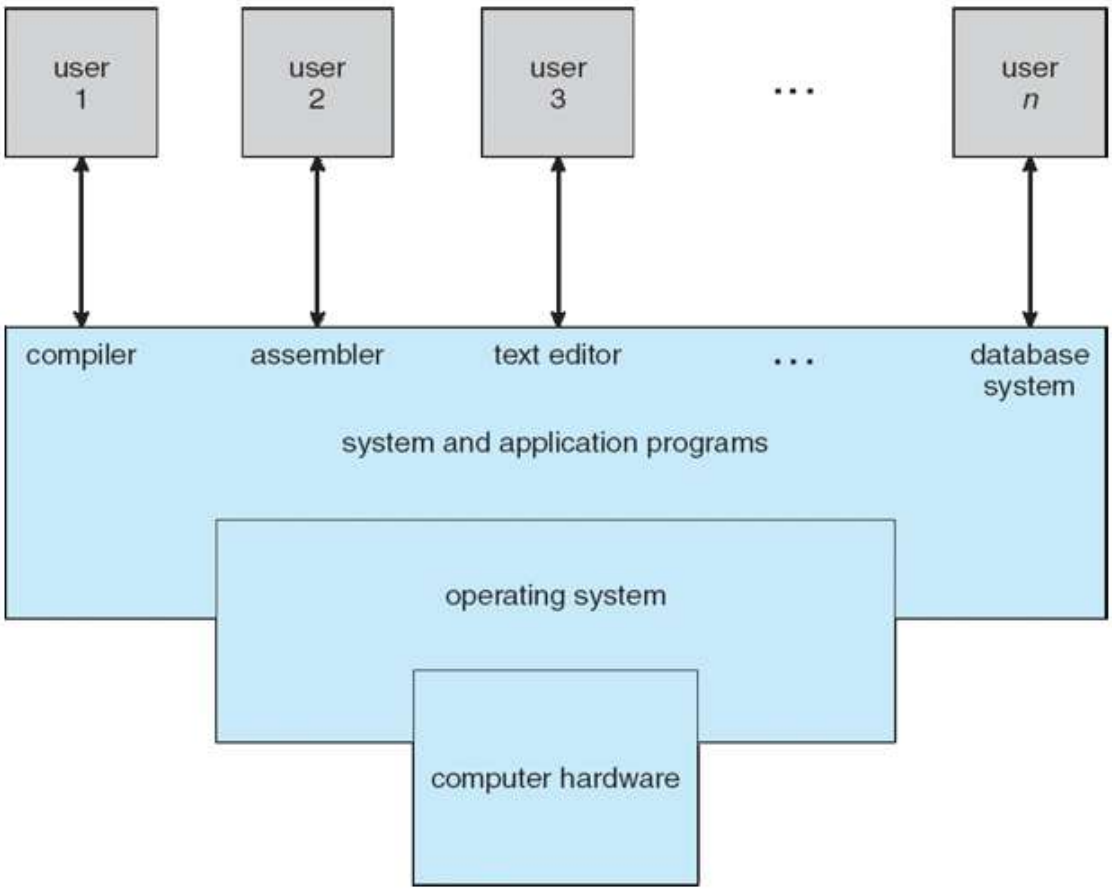
Chapter 1

- Operációs rendszer alapvetően a felhasználó és a számítógép közé beékelődő réteg
- Céljai
  - Végrehajtsa a programokat/feladatokat, könnyítse a felhasználó dolgát
  - Legyen kényelmes
  - Hatékonyan használja a gép erőforrásait → A felhasználót nem érdekli a hogyan, csak az eredmény
- Mi is az oprendszer?
  - Erőforráskezelő
    - Kezeli a megadott erőforrásokat
    - Prioritások alapján ütemezi mely dolgok a fontosak
  - Egy irányító program
    - Végrehajt programokat, meggátolja a hibákat és a nem megfelelő használatot
  - Nincs egy univerzálisan elfogadott definíció
- Vas/Hardver-Oprendszer-Alkalmazások-Felhasználó
  - Az a lánc, amivel effektíve a felhasználó interaktál a számítógéppel
  - Erre a négy részre bontható egy számítógépes rendszer, de egymásra épülnek

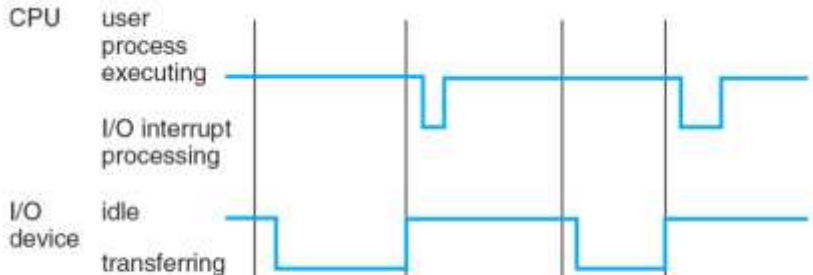


- Mit kezel az oprendszer?
  - CPU
  - Memória
  - I/O eszközök
- Kernel – „*The one program running at all times on the computer*”
  - Az oprendszer része, ami mindennek az alapja, és a legfontosabb feladatokat végzi
  - Kernelen alapul a rendszer maga, és azon minden más
  - Minden ami nem kernel az lehet rendszerprogram, vagy külső alkalmazás
- Számítógép indulása, bootolási folyamat
  - Egy bootstrap program betölt, ami legtöbbször EPROM-ban tárolnak, és firmware-nek hívnak (alapszoftver?)
  - Betölti a kernelt, majd minden mást
- Számítógép üzemelése
  - Ha több eszköz van csatlakoztatva a számítógéphez, akkor egy közös busszal vannak összekötve a közös memóriával
  - Konkurens execution probléma lehet
  - I/O és CPU tud egyszerre konkurensen működni
  - Minden device controllor feladata egy eszköz „irányítása”, és ezeknek van egy helyi buffere
  - A device controller szól a CPU-nak egy megszakítással, hogy elvégezte a feladatát

Első előadás vége

Megszakítások → Minden alapja lényegében

- interrupt service routine címét
  - Regiszterek állapotának tárolása basically
- A megszakítás elmenti az általa megszakított instruction címét
- Trap/Exception → Szoftver/program által tolt megszakítás, ami vagy hiba miatt vagy felhasználói kérés miatt jött létre
- Minden operációs rendszer megszakításokon alapul
- Példák megszakításra
  - Szoftveres megszakítás
    - 0-val való osztás → Kilövi a programot
      - Cizellelható, kivételkezeléssel
  - Pixel megjelenítése, billentyű lenyomása, akár 1 byte mozgatása
    - Ezek alapvetően a CPU feladatai
- Ki manageli a megszakításokat? → A kernel legfontosabb feladata



- Megszakítások két fő iránya
    - Lekérdezés alapú → Polling
    - Vektoralapú → Ez az elterjedtebb

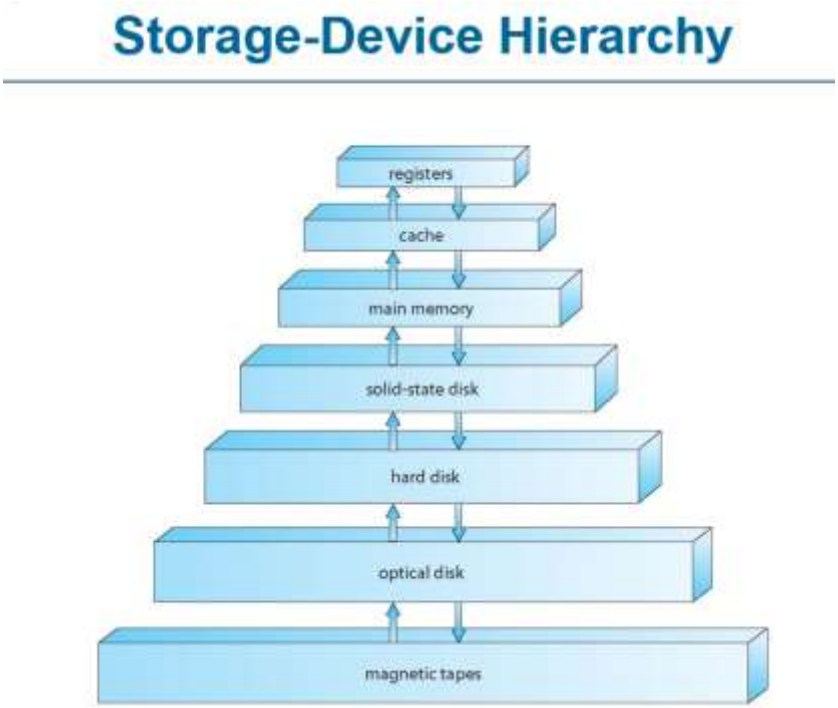
I/O struktúra, I/O műveletek

- 1. Amikor elkezdődik egy I/O művelet, elvesszük a felhasználótól az irányítást, és csak akkor adjuk vissza, ha végzett az I/O művelet
    - Wait instruction → CPU Idle
- 2. Amikor elkezdődik az I/O művelet azonnal visszaadjuk a felhasználónak az irányítást, nem várjuk meg a befejeződést
    - Rendszerhívások → Kérés az oprendszerhez, hogy a felhasználó várja meg a végét az I/O műveletnej

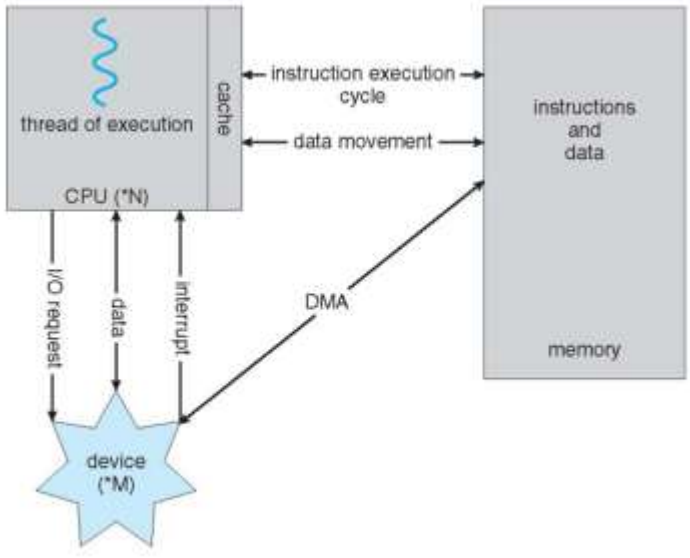
Storage

- Alapok
    - SI mértékegységek -gotta now
    - Alapegység 1 bit → 0 / 1
    - Byte → 8 bit → Legkisebb tárolási egység legtöbb számítógépen
        - Legtöbb számítógép ennél kisebb dolgot mozgatni sem tud
    - Szó → Gépi szó
        - Architektúrától függ, egy egységet jelent az adott architektúrán
        - 64 biten → Egy szó 8 byte
        - 32 biten → Egy szó 4 byte
        - A gép nagyrészt szavakban gondolkozik
        - Ezekkel ellentétben a hálózatszolgáltatók bitben dolgoznak → Ezzel „csalva”, hogy nagyobbaknak tűnjenek a számok
    - Átviteli egysége
        - Hálózat → Bit
        - Storage → Byte
        - Processzor → Gépi szó (architektúra függő, 64 biten 8 byte)
        - I/O rendszer → Egységek/blokkok → Fájlrendszertől függ
    - Random órán leírt dolgok még ide
        - 1 int → 4 byte
        - 2 int → 8 byte –> 64 bites processzornak 16 byte, mert 8-nál kisebbet nem tud kezelni
        - Bit → Byte → Gépi szó → I/O blokkok
- Storage Structure
    - Main Memory
        - Egyetlen nagy „adattároló” melyet a CPU közvetlen elér
        - Random Access
        - Volatile → Áram nélkül elveszik a rajta lévő adat
        - CMOS → Nem igazán ide tartozik, de nagyjából mégis
    - Secondary storage

    - Non Volatile → Kikapcsolás után is megmaradnak rajta a dolgok
    - Nagy méretű
    - HDD/SSD → Disk controller
- Storage Hierarchy
    - Minél nagyobb, annél lassabb
    - Regiszter leggyorsabb, és a legkisebb méretű



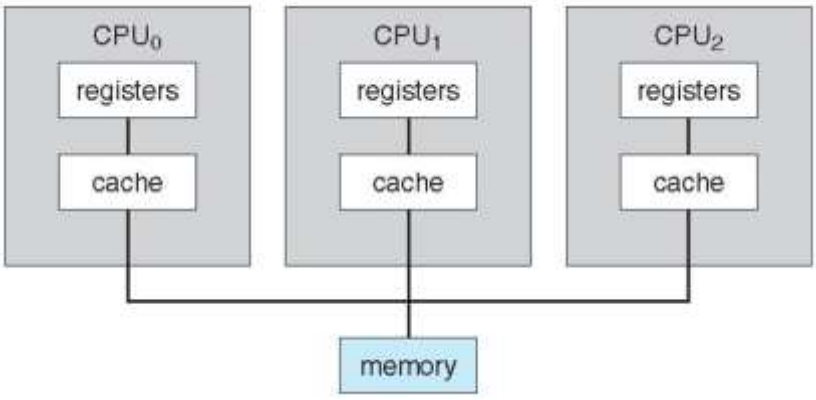
- Caching/Gyorsítótárazás
    - CPU sokkal gyorsabb mint egy HDD/SSD vagy az I/O alrendszer, ezért vezették be a Cachet
    - Átmenetileg „idehozunk” adatot valamelyik lassabb tárolóról
- Közvetlen Memória hozzáférés -DMA
    - Nem a processzoron keresztül mozgassunk adatot eszközről a memóriába, hanem közvetlenül az eszközről a memóriába



