Operációs Rendszerek

1. Tétel: Az operációs rendszer fogalma (több nézőpontból is). Az operációs rendszerek szolgáltatásai (szolgáltatások szerinti komponensek)

Az OS, mint kiterjesztett gép:

- Magasabb absztrakciós szintet biztosít a felhasználó számára.
- Az eszközöket és állományokat szimbolikus neveken engedi kezelni, ezekben magasabb szintű operációkat biztosít.
- Elrejti a részleteket a felhasználó elől, kényelmessé teszi a hardver használatát.
- Egységességet biztosít a hasonló, de részleteikben különböző eszközök kezelésére. (pl.: floppy és hard diszkek)
- Az OS által biztosított virtuális gépet könnyebb programozni, mint az alatta létező hardvert.

Az OS, mint erőforrás menedzser:

- Az OS feladata, hogy elossza az erőforrásokat az ezekért vetélkedő programok számára.
- Az OS-nek hardver-, szoftver-, és emberi erőforrásokat kell menedzselnie.
- A menedzselési feladatkörbe az erőforrások kiosztása mellett beleértjük azok védelmét, használatuk számbavételét.
- Az OS hatékonnyá teszi a hardver használatát.

Az OS szolgáltatásai, szolgáltatások szerinti komponensei:

- Processz menedzsment komponensek.
    - Processz kreációt és terminálódást biztosító funkciók.
    - Processz kontroll és ütemezési funkciók.
- Memória menedzselő komponensek.
    - Memóriahasználat nyilvántartása.
    - Memória allokálás és felszabadítás.
    - Címleképzés segítése.
- Másodlagos tárolók menedzsmentje.
    - Szabad terület menedzselése.
    - Blokk allokáció.
- I/O menedzsment komponensei.
    - Bufferezés.
    - Device driver interface elemek, speciális driver-ek.
- Fájlrendszert megvalósító és menedzselő komponensek.
    - Jegyzékstruktúra megvalósítás.

    - Blokk-hozzárendelési módszerek.
    - Fájlok, jegyzékek készítése, törlése, stb.
- Védelmi komponensek.
- Hálózatkezelő komponensek.
- Felhasználói kapcsolattartó rendszer (CLI, pl.: parancsértelmező, GUI)

2. Tétel: Operációs rendszerek implementációs szerkezetei (Monolitikus kernel, réteges kernel, mikrokernel struktúra)

Monolitikus kernel:

- „Az a struktúra, mikor nincs is struktúra”. Nincs strukturáltság.
- A monolitikus rendszer magja névvel ellátott szolgáltató eljárások gyűjteménye.
- A szolgáltató rutinok hívása:
    - explicit: felhasználó programból rendszerhívással (system call).
    - implicit: exception, interrupt.
- A system call kívülről egyszerű függvény, de valójában ez egy trap: a processzor felhasználói módból kernel módra vált.
- A kernel szolgáltató rutinjai egymást korlátlanul hívhatják.
- A szolgáltató rutinokból a felhasználói programba való visszatéréskor a processzor futási módja visszavált felhasználói módra.

Réteges kernel:

- Rétegezett struktúra, a szolgáltató rutinok rétegekbe szervezettek.
- Előnye:
    - Egy rétegben jól meghatározott, összetartozó funkciók találhatók.
    - Egy réteg magasabb szintű operációkat biztosít a felette lévő számára.
    - Egy réteg elrejti az alatta lévő részleteket.
- Hátrány: hatékonyság probléma (alsó rétegben megvalósított szolgáltatás kivezetése).
- A rétegek között jól meghatározott interfészek vannak, ezek explicit-, illetve könyvtár függvényhívások.
- Ilyen jellegű OS volt a Dijkstra professzor által tervezett THE (1968), rétegei:
    - 0. réteg: Processzor allokálás, multiprogramozás, szinkronizáció.
        - Virtuális CPU-k, minden processztől elrejtve a többi.
    - 1. réteg: Memória-dob menedzsment.
        - Virtuális memória, a processz a teljes címtartományát látja.
    - 2. réteg: Operátor-processz kommunikáció.

        - A processzeknek saját, virtuális konzolok.
    - 3. réteg: I/O menedzsment eszközök.
        - Felette virtuális, absztrakt I/O. Bufferezés minden processz számára.
    - 4. réteg: Felhasználói programok.
        - Független processzek.
    - 5. réteg: Operátor.
        - Az operátor elindít, lelő processzeket.
- A rétegezési koncepciót a korszerű OS-ek természetszerűleg használják. (pl.: virtuális fájlrendszer, dinamikusan linkelt futásidejű könyvtári rutinok (DLL))

Virtuális gépek:

- Egy időosztásos rendszertől elvárjuk:
    - Biztosítsa a multiprogramozást.
    - Biztosítson kiterjesztett gépet, melynek kezelése kényelmesebb, mint egy valós gépnek.
- Virtual Machine Monitor (VMM) fut a hardveren, ami több virtuális gépet emulál, amelyek pontos másolatai valós gépeknek.
- Az emulált gépeken különböző OS-ek futtathatók.
- A VM közvetlenül a hardver feletti réteg.

Mikrokernel – Kliens-Szerver modell:

- Bizonyos rendszerszolgáltatások kiszolgálására önálló processzeket készítenek.
- A szolgáltató processzek programjai önállóan linkelhetők, betöltődhetnek a rendszer egész életére, vagy időlegesen (daemon processzek), futhatnak kernel és felhasználói módban is.
- A szolgáltatások processzek közötti kommunikációval kérhetők. A felhasználói processzek (kliens processzek) üzenetküldéssel kérnek szolgáltatást a szolgáltató processzektől, melyek üzenetküldés formájában válaszolnak a kérésre.
- Ebből fejlődtek ki a hálózati osztott rendszerek.

3. Tétel: Az operációs rendszerbeli folyamat (processz) fogalom. Folyamat kontextus. A taszk és a fonál, azok megvalósításai.

Folyamat (processz):

- A processz egy végrehajtási példánya egy párhuzamosságot nem tartalmazó végrehajtható programnak.
- A processz egy futó program-példány.
- A processz más entitás, mint a program.

- Egy processzhez egyetlen programszámláló regiszter (PC) tartozik, ami egyetlen helyre tud mutatni a program szövegében.
- A klasszikus processz egy szálon fut, a benne futó végrehajtható program nem tartalmaz párhuzamosságot.

Folyamat tartalom (processz kontextus):

- Adatstruktúrákba rendezve minden olyan információ, ami a folyamat futásához szükséges.
- Minden olyan információ, ami a rendszer számára szükséges, hogy a CPU-t a folyamatok között kapcsolja, a folyamatok szekvencialitásának illúzióját biztosítva.
- Tartalma:
    - A program kódszegmensei (instrukciók sorozatai).
    - A program adatszekciói.
    - A processz veremtárai (stack).
    - Regiszter állapotok.
    - A processz menedzselési információi: azonosítási, állapotokat leíró, számlázási, hozzáférési, stb. információk.
- A processz kontextus szemléletmódjai:
    - Hardver kontextus: regiszterek értékei, mint menedzselési információk.
    - Szoftver kontextus: kódszegmensek, adatszegmensek, egyéb információk.
    - Felhasználói szintű kontextus: ami a felhasználói címtartományban van.
    - Rendszer szintű kontextus:
        - Statikus: főleg a menedzselési információk tartoznak ide.
        - Lebegő (dinamikus, volatile): regiszter állapotok.

