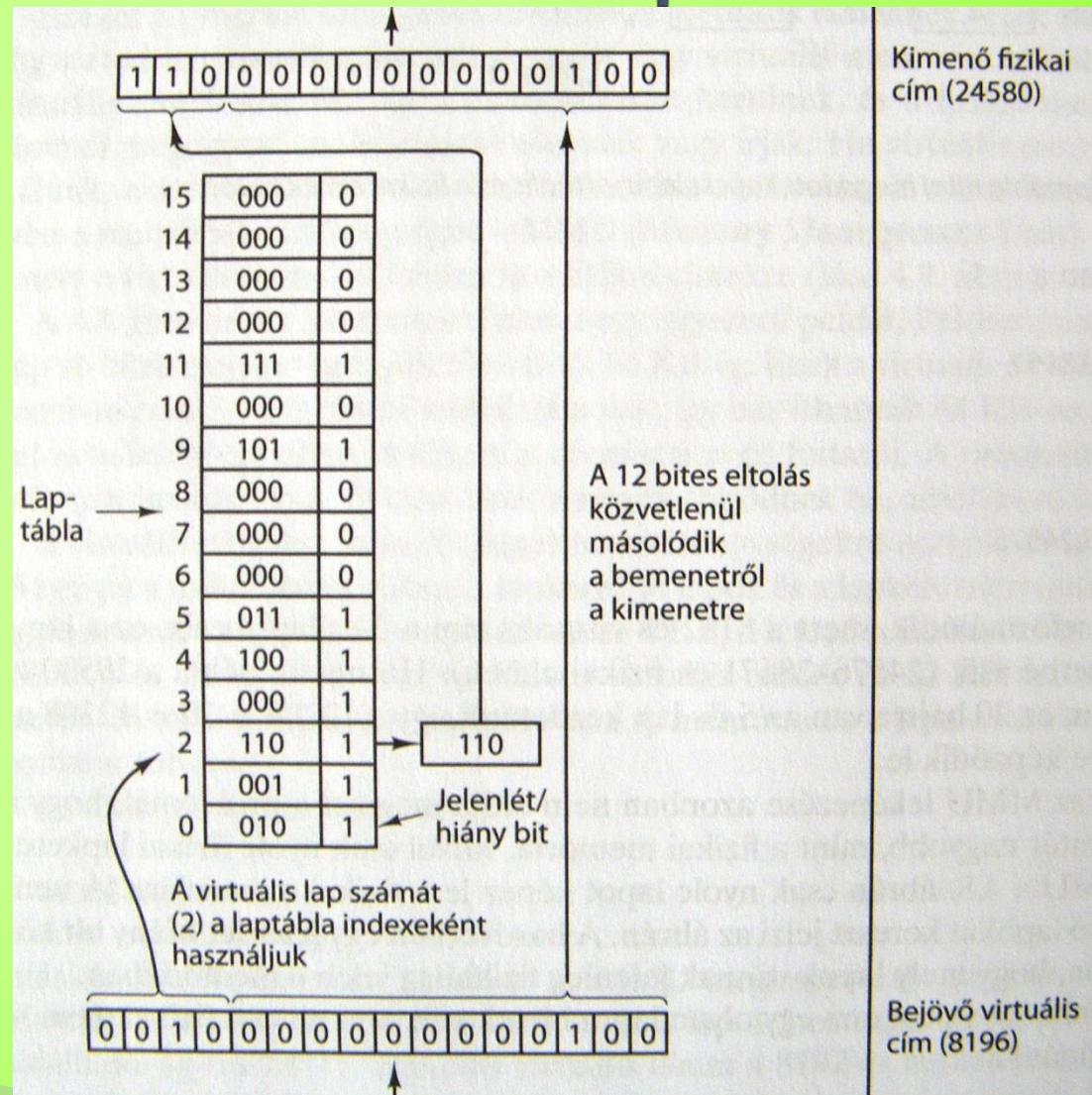
- ▶ A virtuális címtér „lapokra” van osztva. (Ezt laptáblának nevezük)
- ▶ Jelenlét/hiány bit
- ▶ Virtuális–fizikai lapok összerendezése
- ▶ Ha az MMU látja, hogy egy lap nincs a memóriában, laphibát okoz, op.rendszer kitesz egy lapkeretet, majd behozza a szükséges lapot.



MMU működés

Példa: 16 darab- 4kb lap esetén

- ▶ Kettő hatvány méretű lapok
- ▶ A 16 bites virtuális címből az első 4 bit a lapszám, a többi az offset.
- ▶ 1 bit a jelenlét/hiány jelzésére.
- ▶ Kimenő 15 bit kerül a fizikai címsín-re.



Laptábla problémák

- ▶ A korábbi példa (16 bit virtuális címtér, 4 bit laptábla, 12 bit, 4 KB, offset, azaz lapméret) egyszerű, gyors.
- ▶ Egy mai processzor vagy 32 vagy 64 bites.
 - 32 bit virtuális címtérből 12 bit (4kb) lapméretnél 20 bit a laptábla mérete. (1MB, kb. 1 millió elem)
 - minden folyamathoz saját virtuális címtér, saját laptábla tartozik! Ilyen méretű laptábla még elképzelhető!
 - 64 bites virtuális címtér esetén ilyen méretű laptábla megvalósíthatatlan!

Többszintű laptáblák

- ▶ Már 32 bites virtuális címzésnél is gond az egyszerű laptábla használat.
- ▶ Használunk kétszintű:
 - Lap Tábla1 = 10 bit (1024 elem) – felső szintű laptábla
 - Lap Tábla2 = 10 bit – Második szintű laptábla
 - Lapon belüli Offset = 12 bit
 - Előny: mivel egy folyamathoz szükséges a laptábla memóriában tartása is, itt csak 4 db 1024 elemű táblára van (LT1-re, meg a program, adat és verem LT2 táblájára) szükség (nem egymillióra)
- ▶ Tábla és offset bitszámokra klasszikus értékeket választottunk, ezeken módosíthatunk, de lényegi változás nem lesz.
- ▶ A két szintet többszintűre is cserélhetjük!
(Bonyolultabb)

Egy táblabejegyzés szerkezete

- ▶ Tartalmazza a lapkeret számát (fizikai memória és az offset mérete mondja meg, hogy hány bit)
- ▶ Jelenlét/hiány bit
- ▶ Védelmi bit, ha =0 írható, olvasható, ha =1 csak olvasható.
 - 3 védelmi bit: írás, olvasás, végrehajtás engedélyezése minden lapra.
- ▶ Dirty bit (módosítás). Ha 1 akkor módosult a lapkeret memória, azaz lemezre íráskor tényleg ki kell írni!
- ▶ Hivatkozás bit, értéke 1 ha hivatkoznak a lapra (használják). Ha használnak egy lapot, nem lehet lemezre tenni!
- ▶ Gyorsító tár letiltás bit. Ott fontos, ahol a fizikai memória(bizonyos területe) egyben I/O eszköz adatterület is.