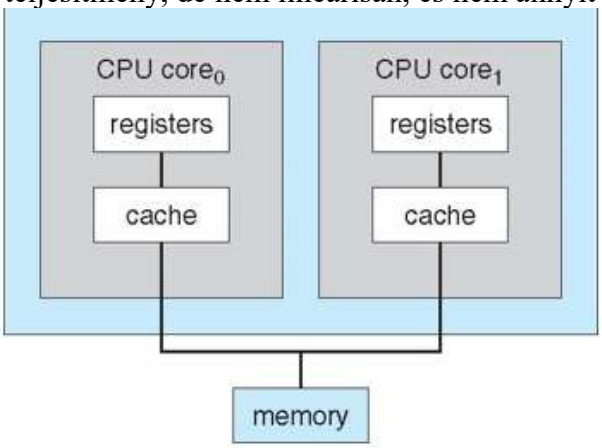
A von Neumann architecture

Számítógép Architektúrák

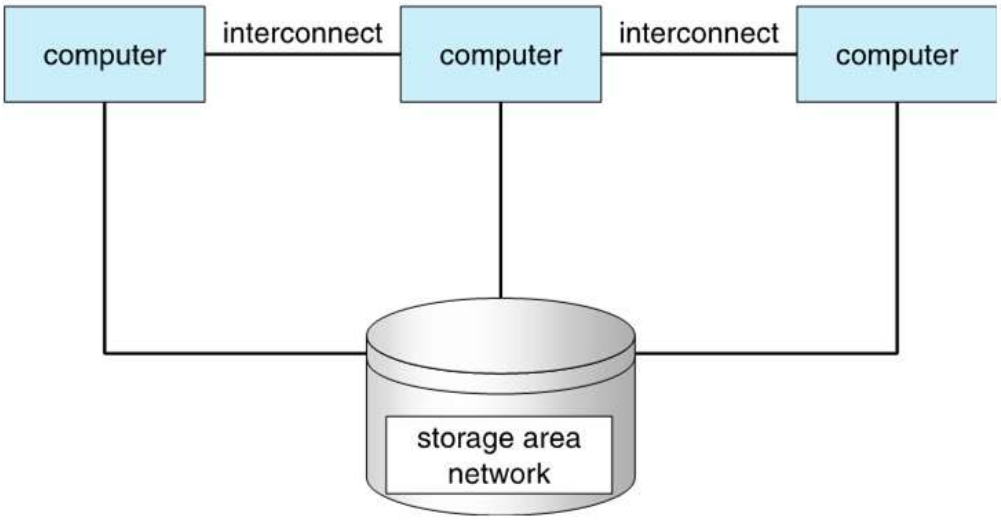
- Processzorok/Multiprocesszorok
- Legtöbb rendszer egy általános célú processzort használ
- Sok rendszernek vannak dedikált special purpose processzorai
- Multiprocesszorok
    - Nagyobb teljesítmény
    - Skálázható
    - Megbízhatóbb
    - Két típusa
        - Aszimmetrikus Multiprocessing
            - A processzorok nem egyenrangúak
            - Business piacra jellemző
            - Saját CPU architektúrák
            - Céltevékenységek
        - Szimmetrikus multiprocessing
            - A processzorok egyenrangúak
            - Konzumer piacra jellemző



    - Nem ugyanaz mint a több mag, fizikailag több processzororról beszélünk
- Multicore processzorok
    - Hoznak teljesítmény, de nem lineárisan, és nem annyit mint 2 fizikai processzor



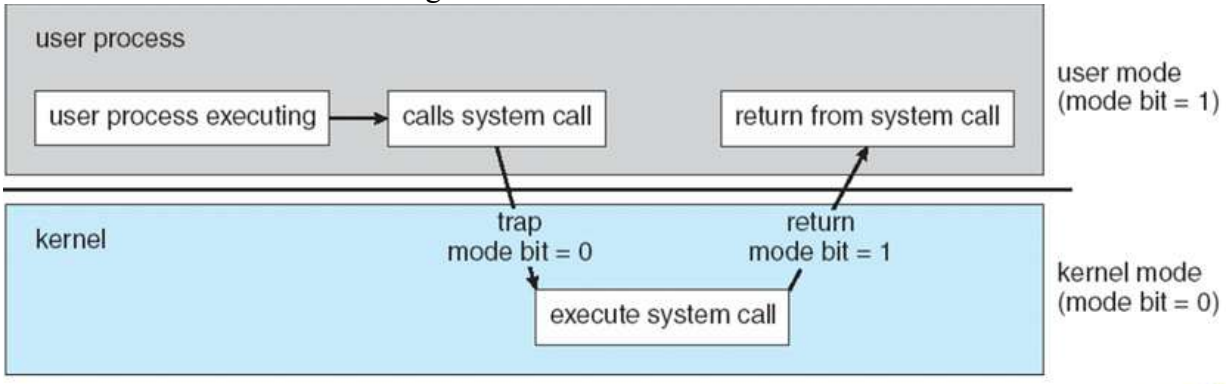
- Clustering
  - Gépek összekötése → gyorsabb mint egy gép magában sokkal
  - Gépek közt optikai összeköttetés
  - Sata 3 → Max 6 Gbit Optika → Több mint 10 Gbit
  - SAN → Storage Area Network



Második előadás vége

Dual mode

- Egy védelmet jelent az operációs rendszernek, és rendszerkomponenseknek
- Dual mode
  - Kernel mode
    - Eszközhozzáférés, file elérése stb.
    - Privilegizált utasítás
  - User mode
  - Módbit → Egy bit ami azt jelöli éppen Kernel vagy User módban van az OS
  - A két mód közötti váltás rendszerhívásokkal valósul meg
  - A lényege, hogy a user nem fér hozzá sok dologhoz, ezzel védve a usert a saját hülyeségétől, és az OS-t is
  - Ára van → Sebesség



Folyamatkezelés

- A folyamat egy olyan program ami éppen fut
- Program alapból passzív entitás → Elindul → Aktív entitás
- Folyamatnak szüksége van
    - CPU, Memory, I/O
        - Ezeket a Kernel végzi
    - Kezdő adat
- Folyamat végezte után fel kell szabadítani minden felszabadítható erőforrást
- Utasításszámláló
    - Betöltött program melyik részén járunk
    - Kezdetben egyszerre egy folyamat megy, és a folyamatokat szekvenciálisan hajtjuk végre
        - Ha van egyszerre több folyamat
            - Közöttük kommunikáció/szinkronizáció megvalósítása
            - Erőforráselosztás több folyamat között
- Folyamatmenedzsment fontos része → Holtpont /Deadlock elkerülése
    - Két folyamat vár egymásra, ezért soha nem fog előre haladni

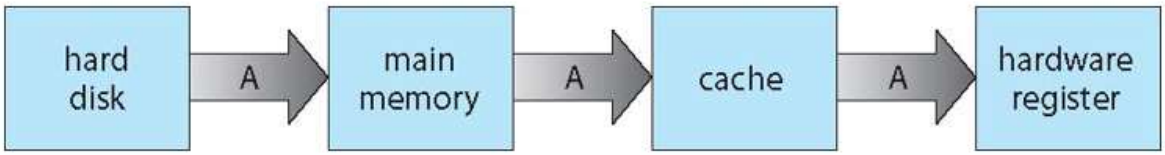
Memória Kezelés

- Egy program KIVÉGZÉSÉHEZ a programnak a memóriában kell lennie, a Neumann elvek szerint
- Az összes adatnak ami a programnak kell, a memóriában kell lennie
- Memória kezelés dönti el
    - Mikor mi van a memóriában
        - Ezzel optimalizálja a CPU kihasználást, és a feedbacket a user felé
- Először fix területeket allokáltak
- Alapvető allokált memória → Verem
- Azon túli memória → Halom
- Minden allokált memóriát program végeztével fel kell szabadítani
- Az Oprendszer nem mindenhol tud segíteni, ha egy program nem deallokál memóriát az szívás lehet

Storage kezelés

- Alapvető logikai egység → File
- Filekezelés központi kérdés
- Minden fájl a népszerű oprendszereknek
    - Egér, billentyű
    - Folyamat is fájl/mappa
    - Így egyszerűbb programozóknak → Mindent lehet olvasni, írni bele
- Filerendszer
    - Blokk méretének megadása

- Másolás, törlés, létrehozás
- Adatmozgatás disk-ről regiszterekbe



Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

I/O alrendszer feladata

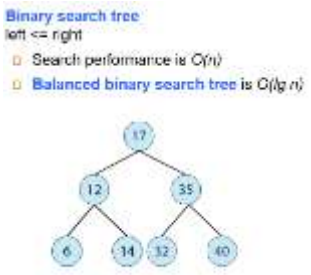
- I/O memory management
- Általános eszközkezelés interface
- Driverek harverekhez

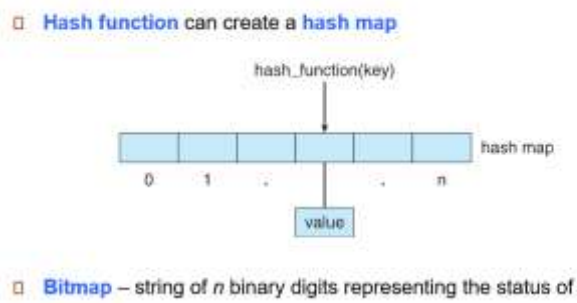
Védelem és biztonság

- Védelem
  - Mechanizmusok, melyek segítségével az erőforráshoz való hozzáférést tudja szabályozni
  - User ID, Group ID, Privilégiumok
  - „Belső védelem”
- Biztonság
  - Kívülről érkező támadás/behatolás elleni védelem

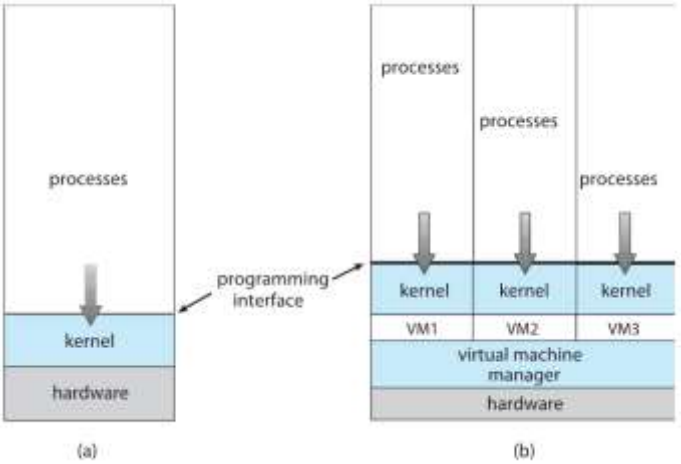
Kernel Adatstrukturák

- Listák
  - Szimpla, Dupla, körkörös
- fák, táblázatok
- Binfa
- Hash Map





Computing Environments



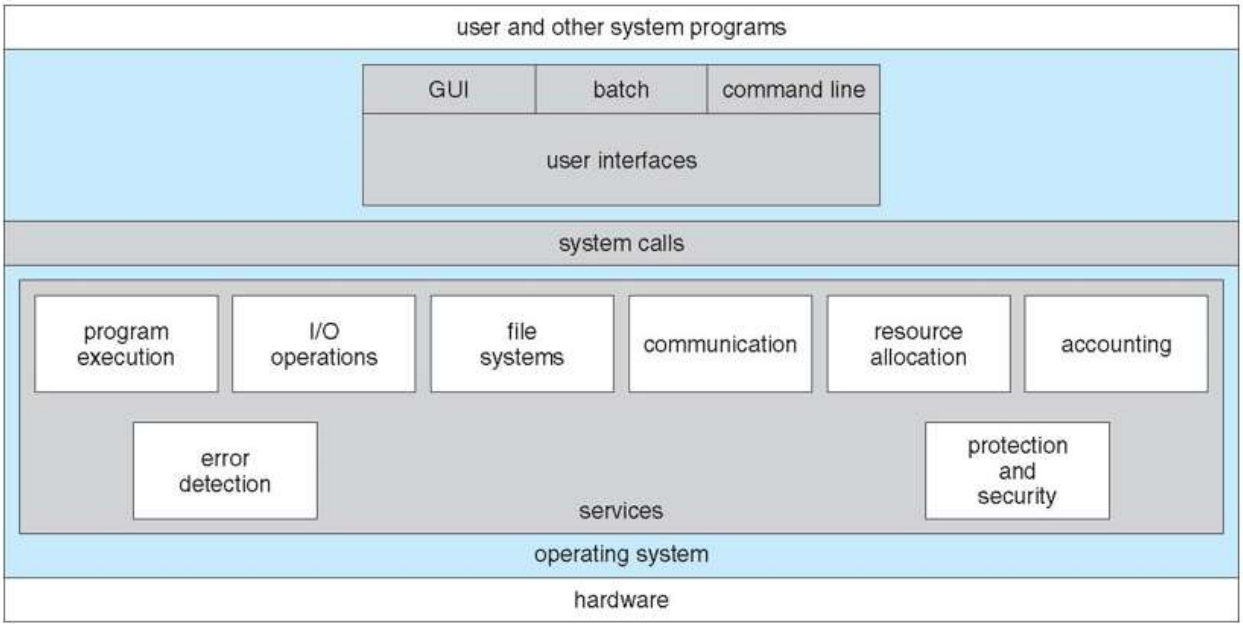
- Csak virtualizációról beszélt Adamkó
- Virtualizáció
  - Erőforráskezelés miatt lett rá igény
  - OS az OS-ben
  - VMM → Virtual Machine Manager
- Valós idejű rendszerek
  - Minden lépésre jön valami válasz, nincs olyan hogy „lefagy”
  - Például egy gyártósor, holdraszállás, bármi ilyen nagyon fontos dolog
  - Business szférában vannak jelen

Chapter 2

Operációs Rendszer -Struktúrák

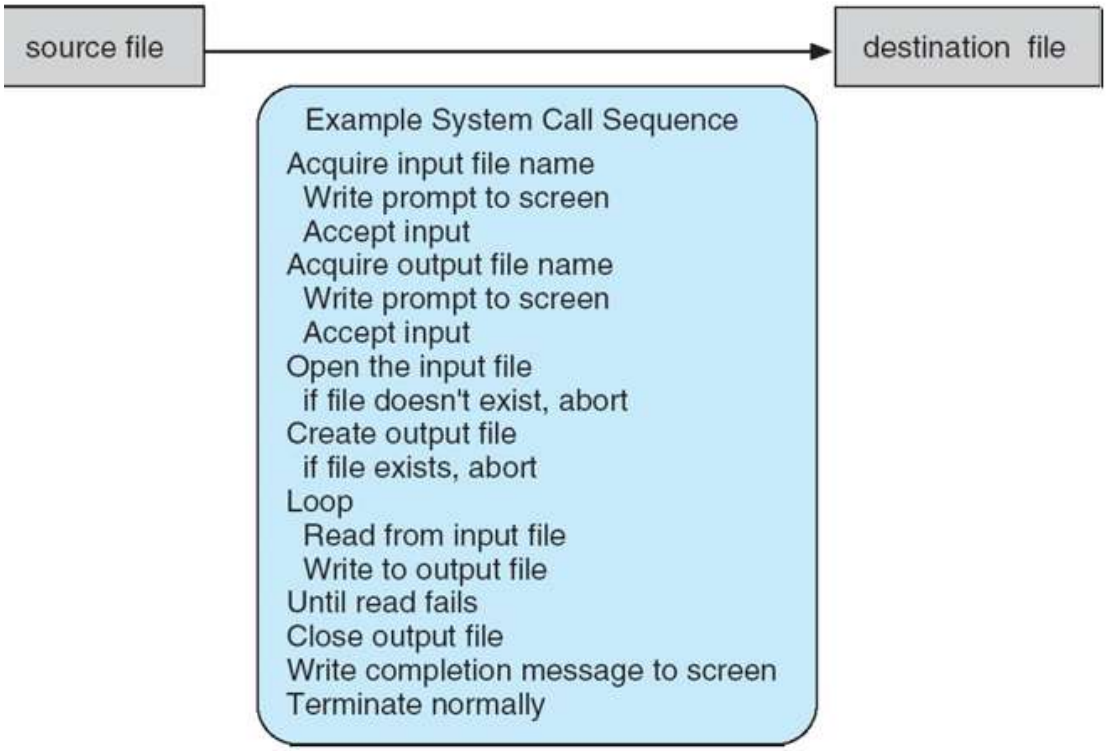
Operációs rendszer szolgáltatásai, feladatai