Kontextus váltások:

- Processz kontextusról rendszer kontextusra:
    - Processz dinamikus kontextusának lementése.
    - Rendszer kontextus betöltése.
- Rendszer kontextusról processz kontextusra:
    - Lementett processz kontextus visszatöltése.
- Processz-processz kontextus váltás (Process Context Switch):
    - „Lekapcsolt” processz kontextusának lementése.
    - „Felkapcsolt” processz kontextusának betöltése.
- A kontextus váltások időigényesek, hardveres támogatással gyorsíthatók (több regiszterkészlet).

A processz kontextus adatstruktúrái:

- Processz tábla:
    - A kernel által kezelt adott méretű táblázat.
    - Többnyire struktúrák készleteként (pl.: láncolt listaként) implementálják.
    - Elemei process table entry-k, vagy más néven process control block-ok (PCB).
    - Minden processznek van benne bejegyzése.
    - A tábla mérete kritikus.
- Process Table Entry/Process Control Block (PCB):
    - Processz tábla bejegyzés.
    - A processzről minden információ elérhető ezen keresztül.
    - A processz létrehozásakor keletkezik.
    - A memóriából nem kisöpörhető, azaz rezidens.
    - Tartalma:
        - Processz menedzselési információk.
        - Mutatók a kontextus további részeire.
        - Mutató a Process Descriptor-ra.
- Processz leíró (Process Descriptor):
    - A processz futtathatóvá válásakor keletkezik.
    - Lehet kisöpörhető is.
    - „Dinamikus” adatokat tartalmaz, melyek a processz futásához is kellenek (volatile kontextus).

Taszk, fonál:

- A modern OS-ek támogatják a konkurens programozást, párhuzamosságot, a processz helyett két másik fogalom: taszk (task), fonál (szál, thread).
- A futó program a taszk, amely több fonalat is tartalmazhat, amelyek egymással is versengenek a CPU-ért.
- Egy taszk egyetlen fonállal a klasszikus processz.
- Taszk:
    - Heavyweight Process.
    - Van statikus kontextusa (címkészlet, erőforrások, menedzselési információk).
    - A végrehajtandó kódot a fonalai tartalmazzák.
    - Fonál nélkül egy taszk passzív entitás.
- Fonál:
    - CPU használat alapegysége.
    - Lightweight Process (LWP).
    - Szekvenciálisan futó instrukciósorozat.

- Van dinamikus kontextusa (regiszterkészlet, verem).
- Osztozik más fonalakkal egy taszk statikus kontextusán.
- A fonalak valódi, vagy pszeudó párhuzamosságban futhatnak.

4. Tétel: A processz állapotok, állapotváltások, processz futási módok. Taszk és fonál állapotok, állapotátmenetek.

Processz állapotok, és állapotátmenetek:

- A processzek életük során különböző állapotokban (state) lehetnek, ezek között különböző állapotátmenetek lehetségesek.
- Alap állapotok:
  - Running: futó állapot, a processzé a CPU.
  - Blocked: blokkolt, alvó állapot, a processz egy esemény bekövetkezésére vár.
  - Ready: futásra kész állapot, a processz csak arra vár, hogy megkapja a CPU-t.
- Alap állapotátmenetek:
  - Wait/sleep/request: várakozás esemény bekövetkezésére.
  - Signal/respond: jelzés esemény bekövetkezésére.
  - Preempt: a CPU elvétele a processztől.
  - Schedule: a CPU kiosztása a processznek.
- Ezekkel az állapotokkal nem szemléltethető egy processz keletkezése, megszűnése, illetve egyéb állapotok és állapotátmenetek.
- Állapotok kiegészítése:
  - Zombie: a processz már exitált, de még nem törlődött.
  - Non-existent: a processz nem létezik.
  - Suspended blocked/ready: a processz ideiglenesen ki van zárva a CPU-ért való versengésből. Az ilyen állapotú processzek esélyesek a kisöprésre. Egy processzt csak egy másik processz függeszthet fel (általában a rendszermenedzser processze).
- Állapotátmenetek kiegészítése:
  - Exit: a processz terminálódott.
  - Delete: az OS elvégezte a szükséges adminisztrációs munkát, a processz törlődött.
  - Create: a processz létrejön, az OS elvégezte a szükséges adminisztrációs munkát, nem kerül rögtön futó állapotba.
  - Suspend/resume: processz felfüggesztése/engedélyezése, középtávú ütemező feladata.
- Processz állapotok nyilvántartása:

    - A kontextusban (rendszerint a PCB-ben) rögzítettek az állapotok, de emellett láncolt lista adatszerkezeteken, sorokon is.
    - Előnye, hogy a láncolt listát könnyebb, gyorsabb kezelnie az erőforrás kezelő rutinoknak, mint végignézni az összes PCB-t.
    - Hátránya, hogy állapotváltáskor több helyen kell módosítani.

Taszk és fonál állapotok, állapotátmenetek:

- A fonál a CPU használat alapegysége, Lightweight Process (LWP). Egyedileg tartozik hozzá PC, regiszterkészlet, és veremtár, osztozik más fonalakkal a taszk címtartományon.
- A taszk, Heavyweight Process, fonalak nélkül passzív entitás. Egy taszk, egyetlen fonállal a klasszikus processz.
- Mivel a fonálnak legtöbb erőforrása megvan a taszkjában, a fonalak közötti CPU kapcsolás „olcsóbb", mint a Process Context Switch. A fonál kapcsolás csak regiszterkészlet kapcsolás.
- A fonál sokban hasonlít a processzhez, lehetnek állapotai, készíthet gyerek fonalakat, stb.
- Kernel Level Threading:
    - A taszknak nincsenek állapotai, a fonalaknak viszont vannak.
    - A kernel ismeri a fonalakat, allokál nekik CPU-t, ütemezi őket, nyilvántartja állapotukat, stb.
    - A fonalak ütemezését a scheduler végzi.
    - A fonalak menedzseléséhez rendszerhívások biztosítottak.
- User Level Threading:
    - Vannak taszk és fonál állapotok is.
    - A kernelnek nincs tudomása a fonalakról.
    - A fonalak állapotát a taszk tartja nyilván.
    - Running taszk állapotban egy fonál blokkolódásakor, a taszk kiválaszt egy másikat, és annak adja a CPU-t.
    - Ready taszk állapotban legalább egy fonál futásra kész.
    - Blocked taszk állapotban nincs futásra kész fonál.
    - A fonalak menedzselését RTL (Runtime Library) rutinok végzik.
    - A fonalak felhasználói módban futnak.