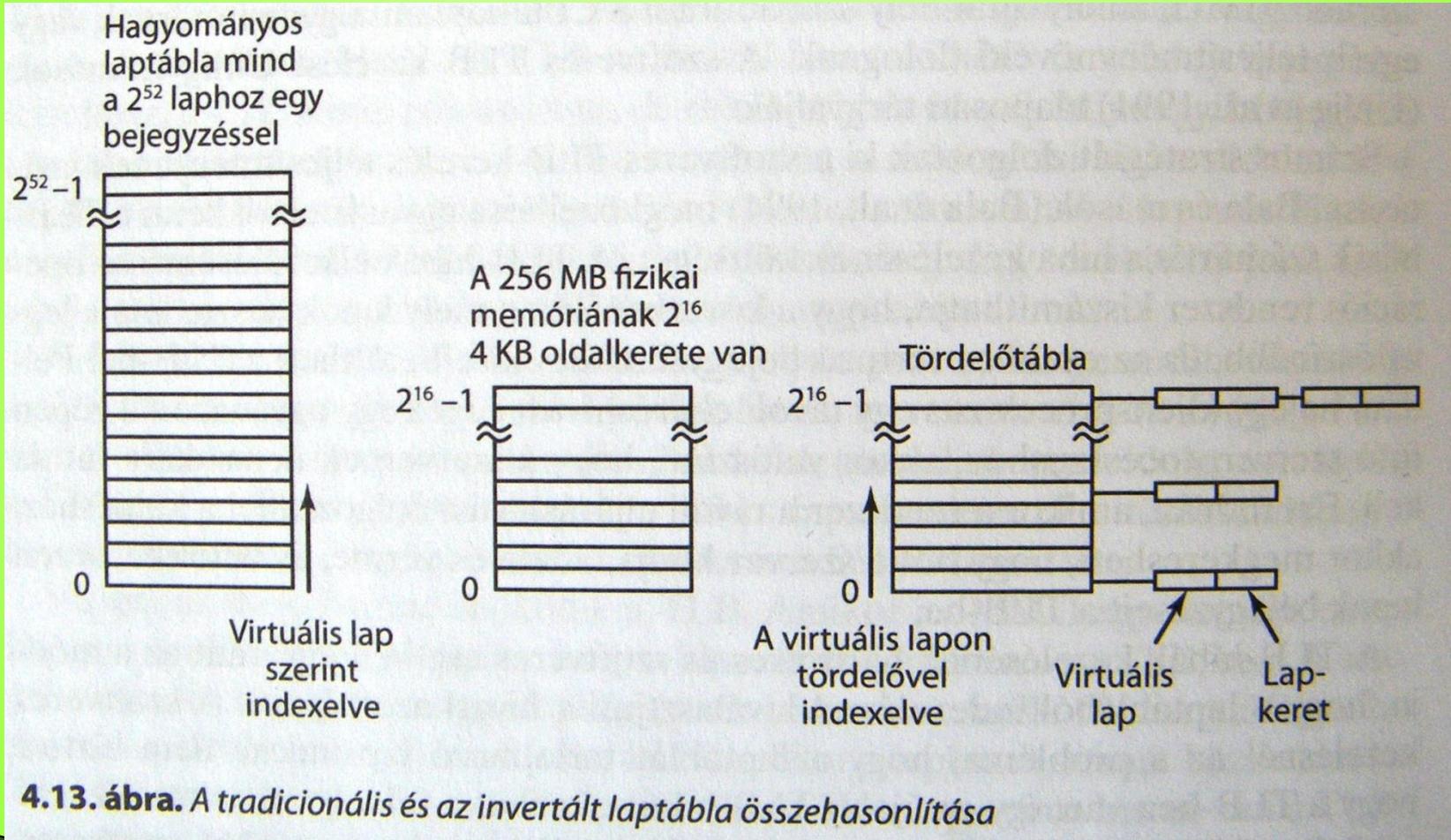
Lapozás helyett(mellett) TLB (Translation Lookaside Buffer)

- ▶ A klasszikus MMU eszközünk lassú, egy memória hivatkozáshoz legalább 1 táblahivatkozás kell, így minimum felére csökken a memórai műveleti idő.
- ▶ Tegyünk az MMU–ba egy kis HW egységet (TLB– asszociatív memória), kevés bejegyzéssel (eleinte 64–nél nem többet)
 - Ma a Nehalem esetében kétszintű TLB van, a TLB2 512 elemű.
- ▶ Szoftveres TLB kezelés
 - 64 elem elég nagy ahhoz, hogy kevés TLB hiba legyen, így a hardveres megoldás kispórolható.
 - Ilyen pl. a Sparc Mips, Alpha, HP PA, PowerPC

Invertált laptáblák

- ▶ Induljunk ki a valós memória méretből.
- ▶ A laptábla annyi elemet tartalmaz, amennyi a fizikai memóriából következik.
 - Sok helyet takarít meg, de nehezebb a virtuális címből a fizikai megadása. (Nem lehet automatikus index használatot végezni!)
 - PL: 4 kb lapméret, 1GB RAM, 2^{18} darab bejegyzés.
- ▶ Kiút: Használjunk TLB-t, ha hibát ad, akkor az invertált laptáblában keresünk, és az eredményt a TLB-be rakjuk.

Invertált tábla példa



Lapcserélési algoritmusok

- ▶ Ha nincs egy virtuális című lap a memóriában, akkor egy lapot ki kell dobni, berakni ezt az új lapot.
 - Kérdés hogyan?
 - Véletlenszerűen? Jobb azt a lapot kirakni amelyiket nem használtak az „utóbbi időben”.
- ▶ A processzor gyorsító tár (cache) memória használatnál, vagy a böngésző helyi gyorsítótránánál is hasonló a helyzet.

Optimális lapcserélés

- ▶ Címkézzünk meg minden lapot azzal a számmal, ahány CPU utasítás végrehajtódik mielőtt hivatkozunk rá!
- ▶ Dobjuk ki azt a lapot, amelyikben legkisebb ez a szám!
- ▶ Egy baj van, nem lehet megvalósítani!
- ▶ Kétszeres futásnál tesztelési célokat szolgálhat!

NRU (Not Recently Used) algoritmus

- ▶ Használjuk a laptábla bejegyzés módosítás (Modify) és hivatkozás (Reference) bitjét.
 - A hivatkozás bitet időnként (óramegszakításnál, kb. 0.02 sec) állítsuk 0-ra, ezzel azt jelezzük, hogy az „utóbbi időben” volt-e használva, hivatkozva.
 - 0.osztály: nem hivatkozott, nem módosított
 - 1.osztály: nem hivatkozott, módosított
 - Ide akkor lehet kerülni, ha az óramegszakítás állítja 0-ra a hivatkozás bitet.
 - 2.osztály: hivatkozott, nem módosított
 - 3.osztály: hivatkozott, módosított
- ▶ Válasszunk véletlenszerűen egy lapot a legkisebb nem üres osztályból.
 - Egyszerű, nem igazán hatékony implementálni, megfelelő eredményt ad.

FIFO lapcserélés, Második Lehetőség.

- ▶ Egyszerű FIFO, más területekről is ismert, ha szükség van egy új lapra akkor a legrégebbi lapot dobjuk ki!
 - Listában az érkezés sorrendjében a lapok, egy lap a lista elejére érkezik és a végéről távozik.
- ▶ Ennek javítása a Második Lehetőség lapcserélő algoritmus.
 - Mint a FIFO, csak ha a lista végén lévő lapnak a hivatkozás bitje 1, akkor kap egy második esélyt, a lista elejére kerül és a hivatkozás bitet 0-ra állítjuk.

Óra lapcserélés

- ▶ Óra algoritmus: mint a második lehetőség, csak ne a lapokat mozgassuk körbe egy listába, hanem rakjuk körbe őket és egy mutatóval körbe járunk.
- ▶ A mutató a legrégebbi lapra mutat.
- ▶ Laphibánál ha a mutatott lap hivatkozás bitje 1, nullázzuk azt és a következő lapot vizsgáljuk!
- ▶ Ha vizsgált lap hivatkozás bitje 0 kitesszük!

LRU (Least Recently Used) algoritmus

- ▶ Legkevésbé (legrégebben) használt lap kidobása.
- ▶ Hogy valósítom meg?
 - HW vagy SW
 - HW1: Vegyük egy számlálót ami minden memória hivatkozásnál 1-el nő. minden laptáblában tudjuk ezt a számlálót tárolni. minden memóriahivatkozásnál ezt a számlálót beírjuk a lapba. Laphibánál megkeressük a legkisebb számlálóértékű lapot!
 - HW2: LRU bitmátrix használattal, n lap, $n \times n$ bitmátrix. Egy k. lapkeret hivatkozásnál állítsuk a mátrix k. sorát 1-re, míg a k. oszlopát 0-ra. Laphibánál a legkisebb értékű sor a legrégebbi!

NFU (Not Frequently Used) algoritmus

- ▶ minden laphoz tegyük egy számlálót. minden óramegszakításnál ehhez adjuk hozzá a lap hivatkozás (R) bitjét.
- ▶ laphibánál a legkisebb számlálóértékű lapot dobunk ki. (a leginkább nem használt lap)
- ▶ Hiba, hogy az NFU nem felejt, egy program elején gyakran használt lapok megőrzik nagy értéket.
- ▶ Módosítsuk: minden óramegszakításnál csinálunk jobbra egy biteltolást a számlálón, balról pedig hivatkozás bitet tegyük be (shr). (Öregítő algoritmus)
 - Ez jól közelíti az LRU algoritmust.
 - Ez a lap számláló véges bitszámú (n), így n időegység előtti eseményeket biztosan nem tud megkülönböztetni.

Lapozás tervezési szempontok I.