- UI-User Interface
  - CLI → Command interpreter
    - Néha a kernelbe építve
    - shellek
  - GUI
    - Felhasználóbarát
  - Batch
- Programvégrehajtás
- I/O műveletek
- Filerendszerhasználat
- Kommunikáció lehetővé tétele folyamatok között
  - Helyben vagy hálózaton
- Error Detection, hibakeresés → Debugging
- Erőforrás kezelés/allokálás
- Nyomkövetés → A felhasználó mit, használ, mennyi erőforrást, stb.
- Védelem és biztonság

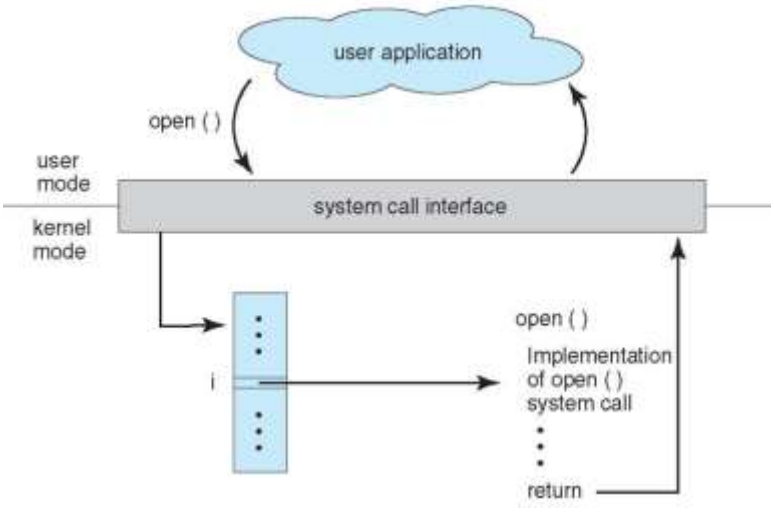


Rendszerhívások

- Általában C vagy C++
- Egy API-n keresztül éred el
  - Application Programming Interface használata közvetlen rendszerhívás helyett
  - Win 32
  - POSIX
  - Java API
- Példa egy másolás rendszerhívására



- Rendszerhívás Implementációja
  - API-n keresztül, az a felülete
    - A felhasználó soha nem éri el az alsó szintet közvetlenül
    - User Mode → Rendszerhívás → Kernel Mód
    - Programozó nem kommunikál közvetlenül a vassal/hardverrel
  - DOS-os időszak → Számítástechnika tévedése



- Rendszerhívás típusok

  create process, terminate process
  end, abort
  load, execute
  get process attributes, set process attributes
  wait for time
  wait event, signal event
  allocate and free memory
  Dump memory if error
  Debugger for determining **bugs, single step** execution
  Locks for managing access to shared data between processes

- +File és eszközkezelés

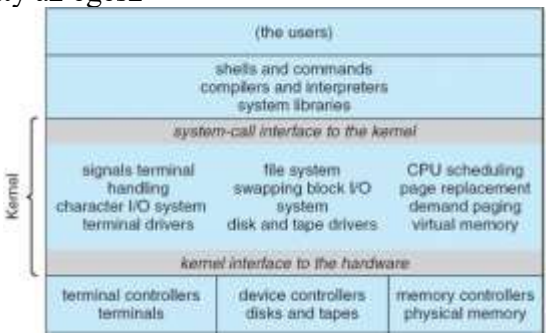
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

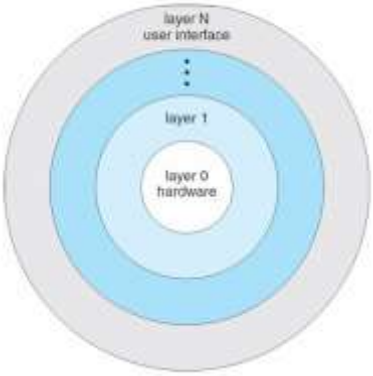
Rendszerprogramok

- Adamkó nem nagyon beszélt róla előadáson, Chapter 2 2.28-2.32 akit érdekel

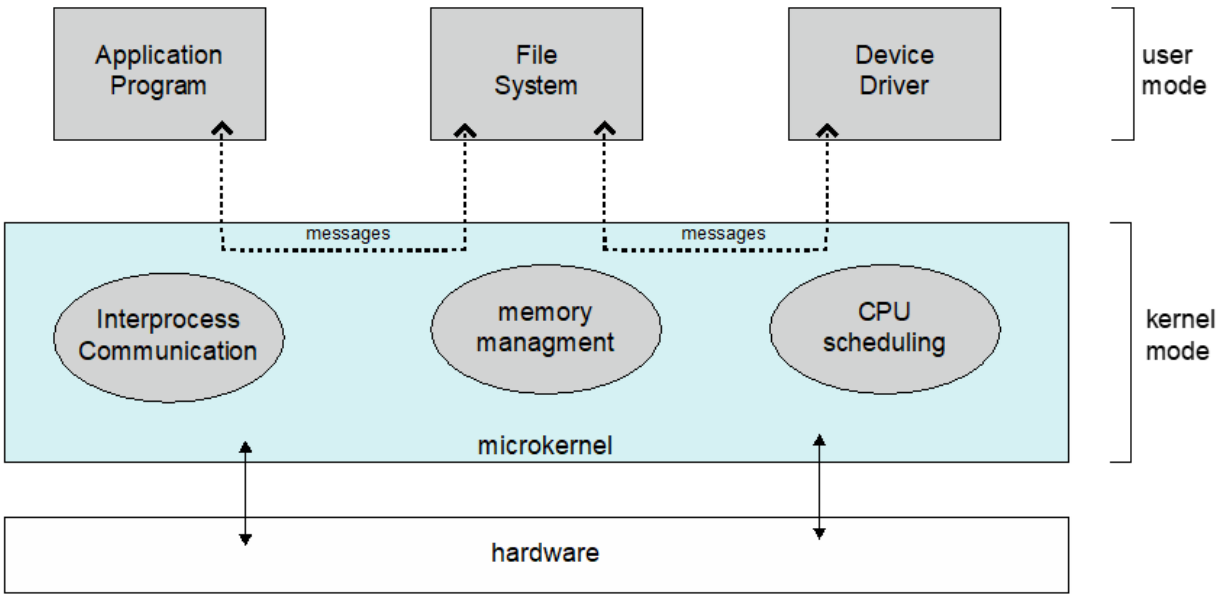
Operációs rendszerek felépítése, design és implementáció

- Meg kell határozni célokat, specifikációkat
- Függ hardvertől, rendszer típustól
- Célok
    - User Goals
        - Egyszerű, kényelmes, könnyen tanulható, megbízható, gyors, biztonságos
    - System Goals
        - Könnyű legyen a designolni, implementálni, fenntartani
        - Rugalmas, megbízható, hibamentes, hatékony
    - Ne úgy mint a DOS
- Két fontos alapelv, melyet el kell különíteni
    - Mit csináljon?
    - Hogyan csinálja?
        - El van rejtve a felhasználótól
- Minden program alapja az oprendszereknél kezdődött
- Nyelvek
    - Régen → assembly
    - Ma → C , C++
    - Az alapok → Assembly
    - Oprendszer főbb részei → C
    - Rendszerprogramok → C, C++, PERL, python, shell script
- Oprendszer Struktúra
    - Egy általános OS egy nagyon nagy program
    - MS-DOS
        - Szimpla
        - Nincs réteg rendszerhívásoknak, közvetlen elérés
        - Egyidejűleg 1 folyamat → 640 Kb memória
        - Legnagyobb szabadság
    - Unix
        - Van réteg a rendszerhívásoknak → Dual mode
            - Kernellel kommunikál
                - 2 interface
                    - Rendszerhívás
                    - Eszközkommunikációs réteg
        - Monolitikus Kernel
            - Ha a kernel valamely része rotty → Rotty az egész
            - All in One megoldás





- Rétegzett architektúra
  - N darab réteg
    - User réteg → Vezérlő réteg → Hardware
  - Seperation of concerns → Felelősségkörök szétválasztása
  - Minden réteg feladata más, csak a sajátjával foglalkozik
  - Legfelső réteg a user interface
  - Nem foglalkozik egyik réteg sem a felette levő réteggel
  -
- Microkernel Architektúra
  - Minél kevesebb dolog Kernel módban
  - Megpróbál a lehető legtöbb folyamatot kiszervezni user módba
    - Ha az elhal, a Kernel működik tovább
  - Könnyű hozzá új funkciót hozzáadni
  - Szolgálatásként képzeljük el a legtöbb dolgot user módban
  - Kommunikáció user modulok között → Message passing
  - Megbízhatóbb
  - Biztonságosabb
  - Cserébe lassabb → Kernel és User mód közötti kommunikáció
  - Példa → Böngészők, MacOs
    - Új funkció hozzáadása → Bővítmény



- Modulok
  - Betölthető kernel modulok → Monolitikus, de moduláris rugalmasabb
  - Linux Disztribúciók, Linux Kernel, Solaris
  - Objektumorientált hozzáállás
  - Olyan, mint a rétegzett, csak rugalmasabb

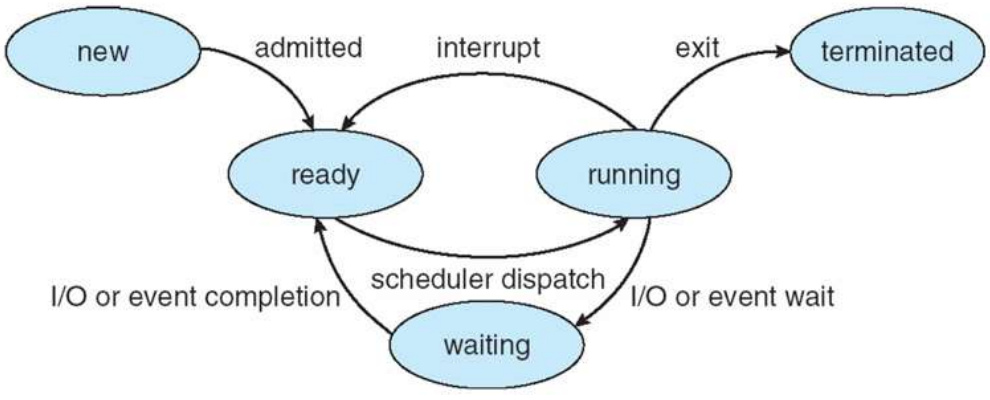
    - Hybrid rendszerek
        - Windows kernel
            - Főleg monolitikus, kis MikroKernel
        - MacOs
- Oprendszer Debugging
    - Nagyon fontos, no paraszt debug
    - Naplófájlok, eseménynapló

Chapter 3 – 5. előadás

Folyamatok

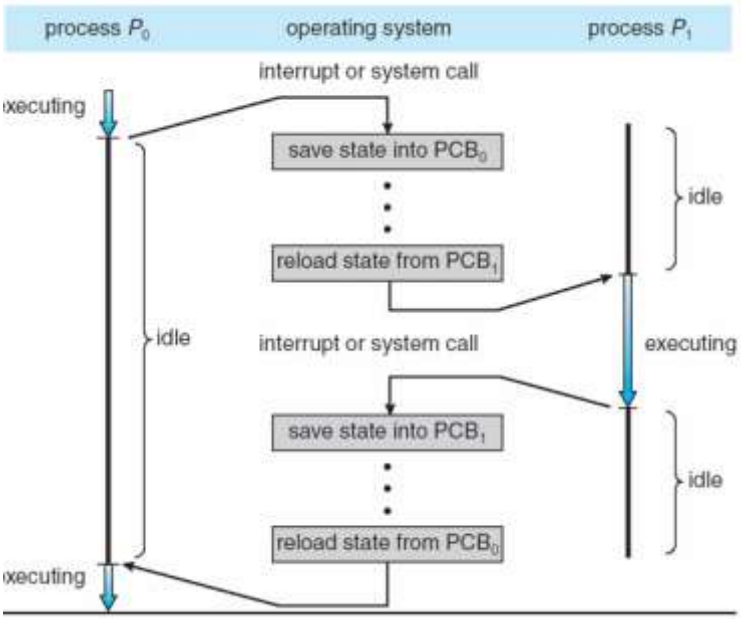
Folyamatok koncepciója

- Batch systems – jobs
- Jobs=Process=Folyamat
- Folyamat
  - Egy KIVÉGZÉS-ben lévő program
  - A KIVÉGZÉS szekvenciálisan kell történjen
  - A folyamat egy aktív entitású programa
  - Egy program lehet egyszerre több folyamat is
- Folyamat részei
  - Utasítások, kódszegmens – Text
  - Utasításszámláló
  - 3 féle memóriaterület
    - Adatszegmens → Van mérete
      - Globális változók
    - Stack/verem
      - Lokális változók, függvényparaméterek
      - Hívási lánc
    - Heap/halom
      - Dinamikusan allokál memóriát a programhoz futási idő alatt
  - Regiszterek
  - Fordítók nem használják ki az oprendszer által nyújtott lehetőségeket, max 1 Mb az alap allokált memória → stack → Legtöbbször elég
    - 32 bit → Max 4 Gb memória
    - 64 bit → Max 16 Gb memória → Heap/Halom
    - Unix, Linux alatt Max Stack méret → 8 Mb → Kernellel lehet változtatni
- Folyamat állapotai
  - Létrejövés/New
  - Futó/Running
  - Várakozó/Waiting
  - Készen/Ready
  - Vége/Terminated



Process Control Block – PCB

- Folyamat állapota
- Utasításszámláló
- Folyamatazonosító → Process Number
- CPU regiszterek
- Folyamatütemező
- Memóriakezelő/limit, allokált memória
- Nyomkövetési dolgok, Időlimit a program végrehajtására
- I/O állapot, allokált I/O eszközök
- A kernelnek ez a blokk a/egy folyamat