5. Tétel: Hiba és eseménykezelés: esemény, állapot, jelzés fogalmak. Kivételes esemény és megszakítás összevetése.

Esemény:

- Folyamatok (taszk, processz, fonál, rutin, utasítás, instrukció) futását befolyásoló, váratlan időpontban bekövetkező történés, amire reagálni kell, le kell kezelni.
- Lekezelése:
    - Az esemény bekövetkezésekor a vezérlés átadódik egy másik instrukciófolyam számára, amely kiszolgálja a bekövetkezett eseményt.
    - A kiszolgáló lefutása után, az eseménytől függően, folytatódik a megszakított folyamat végrehajtása, vagy máshova adódik a vezérlés.
- Fajtái a processz szemszögéből:
    - Belső esemény: a processzorban fellépő kivételes esemény (exception).
    - Külső esemény: nem a processzortól érkezik, megszakítás (interrupt).
- Fajtái az előállító entitás szerint:
    - Hardver által generált és detektált: hardveres interrupt, error.
    - Szoftver által generált és detektált: szoftveres interrupt, szoftver által generált exception, szűkebb értelmű események.

Feltétel (condition), állapot:

- Az az információs állapot, ami előáll, mikor egy esemény bekövetkezik.
- Beszélhetünk hardver-, és szoftver condition-ről.
- Ha az információs állapot hibára utal, hiba-, vagy exception condition-ről beszélünk.

Jelzés (signal):

- Esemény bekövetkezésekor (feltétel állapot előállásakor) jelzés keletkezik.
- Jelzések keletkeznek normál futás közben események hatására, ezek értesítik a CPU-t, vagy a processzeket az eseményről.

Megszakítások:

- A legalacsonyabb szintű események.
- A processz szemszögéből külső események.
- Előállítója általában hardver, de lehet szoftveres is.
- Lekezelői az OS interrupt kezelő (IT handler) rutinjai.
- Lekezelése:
    - Befejeződik az aktuális instrukció, lementődik a dinamikus kontextus.
    - Lefut a lekezelő rutin.
    - Betöltődik a dinamikus kontextus, visszatér a soron következő instrukcióra.
- A bekövetkezett, de még le nem kezelt megszakítások hardveres sorokon várakoznak.
- A megszakítások rendelkezhetnek prioritással, a magasabb prioritású interrupt megszakíthatja az alacsonyabb prioritású kiszolgálását.
- Fajtái:

    - Maszkolható: kiszolgálását az OS ideiglenesen letilthatja.
    - Nem maszkolható: nem lehet megakadályozni a kiszolgálását.

Kivételek:

- Szinkron jellegű, de váratlan események.
- Lehetnek alacsony szintűek (pl.: overflow, osztás hiba) és magas szintűek (pl.: laphiba).
- Klasszikus kivétel bekövetkezésekor az éppen futó entitás nem fejezhető be, le kell kezelni a kivételt, a felfüggesztett entitást újra kell futtatni, vagy folytatni kell.
- A lekezelés utáni folytatás a lekezelés eredményességétől függ.

Interrupt és exception különbsége:

- Az interrupt aszinkron, kiszolgálása két instrukció között megtörténik.
- Az exception szinkron, de váratlan. Tipikus példa az úgynevezett „Page Fault”.
- Az interrupt nem kapcsolódik a futó processzhez, kiszolgálása többnyire nem a futó processz javára történik.
- Az exception a futó processz instrukciófolyamának kiegészítése, kiszolgálása a futó processz javára történik.
- Az interrupt kiszolgálásánál prioritási szinteket vesz figyelembe a rendszer, magasabb prioritású megszakíthatja az alacsonyabb prioritású kiszolgálását.
- Az exception kiszolgálását más exception nem szakíthatja meg.
- Az interrupt kiszolgálására lehet közös rendszer verem.
- Az exception kiszolgálására processzenkénti kernel verem.

6. Tétel: Processzor idő ütemezése (scheduling). Kívánalmak, technikai alapok. Ütemezési stratégiák és állapotok összefüggései helyzetek. Ütemezési algoritmusok.

A processzor idő ütemezése:

- Az ütemezés az erőforrás kezelés egyik lehetséges módja.
- Az erőforrás (esetünkben CPU) használatát teljes egészében egy OS komponens, az ütemező (scheduler) vezérli.
- Előnye: teljesen kézben tartható az erőforrás kezelése.
- Hátránya: költséges megoldás (teljesítmény).
- Ütemezési szintek:
    - Long-term-scheduling: a create állapotátmenetért felelős.
    - Medium-term-scheduling: a suspend, resume állapotátmenetekért felelős.
    - Short-term-scheduling: a preempt, schedule állapotátmenetekért felelős.

Kívánalmak, technikai alapok:

- Pártatlanság: minden processz (taszk) korrekt módon, de nem feltétlenül egyenrangúan kapja meg a CPU-t.
- Hatékonyság: a CPU a lehető legnagyobb százalékban legyen kihasználva.
- Válaszidő: elfogadható legyen, minimalizálására kell törekedni.
- Fordulási idő: minimalizálni kell a kötegelt munkák indulása és befejezése között eltelt időt.
- Teljesítmény: időegységre eső job, taszk, processz feldolgozást maximalizálni kell.
- Technikai alapot nyújt, hogy a korszerű rendszerekben mindig van óraeszköz, ami periodikusan megszakításokat generál. Ez lehetőséget biztosít arra, hogy az időt szeletekre bontsuk, az erőforrás felhasználás idejét mérjük, illetve időközönként az állapotokat kiértékeljük, processzek közti kapcsolást valósítsunk meg.

Ütemezési stratégiák, döntési helyzetek:

- A döntési stratégia azért szükséges, hogy a scheduler választani tudjon, mely futásra kész processz kapja meg a CPU-t.
- Non-preemptive (nem beavatkozó) stratégia:
    - Run-to-completion jellegű: ha egy processz megkapja a CPU-t, addig használja, amíg a feladatát el nem végzi.
    - Cooperative (együttműködő): ha egy processz megkapja a CPU-t, saját döntése szerint lemondhat róla.
- Selective preemptive (szelektív beavatkozó) stratégia: bizonyos processzek futásába nem lehet beavatkozni (rendszer processzek), más processzektől elveszik a CPU-t, még ha az nem is mondana le róla.
- Preemptive (beavatkozó) stratégia: bár a processzek nem mondanak le a CPU-ról, bizonyos körülmények között, beavatkozva elveszik tőlük. Azokat az OS-eket, amelyeknél létezik a beavatkozás lehetősége, valódi időosztásos rendszereknek nevezzük.
- Beavatkozó ütemezési stratégiáknál az ütemező feladatai:
    - Döntés a beavatkozásról.
    - Döntés a kiosztásról.
    - Context Switch elvégzése.
- Ütemezési döntési helyzetek a következő esetekben léphetnek fel:
    - 1. Egy processz futó állapotból blokkolt állapotba kerül. (pl.: I/O kérés esetén)
    - 2. Egy processz futó állapotból ready állapotba kerül. (pl.: megszakítás esetén)
    - 3. Egy processz blokkolt állapotból ready állapotba kerül. (pl.: I/O befejeződése)
    - 4. Egy processz terminálódik.
- Ha ütemezési döntés csak 1. és 4. helyzetben lép fel, az ütemezés non-preemptive.
- Ha ütemezési döntés a 2. és 3. helyzetben is lehetséges, az ütemezés preemptive.
- Ütemezésnél figyelembe vehetők a processzek futásának különböző szakaszai:

    - CPU burst (CPU lázas).
    - I/O burst (I/O lázas).
    - CPU bound (számolásigényes, hosszú CPU lázas szakaszok).
    - I/O bound (rövid CPU lázas, főleg I/O csatornákat használó szakaszok).
- A beavatkozás döntési algoritmusai lehetnek:
    - Időszeleten alapuló: egy processz csak meghatározott időszeletnyit futhat egyhuzamban.
    - Prioritáson alapuló: a processzek rendelkeznek egy prioritás értékkel, ez alapján választ a scheduler.
        - Statikus prioritás: a processz élete során állandó. Starvation veszélye.
        - Dinamikus prioritás: a szükséges esetekben újra számolódik.
    - Vegyes.

Ütemezési algoritmusok:

- FCFS (First Come – First Served):
    - A ready állapotú processzek beérkezési sorrendben kapnak CPU-t.
    - Non-preemptive ütemezőknél volt használatos.
    - Előnye: megvalósítása egyszerű.
    - Hátránya: convoy effect alakulhat ki, azaz CPU lázas processz lefogja a CPU-t.
- SJF (Shortest Job First):
    - A ready állapotú processzek közül az kapja meg a CPU-t, amelyiknek a legkisebb a várható futási ideje.
    - Régi rendszerekben a programozónak kellett megadni a várható futási időt, később is használták, például printer spooling sorok kiszolgálására.
    - A várható futási idő becslésére idősorozatokkal jellemezhető feladatoknál az „aging” algoritmust használta.
- Round-Robin:
    - FCFS + időszeletes beavatkozás.
    - A ready processzek érkezési sorrendben kapnak CPU-t, majd az időszelet lejárta után beavatkozás.
    - Előnye: egyszerű, kis költségű, minden processz megkapja a CPU-t.
    - Hátránya: nem vesz figyelembe fontosságot.
- Policy Driven (ígéret vezérelt):
    - n számú processz esetén egy processz a CPU 1/n-ed részét kapja.
    - Mérni kell a rendszerben eltöltött időt (életidő), a felhasznált CPU időt, a jogosultságidőt (életidő/n), és a prioritást (jogosultságidő/CPU idő).
    - Hátrány: nem vesz figyelembe fontosságot.

    - Speciális esete a valós idejű időkiosztás.
- Sorsjegyes:
    - A processzek sorsjegye(ke)t kapnak, döntéskor sorshúzás, a nyertesé a CPU.
    - Figyelembe vehető a fontosság, több sorsjegyet kap a fontosabb processz, nagyobb esélye van.
    - Kisebb fontosságú processznek is van esélye.
    - A fontosság csak hosszabb távon érvényesül.
- Általános prioritásos:
    - A ready processzek közül a legmagasabb prioritású kapja a CPU-t.
    - A prioritásszámító képlettől függ, hogy teljesülnek-e az ütemezéssel szembeni elvárások.
    - A prioritás függhet:
        - Az eddigi CPU használati időtől.
        - A memóriaigénytől.
        - Erőforrás használati szokásoktól.
        - Külső prioritás értéktől.
        - Időszerűségtől.
        - Stb.
- Többszintes prioritás-sorokon alapuló (Multilevel (Feedback) Queue Scheduling):
    - A prioritásszámító képlet véges számú diszkrét értéket ad (prioritási szintek).
    - Több azonos prioritású processz is létezhet, másodlagos döntési algoritmus kell.
    - A prioritás dinamikus kell legyen.
    - Kell alap prioritás, tudnia kell csökkenni (CPU idő növekedésével arányosan), tudnia kell növekedni (feedback).
- UNIX időkiosztás:
    - Round Robin with Multilevel Feedback Queue jellegű.
    - A processzeknek user-mode vagy kernel-mode prioritásértéke lehet.
    - A user-mode prioritás szintekre sorolódnak a felhasználói processzek, dinamikus prioritás, van beavatkozás.
    - A kernel-mode prioritás szintekre sorolódnak a rendszer funkciók, statikus prioritás, nincs minden szintre beavatkozás.
    - Prioritás számításnál a kisebb érték jelenti a nagyobb prioritást.
    - p-usrpri = PUSER + p-cpu / const2 + const3 * p-nice (const2 = const3 = 2)
- Linux O(1) scheduler: processzek számától függetlenül konstans idő alatt végzi el a döntést.
- Linux CFS (Completely Fair Scheduler): O(logN) komplexitás.

7. Tétel: Kölcsönös kizárás. Alapfogalmak (Kölcsönös kizárás, kritikus szakasz, belépési szakasz, kilépési szakasz stb.), követelmények (biztonsági, előrehaladási, korlátozott várakozási, platform). Megvalósítás: szemaforok működési mechanizmusa.

Kölcsönös kizárás, alapfogalmak:

- Kölcsönös kizárás (Mutual Exclusion): a közös erőforrásokért vetélkedő processzek közül egy és csakis egy kapja meg a jogot az erőforrás használatra, ennek módszerei, eszközei, technikái.
- Kritikus szakasz (Critical Section): a folyamaton belüli kódrész, melyen belül a kölcsön kizárást, vagy szinkronizációt, kell megvalósítani.
- Belépési szakasz (Entry Section): a folyamaton belül az a kódrész, ahol kéri az engedélyt a kritikus szakaszba való belépésre.
- Kilépési szakasz (Leave Section): az a kódrész, ahol a processz elhagyja a kritikus szakaszt.
- Holtpont (Deadlock): az az állapot, amely akkor következhet be, amikor két vagy több folyamat egyidejűleg verseng erőforrásokért, és egymást kölcsönösen blokkolják.

Kölcsönös kizárás megvalósításával szembeni követelmények:

- 1. Biztonsági kívánalom: ha egy processz kritikus szakaszában fut, más processz ne léphessen be kritikus szakaszába.
- 2. Előrehaladási kívánalom: nem kritikus-, és nem belépési szakaszban futó processz ne befolyásolja mások belépését.
- 3. Korlátozott várakozási kívánalom: egy processz se várakozzon a végtelenségig belépésre, azért, mert egy másik, újból bejelentve igényét megelőzi.
- 4. Hardver és platform kívánalom: ne legyenek előfeltételeink a hardverre, az OS ütemezésére, stb.
- 5. Teljesítmény kívánalom: ne legyen tevékeny várakozás.