► Munkahalmaz modell

- A szükséges lapok betöltése. (Induláskor–előlapozás) A folyamat azon lapjainak fizikai memóriában tartása, melyeket használ. Ez az idővel változik.
- Nyilván kell tartani a lapokat. Ha egy lapra az utolsó N időegységben nem hivatkoznak, laphiba esetén kidobjuk.
- Óra algoritmus javítása: Vizsgáljuk meg, hogy a lap eleme-e a munkahalmaznak? (WSClock algoritmus)

► Lokális, globális helyfoglalás

- Egy laphibánál ha az összes folyamatot (globális), vagy csak a folyamathoz tartozó (lokális) lapokat vizsgáljuk.
- Globális algoritmus esetén minden folyamatot elláthatunk méretéhez megfelelő lappal, amit aztán dinamikusan változtatunk.
- Page Fault Frequency (PFF) algoritmus, laphiba/másodperc arány, ha sok a laphiba, növeljük a folyamat memóriában lévő lapjainak a számát. Ha sok a folyamat, akár teljes folyamatot lemezre vihetünk.(teherelosztás)

Lapozás tervezési szempontok II.

- ▶ Helyes lapméret meghatározása.
 - Kicsi lapméret
 - A „lapveszteség” kicsi, viszont nagy laptábla kell a nyilvántartáshoz.
 - Nagy lapméret
 - Fordítva, „lapveszteség” nagy, kicsi laptábla.
 - Jellemzően: $n \times 512$ bájt a lapméret, XP, Linuxok 4KB a lapméret. 8KB is használt (szerverek)
- ▶ Közös memória
 - Foglalhatunk memóriaterületet, amit több folyamat használhat.
 - Elosztott közös memória
 - Hálózatban futó folyamatok közti memória megosztás.

Szegmentálás

- ▶ Virtuális memória: egy dimenziós címtér, 0-tól a maximum címig (4,8,16 GB, ...)
- ▶ Több programnak van dinamikus területe, melyek növekedhetnek, bizonytalan mérettel.
- ▶ Hozzunk létre egymástól független címtereket, ezeket szegmensnek nevezzük.
 - Ebben a világban egy cím 2 részből áll: szegmens szám, és ezen belüli cím. (eltolás)
 - Szegmentálás lehetővé teszi osztott könyvtárak „egyszerű” megvalósítását.
 - Logikailag szét lehet szedni a programot, adat szegmens, kód szegmens stb.
 - Védelmi szint megadása egy szegmensre.
 - Lapok fix méretűek, a szegmensek nem.
 - Szegmens töredézettség megjelenése. Ez töredézettség összevonással javítható.

Pentium processzor virtuális címkezelése I.

- ▶ Sok szegmens lehet, egy szegmens virtuális címtere: 2^{32} bájtos (4GB)
 - XP, Linuxok egyszerű lapozásos modellt használnak, egy folyamat–egy szegmens– egy címtér.
 - Szegmensek száma: Pentium–ban 16000, a TSS szegmens értékéből, processzor tulajdonság.
- ▶ LDT– Local Descriptor Table
 - minden folyamatnak van egy sajátja.
- ▶ GDT– Global Descriptor Table
 - Ebből 1 van ezt használja mindenki.
 - Ebben találhatók a rendszerszegmensek (rendszer is)
- ▶ Fizikai cím: Szelektor + offset művelet elvégzése

Pentium processzor virtuális címkezelése II.

- ▶ Szegmens elérése: 16 bites szegmens szelektor
 - Ennek alsó 0–1. bitje a szegmens jogosultságát adja
 - 2. bit=0=GDT-ben, 1=LDT-ben van a szegmens leíró.
 - Felső 13 bit a táblázatbeli index. (8192 elemből áll minden két tábla)
- ▶ Az LDT, GDT tábla elem 32 bites, ebből kapjuk bázis címet, amihez az eltolást hozzáadjuk. (eredmény is 32 bit!) Ez adja a lineáris címet.
- ▶ Ha a lapozás tiltott, ez a valós fizikai cím, ha nem tiltott akkor kétszintű laptábla használat: lapméret 4KB, 1024 (10 bit) elemű laptáblák.
- ▶ A gyorsabb lapkeret eléréshez TLB-t is használ.

Pentium processzor védelmi szintjei

- ▶ 0- Kernel szint
- ▶ 1- Rendszerhívások
- ▶ 2- Osztott könyvtárak
- ▶ 3- Felhasználói programok
- ▶ Alacsonyabb szintről adatok elérése engedett.
- ▶ Magasabb szintről alacsonyabb szintű adatok (0–1 szint) elérése tiltott.
- ▶ Eljárások hívása megengedett, csak ellenőrzött módon (call szelektor)
- ▶ Felhasználói programok osztott könyvtárak adatait elérhetik, de nem módosíthatják!

Köszönöm a figyelmet!

zoltan.illes@elte.hu

A nyílt forráskód napjainkban

Hargitai Zsolt

üzletfejlesztési vezető

zsolt.hargitai@novell.com

NetIQ Novell SUSE Magyarországi Képviselet



Napirend

- Nyílt forráskódú szoftverek története
- Elterjedtség
- Nyílt forráskódú fejlesztési modell
- Ingyenes és fizetős megoldások. Mikor melyiket válasszuk?
- SUSE megoldások

Nyílt forrás kódú szoftverek története

Történelem, fogalmak

Projektek

- GNU projekt - Stallman - 1983
- Linux - Torvalds - 1991
- Szoftverek, szervezetek
 - Szabad szoftver - FSF
 - Nyílt forrás kódú szoftver - OSI
 - FLOSS
- Licencek
 - GPL, LPGL, BSD, Mozilla Public license

Ennél sokkal több

- Nyílt szabványok támogatása
- Pozitív társadalmi hatások
 - Innováció elősegítése
 - Magánszemélyek, közösségek, oktatási intézmények és vállalatok együttműködése
 - Országokon átívelő projektek
- Üzleti lehetőségek
 - Piacképes szaktudás, álláslehetőségek
 - Vállalkozások, szoftverfejlesztés nyílt forrás kód alapon
 - Nyílt forrás kódú projektek támogatása
 - Vállalkozások indítása kisebb beruházási költséggel