- Másodpercenként 8-9 folyamatváltás
- Multiprogramozottság ára → Idő
    - Folyamat állapotának elmentése → Másik folyamat betöltése → Megszakítás Másik folyamat végrehajtása → Elmentése → eredeti folyamat visszatöltése

Szálak – Threads

- Eddig a PCB csak szekvenciális végrehajtásra volt képes → Mert 1 utasításszámláló
    - Most több utasításszámláló → Több „szálon futhatnak” folyamatok

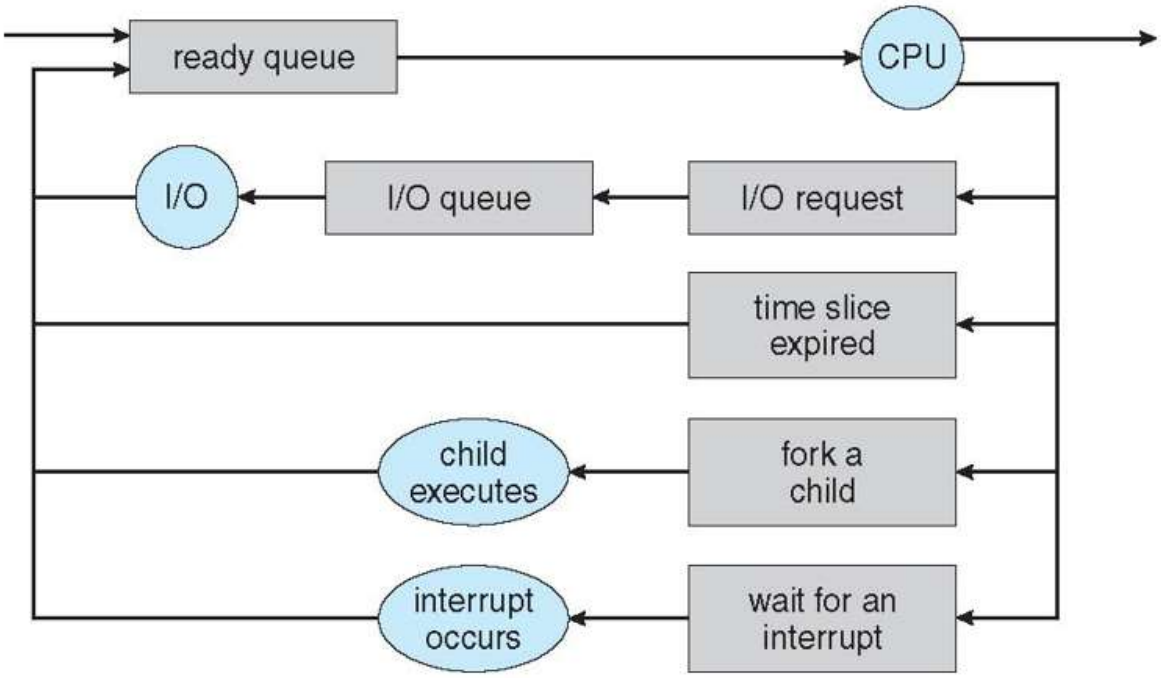
Folyamathierarchia

- Gyökér-Szülő-Gyermek → Fa alakú → Gyökér +levelek
- Ősfolyamat → Elindítja az összes többi folyamatot
- 1 folyamatnak 1 szülőfolyamat, bármennyi gyerekfolyamat
- Fork → Folyamat elindít egy másik folyamatot
    - Megvárja a szülő folyamat a gyermek folyamat lefutását

- Ha a szülő folyamat leáll → Ki kell gyalulni az összes alatta lévő gyermek folyamatot
    - Orphan/árva folyamatok → Keresni kell új szülőt →Gyökérfolyamat → init → Időnként elindít egy wait rendszerhívást, az árva folyamatok lefutnak, vagy ha nem kigyalulja a megmaradt folyamatokat a kernel
    - Egész fának valahogy el kell tűnnie
- Ha a szülő nem foglalkozik azzal hogy a gyerek „kész van”, azaz nem hívja meg a wait-et, a folyamatból Zombie folyamat lesz
    - Leadja az erőforrást, terminált a folyamat, de megmarada a PCB blokkja

Folyamatütemezés

- Cél → Maximalizálni a gépidőt, folyamatok közötti váltás ennek elérése érdekében
- Ütemező → Választ melyik folyamatot végezze el következőleg
- Sorban vannak a programok → Versengenek az erőforrásért
    - Ütemező láncolt listából választ
- Kernel listákat kezel, folyamatokból fát épít
- Ütemező sorok
    - Job queue
        - Az összes folyamat a rendszerben
    - Ready queue
        - Az összes folyamat, ami készen áll a KIVÉGZÉSRE
    - Device queue
        - A folyamatok amik I/O eszközre várnak
    - A folyamatok ezek között lépkednek



- Context switch
    - „A” folyamatról váltunk B folyamatra
    - A fut → A ment → B betöltődik → B fut → B ment →A betöltődik → A fut

Ütemezők

- Rövidtávú
    - Laptop, telefon, PC → Round Robin
    - Kiválasztja melyik programot futassa
    - Néha az egyetlen ütemező a rendszerben
    - Ő a sebesség
- Hosszútávó ütemező
    - Lassú → Akár percek
    - Kiválasztja melyik program kerüljön ready queue-ba
    - PCB sokkal nagyobb ablak
    - Folyamatosan becsülgetnek kinek lenne célszerű adni gépidőt
        - Folyamatok kategorizálása
            - I/O vagy CPU intenzív
            - Hogy maximalizálja mindkettőt
    - Folyamatos terhelés
    - Régi mainframek

5. Előadás vége

Böngésző példák

- Van egy pár böngésző, ami egy folyamatként működik
- Chrome
    - Minden lap egy külön folyamat amit nyit

Folyamatok lehetnek

- Különálló, független
- Kooperáló
    - Hatással lehetnek rá más folyamatok, és hatással lehet más folyamatokra
    - Okok erre
        - Információ megosztás
        - Számolás felgyorsítása
        - Modularitás
        - Kényelem
    - InterProcess communication → IPC
        - Osztott memória
        - Message Passing

- Producer- Consumer Process
    - Producer → Előállít információt
    - Consumer → Megkapja az információt
        - Van egy buffer
            - Limit nélküli
            - Limitált

Interprocess Communication

- Osztott memória
    - Egy adott memória rész meg van osztva folyamatok között
    - Ez a kommunikáció felhasználói folyamatok irányítása alatt van, nem az OS irányítása alatt
    - Fő feladat → Egy mechanizmus arra, hogy felhasználói folyamatok tudjanak szinkronizálni egymás között amikor osztott memóriát használnak
- Message Passing → Üzenet átadás
    - Egy másik mód arra, hogy folyamatok egymás között tudjanak kommunikálni
    - Osztott változókkal teszik meg ezt a kommunikációt
    - Send – Receive → Communication link a folyamatok között
    - Message size → Fix, vagy változó
    - Communication link implementációk
        - Fizikai
            - Osztott memória
            - Hardver busz
            - Hálózat
        - Logikai
            - Közvetlen, vagy közvetett
            - Közvetett
                - Mailboxok használata → Portok
                - Minden mailboxnak saját ID
                - Csak akkor van link két folyamat között, ha egy mailboxhoz tartoznak
                - Egy li
            - Szinkronizált, nem szinkornizált
            - Automatic or explicit buffering
    - Közvetlen link
        - Send, Receive
        - Linkek automatikusan jönnek létre
        - Egy linkhez pontosan 2 folyamat van, es 2 folyamathoz 1 link

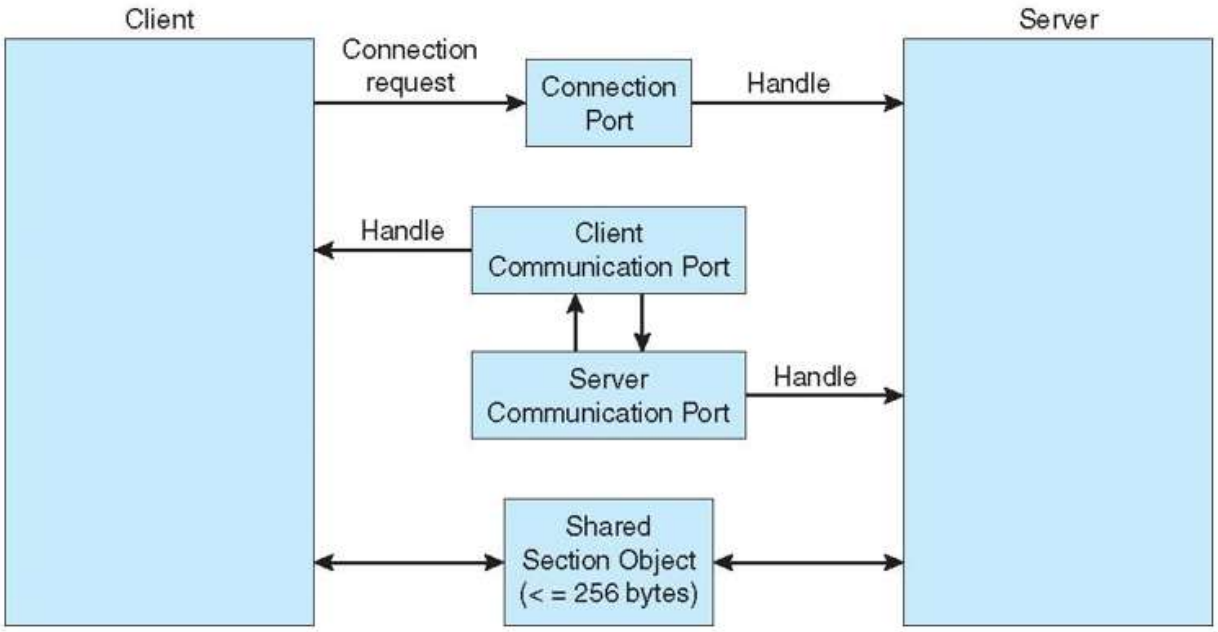
- Közvetett link
  - Mailboxok használata → Portok
    - Minden Mailboxnak saját ID
    - Csak akkor tud két folyamat kommunikálni ha ugyanahhoz a mailboxhoz csatlakoznak
  - Tulajdonságai
    - Csak akkor jöhet létre, ha egy a mailboxuk
    - Egy link több folyamathoz is tartozhat
    - Minden folyamatpárhoz akár több link is tartozhat
    - Lehet Egy vagy kétirányú
  - Műveletek mailboxokkal
    - Új létrehozása
    - Üzenet fogadása, küldése
    - Mailbox törlése
    - Alapművelet→ send, receive
  - Probléma
    - P1 küldi üzenetet mailboxra, amihez P2 és P3 csatlakozik
      - Ki kapja meg az üzenetet?
    - Megoldás
      - Egy link max 2 folyamathoz tartozhat
      - Egyszerre egy folyamat hajthat végre receive operationt
      - Kiválasztunk önkényesen egy receivert
        - Sender megkapja ki kapta az üzenetet

- Szinkronizáció
  - Message Passing lehet Blocking vagy Non-Blocking
  - Blocking → Szinkronizált
    - Blocking send → a küldő blokkolva van, amíg nem kapja meg a receiver
    - Blocking Receive → A receiver blokkolva van, amíg nem elérhető az üzenet
  - Non – Blocking → Aszinkronizált
    - Non-Blocking send → A küldő elküldi az üzenetet, és megy is tovább, nem vár
    - Non-Blocking receive
      - A receiver vagy érvényes üzenetet kap
      - Vagy null üzenetet
  - Ha send és receive is blokkolt akkor → Randevú
- Buffering
  - Üzenetet sora (queue)
  - Implementáció háromféle lehet
    - Zero capacity → Nincs egy üzenetsor → Küldőnek várnia kell a receiverre
    - Bounded capacity → Véges számú üzenet tárolása
      - Ha tele van a sender vár

- Unbounded capacity → Végtelen számú üzenet tárolása
  - Sender soha nem vár

IPC rendszer példák

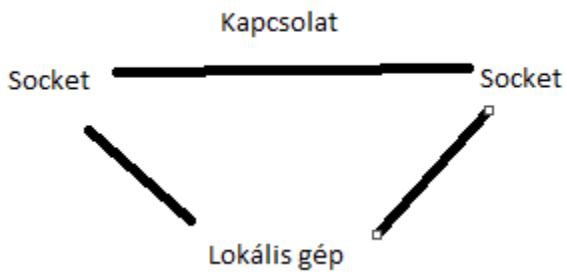
- POSIX osztott memória
  - Mach
- Windows

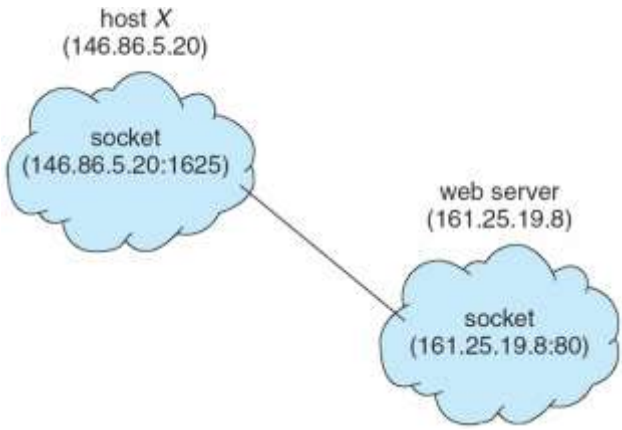


6. Előadás vége

Socket

- Unix → Miért legyen minden lokálisan?
  - Erre a kérdésre mindig az a válasz, hogy nem kell annak lennie
  - Host → Socket → Kapcsolat → Socket → Webserver
  - Lokálisan is lehet két socket között kapcsolat, azaz lehet kliens a saját gép is
  - Portok használata





Remote Procedure Calls - RPC

- Egy nyelv, amiben le lehet írni a végződéseket, socketeket
- Socket interface újragondolva
- Szükség van egy third party szoftverre → Az adott nyelven elérhetővé tegye a rendszerhívásokat
    - ORB – Object Request Broker
        - Segít megtalálni a távoli helyen a végpontot
- Socket interface egy absztrakciója
- Windowson – Matchmaker
- IPC → Gépen belül
- RPC → Gépen kívül

Csővezetékek

- Egy folyamat standard outputját összedrótozza egy folyamat standard inputjával
- Mivel minden fájl, ezek is
    - Három eszközfájl lényegében
- Fd → File descriptor
- Névtelen és nevesített csővezeték

