„A” tételsor

(Informatikai alapok, Számítógép-architektúrák I., Operációs rendszerek, Számítógép- hálózatok)

1. A processzor felépítése, utasításkészlete. Utasítások szerkezete, címzési módok. Utasításszámláló és utasítás-regiszter. Az utasítás-feldolgozás elemi lépései.

A processzor a számítógép központi feldolgozó egysége (CPU). A processzor egyik fő része az **ALU** (aritmetikai és logikai egység), ez az egység felelős a numerikus (pl összeadás, kivonás) és logikai műveletek (pl. ÉS, VAGY) elvégzésért. Másik fő része a CPU-nak a **regiszterek**, amelyek kisméretű, gyors elérésű írható-olvasható memóriák. Ezekből több is található a processzoron belül, nagyságrendileg akár 1000 darab általános célú regisztert is tartalmazhat. A processzor fontos része még a **cache memória**, ami kis tárkapacitású, és nagyon gyors átmeneti tároló. A **CPU vezérlőegysége a CU** vezérli a számítógép részegységeit (pl. perifériák megszakítás kérelme).

Egyes platformok **utasításkészlete** „komplex” utasításokat is tartalmaz. Egyetlen komplex utasítás elvégzi, ami más számítógépeken sok normál utasítás feladata lenne. Az utasításkészlet tehát azt határozza meg hogy mik azok az elemi szintű gépi kódú utasításoknak, amelyek végrehajtására a processzor a legalsó (hardver) szinten képes. A komplex utasításkészlettel rendelkező számítógépek (**CISC**) sok olyan, specializált utasítással bírnak, melyeket a programok csak ritkán használnak. A csökkentett utasításkészletű számítógépeket (**RISC**) úgy egyszerűsítették le, hogy csak azokat az utasításokat valósították meg bennük, melyek gyakran előfordulnak a programokban.

Egy utasítás a **műveleti kódból**, a **címrészből** és a **módosító részből** áll. **A címzési mód lehet pl. relatív vagy abszolút**. A processzor órajelre hajtja végre az utasításokat, pl. ha 2GHz-es egy processzor az azt jelenti hogy másodpercenként 2 milliárd elemi műveletet képes végrehajtani.

Feldolgozáskor a processzor az **utasításszámláló** tartalma alapján kikeresi az utasítást és átviszi a vezérlőegység **utasításregiszterébe (IR).** Az **utasítás feldolgozásakor** a CPU kiolvassa az utasítást, dekódolja, órajelre végrehajtja majd az eredményt beírja a memóriába.

2. A verem fogalma és működése, a veremmutató regiszter. A vermet kezelő utasítások. A verem alkalmazása szubrutinok kezelésében. A szubrutinra vonatkozó utasítások.

A verem a számítástechnikában **lehet szoftveres**, pl. egy programozáskor használt **adatszerkezet** az adatok tárolására, **de létezik hardveres, konkrét megvalósítása is**. A verem valamilyen **véges számú azonos típusú elem tárolására alkalmas**. Különleges jellemzője hogy a benne tárolt adatokat nem lehet csak úgy tetszőlegesen elérni, mindig vagy csak a verem tetején vagy alján lévő adatot lehet elérni.

Alapvetően két féle verem megoldás létezik, a **LIFO** esetében (last in, first out) az utoljára a verembe helyezett elemet lehet elsőként kivenni. A **FIFO** verem esetén (first in, first out) pedig ahhoz az elemhez lehet elsőként hozzáférni amit legelsőként tettünk a verembe. A szemléltetés és egy ócska példa kedvéért ha veszünk egy üres Pilóta kekszes zacskót amit csak az egyik végén vágtunk ki, majd egyesével belerakjuk a kekszeket akkor az kvázi LIFO verem lesz mert az utoljára beletett kekszet tudjuk elsőként kivenni.
Veremkezelő utasításból kettőt ismerünk: van a **push** és a **pop**. LIFO verem esetén push utasításkor új elem kerül a verem tetejére, pop utasítás esetén pedig kivesszük a verem legfelső elemét.

A verem a memóriában helyezkedik el, éppen ezért a címe, mérete változó, ezért szükség van egy **veremmutató regiszterre** ami egy pointerként mutat a verem aktuális memóriacímére hogy mindig el tudjuk érni a verem tartalmát.