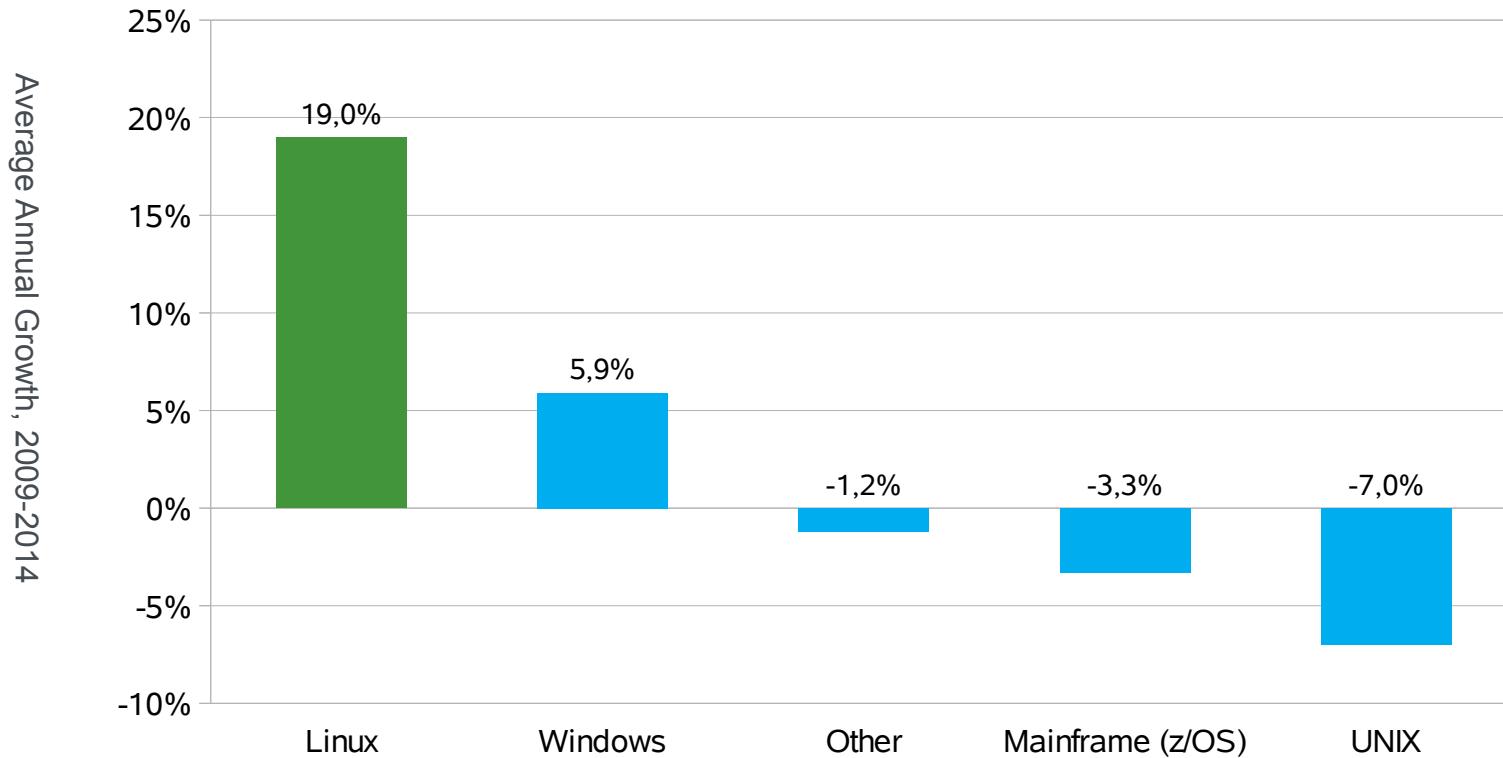


Elterjedtség

Miért a Linux és miért most?

A leggyorsabban növekvő operációs rendszer

IDC: Worldwide Server Operating System Revenue Growth, 2009-2014



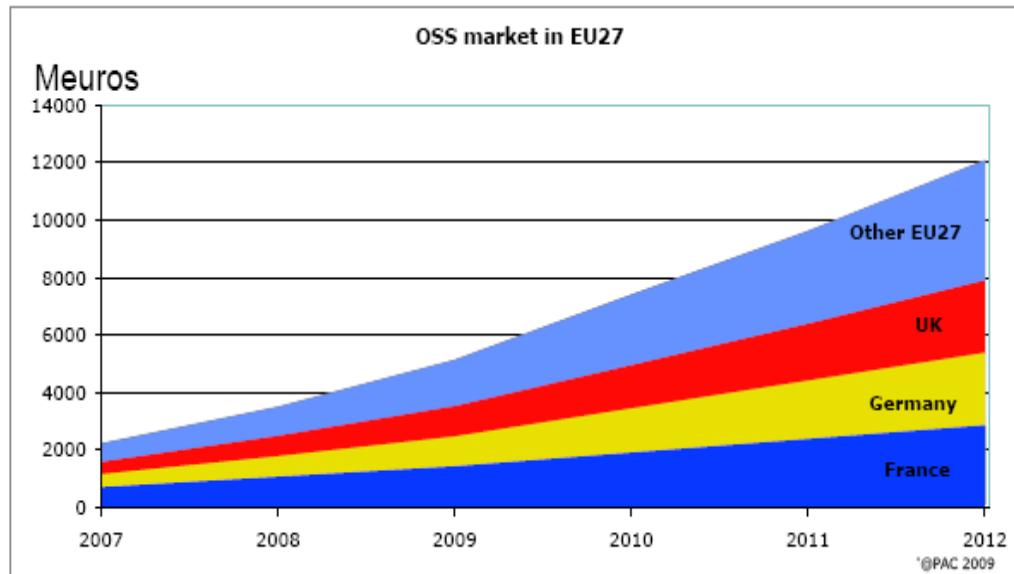
Source: IDC, "Worldwide Operating Environments 2010–2014 Forecast: A First Look" March 2010



Nyílt forráskód az Európai Unióban



Az Európai Unió tagállamainak nyílt forráskódú szoftver-piaca 2007-től 2012-ig



Forrás: Európai Bizottság, **Economic and Social Impact of Software & Software-Based Services**, http://cordis.europa.eu/fp7/ict/ssai/docs/study-sw-2009v2_en.pdf

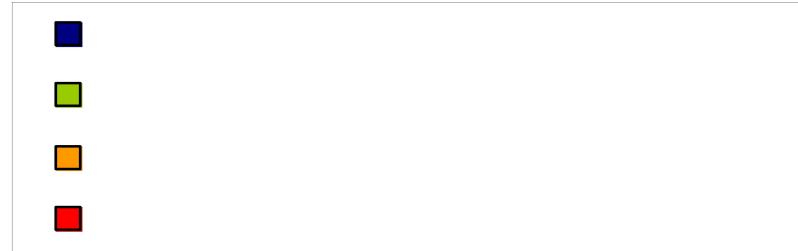
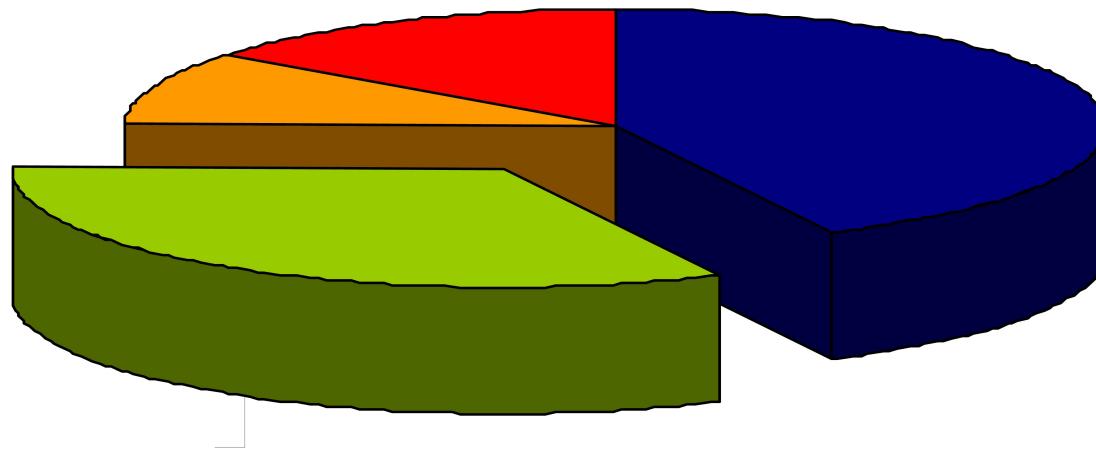