7. előadás eleje

Chapter 4

Szálak – 7. előadás

Szálak koncepciója

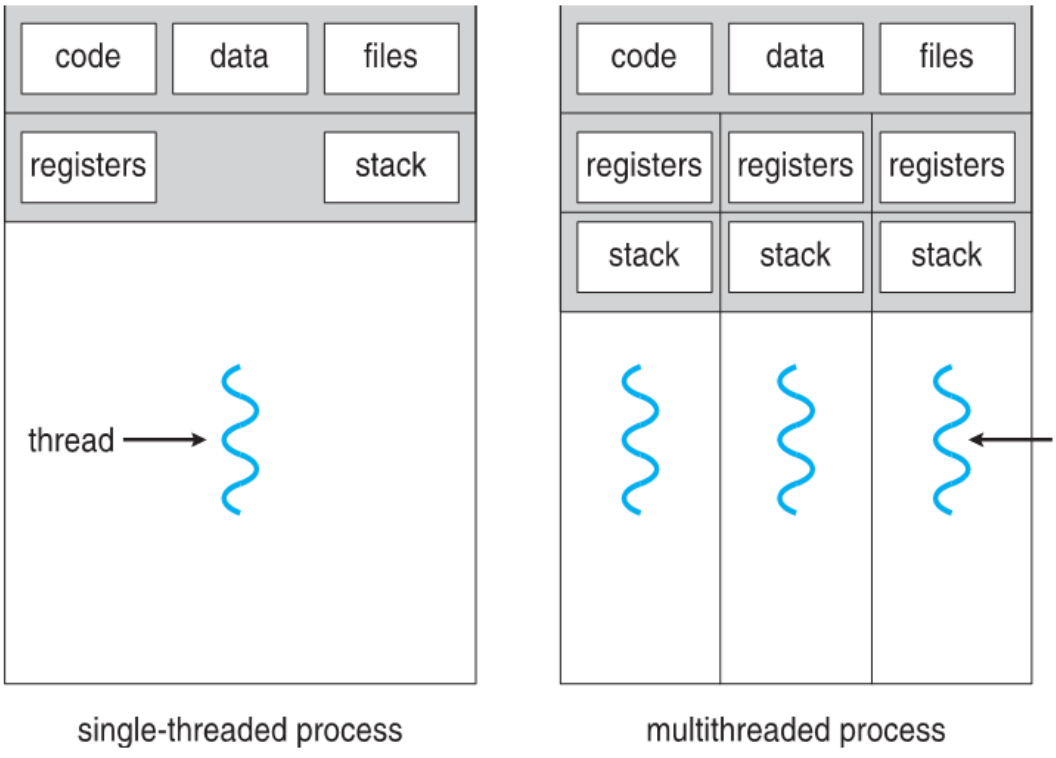
- Szimulált párhuzamosság lényegében
- Egy fizikai processzoron belül több logikai egység
- Ne legyen felára a multiprogramozottságnak → Megspórolja a Context Switchet
- Ütemező → Nagyon jó, de elviszi gépidő 15-17%-át
- Szálak alkalmazása sincs ingyen, de töredékére esik az elvesztegetett gépidő
- Előnyök
    - Reszponzívabb
    - Erőforrás megosztás
    - Hatékonyabb, mint a context switch
    - skálázható

Multicore Programming → Valódi párhuzamosság

- Több fizikai mag a processzorban
- Párhuzamosság nem egyenlő konkurenciával
- Nem lineáris a növekedés, 2 mag nem egyenlő 2 külön processzor, nincs 2 ALU
    - Amdahl törvénye

Szálakban gondolkozás

- Egy adaton több szeparált szolgáltatás
- Párhuzamosítás → Felismerése nagyon nehéz, mert az ember szekvenciálisan gondolkozik



User, vagy Kernel szál?

- Posix Pthreads
- Windows threads
- Java Threads
- Kernel Threads
- Many to One → Több felhasználói szálhoz egy kernel szál
    - Not good, nem használják már
    - Solaris Green Threads
- One to one → Egy felhasználói szálhoz egy Kernel Szál
    - Mai szálkezelés
    - Windows, Linux, Solaris 9
- Many to Many
    - We no likey
    - Windows sem szereti, de van Thread Fiber
    - Solaris 9 előtt

Threadkönyvtárak → Programozóknak API-kat nyújt szálkezeléshez

Jelkezelés

- POSIX-es megvalósítás UNIX-ban
- Jelzik egy folyamatnak, hogy történt valami → Eseményjelző
    - A folyamat ezekre az eseményekre reagálhat
- Például a Ctrl+C
    - Ha erre reagál → Békés rotty
    - Ha nem reagál erre a Kernelnek küld jelet → Kernel kilövi → Nem békés rotty
- Minden jelnek van egy alapvető kezelője, ami lefut a jel adása után
    - Ezt felülírhatja egy user defined kezelő
- Kill 15 → Békés → Légyszi állj le köszi csá
- Kernel kill → Nem annyira békés → Először lő aztán se kérdez

Thread Local Storage

- Lehetőséget ad minden szálnak, hogy meglegyen a saját adatpéldánya/memóriája
    - Látható függvényhívások között is, nem olyan mint egy helyi változó
    - Hasonlít a static váltzóhoz
        - Static → Fix a memóriacíme
- Thread vezérlés hasonlít a folyamatokhoz

Chapter 5

Folyamat szinkronizáció – 8. előadás

Alapvetések eddig

- Producer – Consumer koncepció
  - Működhet, ha időben nem közel történik két dolog
- Folyamatok mehetnek konkurensen
  - Meg lehet szakítani, részben elvégezve a folyamatot
- Konkurens elérés osztott adathoz adat inkonzisztenciához vezethet
  - Későbbiekben erre lesz példa mi is ez az inkonzisztencia
  - Módok kellenek arra, hogy elkerüljük ezt

Race Condition - Versenyhelyzet

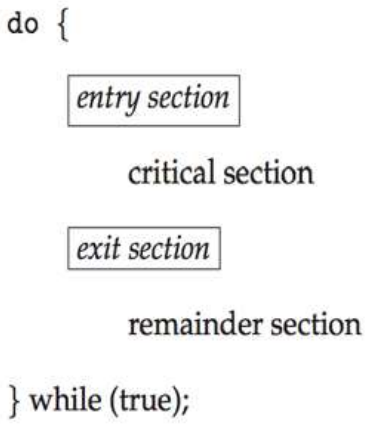
- Consider this execution interleaving with "count = 5" initially:

```
S0: producer execute register1 = counter          {register1 = 5}
S1: producer execute register1 = register1 + 1    {register1 = 6}
S2: consumer execute register2 = counter          {register2 = 5}
S3: consumer execute register2 = register2 - 1    {register2 = 4}
S4: producer execute counter = register1          {counter = 6 }
S5: consumer execute counter = register2          {counter = 4}
```

- Példa az adat inkonzisztenciára
  - Mivel a producer és consumer bármikor megszakítható, ezért hiba lehet, ahogy itt is, keresztül húzza egymást a producer és consumer
  - Ha mondjuk így egy tömb x.-dik elemét keressük, és egy 10 elemű tömbnél a counter 15, az még rosszabb, mert nem vesszük észre a hibát, és kiírhat valami memória szemetet
- Kell valami megoldás, a versenyhelyzet elkerülésére → Kritikus szakasz

Kritikus szakasz – Critical Selection

- A megoldás a versenyhelyzetre
- Bevezetünk egy fogalmat, a kritikus szakaszt → Minden folyamatnak van ilyen része
  - Közös változók változtatása, fájlírás, egy táblázat frissítése stb.
  - Amikor egy folyamat a kritikus szakaszban van, akkor más folyamat nem lehet a kritikus szakaszában
- Alapelképzelés



- Kritikus szakasz védelme
  - Kölcsönös Kizárás
    - Egyszerre csak egy folyamat lehet kritikus szakaszában
  - Haladás
    - Ne lehessen egy folyamat kritikus szakaszát végtelenségig halasztani
    - Ha nincs egy folyamat se kritikus szakaszba, szerzünk egyet ami abba akar menni
  - Ütemezett várakozás
    - Bármely program véges időn belül be tudjon lépni a kritikus szakaszába
    - Van egy limit egy folyamat mennyiszer kérheti a kritikus szakaszba való belépést
- Kritikus szakasz kezelése OS-ben
  - A kernel módban történik a kritikus szakasz
  - Kétféle hozzáállás
    - Nem preemptív
      - Ha folyamat Kernel módban van, nem lehet tőle elvenni a gépidőt
      - Versenyhelyzet lényegében nem fordulhat elő
      - Ilyen nem nagyon van
    - Preemptív
      - Nem számít milyen módban van, ha ütemező el akarja elveszi tőle a gépidőt
      - Összes modern OS
      - Gyorsabb, reszponzívabb → De megszakadhat bármikor
        - Mi legyen, ha kritikus szakaszban veszi el?
  - Peterson megoldása
    - Load és Store atomi utasítások, nem lehet megszakítani
    - Két közös változó folyamatok között
      - Int turn
        - Megmondja kinek a „köre” a kritikus szakaszba lépni
      - Boolean flag[i] →alapból hamis
        - Megmondja egy folyamat készen áll-e belépni a kritikus szakaszba
        - Flag[i]=true → I.-dik folyamat készen áll
    - Teljesül a
      - kölcsönös kizárás
      - Haladás
      - Ütemezett várakozás

```
do {
        flag[i] = true;
        turn = j;
        while (flag[j] && turn = = j);
                critical section
        flag[i] = false;
                remainder section
} while (true);
```

Hardveres szinkronizáció

```
do {
      acquire lock
              critical section
      release lock
              remainder section
} while (TRUE);
```

- Locking/Zárolás
  - Belép kritikus szakaszba → Megkéri a zárat → Végez → Elengedi a zárat
  - Hardveres támogatás kritikus szakasz implementációjához
  - Atomi utasítás → Nem lehet megszakítani, mert hardverszintű
    - Letiltja a CPU az ütemezőt
  - 2 folyamatra biztosan működik
- Compare and Swap
  - Ugyanaz, mint a Lock, csak van benne egy if, hogy olyan állapotban van-e a zár mint vártam
  - 2 folyamatra biztosan működik

```
do {
     while (compare_and_swap(&lock, 0, 1) != 0)
     ; /* do nothing */
   /* critical section */
lock = 0;
   /* remainder section */
} while (true);
```

- Megoldás több folyamatra → Waiting
  - Van egy waiting segéd görbe
  - Egy folyamat kritikus szakaszban → Letárolja az össze dolgát → Exit section → Végigmegy folyamatosan egy tömbön, amiben azt tároljuk melyik folyamat várakozik éppen hogy kritikus szakaszba lépjen
    - Ha van, kijelöli a következőt
    - Ha nincs, kinyitja a zárat, és a következő folyamat amelyik majd egyszer kritikusba akar lépni beléphet azonnal
- Eddig voltak Hardveres Lockok, innentől szoftveres

Szoftveres szinkronizáció Megoldások

- A hardveres megoldások komplikáltak, és nem minden programozó számára elérhetőek
- Mutex Lock
  - Kritikus szakasz
    - Kezdete → Acquire lock
    - Vége → Release Lock
  - Egy boolean változó, hogy nyitott-e a zár
  - Probléma→ Összes folyamat be akar lépni → Busy waiting
    - Gépidő égetés
    - Spinlock

```
acquire() {
    while (!available)
       ; /* busy wait */
    available = false;;
}
release() {
    available = true;
}
do {
  acquire lock
    critical section
  release lock
    remainder section
} while (true);
```

- Semaphore
  - Egy szofisztikáltabb mutex lock
  - Egy erőforráshoz több folyamat
  - S → Egész szám mennyi folyamat lehet egyszerre kritikus szakaszban
    - Ha nulla, akkor maximumon van jelenleg futó kritikus folyamatok száma
  - Két atomi utasítás
    - Wait
      - Meg akarja várni amíg erőforrást kap
    - Signal
      - Le akarja adni az erőforrást

```
wait(S) {
    while (S <= 0)
        ; // busy wait
    S--;
}
```

```
signal(S) {
    S++;
}
```

  - Kétféle semaphore
    - Bináris semaphor
      - S =0 vagy 1 → Lényegében egy Mutex Lock
    - Számoló semaphor
      - S értéke tetszőleges
  - Sok problémát megold
  - Garantálni kell, hogy egy folyamat nem hajthatja végre egyszerre a wait-et és signal-t
    - Wait és Signal a kritikus szakaszban
    - Így viszont keletkezhet Busy waiting
      - Kevés folyamatnál nem gáz
    - Alkalmazások sok időt tölthetnek a kritikus szakaszban, szóval ezt we no likey
  - Semaphore implementáció Busy Waiting nélkül
    - Minden semaphore-nak van egy waiting sora (queue)
      - Ez tartalmaz egy értéket
      - És egy mutatót a sor következő rekordjára
    - Block művelet
      - A folyamatot ami meghívta a megfelelő waiting queue-ba teszi
    - Wakeup művelet
      - Kivesz egy folyamatot a waiting q ueue-ból, beteszi ready-be

```
wait(semaphore *S) {
    S->value--;
    if (S->value < 0) {
        add this process to S->list;
        block();
    }
}

signal(semaphore *S) {
    S->value++;
    if (S->value <= 0) {
        remove a process P from S->list;
        wakeup(P);
    }
}
```

Deadlock/Holtpont és Starvation/Éhezés

- Deadlock
  - Kettő vagy több folyamat határozatlan ideig vár egy eseményre, ami az egyik másik várakozó folyamat által keletkezhetne, tehát nem jutunk sehova

- Starvation
  - Soha nem tud kijutni a semaphore sorból ahova beragadt
- Megoldások → Prioritások

Klasszikus szinkronizáció problémák

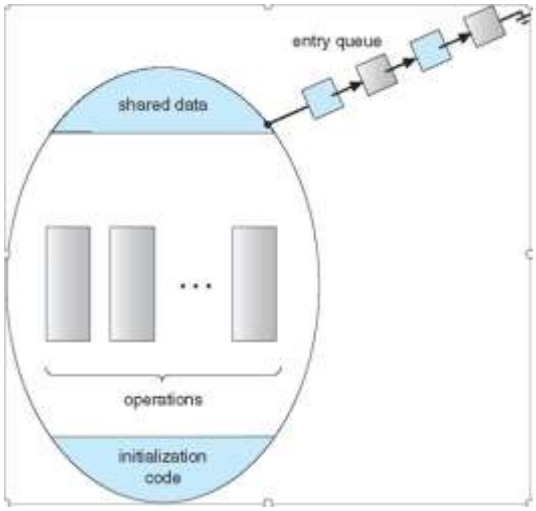
- Bounded Buffer
- Readers Writers Problem

