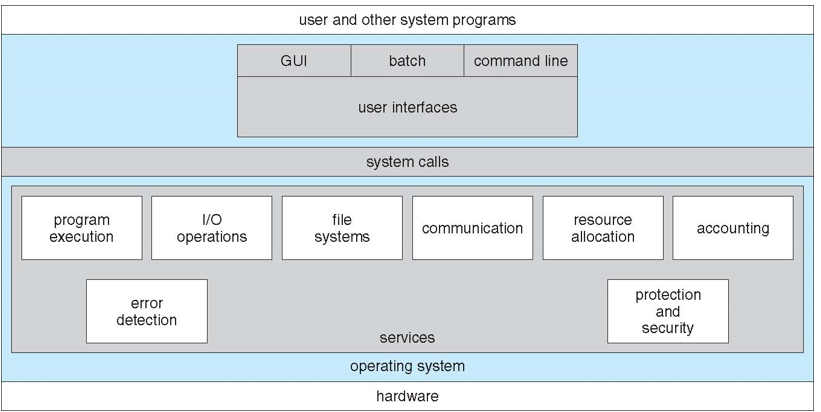
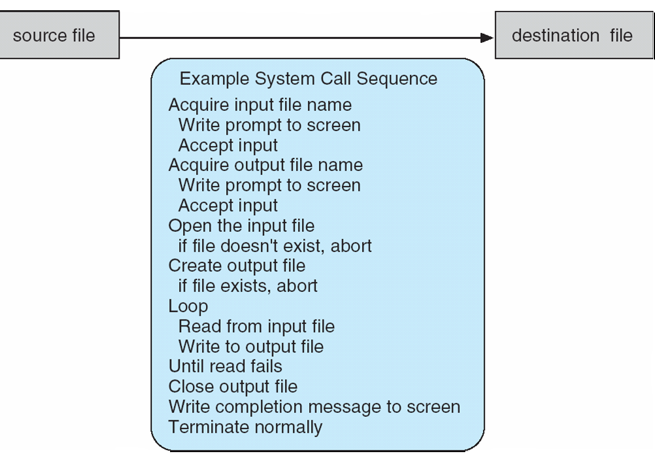
**Chapter 2**