Elterjedtség Magyarországon

Jelenlegi használat

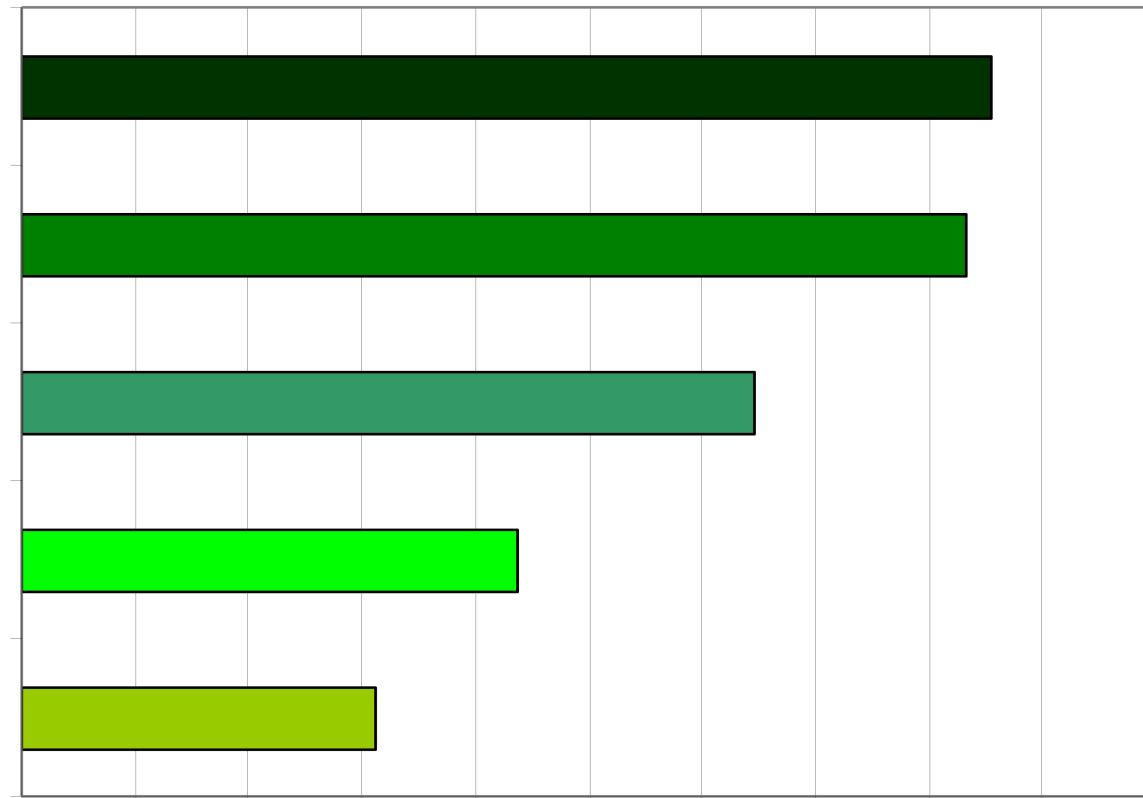
1. Mi

re



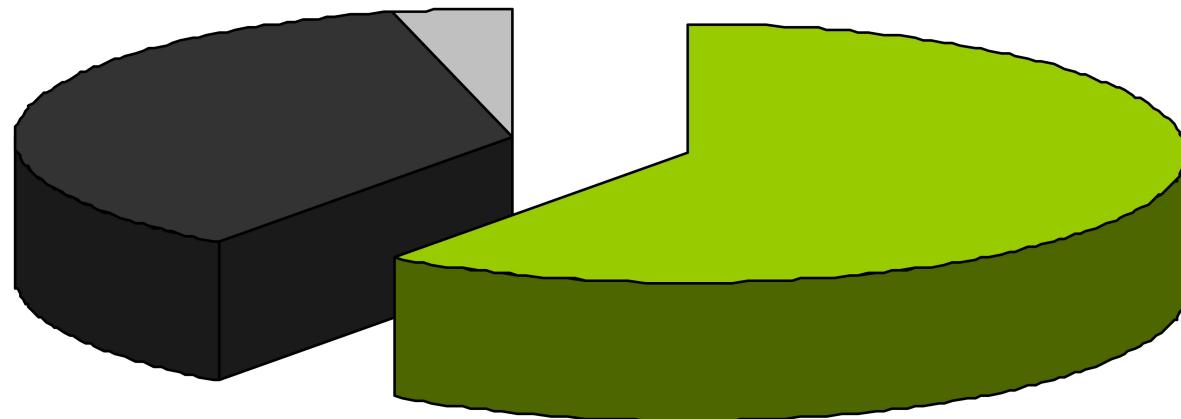
Miért használják?

2.1.



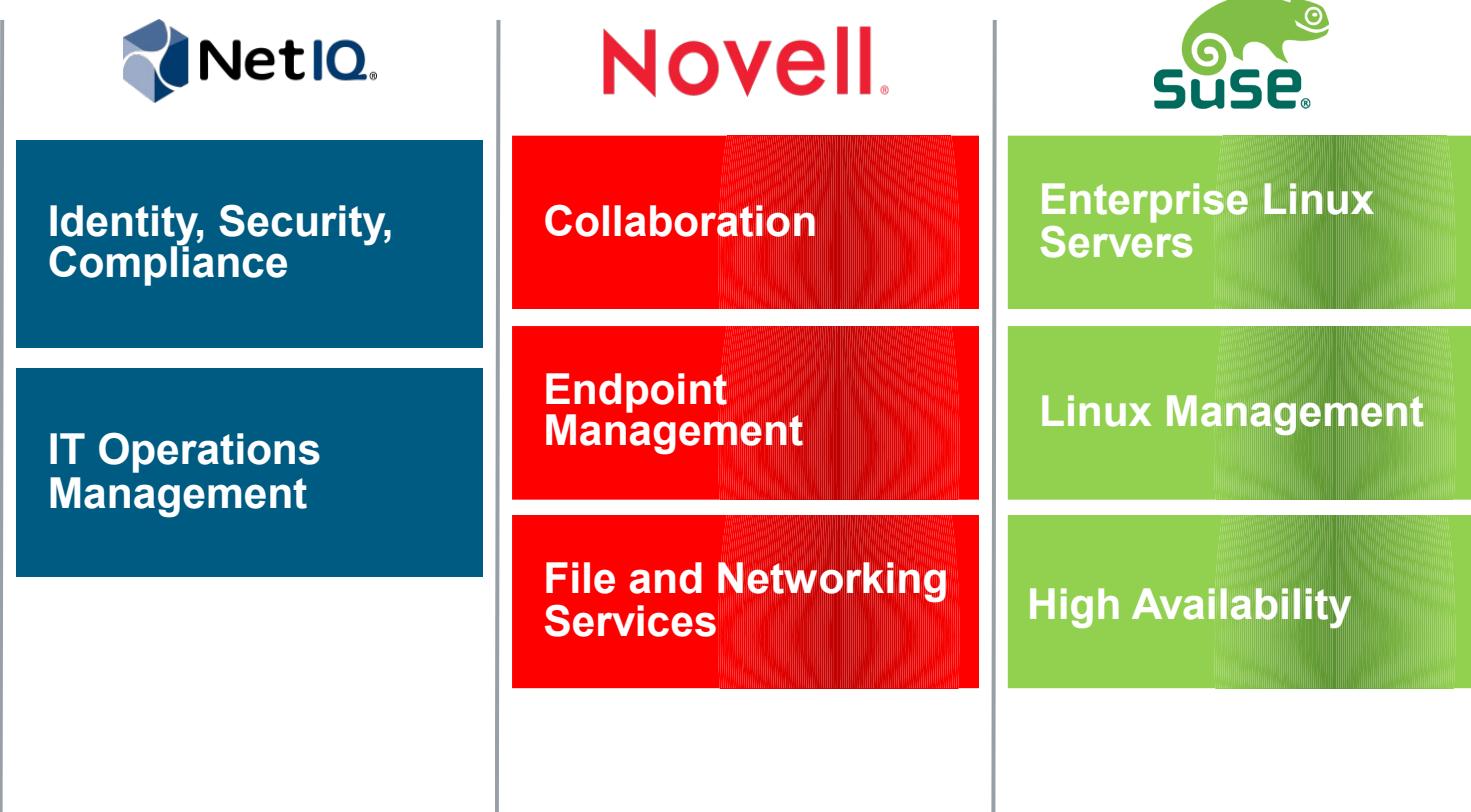
Tovább terjed a Linux?

4. Hog



A Novell és a nyílt forrás kódú fejlesztés

A cégsoport



Mi az hogy SUSE?

Trivia

Software- und System-Entwicklung, vagyis S.u.S.E



Konrad Zuse



A Novell, illetve a SUSE által támogatott projektek



Fejlesztési modell



Ingyenes és fizetős megoldások.
Mikor melyiket válasszuk?
Hogyan alkalmazzuk?

Szoftver és támogatás

- Disztribúciók
 - openSUSE, Ubuntu, Fedora
 - Testre szabott disztribúciók, alkalmazások
 - openSUSE Build Service
- Támogatás
 - Fórum, levelező lista
- Magyar nyelvű dokumentációk
 - hu.opensuse.org
 - openSUSE 11.1 reference guide – 634. o
 - SLED 10 SP2 leírások – KDE és GNOME felhasználói kézikönyv összesen 800 o.
 - KDE és Gnome gyorskalauz - 10.o



Oktatás

- HUEDU projekt oktatási intézmények számára
 - Szoftver, támogatás, oktatás, szakmai gyakorlat
 - <http://huedu.hu>
- Érettségi Szoftvercsomag
 - openSUSE, Sulix, UHU-Linux
- Tankönyvsorozat
 - Kezdő ismeretek
 - Az openSUSE operációs rendszer
 - Az OpenOffice.org Impress, Writer, Calc (3 kötet)
 - openSUSE – önállóan otthon és az irodában (az önálló otthoni és irodai openSUSE-használat Nagy Képeskönyve)



Fizetős, linux alapú megoldások

- Vállalati Linux szerverek
 - RedHat Enterprise Linux
 - SUSE Linux Enterprise Server
- Linux alapú infrastruktúra
 - Novell Open Workgroup Suite
- Csoportmunka
 - Novell és IBM megoldások
 - Alkalmazások
 - Üzleti alkalmazások
 - Adatbáziskezelők
 - Webes szolgáltatások



Miért SUSE?

Miért SUSE?



Miért SUSE?