Dining Philosphers Problem – A kajáló filozófusok problémája

- 5-en ülnek egy körasztal körül
- Két állapota van a filozófusoknak → Esznek, vagy gondolkodnak
- Nem foglalkoznak a mellettük ülővel
- Időnként enni akarnak, amihez fel kell venni a jobb és bal oldalukról egy-egy evőpálcikát
- Megosztott adat
  - Egy tál rizs az asztal közepén
  - 5 darab evőpálcika
- Először ha enni akarnak felveszik a jobb oldalukról a pálcikát, majd a bal oldalról
  - Ha nincs egy oldalon pálcika várnak, hogy a mellettük ülő befejezze az evést
- Probléma → Mind az öten egyszerre akarnak venni
  - Mind felveszik az egyik oldalukról a pálcikát és várnak, hogy a másik oldalukon ülő befejezze a kajálást, mert ott nincs pálcika
  - Mind az öten egymásra várnak, és soha nem fognak előrébb jutni → Deadlock
- Megoldások
  - Max 4 filozófus ülhessen az asztalnál
  - Csak akkor vegyenek fel pálcikát ha egyszerre mind a kettő elérhető
    - Felvevés kritikus szakasz
  - Több evőpálcika

Monitorok

- Egy magas szintű absztraktció, ami lehetővé teszi kényelmesen és hatékonyan a folyamatok szinkornizációját
- Absztrakt adattípus
  - Belső változók csak a belső kódnak elérhetőek → Class lényegében
- Egyszerre csak egy folyamat lehet aktív a monitorban




```
monitor DiningPhilosophers
{
   enum { THINKING; HUNGRY, EATING) state [5] ;
   condition self [5];

   void pickup (int i) {
          state[i] = HUNGRY;
          test(i);
          if (state[i] != EATING) self[i].wait;
}

   void putdown (int i) {
          state[i] = THINKING;
                   // test left and right neighbors
           test((i + 4) % 5);
           test((i + 1) % 5);
}
```

Chapter 6

CPU ütemezés – 9. előadás

Alapvetések

- Gépidő és I/O égetés miatt jött a Multitasking ötlete
- Csakis multiprogramozottsággal érhető el optimális CPU kihasználtság
    - Ára van → Folyamatok közti váltás ideje
- Első 8 miliszekundomban van a folyamatok legintenzívebb része
    - Utána csak vár, nem effektív
    - Megoldás → 8 ms → Másik folyamat → 8 ms → Másik folyamat
    - Lényegében ez a CPU ütemezés

Folyamatok 4 állapota

- Waiting
- Ready tor un
- Running
- Terminated

Ütemező

- Rövidtávú ütemezőről beszélünk leginkább
    - Választ a ready queue-ban lévő folyamatok közül, és ad neki gépidőt
        - A ready queue sokféle sorrendben lehet
- Mit tehet az ütemező?
    - 1. Running → Waiting
    - 2. Running →Ready to run
    - 3. Waiting → Ready tor un
    - 4. Running → Terminated
- Nem preemptív ütemezőnél csak az 1. és 4. állapot van
- Preemptív ütemezésnél mind a 4, mivel át kell gondolni a
    - Közös adathozzáférést
    - Preemption kernel módban
    - Megszakítások fontos OS folyamatok közben
- Ma minden ütemező preemptív
    - Kapsz gépidőt, bármikor elvehető
    - Windows 95-nál váltott a Microsoft preemptívre
    - Nagyon fontos a reszponzivitás → A felhasználó türelmetlen
- Kell valami ami elvégzi a váltást két folyamat között, a context switchet
    - Elmentés, másik betöltése
    - Dispatcher/Diszpécser

Dispatcher

- Lehetővé teszi a CPU-nak a
    - Kontextus váltást
    - User mode váltást
    - User programban a megfelelő helyre ugrást, hogy újraindíthassa a programot
- Előző folyamat mentése, másik betöltése időbe kerül → Dispatch latency
    - A multiprogramozottság ára

Ütemező szempontjai

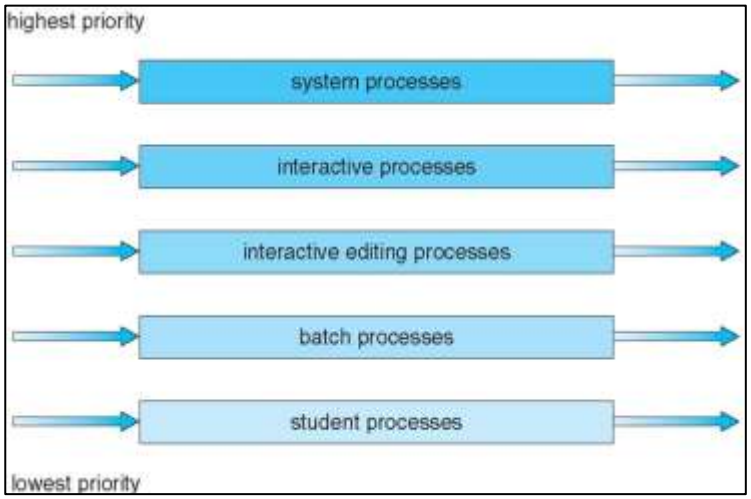
- Processzor kihasználtsága → Maximalizálni
- Kibocsátóképesség /Throughput → Maximalizálni
    - Mennyi ciklust végez el, mekkor az időablak
- Idő → Minimalizálni
    - Megfordulási → Egy adott folyamat KIVÉGZÉSE
    - Várakozási idő → Mennyit vár egy folyamat a ready queue-ban
    - Válasz idő → A kéréstől az első válaszig eltelt idő, nem feltétlenül output
- Mind egyszerre nem lehet tökéletes

Ütemező típusok

- First Come First Served – FCFS
    - Sorban, szekvenciálisan ahogy beérkeznek → Feltorlódás
    - Konvoj effektus → Egy rövid, gyors folyamat beragadva egy hosszú folyamat mögé
    - Magas várakozási idő
    - Lehet jó is, de nem igazán
- Shortest Job First
    - Legkevesebb időt igénybevevő feladatok először
    - Minimum átlag várakozási idő
    - Konvoj hatás lehetősége
    - Becsülgeti a következő CPU burst hosszát, aszerint osztja be
    -
    - Preemptív változat → Shortest remaining time first
        - We likey
- Priority Scheduling
    - Magas prio előre, alacsony hátra
    - Jó lehet, de probléma → Starvation
    - Starvation → Lehet az alacsony priojú feladatok soha nem futnak le
        - Megoldás → Aging → Idő elteltével a ready queue-ban lévő folyamatok prioja növekszik

- Round Robin
  - PC-knél ez a befutó
  - Bevezet egy időablakot → Időquantum → 10 ms < Quantum < 100 ms
    - Context Switch nagyjából 1 ms, tehát marad 9 ms minimum gépidő
    - Az 1 ms a multitasking ára
  - Amint lejár a quantum fogja a következő folyamatot

Multilevel queue



- Két ready queue-re bomlik
  - Háttérfolyamatok
  - Előtérben lévő folyamatok
  - Egy adott folyamat csak 1-ben lehet,
    ott permanensen
- Előtér
  - Ineraktív alapvetően
  - Round Robin
  - 80% gépidő
- Háttér
  - Szolgáltatások, rendszerfolyamatok
  - Nem interaktálnak a userrel
  - Bármi lehet FCFS/SJF
  - 20 % gépidő

Multilevel feedback queue

- 3 külön queue
  - Q0 → 8ms
  - Q1 → 16 ms
  - Q2 → FCFS
- Minden folyamat bekerül először Q0-ba, ha nem végez 8 ms alatt, megy tovább Q1-be, ha nem végez az az ottani +16 ms alatt sem, akkor megy Q2-be ahol sima FCFS van
- Ha elég egy folyamatnak Q1, akkor oda is tér majd vissza

Eddig csak folyamatütemezésről beszéltünk

Szálütemezés → Nice to have, elég a folyamat

- Same but different mint a folyamatkezelés
- User és Kernel level threadek

Multiple Processor Scheduling

- Több processzornál komplexebb az ütemezés
- CPU-k közötti mozgatás → Nem effektív, gépidőt vesz el
- Processzor affinitiás → Mennyire „ragaszkodjon” egy folyamat a processzorhoz amin fut
    - Általában folyamatoknál ez az érték nagy
    - Hogy ne tegye át másik procira a folyamatot, ezzel időt veszítve
    - Soft affinitás, hard affinitás
    - Folyamat egy procin belül szeretne maradni → nem mindig megoldható
- Processing típusok
    - Homogén processzorok→ Egy multiprocesszoron belül
    - Assymetric multiprocessing
        - Master CPU-k és Slave CPU-k
    - Symmetric Multiprocessing – SMP
        - CPU-k egyenrangúak
- Load Balancing
    - SMP esetén összes processzornak kell „adni feladatot”, hogy hatékony legyen
    - Load Balancing → Megpróbálja elosztani a terhet CPU-k között
    - Pull migration
        - Ha vannak vegetatív magok/processzorok, áthúzza a terhelt prociról
        - Békés
    - Push migration
        - Terhelt processzorról letolja a terhelést
        - Nem a legbékésebb

Multicore Processzorok

- Egy fizikai chipen belül több processzor
- Nem olyan hatékony 2 mag mint 2 processzor
    - Elveszted a plusz Cache-t
    - Az ALU-t → Aritmetikai és Logikai egység
- De mindenképpen javulás egy maghoz képest, viszont közel nem lineáris a javulás

Valósidejű Processzorütemezés

- X időn belül kell valami response/feedback a folyamattól
    - Nincs olyan hogy szimplán „lefagy”
    - Ha nem készül el időn belül, hibának veszi, elkezdi kiértékelni, javítani
        - Hibakezelő
- PC-kre egyáltalán nem jellemző

Chapter 7

Memóriakezelés – 10. előadás

Alapvetések

- A programnak a storageból a memóriába kell kerülnie, hogy le tudjuk futtatni
- A CPU csak a Main memóriát és a regiszterek éri el
- A memory unit csak címek sokaságát, illetve olvasás írás requestet lát
- Cache a memória és a CPU között van
- Memóriát védeni kell hogy megfelelően tudjon működni
- Base és limit regiszterek meghatározzák a memóriacímek tartományát ami elérhető a usernek
- A CPU-nak meg kell néznie hogy a user módban generált memória eléréshez joga van-e a usernek

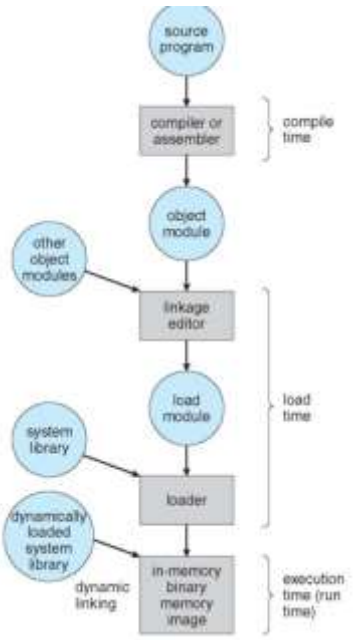
Hardveres védelem

- Ha rossz memóriacímre hivatkozik a program → Kilövi a programot

Cache szükségessége

- CPU gyorsabb mint a memória → Stall → CPU vár az adatra a memóriából
- Ezért hozták létre a Cachet
    - Nagyon gyors, és nagyon drága tárolóegység kis mérettel
    - Régen alaplapon, most a processzoron
    - Megpróbálja kiküszöbölni a Stalling-ot

Memória Címek



- Utasítások és adatnak memóriacím adás, memória allokálás három időben történhet
    - Fordítás ideji cím
        - Fordítás előtt tudjuk hol az adat
        - Abszolút memória cím/Direkt memória cím
    - Betöltés ideji cím
        - Olyan kód kerül előállításra, ami nem direkt beégetett kód, memória cím, csak annyit mond kell neki 4 byte memória
            - Megmondja melyik memória címtől kezdődik az ő helye
            - Relatív memória cím
    - Futási idejű cím
        - Nincs megadva betöltéskor az allokált memória cím
        - Mondod a gépnek mennyi memória kell → Ad

Logikai vagy fizikai memóriacím

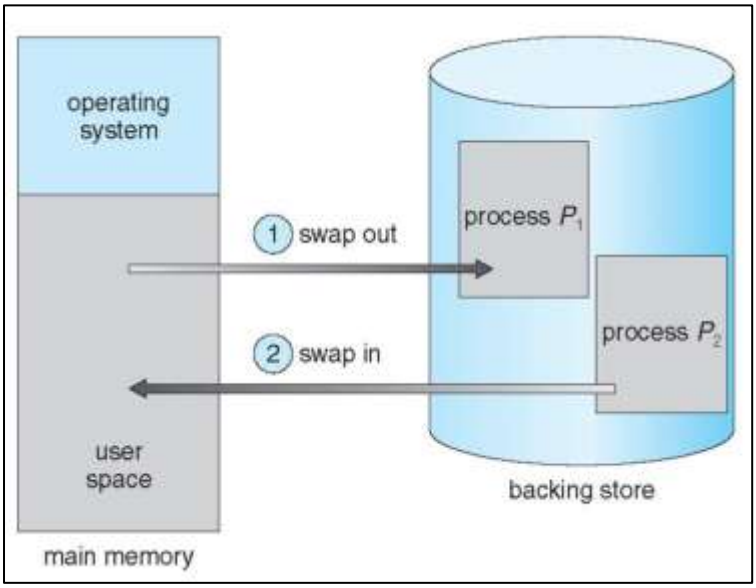
- Logikai/Virtuális memóriacím
  - CPU által generált
  - Lassabb, de megbízhatóbb
- Fizikai memóriacím
  - Memory unit által látott
- Adress space→ Egy program által generált összes Logika/Fizikai memóriacím
- Memory Management Unit – MMU
  - Logika címhez fizikai címet sorsoló hardver eszköz
  - Dinamikus cím
  - Felhasználói programok soha nem foglalkoznak fizikai memóriacímmel, csak logikaival

Linkelés

- Statikus linkelés
  - #include header.h → Beleépíti a programba a headert lényegében
    - Egyszerű címzés
    - Több memória kell hozzá, nem olyan hatékony
- Dinamikus linkelés
  - Stub/Csonk → Megnézi az adott függvénykönyvtár be van-e már töltve
    - Csökken az induló memóriahasználat → Idővel ahogy használjuk felkúszhat ugyanoda
  - Dll → dynamic link library → Könyvtárak max 1 példányban a memóriában

Swapping

- Alapötlet → Ha kevés a memória/RAM → Told bele más tárolóba a folyamatot
  - Ez a tároló a háttértár
  - Nagyön költséges feature
- Ha az a folyamat kell éppen ami a háttértáron van →Swap
  - Kitol RAM-vól valamit, ami az ütemező szerint nem kell és betolja a RAM-ba ami kell a háttértárról
  - Úristen very lassú



Többpartíciós allokálás

- Első oprendszerek → Fix memóriaterület adva egy folyamatnak
  - We no likey
- Változó méretű partíció/memóriaterület allokálás
  - Minden folyamatnak annyi memóriát ad, amennyit kér
    - Annak a folyamatnak majd deallokálni is kell a mennyiséget, amit allokált, itt lehetnek problémák
  - Összefüggő memóriaterület kell legyen →
    Ha nincs el sem indul, vagy vár amig lesz
  - Itt a probléma a lyukak kezelése, miként, hova allokálunk memóriát?