Kölcsönös kizárás megvalósítások:

- Megszakítás letiltás:
    - Belépési szakaszban letiltunk minden megszakítást, kilépési szakaszban engedélyezzük azokat.
    - Csak egy CPU esetén jó. (sérül a 4. követelmény)
    - Hiba esetén deadlock.
- Egyszerű zárolás változó:
    - Egy közösen használt zárolás változó.
    - A változó használata szintén kölcsönös kizárást igényelne.
    - Sérül az 1. és 5. követelmény.
- Egyszerű váltogatás:

    - Egy változóval nyomon követi, ki következik.
    - Sérül a 2., 4. és 5. követelmény.
- TSL instrukció (Test and Set Lock):
    - Egy processzor utasítás, amely egy megszakíthatatlan (atomi) lépésben leellenőrzi a zárolásváltozót és beállítja.
    - Sérül a 4. és 5. követelmény.
    - Hasonló a CAS (Compare and Swap) instrukció, amely atomi lépésben összehasonlít és kicserél értékeket.
- Szemafor:
    - A klasszikus szemafor egy pozitív egészt tartalmazó változó és egy hozzá tartozó várakozási sor.
    - Két atomi operáció végezhető rajta (inicializálást leszámítva): DOWN és UP.
    - DOWN:
        - Ezen blokkolódhat a hívója.
        - Ha a változó nagyobb, mint nulla, értékét csökkenti eggyel.
        - Ha a változó értéke nulla, a processz várakozó sorba kerül.
    - UP:
        - Szignáloz, hogy felébredjen egy blokkolódott processz.
        - A változó értéke növekszik eggyel, ha nulla volt, szignáloz.
    - Az 1., 2. és 4. követelményeket kielégíti, a 3. az UP megvalósításától, az 5. a várakozás megvalósításától függ.
    - Bináris spinlock szemafor:
        - Csak false és true értékeket vehet fel.
        - A blokkolódást tevékeny várakozással oldja meg.
        - Előnye, hogy nincs kontextus váltás.
        - Sérülhet az 5. követelmény.
    - Blokkolós számláló szemafor:
        - Az UP és DOWN operációk mellett megjelenik az ncount, amely a várakozási soron blokkolt processzek számát adja meg.
        - Szignállal ébreszt várakozó(ka)t.
        - Minden követelményt kielégít.
- Monitor:
    - Magasabb szintű kölcsönös kizárást és szinkronizációt biztosító mechanizmus.
    - Megvalósítása általában szemaforok használatával történik.
    - A nyelv blokkműveleti szintaktikáját használja, blokk nyitáskor szemafor DOWN, záráskor szemafor UP.

- RCU (Read-Copy-Update):
    - Nincs blokkolás, gyors, nem áll meg a folyamat.
    - Láncolt listás adatszerkezetekhez használható.
    - Másolat készül az adatszerkezetről, külön kell karbantartani.
    - Szinkronban frissül, esetleg bevárja a frissítéseket.

8. Tétel: Holtpont jelenség. Kialakulása, klasszikus példák, kialakulás feltételei. Megoldási lehetőségei.

Holtpont (Deadlock):

- Az az állapot, amely akkor következhet be, amikor két vagy több folyamat egyidejűleg verseng erőforrásokért, és egymást kölcsönösen blokkolják.
- A holtpontban lévő processzek nem tudják folytatni futásukat, az általuk lefoglalt erőforrások nem tudnak felszabadulni, az ezeket az erőforrásokat igénylő processzek szintén holtpontba kerülnek.

Holtpont kialakulása:

- Kialakulhat erőforrásokért vetélkedés közben, szinkronizáció közben, eseményekre várakozáskor, vagyis olyan esetekben, amikor kölcsönös kizárást használunk.
- Kialakulásához mindenképp egynél több kölcsönös kizárást igénylő erőforrásra van szükség.
- Kialakulás feltételei:
    - Kölcsönös kizárás: legyenek olyan erőforrások a rendszerben, amelyeket a folyamatok csak kizárólagosan használhatnak.
    - Foglalva várakozás: legyen olyan folyamat, amelyik lefoglalva tart erőforrásokat, miközben más erőforrásokra várakozik.
    - Nincs erőszakos erőforrás-elvétel: a rendszerben minden folyamat addig birtokolja a lefoglalt erőforrás(oka)t, amíg önszántából le nem mond róla.
    - Körkörös várakozás: a folyamatok az egymás által lefoglalt erőforrásokra várakoznak.

Klasszikus példák:

- Termelő-fogyasztó probléma.
- Ebédlő filozófusok problémája: öt filozófus ül egy asztal körül, mindegyik előtt egy soha ki nem ürülő tányér spagetti, a tányérok között egy-egy villa. Spagettit enni csak két villával lehet. A filozófusok felváltva esznek és gondolkodnak. Ha valamelyik megéhezik, megpróbálja megszerezni a tányérja melletti mindkét villát, ha sikerül, eszik, majd leteszi a villákat és folytatja a gondolkodást.
- Alvó borbély probléma: borbélyüzletben van egy vagy több borbély és ugyanannyi borbélyszék, ezen kívül N szék a várakozó vendégeknek. Ha nincs vendég, a borbély alszik a borbélyszékében. Ha vendég jön, felkelt egy borbélyt. Ha vendég jön, de minden borbély dolgozik, ha van üres várakozószék, leül és vár, ha nincs, távozik.

Megoldási lehetőségek:

- Strucc megoldás:
    - Nem igazi megoldás. Nem törődünk a problémával, hiszen kicsi az előfordulási esélye.
    - Előnye, hogy költsége nulla.
    - Hátránya: ha mégis előfordul, előbb-utóbb „lefagy” a gép.
    - Kritikus rendszerekben nem használható.
- Detektálás és megszüntetés:
    - Fontos minél hamarabb detektálni.
    - Ellenőrzés rendszeres időközönként:
        - Gyakran: hamarabb detektál, költséges.
        - Ritkán: később detektál, alacsony költség.
    - Ellenőrzés eseményhez kötötten:
        - Minden erőforrás lefoglalás után: túl nagy költség.
    - Detektálás:
        - Egypéldányos erőforrások esetén a körkörösség figyelésével.
        - Többpéldányos erőforrások esetén a kielégíthető processzeket megkereső algoritmussal.
    - Megszüntetés:
        - Erőforrás elvételével: általában a processz sérülése a következmény.
        - Processz lelövésével: minden holtpontban lévő lelövése túl drasztikus, egyesével lelőni és újra detektálni költséges és döntési algoritmus kell.
    - Kármentés: a processzekhez közbenső mentési pontok tárolhatók, nagy költségigény.
- Megelőzés:
    - Megelőzés valamelyik kialakulási feltétel kiiktatásával:
        - Nincs teljes körű megoldás.
        - Kölcsönös kizárás nem iktatható ki, de spooling technikával csökkenthető az azt igénylő erőforrások száma.
        - Foglalva várakozás kiiktatásához egy processz egyszerre kell kérje az összes erőforrását, túl nagy hatékonyságromlást eredményez.
        - Erőszakos erőforrás elvétel csak menthető állapotú erőforrások esetén iktatható ki.
        - Körkörösség kialakulása sorrendi foglalással kiiktatható, de nem minden esetben.
    - Megelőzés állandó mérlegeléssel:

        - Minden erőforrás kéréskor megvizsgálni, hogy biztonságos állapotban marad-e a rendszer.
        - Nem használható minden erőforrás esetén.