Vendor Lock-in kizárva

- Széles körű hardver támogatás



X86



Blades



SAN



Cluster



Mainframe

- Azonos platform minden környezetben
 - VMware, Hyper-V, KVM, Citrix XenServer, Xen
 - Publikus felhők
- Kereszplatformos támogatás és felügyelet
 - Szabványokra épülő rendszer felügyelet
 - Irányítsa teljes Linux állományát egy konzolról
- Nyílt forrású ökoszisztéma

Vendor Lock-in kizárva

Vezető szerep a nyílt forrású közösségekben



Miért SUSE?



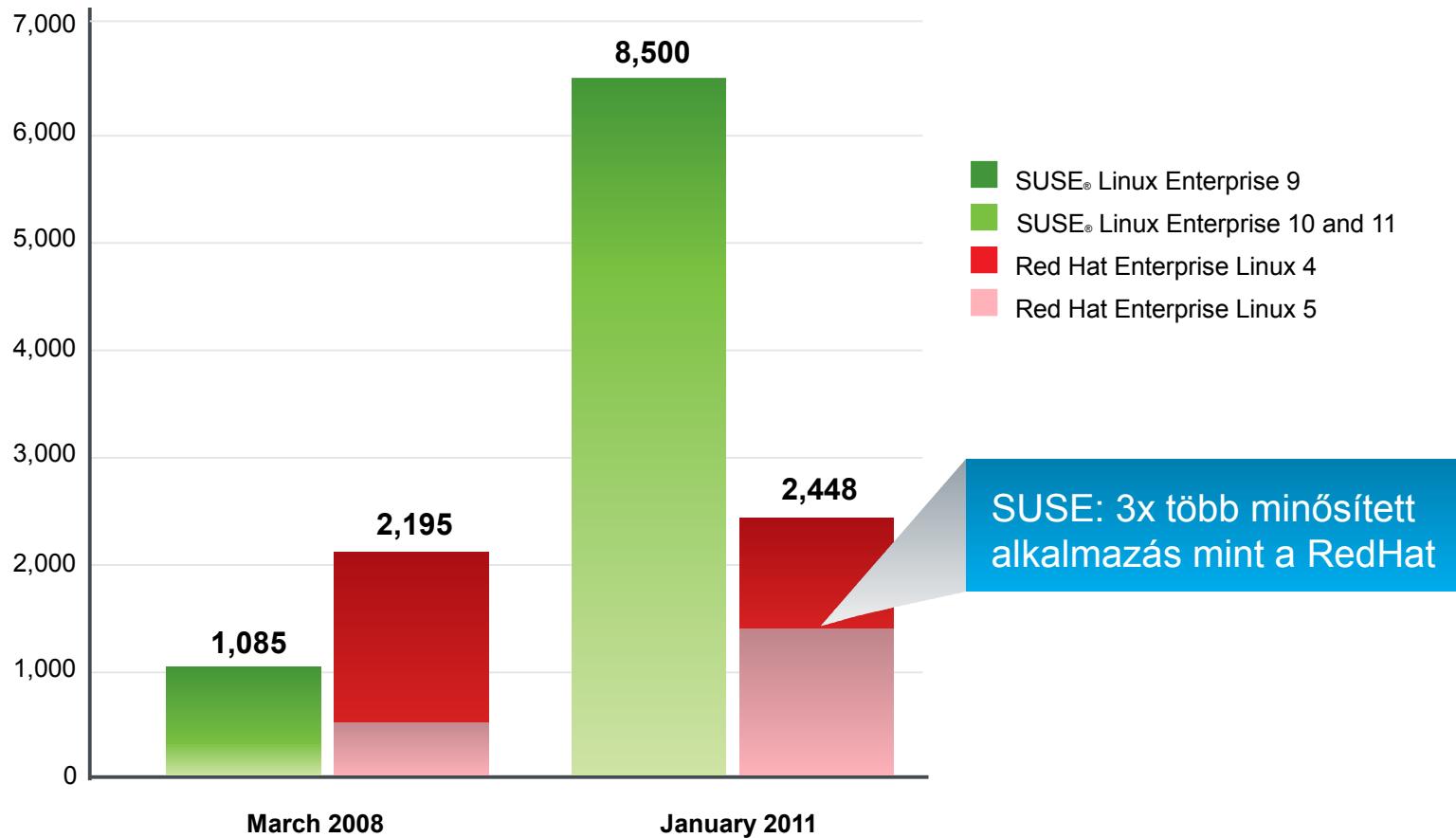
Vendor lock-in
kizárva

Rugalmasság
Döntési szabadság

Nagyvállalati
Minőség

Döntési szabadság és rugalmasság

A legtöbb minősített alkalmazás



6,080 ISV applications are certified on SUSE Linux Enterprise, from 1,564 vendors.
2,869 ISV applications are certified on Red Hat Enterprise Linux, from 1,594 vendors.
Source: Novell and Red Hat Websites, Oct 2010



Miért SUSE?