Dinamikus Memóriaallokálás – Típusok

- First fit
  - Megyünk végig a memórián, és az első összefüggő üres helyre ahova tudjuk „berakjuk", ott allokálunk neki memóriát
- Best fit
  - Oda rakjuk, ahol a legkisebb a lyuk, ami még elég a programnak
    - Legkevesebb a pazarlás
- Worst fit
  - A lehető legnagyobb összefüggő memóriaterületen allokáljunk a programnak memóriát
- First-fit és Best-fit vált használttá
  - Ez viszont feldarabolja a szabad helyet → Külső töredezettség
- Külső töredezettség
  - Van elég memória, de nem összefüggő terület
  - Kell egy „töredezettségmentesítő"
  - Egy gondolat a megoldásra → Compaction/Tömörítés
    - Rendezzük újra az összes memória blokkot, hogy egy nagy üres memóriaterület maradjon
    - Csak gondolat maradt, mert ha egy program ugyanarra a memóriacímre hivatkozik, mint tömörítés előtt, nem fog működni, illetve az I/O műveletek sem szeretnék
    - We no likey összességében

Megoldások töredezettségre

- Szegmentálás
  - Daraboljuk fel a programot → A memóriában több kisebb darab kell nem egy nagy összefüggő memóriaterület
  - Hagyományos, elavult
  - Természetes következménye a Paging/Lapozás
  - A logikai memóriacím több részre bomlik
    - Segment Number
    - Szegmenten belüli cím
  - Szegment tábla
    - Base
      - Megmondja melyik fizikai címen kezdődik egy szegmens
    - Limit
      - Megadja a szegmens hosszát
  - Ezzel nincs belső töredezettség, és nagyban csökken a külső töredezettség
- Lapozás/Paging
  - Vissza a gyökerekhez → Felosztjuk a memóriát fix méretű keretekre
    - 512 byte – 16 Mbyte
  - Ezzel kapunk n darab lapot/paget
  - Nyilvántartás kell a foglalt és üres keretekről
  - Page Table → Logikai címet lefordítja a gépnek fizikai címre
  - Nincs külső töredezettség, de bejön a belső töredezettség
  - 80-as években tökéletes, de bajos ahogy nőtt a memória méret
    - Sokáig tart 10000000 lap között megkeresni amit akarunk
    - Megoldás → Több szintű lapozás
- Több szintű lapozás
  - Ha van 10000000 lapunk, felosztjuk 100000 lapos blokkokra, és felindexeljük ezeket a blokkokat → Tudjuk, hogy a 3. blokkban van valami, nem kell az összesen végigmenni → gyors
- Osztott Lapozás
  - Bent vannak a lapokon a függvénykönyvtárak → Nem kell többször betölteni a memóriába, nem foglal feleslegesen memóriát
  - Stub-ok használata
  - Külső töredezettség nincs

Chapter 9 (8-skip)

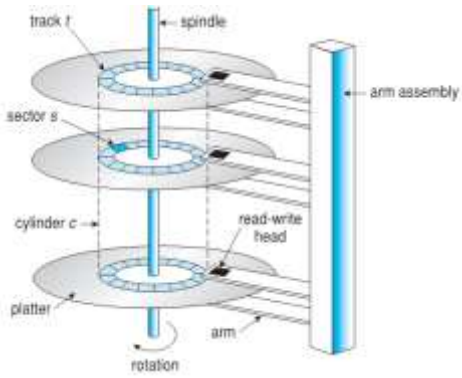
Mass Storage System/Tárolórendszer – 11. előadás

Alapvetések

- Neumann elv → Memóriába kell rakni valamit, ha KI AKARJUK VÉGEZNI, de ez adat nem jó ha elveszik ha nincs áram → Háttértárak megjelenése

Háttértártípusok

- Mágnesszalag
  - Lassú, de ár/kapacitás és megbízhatóság szempontjából nagyon jó
  - Szekvenciális olvasás
- Merevlemez / Hard Disk Drives
  - Író – Olvasó fej → Nem ér hozzá a tányérhoz (jobb esetben)
    - Mozgó alkatrész → Not good, nem jó rugdosni
    - Fejmozgatási stratégia fontos lehet
  - Tányérok → Sávok → Szektorok (min 512 byte)
  - RPM 4200-15000
  - Kapacitás 500 – 10+TB
  - Örökéletre megőrzi, ha nem zavarja be valami mágneses tér
  - Könnyen eltávolíthatóak
  - Az alaplap kettéválasztja a bővítő eszközt az I/O-tól
    - PCI bus-t ezért nem használják
  - Seek time → 3 ms – 12 ms
  - SATA-t használt



- EIDE/ATA, SATA, USB
  - ATA – Advanced Technology Attachment
  - EIDE – Electronic integrated device
  - SATA – Sequential ATA Elméleti max sebesség → 6 Gbit→ Gyakorlatban 1 Gbit
  - USB – Universal Serial Bus
  - Ezek mind azt írják le hogy kerül át az adott adat a háttértárról a memóriába
  - Lehetséges probléma → Késlekedési idő → Lemez ütemezési stratégia

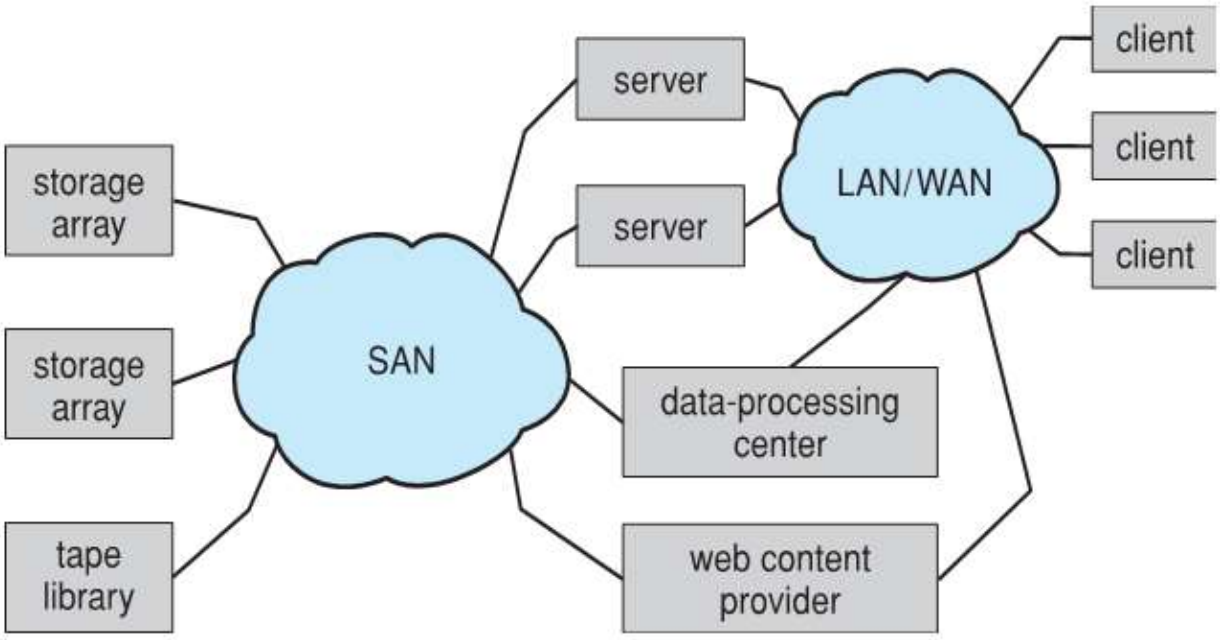
- Solid State Drive /SSD
    - Non-Volatile memory used like a hard-drive
    - Megbízhatóbb, mint a HDD
    - Sokkal drágább
    - Kisebb élettartam → A sok írást nem szereti
    - Kisebb kapacitás
    - Sokkal gyorsabb, mint egy HHD
    - PCI bus-hoz csatlakozik, hogy a sebességet ki tudja használni
    - Nincs mozgó alkatrész, nincs seek time

Háttértár struktúra

- Szektor alapú → Egymás után sok szektor
    - 1 szektor 512 Byte
    - Szektorok sorszáma 0.-tól kezdődik
- Amikor formázunk egy tárolót, szektorokból logikai blokkot készítünk
    - NFTS → Egy blokk lehet 4096 byte → 8 szektor
    - Tárterületvesztés lehet → 1 byte-os file elfoglal 4096 byte-ot

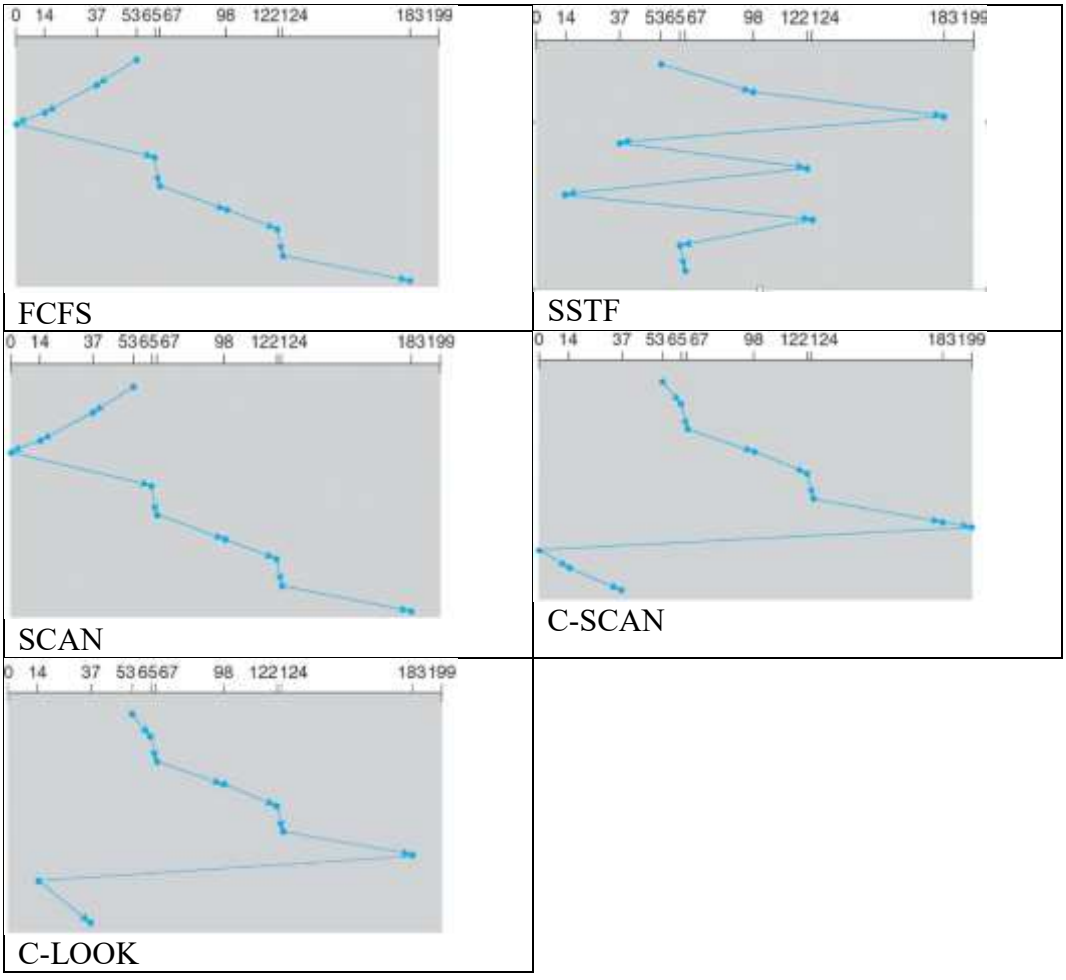
Tároló alhálózat / Storage Area Network SAN

- Elmaszkolja, hogy mi és miért látszódik
- Hálózaton kersztüli elérés → Gyorsabb mint az I/O
- Nagyon gyors, nincs adatvesztés
- Nagyon drága



Lemezütemezési Stratégiák

- Lokálisra törekszünk, de lehet bárhol → Mi most lokálissal foglalkozunk
    - A kérdésre, hogy muszáj-e, hogy ez lokális legyen, mindig nem a válasz
- Író-Olvasó fej mozgatása, csesztetése
- HDD-n ütemezési stratégiák
    - FCFS First Come First Served
        - Lassú
    - SSTF Shortest Seek Time First
        - Sokkal jobb, gyorsabb
    - SCAN – Természetes legyező mozgás
        - Csökkenti a szükséges fejmozgásokat
        - Teljesen elmegy mindkét oldalig → Felesleges
    - C-SCAN
        - Mint a Round Robin → Folyamatos legyező mozgás körbe-körbe
    - C-LOOK
        - C-SCAN, de kiveszi a felesleges mozdulatokat, nem megy ki a legszélére, ha nem kell



Melyik a legjobb lemezütemezési stratégia?