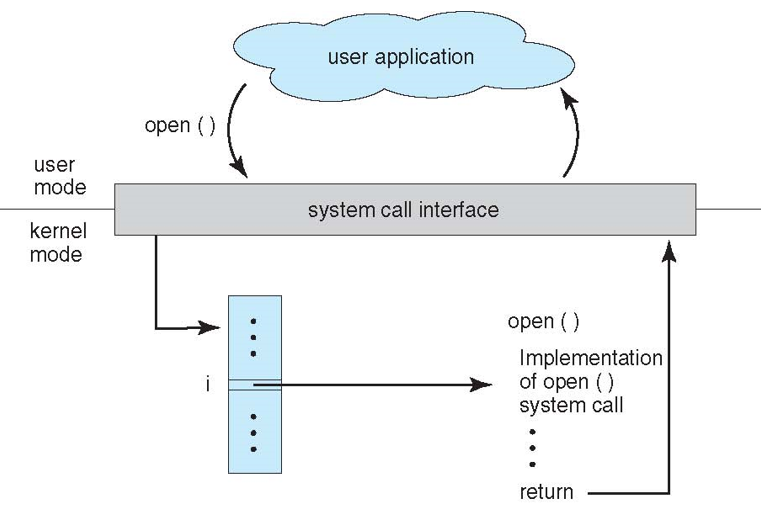
Operációs Rendszer -Struktúrák

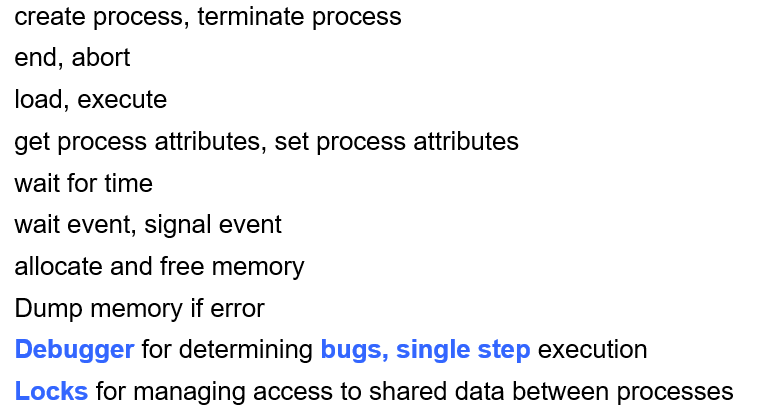
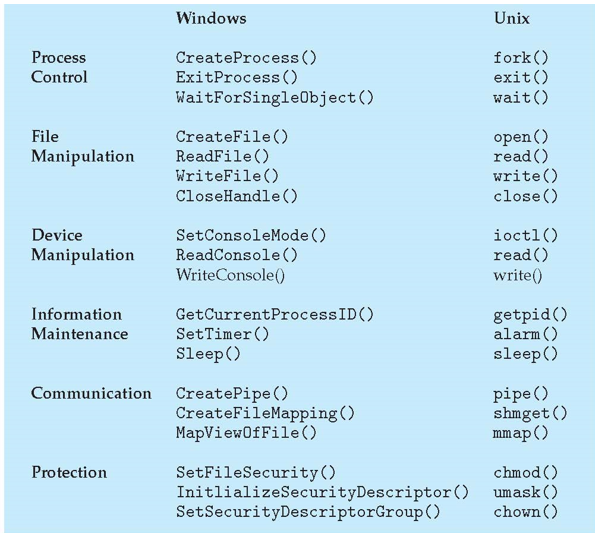
Operációs rendszer szolgáltatásai, feladatai

* 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



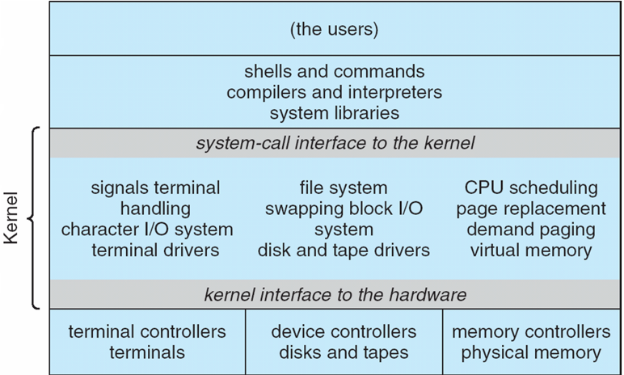
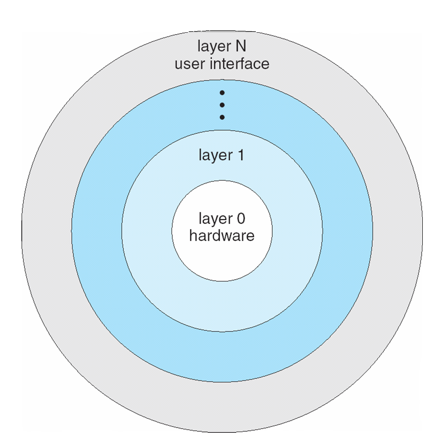
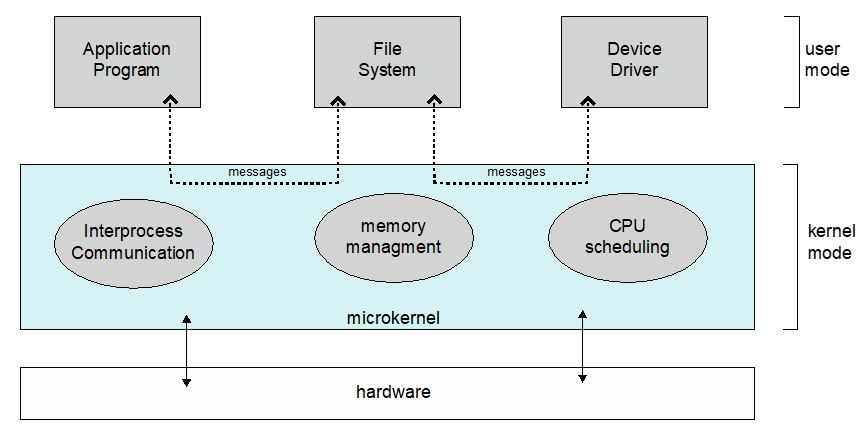
* + Rendszerhívás típusok
  + +File és eszközkezelés

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 stb.
* 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