Vendor lock-in
kizárva



Rugalmasság
Választási szabadság

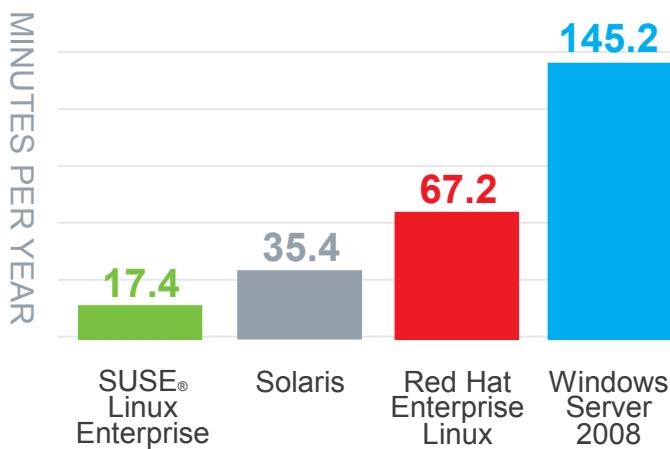


Nagyvállalati
Minőség

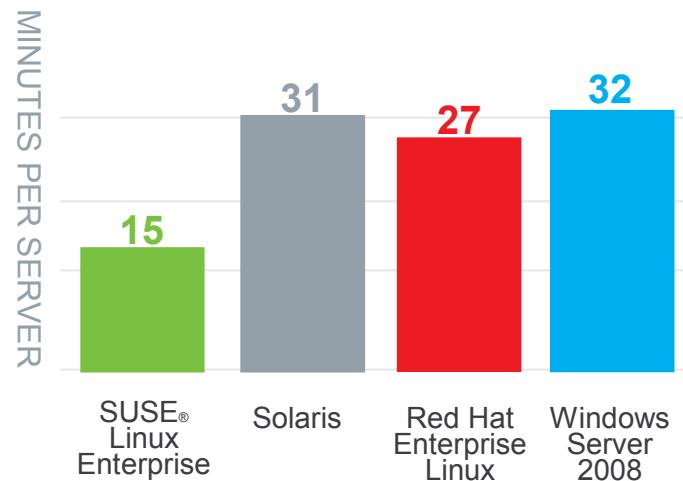
Nagyvállalati minőség

Megbízható, fenntartható platform

- Legalacsonyabb nem tervezett kiesés



- Legalacsonyabb átlagos frissítési idő – tervezett leállás



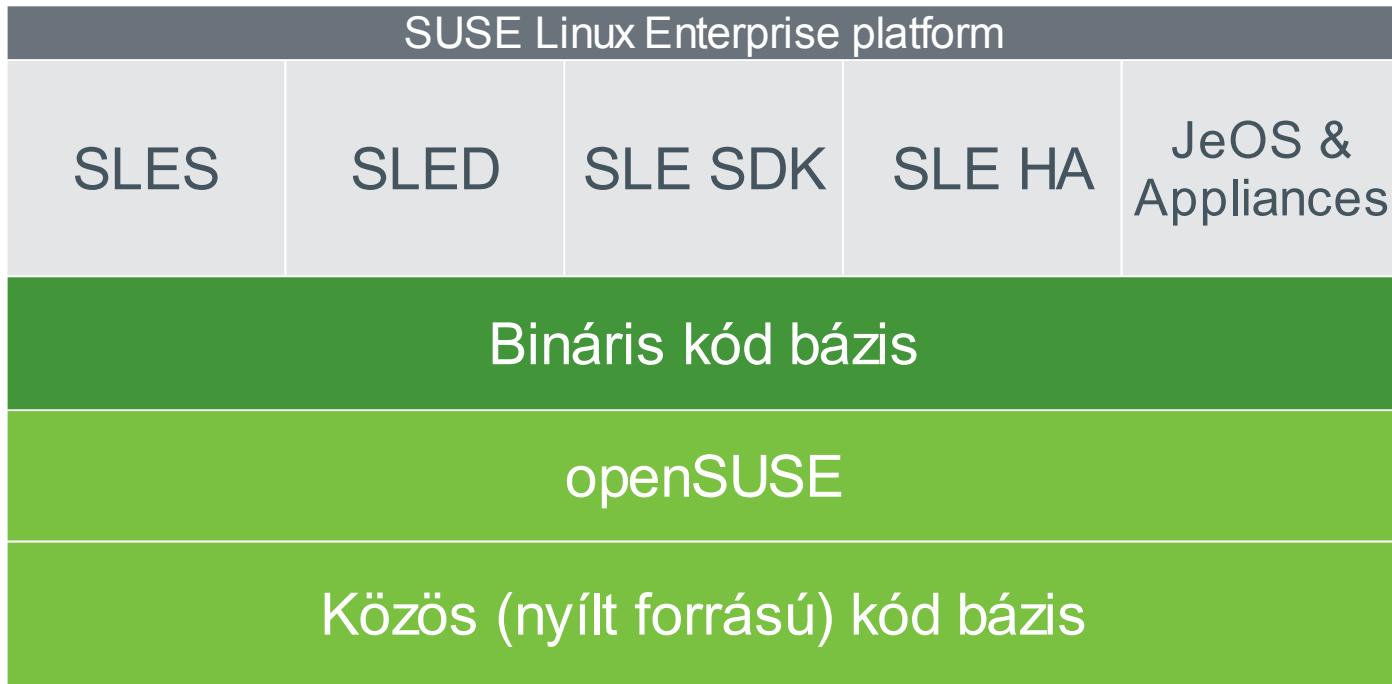
Source: Information Technology Intelligence Corporation, ITIC 2009 Global Server Hardware and Server OS Reliability Survey, July 2009

Miért SUSE?



Miért SUSE?

Közös kód bázis



Szilárd alap a SUSE Linux Enterprise termékek számára
Teljes körűen támogatott alap rendszer – L3 szintig



Miért SUSE?

MAINFRAME LINUX

OVER 80%

of all Linux running on mainframe computers is SUSE Linux Enterprise Server

LINUX IN AEROSPACE AND DEFENSE

Nearly 80% of the US Fortune 500 aerospace and defense companies use SUSE Linux Enterprise Server



SAP ON LINUX

OVER 70%

of all SAP running on Linux runs on SUSE Linux Enterprise Server

LINUX IN RETAIL

NEARLY 70%

of the US Fortune 100 general merchandisers, specialty retailers, and food and drug stores use SUSE Linux Enterprise Server

LINUX IN AUTOMOTIVE



SUSE Linux Enterprise Server is used by nearly all of the world's major automobile manufacturers

MOST CERTIFIED APPLICATIONS

Over 8500 applications are certified and supported on SUSE Linux Enterprise Server, more than any other Linux distribution

8500 OVER

LINUX IN HPC



Half of the world's largest supercomputer clusters use SUSE Linux Enterprise Server

BEST LINUX SUPPORT

SUSE offers better Linux support than Red Hat or Oracle



LINUX IN GLOBAL FORTUNE 100

Over two-thirds of the global Fortune 100 use SUSE Linux Enterprise Server



LINUX IN CHINA



SUSE Linux Enterprise Server is the most widely used commercial enterprise Linux distribution in China — more than Red Hat

MOST CERTIFIED HARDWARE

Over 13,500 hardware systems are certified and supported on SUSE Linux Enterprise, more than any other Linux distribution

13,500

Miért SUSE - Stratégiai együttműködések



Partnerség 2006-2015

További befektetések a SUSE Linux népszerűsítésébe

Kiterjesztett támogatás

Hosszú távú műszaki együttműködés

Közös Interop Lab: a két fél mérnökei minden nap együtt dolgoznak



OEM partner, dedikált SUSE változattal: SUSE Linux Enterprise Server for VMware

Vmware szoftver készülékek SUSE Linux Enterprise Serveren

vSphere 4 óta SUSE Linux Enterprise Server előfizetést tartalmaz a termék



>70% részesedés az SAP Linux rendszerek közt

SUSE Linux Enterprise Server for SAP:

- SAP-ra szabott telepítő
- Saját frissítési csatorna
- Saját életciklus
- Fürttel csomagolva

Fejlesztési referencia platform

Kizárolagos HANA partner



Miért SUSE – Kiterjedt ökoszisztemáma

3,200

Megoldás
Szállító

600

Képzési
partner

1,300

Technológiai
Partner

8,500+

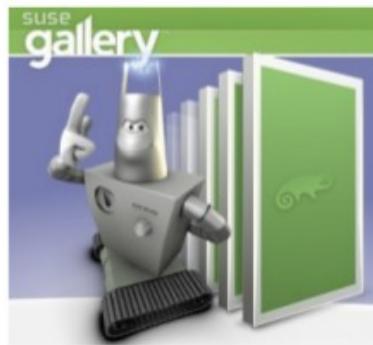
Tanúsított
SUSE
Linux
Enterprise
termék



Miért SUSE – Nyílt forrású innováció



Open Build Service



SUSE Gallery



...