9. Tétel: A memóriamenedzselés feladatai. Memória mint erőforrás. Címleképzés fajták. A fix partíciós és a változó partíciós memóriakezelés.

Memóriamenedzselés feladatai:

- A memóriamenedzselés fontos OS-beli feladat, de szoros az együttműködése a hardverrel is.
- Két fő feladat: a memória allokálás, és a címleképzés segítése.
- Memória allokálás:
    - Statikus: processz keletkezésekor címtartományt és memóriát kell biztosítani.
    - Dinamikus: a processz futása során további memóriát igényel, ami címtartomány bővítéssel járhat.
    - Mai rendszerekben kisöprési/kilapozási területek biztosításával és leképzési táblák létrehozásával/bővítésével jár.

A memória, mint erőforrás:

- A processz szemszögéből erőforrás, amire szüksége van.
- Véges mennyiségű, több példányos erőforrás, ahol egy cellát egy időben csak egy processz használhat.
- Erőforrás kezelés külön OS komponens, a Memory Management (MM) segítségével.
- Az MM feladatai:
    - Memória foglalások kezelése:
        - Memória allokálása a processz számára.
        - Processz számára lefoglalt, és szabad memória területek nyilvántartása.
        - Lehetőség biztosítása „extrém használatra”. (pl.: olyan processzek, amik több memóriát igényelnek, mint az összes fizikai memória)
    - Memória területek védelme: processzek ne férjenek hozzá egymás területeihez.
    - Osztott memória biztosítása: processzek használhassanak közös területeket.

Címleképzés, címkötődés:

- A program instrukcióiban a címek programkészítés közben keletkeznek, de a memóriafoglalás csak a processz létrejöttekor történik. Nem lehet tudni, futtatáskor mely memória területek lesznek szabadok.
- Mikor történik címkötödés?
    - Fordítás során: ha van kötődés, abszolút-, ha nincs, relatív címek generálódnak a tárgymodulokban.

    - Taszképítés során (linkeléskor): ha van kötődés, abszolút-, ha nincs, relatív címek generálódnak a végrehajtható modulokban. Relatív címek esetén a program áthelyezhető.
    - Betöltés során: ha van kötődés, az áthelyezhető címek betöltéskor abszolút címekké válnak, ha nincs, a processz kódszegmensének címei még relatív címek.
    - Végrehajtás során: dinamikus a kötődés. A processz kódjában az instrukciók logikai címeket tartalmaznak, melyeket a CPU átképez fizikai címekre.
- Valós címzésű rendszerek:
    - Programkészítéskor meghatározott címekre van szükség.
    - A processz csak a meghatározott címekre tud betöltődni, amíg az nem szabad, nem tud futni.
    - Csak single taszkos környezetben célszerű.
- Relatív címzésű rendszerek:
    - Programkészítéskor relatív címek keletkeznek.
    - A relatív címek egy bázis címhez képesti címek.
    - A bázis cím a program betöltődésekor határozódik meg.
    - Minden címhivatkozás leképzésre szorul, ehhez hardveres támogatást a CPU MMU egysége ad.
    - CPU => Logikai cím => MMU => Fizikai cím => Memória.
    - Előnye, hogy a processzek áthelyezhetők.
    - Hátránya, hogy a processz csak egybefüggő területen helyezkedhet el, illetve csak a fizikai memóriánál kevesebbet használhat.
- Virtuális címzésű rendszerek:
    - A programok saját virtuális címkészlettel rendelkeznek, virtuális címeket használnak.
    - Az OS minden virtuális címhez biztosít egy tárolót, nem feltétlenül a memóriában.
    - Nyilvántartásához a címtartományt darabokra bontjuk, a nyilvántartásba csak a darabok kezdetei kerülnek.
    - A címhivatkozások leképzésre szorulnak, hardveres támogatást nyújt a CPU MMU egysége. A címképzés a processz számára transzparens.
    - Előnye, hogy a processz memóriaterületeinek nem feltétlenül kell egybefüggőnek lenni, több darabban is lehet a memóriában, vagy más tárolón. A programok tetszőleges méretű virtuális címtartományt használhatnak.
    - Hátránya a bonyolult címképzés.

Fix partíciós memóriakezelés:

- A memóriaterület (az OS által lefoglaltat leszámítva) fix méretű és számú partíciókra (nem feltétlenül egyforma méretűek) van felosztva. (bootoláskor meghatározott)

- Egy partíción egy processz helyezkedik el.
- Valós címzésű rendszereknél partíciónként várakozó sorok, relatív címzésűnél közös várakozó sor a processzeknek.
- Allokálás: a processz megkap egy teljes partíciót. Dinamikus allokáció nem okoz gondot, amíg a processz ki nem növi a partíciót, ha kinövi, nagyobba helyezik.
- A legnagyobb partíció méretétől nagyobb processzek csak overlay technikával futtathatók.
- A partíciók védelme a határcímek figyelésével megoldható.
- Osztott memória csak OS területen biztosítható, hivatkozás csak rendszerhívásokon keresztül lehetséges.
- Előnye, hogy multitaszkos környezetben is használható, illetve költségigénye kicsi.
- Hátránya, hogy rossz a helykihasználtság, belső töredezettség van, és a processzek száma korlátozott.
- Belső töredezettség: egy memória partíción belül kihasználatlan területek vannak.
- Overlay technika: a programok egy része „szétvágható” olyan részekre, amelyek időben elkülönülten dolgoznak.

Változó partíciós memóriakezelés:

- Egy processz egy partíció, de a partíciók száma és mérete nem rögzített.
- Relatív címzésű rendszereknél használható.
- Allokálás:
    - Elhelyezés egy üres partícióba, amelyből így lesz egy foglalt és egy új üres.
    - Elhelyezési stratégiák: first fit, worst fit, best fit.
    - Külső töredezettség: kihasználhatatlanul kicsi memória partíciók.
    - Dinamikus allokációnál, ha a következő partíció szabad, megnövelhető, ha nem, akkor át kell helyezni.
    - Processz befejezésekor össze kell vonni az egymást követő szabad partíciókat.
- Partíciók és szabad területek nyilvántartása:
    - Bittérképes: minden blokkról 1 bit jelzi, hogy szabad-e.
    - Láncolt listás: minden partíció láncolt listán bejegyezve.
    - Buddy rendszer: csak 2 hatványai méretű partíciók, minden mérethez külön lista.
- Nagyméretű processzek itt is csak overlay technikával használhatók.
- Védelem a processz kontextusban tárolt határcímek figyelésével megoldható.
- Osztott memória csak OS területen, rendszerhívásokkal kezelhető.
- Előny: kicsi belső töredezettség, nagyobb helykihasználtság.
- Hátrány: processzek csak egybefüggő területen helyezhetők el, több OS feladat.