A veremnek a **szubrutinok feldolgozásakor nagy szerepe van**, a processzor azokat a **memóriacímeket menti el** verembe, ahova az egyes eljárások befejeztével visszatér, illetve a szubrutinok kezdetén elmeni azokat a **regisztereket amelyeknek az értéke meg fog változni** a végrehajtás során.
Szubrutinokra vonatkozó utasítások a **CALL** (hívás) és **RET** (visszatérés).

3. A Neumann-elvek. Utasítás- és adatfolyam (SISD, SIMD, MISD és MIMD architektúrák). Adatok számítógépes ábrázolása (fixpontos, lebegőpontos, BCD, vektoros adatok, karakterek).

Neumann elvek:

- Teljesen elektromos működés
- Kettes számrendszer használata
- Belső memória használata
- A programok és az adatok egy memóriában tárolódjanak (tárolt program elve)
- Soros utasítás végrehajtás (mai gépek már párhuzamos végrehajt használnak)
- Programozhatóság hogy ezáltal univerzális gépünk legyen (Turing-gép)

A **SISD, SIMD, MISD és MIMD** rövidítések arra vonatkozóan hogy egyidejűleg hány utasítást és hány adatfolyamot képes kezelni az architektúra.

SISD = single instruction, single data stream = egy utasítás, egy adatfolyam
MIMD = multiple instruction, multiple data stream = több utasítás, több adatfolyam
A rövidítések magát magyarázzák ha megjegyzed mi mit rövidít...
Leegyszerűsítve **a fixpontos számábrázolást** egész számok tárolására használjuk. Azért fix mert a tizedespont helye fixen a szám végén van, pl: 123.

A **lebegőpontost** (float) törtszámok, valós számok tárolására is alkalmasak. Azért lebegőpontos mert a tizedespont "lebeg", a számjegyen belül bárhova kerülhet, pl: 1.23 12.3 123.

A BCD kódolás (**binárisan kódolt decimális** – Binary-Coded Decimal) a decimális (10-es számrendszer beli) számok kódolási formája, minden számjegyet egy bitsorozat ábrázol, pl: 82: 1000 0010

Vektoros adatok, például egy grafika tárolásakor geomatriai primitíveket: pontokat, egyeneseket, görbéket, sokszögeket használunk a kép leírására. Például egy egyenest vektorgrafikusan úgy tárolunk hogy a kezdő és a végpontot tároljuk csak mert ebből bármikor előállítható az egyenes bármekkora méretben. Ennek ellentéte a rasztargrafika ahol pixelekkel írjuk le az egész egyenest és ebből adódan ha jól belenagyítunk akkor jó "pixeles", szemcsézett lesz a képünk.
A vektor processzorok (ha egy processzor utasításkészlete tartalmaz ilyet) nagyon meg tudnak gyorsítani egyes munkafolyamatokat. Mai gyakorlatilag összes korszerű processzor tudja ezt...

Katakterek kódolására több féle szabvány is létezik,legelterjedtebb kettő az **ASCII** és az **Unicode** (pl: UTF-8 kódolás). Az ASCII mindössze 128db különböző karaktert tud reprezentálni, ez arra elég volt hogy lefedi az angol abc-t, a fennmaradó karakterek pedig vezérlő funkciót töltenek be (pl: DEL = delete, BS = backspace).

Az Unicode több mint 100.000 karaktert tartalmaz, amellyel gyakorlatilag a valaha volt összes modern és történelmi abc-t lefedi. Azért Unicode a neve, mert univerzális kódolási ipari szabványként hozták létre. Szoftverfejlesztői oldalról nézve rengeteg szívástól megkíméli a fejlesztőket az Unicode, mindig használjunk Unicode-ot. Használatával kivédhetőek a bosszantó karakterkódolási hibák, pl tipikusan az 'ő' 'ű' betűnél szétcsúszik egy rakás másik kódolási szabvány, és sorolhatnám...

4. Az utasítás-feldolgozás gyorsítása párhuzamosítással. A pipelining lényege, szuperskalár processzorok. Fellépő problémák és kezelésük.