- Nincs legjobb, szituációfüggő
- Leggyakrabban az SSTF az optimális
- SCAN / C-SCAN → Heves I/O műveleteknél jó
- Fájlrendszerhez hasonlóan, minden használathoz megvan mi az ideális

Swap – Space Management

- Linuxon → Magától csinál egy Swap partíciót ahhoz megfelelő fájlrendszerrel
    - Lehet állítani
    - Really good
- Windows
    - Gyökérbe berakunk egy Swap fájlt
    - Mi a fasz
    - Lol

Raid Struktúra

- Miért van rá szükség?
    - 1 db disk a hardver bus maximum 50%-át használja ki
        - „50% ott van parlagon”
    - Erre a megoldás a Raid Tömbök
- Mit jelent a Raid struktúra?
    - Több disk összekötve, hogy maximálisan kihasználjuk a sávszélességet
    - 2 disk = 2* 50% =100% → We likey
- RAID típusok
    - RAID 0 – Nem redundáns csíkozás
        - Nagyon gyors, a megbízhatóság oltárán
        - Felcsíkozza az adatokat → Egyik csík egyik diskre, másik másikra
            - Ha elszáll az egyik elszáll az egész
        - Nem redundáns, minden adat csak egyszer van meg
            - Adatvesztés veszélye fennáll
    - RAID 1 – Mirrored Disks
        - Nagyon megbízható, a sebesség oltárán
            - Minden másolást effektíve kétszer végez el mindkét diskre
        - Minden adat kétszer van meg
            - Ha elszáll az egyik, ott a másik, no problem
    - RAID 2,3,4,5
        - Hibrid megoldások
        - Csíkozott, és redundáns, stb

Chapter 10

File System Interface/ Fájlrendszerek – 12. előadás

File koncepciója

- Összefüggő/folytonos logikai címtér
- Háttértár oldja meg a többit → Háttértárnak nem feltétlenül folyamatos, usernek igen
    - Ez a töredezettség
- UNIX-os világban szó szerint minden fájl
    - Eszközök
    - Parancsok
    - Standard Kimenet/bement/error kimenet
    - Anyád
- File típusok → Lényegében végtelen van
    - Bináris
    - Szöveges
    - Végrehajtható
    - Ezek csak felhasználó számára fontosak a gépnek minden csak egy bytesorozat
    - Kiterjesztés → Plusz infó, nem kötelező (Windowson kinda)

Fájl tulajdonságai

- Név
- File ID → Külön azonosító a fájlrendszeren belül → Ez nem azt jelenti hol van a disken
- Típus → Néhány rendszernek kell
- Location → Hol van a fájl magán a disken
- Méret → byteban
- Védelmi lehetőség/Jogosultság
- Időbélyeg
    - Létrehozás ideje
    - Utolsó módosítás ideje
    - Utolsó hozzáférés ideje
    - Lehet több is, ez alap

File műveletek

- Létrehozás – Törlés
- Írás – Olvasás
- Pozícionálás fájlon belül → Seek → Létezik aktuális pozícióra mutató adat
- Truncate/Csonkolás → Kitörli a tartalmát, név, tulajdonság megmarad
- Open – Close

Megnyitott fájlok kezeléséhez szükséges adatok

- Oprendszer nyilvántartja mi van megnyitva jelenleg
  - Ezért nem lehet átnevezni egy fájlt, ami meg van nyitva
- File Pointer → Hol vagyunk az aktuális fájlban
- Open számláló → Hányszor lett megnyitva
- Location
- Hozzáférési jog
- Amíg nincs konkurens hozzáférés addig ez pacek

Nyitott fájl Zárolás

- Olvasási Zár/Osztott zár → Akármennyi folyamat hozzáférhet egy fájlhoz
- Írási zár/Exkluzív zár → Én írok a többieknek kuss
  - Adatbázisok

Java dolgok nem kellenek innen

File kiterjesztések, típusok

- Executable
  - Linuxban nem kell
  - Windowsban igen
- Object → Tárgykód
- Batch
  - Minden rendszerben van
  - Linux.sh
  - Windows.bat
- Könyvtárak=Függvénygyűjtemények
  - Lib, dll
  -

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Fájl Struktúra

- Eddig → Összefüggő logikai címtér
- Hogyan olvassuk ? → Byte-osával → Not good
    - Itt is mint fájlrendszerben ráépítünk egy logikai blokkot
        - Átvitel egység
    - Szöveges fájl → Átviteli egység ugyanaz mint a bytesorozat
        - Bináris, egy kódtáblával
            - Ascii
            - UTX-8 stb.
    - Bináris fájl
        - Telefonkönyvet binárisan → Szövegesen / Struct-al C-ben
            - Struct
              {
              String name;
              Int number;
              }
        - Ha megmondom az átviteli egységet, akkor nem byte-onként olvas, hanem structonként → Egyszerre egy név-telefonszám párt olvas → Hatékonyabb mint byte-onként
    - Az eddigi szekvenciális olvasást, hozzáférést felváltja a direkt hozzáférés
        - Logikai rekordok, 1 Struct
        - Speciális szekvenciális ha úgy nézzük

Hogy kerül ki a diskre a fájl? - Directory Structure

- A Disk-et partíciókra tudjuk bontani
    - Ezeket a partíciókat köteteknek is nevezzük
    - Lemez egy része
- Cél a fájlok tárolása → Ehhez kell egy fájlrendszer
    - Fájlok rendszeres tárolására alkalmas megoldás
    - Vannak általános, és speciális célokra kifejlesztett fájlrendszerek

Fájlrendszertípusok

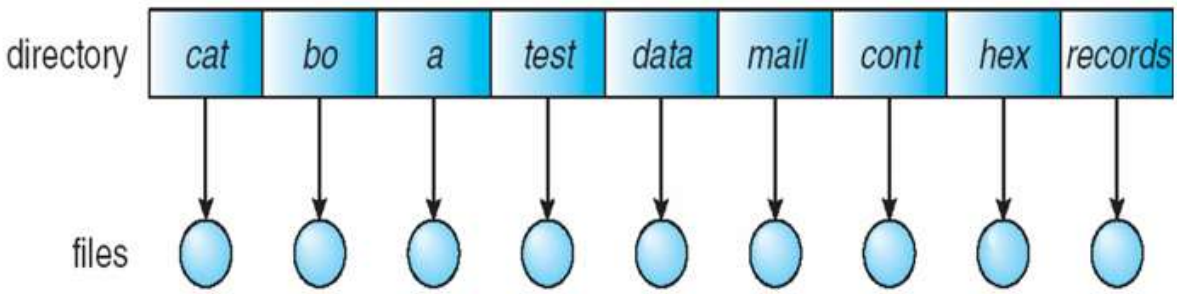
- Szükségszerűen, sok a különbség köztük
- Procfs → Minden fájl, folyamatok is
- Hálózati fájlrendszerek
    - Szükséges biztosítani a kötet távoli elérését

Könyvtárműveletek

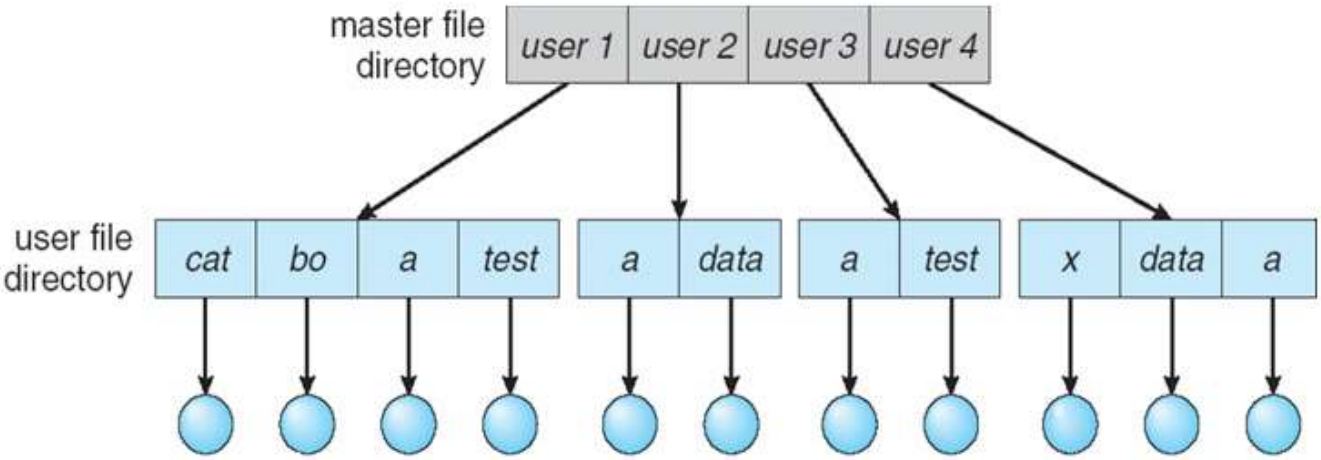
- File keresése
- Fájl létrehozása – Törlése
- Filok listázása
- Filok átnevezése
- Filerendszer átjárása

Directory Organization

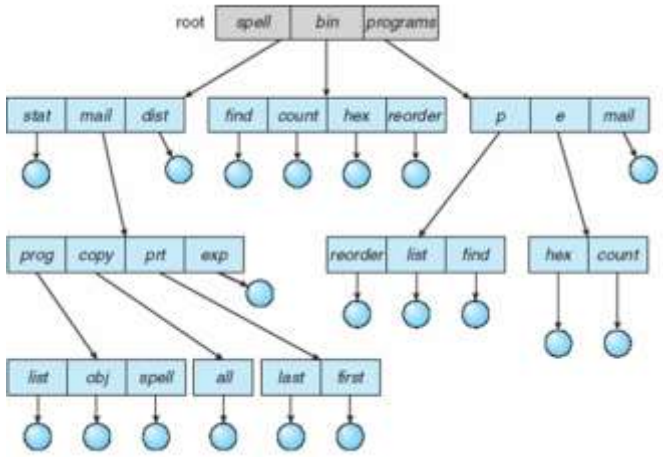
- Kezdetben
  - 1 Partíció → 1 Könyvtár
- Egyszintű
  - Kezdetben minden fájl egy mappában, az egész disk egy mappa
  - Nem lehet két ugyanolyan nevű file-om
  - Két felhasználó nem adhatja ugyanazt a filenevet
  - Amíg kevés file, addig jó, hatékony, gyors
  - Nincs lehetőség csoportosításra



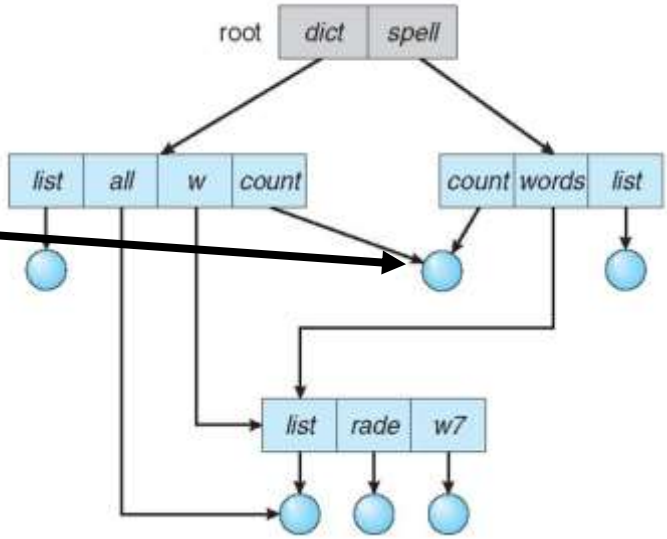
- Kétszintű
  - Userenként kétszintű → Minden usernek van egy egyszintű kötete lényegében
  - Megjelenik az elérési útvonal
  - Lehet két ugyanolyan nevű fájl két külön usernek
  - Hatékony, gyors
  - Nincs lehetőség csoportosításra



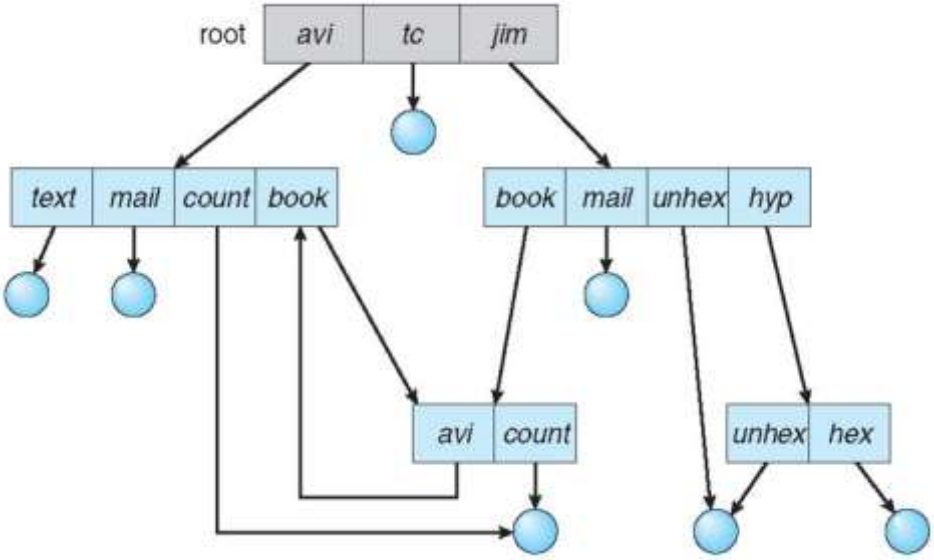
- Fastruktúra
    - Egy gyökérmappa, és annak almappái
        - Minden mappának lehet almappája
    - Relatív és abszolút elérési útvonal megjelenése
    - Csoportosítás lehetséges
    - Gyors, hatékony keresés
    - A bejárása egy valós probléma lett
    - Hogyan oldjuk meg, hogy különböző elérési útvonalakon elérjünk ugyanoda?



- Aciklikus-gráf struktúra
    - Tud mindent, amit a Fa, csak megoldja a bejárást
    - Osztott listák, és alkönyvtárak
    - Ha valami két irányból közelíthető meg, hogy letöröljük mindkét irányból le kell törölni
        - Dangling Pointer problénája
            - Linkek megjelenése



- Általános Gráf Struktúra



Linkek

- Hard Link
    - Nem jön létre új fájl effektíve
    - Csak köteten belül működik
    - Egy pointer lényegében
- Soft link/szimbolikus link
    - Egy fájl amiben van egy elérési út
    - Lehet broken link, ha az adott helyen ahova mutat nem létezik az a fájl/mappa
    - Fájlrendszereken átívelhet → Nincs kötethez kötve

Filerendszer Csatolás

- UNIX alapú OS-nél csak becsatoljuk egy mappa alá és világbéke
    - Egy nagy fa van amihez hozzácsatoljuk a kötetet
- Windows
    - xd
    - Max 24 kötet lehet mert annyi maradt az ABC-ből
    - Lol

File Sharing/Megosztás

- Hogyan oldom meg a hozzáférést az állományaimhoz?
    - Jogosultságkezelés, csoportok
        - Linuxon pacek
        - Windows már megint nem tudom mi a faszt csinál
- Lokálisnak kell lennie?
    - Szokásos válasz→ Nem

Remote File Systems

- Távoli kötetet csatolunk fel
    - Nincs lényegi változás
    - UNIX NFS
    - Windows CIFS

Védelem

- Hozzáférés típusai
    - Írás – Olvasás
    - KIVÉGZÉS
    - Hozzáfűzés
    - Törlés
    - Listázás

Hozzáférési listák és csoportok

- Linux
  - Owner
  - Group
  - Other/Guest/Public
- Windows
  - Whatever the fuck this is
- A Windowsos jó, de a Linuxos működik