10. Tétel: Lapozós virtuális memóriamenedzselés működése. Allokálás, nyilvántartás, címleképzés. Laphiba fogalma, kezelése, kilapozási stratégiák. Előnyök, hátrányok.

Virtuális memóriakezelés:

- Virtuális címzést használ.
- Virtuális címeknek fizikai rekesz a memóriában, vagy a háttértáron.
- A címleképzéshez nyilvántartás kell, ennek méretének csökkentéséhez a virtuális címtartományt blokkokra kell osztani.
- Blokkokra osztás fajtái:
    - Lapozásos.
    - Szegmentálásos.
    - Vegyes.

Lapozós virtuális memóriakezelés:

- Allokálás:
    - A processz memória területét azonos méretű blokkokra, lapokra osztjuk.
    - A fizikai memória ilyen méretű lapkereteket tartalmaz.
    - A lapok bármely szabad lapkeretben, vagy háttértárolón elhelyezhetők.
- Címképzés:
    - Az OS minden processzhez létrehoz egy laptáblát.
    - A laptáblában annyi bejegyzés, ahány lapra feloszthető a processz memória területe.
    - Laptábla bejegyzés tartalma:
        - A lap a memóriában van-e (valid/invalid).
        - Helye a memóriában (lapkeret címe).
        - Helye a háttértáron (rendszerfüggő).
        - Egyéb adatok.
    - A processz dinamikus kontextusában tartalmazza a laptábla kezdőcímét.
    - Ha egy hivatkozott lap nincs a memóriában, laphibáról beszélünk.
    - A címképzés gyorsításához hardveres támogatást ad az MMU.
- Laphiba:
    - A hivatkozott lap nincs betöltve lapkeretbe.
    - Laphiba esemény generálódik, a processz blokkolódik, az OS pager komponense kezeli.

    - A pager betölti a lapot valamelyik üres lapkeretbe, ha nincs üres, kilapozással csinál helyet.
    - Laphiba kezelése után a processz folytathatja futását a laphibát kiváltó utasítással.
- Védelem: a leképzési mechanizmus biztosítja, hogy a processz csak a saját memóriaterületeihez fér hozzá.
- Osztott memória: közös lapokkal megvalósítható.
- Előnyök:
    - Nincs külső töredezettség.
    - Kicsi belső töredezettség.
    - Könnyű elhelyezés, bármely lap bármely lapkeretben elhelyezhető.
    - Nagy memóriaigényű processzek is kezelhetők.
- Hátrány: a címképzés ideje, a nyilvántartás mérete és kezelési ideje megnőtt.
- Gyorsítási lehetőségek:
    - Nagyobb lap és lapkeret méret: kisebb laptábla, kevesebb laphiba, hosszabb a betöltés.
    - Többszintű laptáblák: bonyolultabb címképzés, nagy processzhez is kis laptábla.
    - Invertált laptábla: nem processzenként egy laptábla, hanem egy közös, aminek annyi bejegyzése van, ahány lapkeret.
    - Asszociatív tárak: laptábla mellett párhuzamosan asszociatív tárban is keres.

Lapkezelés:

- Working Set: azon lapok halmaza, amelyeket a processz egy időintervallumban használ.
- Belapozási stratégiák:
    - Igény szerinti: akkor lapozzuk be a lapot, ha hivatkozás történik rá.
    - Előérző: lokalitás elveket figyelembe véve, hivatkozotton kívül még néhány lapot belapoz.
    - Mohó: belapoz, amennyit csak tud.
- Kilapozási stratégiák:
    - Ha a processznek már nincs szabad lapkerete, a pager-nek ki kell lapoznia egy lapot a memóriából.
    - Lokális kilapozási stratégia: csak a processz saját lapkeret készletéből lapoz ki.
    - Globális kilapozási stratégia: más processzek lapkeret készletéből is kilapozhat.
    - FIFO:
        - A legrégebben belapozott lapokat lapozza ki először.
        - Hátránya, hogy a régen belapozott, de gyakran használt lapok is kilapozódnak.

    - Második esélyes FIFO (óra algoritmus):
        - Belapozási sor (körkörös lista), minden laphoz hivatkozás bit.
        - Legrégebben belapozott lapra, ha nem történt hivatkozás, kilapozódik, ha történt, a sor végére kerül.
    - LRU (Least Recently Used):
        - Hivatkozások ideje alapján sor.
        - Ha egy lapra hivatkozás történik, a sor végére kerül.
        - Legrégebben használt lap lapozódik ki.
    - LFU (Least Frequently Used):
        - A legritkábban használt lap lapozódik ki.
        - Nyilván kell tartani a hivatkozások számát.
    - NRU (Not Recently Used):
        - Nem túl gyakran hivatkozott lapok lapozódnak ki.
        - Megvalósítások: utolsó 8 időintervallum históriája, módosítás bites.
- A kilapozás gyorsítása érdekében a pager szabad CPU időben a módosult lapokat kiírja a háttértárra, így kilapozáskor nem kell a mentéssel foglalkozni.

11. Tétel: Eszköz driver-ek funkciói, felépítésük. Eszköz osztályok.

Eszköz osztályok:

- Karakteres eszközök:
    - Strukturálatlanul kezelhető.
    - A driver fölött lehet szűrés.
    - Diszkek is kezelhetők ilyen módon.
    - Pl.: billentyű, soros port, képernyő, stb.
- Strukturált (blokkos) eszközök:
    - Blokkos I/O-val kezelhetők.
    - Fájlrendszer is kialakítható rajtuk.
    - Buffer-cache gyorsító mechanizmuson át kezelhetők.
    - Pl.: diszkek.

I/O rendszer:

- Felhasználó szemszögéből:
    - Szimbolikus neveken lát eszközöket, lát fájlokat, hierarchiát.
    - Felhasználói felülettel kezeli.
    - Ismer tulajdonossági és hozzáférési kategóriákat, beállít ilyeneket.

- Programozó szemszögéből:
    - Stream-eket lát.
    - Nyithat és zárhat csatornákat.
    - Adatokat mozgathat azonosított csatornákon.
- OS szemszögéből:
    - Külön I/O alrendszere van.
    - Feladata: vezérlés, interfész, védelem és menedzsment biztosítás.
    - Alapelvek:
        - Réteges struktúra.
        - Eszközfüggetlenség biztosítása.
        - Célszerűen elosztott hibakezelés.
        - Szinkronitás-aszinkronitás biztosítása.
        - Osztható és dedikált eszközök, fájlok is kezelhetők.
- Rétegei:
    - Felhasználói réteg: rendszerhívások az eszközök kezeléséhez.
    - Eszközfüggetlen réteg: elrejti az eszközök különbségeit, cserélhetőséget, magasabb szintű kezelhetőséget biztosít.
    - Eszközfüggő réteg: eszköz meghajtókat, más néven drivereket, tartalmaz.