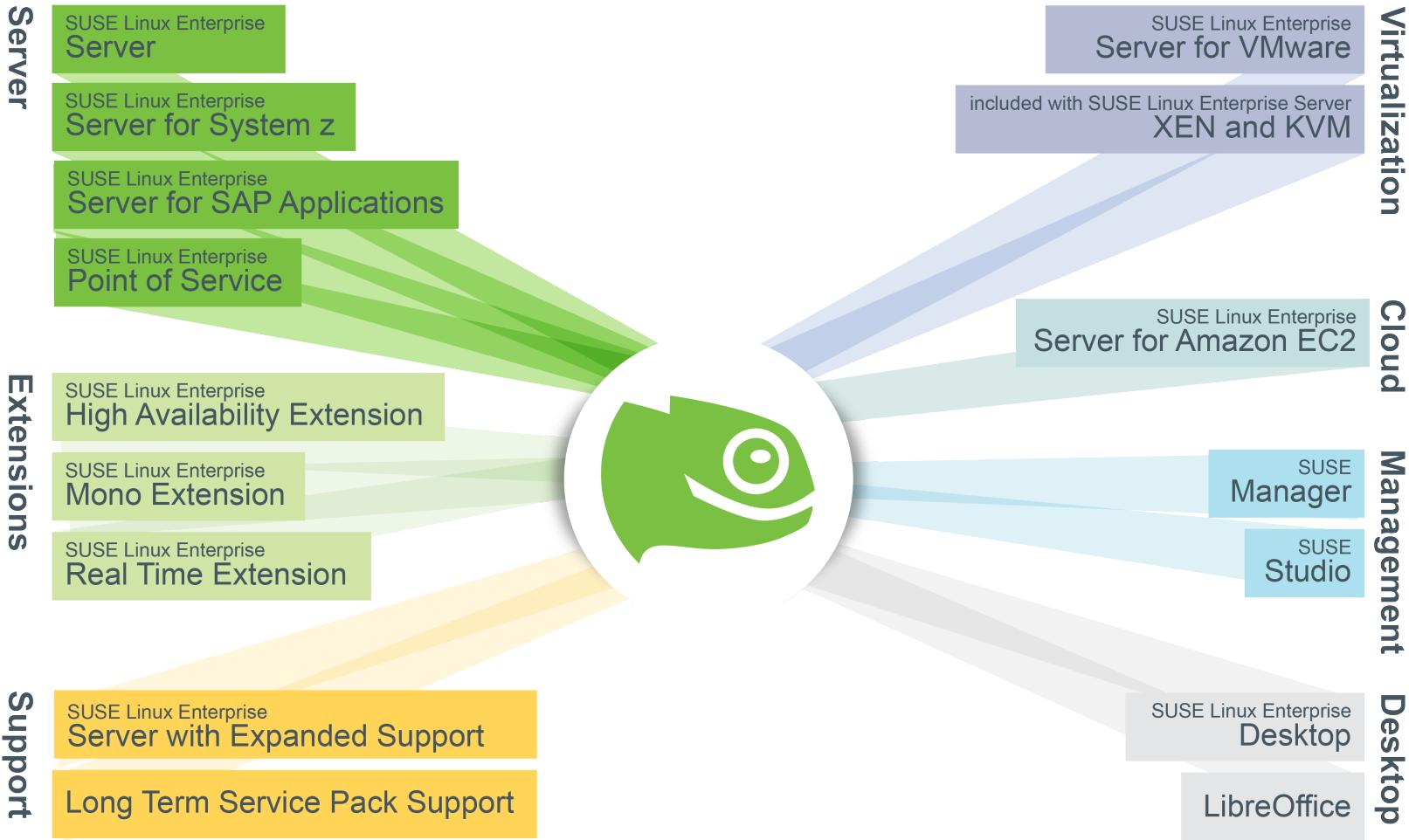


Ügyfelek



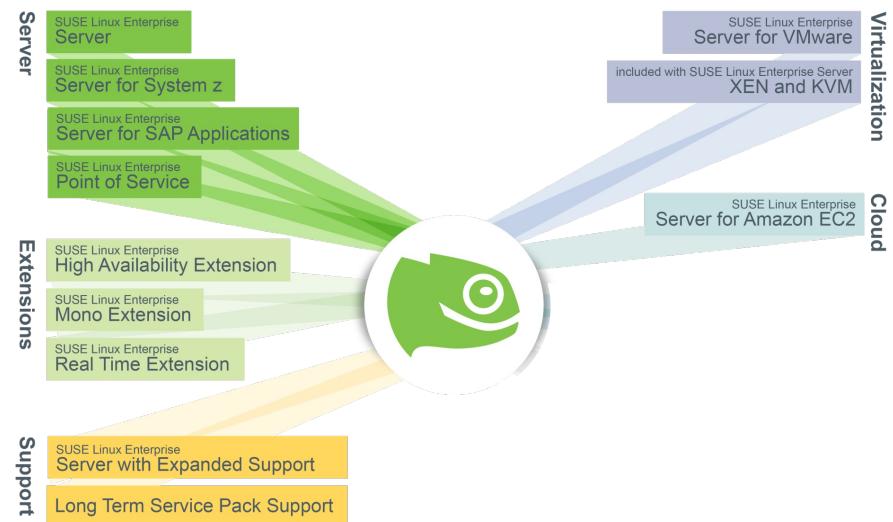
SUSE termékek

Teljes portfólió



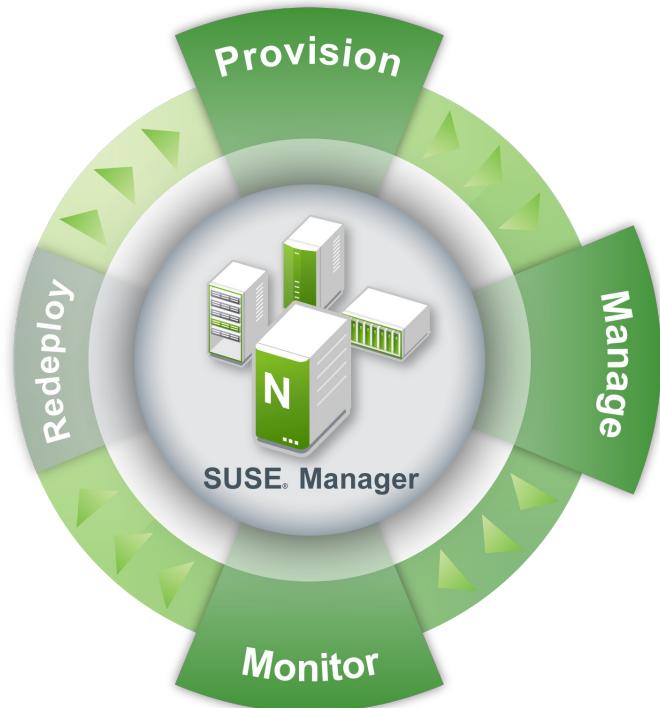
SUSE Linux Enterprise Server

- Linux mindenhol – desktop, PoS, készülék, szerver, mainframe, SAP
- Magas teljesítmény
- Kedvező költségek
- Interoperabilitás



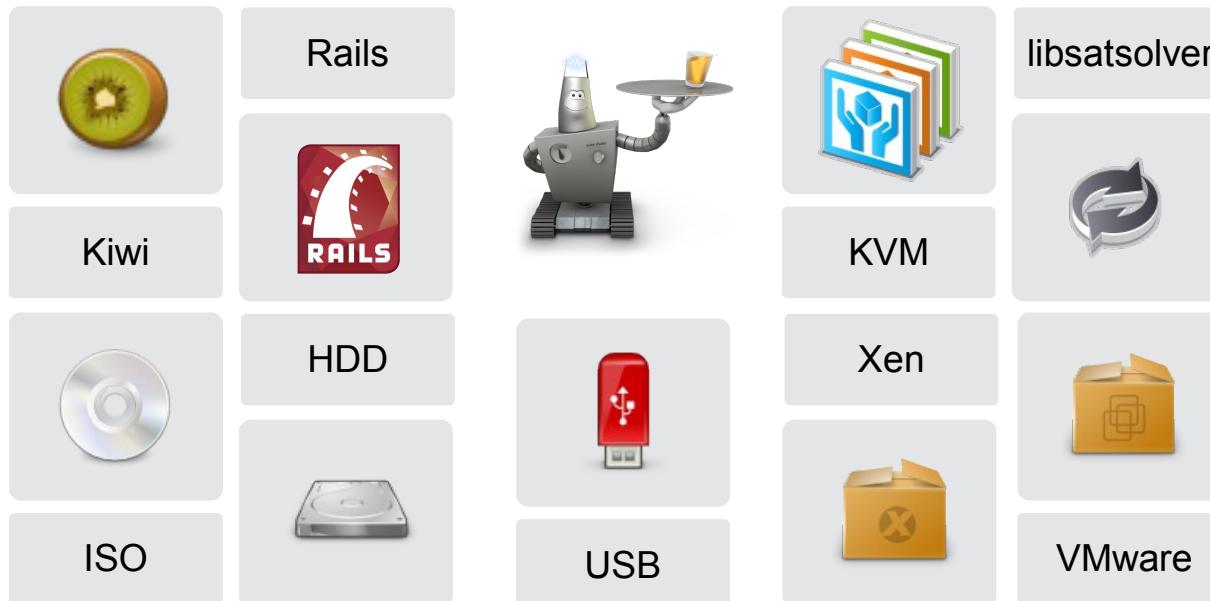
SUSE Manager

- Egységes felügyelet SUSE és RedHat Linuxhoz
- Patch és csomag kezelés
- Konfiguráció kezelés
- Telepítés automatizálása
- Monitorozás



SUSE Studio

- Rendszerképek gyors létrehozása
- Alkalmazások integrációja terítés előtt
- Megkönnyíti a terítést



SUSE megoldások helye a piacon

Piaci helyzet



A költségcsökkentés kényszere
velünk marad a belátható jövőben



Virtualizációs sűrűség növekszik



Számítási kapacitás növekszik



SUSE az adatközpontban

Linux virtualizálva és felhőben

UNIX – Linux migráció

SAP Linuxon



További projektek

Licencköltség optimalizálás MS Office → OpenOffice

- Dokumentumformátumok, oktatás, MS Office integráltság

Internet Explorer → Firefox

- Egy-két IE függő alkalmazás

Desktop átállás: Windows → Linux

- Alkalmazások
- Megoldás: átírás, emulátor, virtuális gép, terminálserver

Vékony/hibrid kliens



Nyílt forrás kódú referenciák

Webkiszolgáló infrastruktúra kialakítása (SLA 99,999%)

Komplex DMZ infrastruktúra üzembe állítása

Virtualizációs rendszer

Informatikai infrastruktúra migrációja (vékonykliens, outsource)

Irodai programcsomag bevezetése (1000 felhasználó)

Testre szabott disztribúció készítése

Heldesk-rendszer kialakítása





NetIQ Novell SUSE
Magyarország
1124 Budapest
Csörsz u. 45.

+36 1 489 4600
www.suse.hu

Csatlakozás:
huedu.hu

This document could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein. These changes may be incorporated in new editions of this document. SUSE may make improvements in or changes to the software described in this document at any time.

Copyright © 2011 Novell, Inc. All rights reserved.

All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States. All third-party trademarks are the property of their respective owners.