Eszköz driverek:

- Szerepe egy eszközfajta kezelése, adatmozgatás a központi memória és a kontroller között.
- Rutinkészlet, táblázatok és pufferek alkotják.
- Az I/O alrendszer legalsó része. Felette lévő rétegből hívással, alatta lévőből megszakítással szólítható.
- Alapvetően három részből állnak:
    - Autokonfigurációs- és inicializálórutinok: driver betöltésekor hívódnak.
    - I/O kérelmeket kiszolgáló rutinok: felső rétegből call jelleggel hívódnak.
    - Megszakítás kiszolgáló rutinok: alsó rétegből aszinkron módon hívódnak.
- Minden konkrét eszközfajtához saját driver szükséges.
- Drivereket az eszközgyártó készíti, nem feltétlenül az OS része.

12. Tétel: Fájlrendszer megvalósítási feladatok. Jegyzékszerkezetek. Szabad blokk menedzselési lehetőségek. Fájl attribútum rögzítési lehetőségek (ezen belül fájl testet képező blokkok rögzítési lehetőségei).

Fájl:

- Valamilyen szempontból összetartozó adatok névvel ellátva.
- Struktúrák, szervezettség:
    - A fájl bájtok sora: nincs strukturáltság, a processzek strukturálhatnak.
    - A fájl rekordok sora: fix, vagy változó rekordhossz, a rekordok csoportosítva vannak, mezőket tartalmaznak.
    - A fájl indexelt szervezésű rekordokból áll: a rekordok nem feltétlenül egyforma hosszúságúak, a gyors keresést egy vagy több kulcsmezővel biztosítják.
- Elérés:
    - Soros, szekvenciális: egy adatelem eléréséhez az előtte lévőkön át kell jutni.
    - Random, vagy direkt: egy adatelem elérése független a többitől. (seek)
- Típusok:
    - Közönséges fájlok.
    - Jegyzékek: bejegyzéseket tartalmaznak további fájlokról.
    - Speciális fájlok. (pl.: UNIX-ban eszközt azonosítanak)
- Attribútumok:
    - Név.
    - Készítési-, utolsó módosítási dátum.
    - Tulajdonossági, védelmi információk.
    - Szervezettségi adatok.

Fájlrendszer:

- Megoldandó feladatok:
    - Rögzíteni, hogy egy fájlhoz mely blokkok, milyen sorrendben tartoznak.
    - Szabad blokkok nyilvántartása, menedzselése.
    - Fájlokhoz attribútumok rögzítése, jegyzékstruktúra kialakítása.
- Blokk hozzárendelés fájlokhoz:
    - Folytonos allokáció:
        - A fájl blokkjai egymás után helyezkednek el, folytonosan.
        - Csak a kezdőblokk címének és a hossznak a tárolására van szükség.
        - Előnye, hogy egyszerű a nyilvántartása, és a blokkok keresése.
        - Hátránya, hogy nagy a töredezettség, a kihasználatlan terület, problémát jelent a hozzáfűzés a fájlhoz.
    - Láncolt lista allokáció:
        - Fájl blokkjainak elhelyezése több darabban, nem feltétlenül folytonosan.
        - Az első blokk címének tárolása szükséges.
        - A blokkokban mutató a következőre, az utolsóban véget jelző NULL pointer.
        - Előnye, hogy egyszerű, nincs töredezettség, könnyű a hozzáfűzés.
        - Hátránya, hogy a fájl blokkjainak eléréséhez végig kell iterálni a listán.
    - Láncolt listás allokáció indextáblával (FAT):
        - A fájl blokkokra osztva van elhelyezve a partíción, nem feltétlenül folytonosan.
        - Az első blokk címét kell tárolni.
        - A következő blokkra mutató pointer a blokk helyett az indextáblába kerül.

        - Az indextábla a partíció egy fix helyén található, a partíció minden blokkjához van benne egy bejegyzés.
        - Előnye, hogy nincs töredezettség, egy blokk megkereséséhez csak az indextáblán kell végig követni a mutatókat.
        - Hátránya, hogy nagy partíciókon nagy az indextábla (cluster-ek létrehozásával megoldható), sérülékeny (az indextábla másolatával megoldható).
    - Az i-node allokáció (pl.: UNIX fájlrendszere):
        - Fájlok elhelyezése blokkokra osztva a partíció tetszőleges blokkjaiba.
        - Minden fájlhoz egy többszintű táblázat arról, mely blokkja, mely blokkba van elhelyezve.
        - A fájl adatai, a blokkcímek fő táblázata és egyéb adatok egy i-node-ban találhatók. A jegyzékben csak a fájl neve és az i-node száma van.
        - Az i-node-k a partíció egy elkülönített területén vannak elhelyezve.
        - Előnye, hogy nincs töredezettség, egy blokk megtalálásához kevesebbet kell olvasni, mint az előző módszereknél, a fájl helyére vonatkozó információk egy helyről elérhetők.
        - Hátránya, hogy a szabad blokkok külön nyilvántartást igényelnek.
- Szabad blokkok nyilvántartása:
    - Bittérképes:
        - A partíció meghatározott területén helyezkedik el a bittérkép.
        - Egy-az-egy megfeleltetés a bitek és a blokkok (vagy cluster-ek) között.
        - A bit 0 (szabad) vagy 1 (foglalt) értéke jelzi a foglaltságot.
        - Gyorsítható in-core memória másolattal.
        - Könnyen módosítható, szabad blokkok keresése az előzőtől indítva.
    - Láncolt listás:
        - Fix helyről indulva egy blokk mutatókat tartalmaz szabad blokkokra.
        - Viszonylag kevés blokk lánca tartalmazhatja az összes szabad blokkot.
        - Módosítás nehezebb.
- Jegyzékstruktúra:
    - A jegyzék egy fájl, ami bejegyzéseket tartalmaz más fájlokról.
    - A jegyzékhez is tartoznak blokkok, ezekben lehetnek állandó, vagy változó hosszúságú bejegyzések.
    - Bejegyzések tartalma:
        - Fájlnév.
        - Az i-node szám.
        - Kezdőcím, hossz.
        - Egyéb attribútumok, stb.
    - A bejegyzések struktúrája befolyásolja a jegyzéken belüli keresést, ez lehet: lineáris, vagy hash-táblás.
    - Keresések gyorsítására lehetőségek:
        - Gyökérjegyzék tartalma mindig a memóriában.
        - Munkajegyzék tartalma mindig a memóriában.
        - A jegyzék belső szerkezetét jól kereshetővé tenni.
        - Fájl megnyitásakor FCB (File Control Block) létrehozása.