A párhuzamos utasítás feldolgozásnak az a nagy előnye hogy az arra alkalmas programoknak a **végrehajtását meggyorsítja**. Sajnos **a legtöbb alkalmazás nem tudja kihasználni a párhuzamosítás előnyeit**. Például ha van egy 2 magos és egy 8 magos processzor akkor a 8 magoson nem fognak 4x gyorsabban végrehajtódni a programok, ez csak egy elméleti maximum...

Körülbelül a 70-es években jöttek rá, hogy egy feladat megoldását nem csak szekvenciális módon lehet meghatározni. A soros feldolgozásnál időben egymás után történik a feldolgozás. Párhuzamos feldolgozás esetén időben egyszerre több szálon is folyhat a végrehajtás. Általánosan elmondható hogy bonyolultabb feladatok esetén azok komplexitását párhuzamos feldolgozással lehet hatékonyabban megoldani.

A **pipelining** (futószalagos feldolgozás) neve onnan adódik hogy a gyárban futószalagos termelésnél minden munkás csak egy-egy részletmunkát végezhet, a szakmunka betanított munkává alakul, gépszerű, egyszerű műveletekből áll. A processzorban is pont ugyanez történik, csak itt a **futószalag** egy adott végrehajtó egység, például egy **fixpontos vagy lebegőpontos művelet-végrehajtó egység**. Az egyes részműveleteket **időben párhuzamosan** hajtjuk végre. Ha az adott **hardveregység felszabadul**, akkor ezt igénybe vehetjük **egy következő utasítás** elemi lépésének végrehajtására.

Ha egy gépi ciklus alatt egy processzor több utasítást is képes végrehajtani azt **szuperskalár processzornak** nevezzük. Mivel több szálon történik a végrehajtás, nagyon fontos ezek **szinkronizációja**. Ennek megvalósítása nagy kihívás hardveresen és szoftveresen programozás közben is ha szálkezeléssel foglalkozunk...

Egy tipikusan fellépő probléma: vannak olyan típusú utasítások, amiknek teljesen végig kell haladnia a futószalagon, mielőtt a végrehajtás folytatódhat. Különösen **feltételes elágazások**nak kell tudniuk egy előzetes utasítás eredményét eldönthető, hogy melyik ágat kell választani. Például, egy olyan utasításban, ami azt mondja „ha x nagyobb mint 5, akkor tedd ezt, különben tedd azt” először meg kell várni, amíg az x változó konkrét értéket kap, mielőtt eldönthető lenne, hogy ezután melyik ágon kell folytatni a végrehajtást.

A megoldás, vagy az egyik lehetséges megoldás a **spekulatív végrehajtás**, ami elágazás jövendölésként van nyilvántartva. Valóságban, az elágazás egyik oldala sokkal gyakrabban lesz meghívva, mint a másik, így sok esetben helyes ha egyszerűen azt mondjuk: „x valószínűleg kisebb lesz ötnél, kezdd el annak a végrehajtását”. Ha a jóslatról kiderül, hogy helyes, hatalmas mennyiségű időt meg tudunk spórolni. A modern processzorkialakításokban már meglehetősen összetett jósló-előrejelző rendszereket alkalmaznak, amik figyelembe veszik a korábban végrehajtott elágazások eredményeit, hogy nagyobb pontossággal jósolják meg a jövőt. Gyakorlatilag amíg a processzor vár az adatra, az üresjáratban előre elkezd kiszámolni olyan ágakat amiken majd feltételezhetően folytatódni fog a végrehajtás.

5. Az aritmetikai-logikai egység és regiszterei (akkumulátor, flag). Fixpontos és lebegőpontos műveletek, ezek végrehajtásának egységei. Logikai műveletek.

Az **ALU** (arithmetic and logic unit) a processzor egy részegysége. Ez az egység felelős a numerikus (pl összeadás, kivonás) és logikai műveletek (pl. ÉS, VAGY) elvégzésért. Az összeadás-kivonás mellett képes fixpontos és lebegőpontos szorzásra és osztásra is.

Leegyszerűsítve **a fixpontos számábrázolást** egész számok tárolására használjuk. Azért fix mert a tizedespont helye fixen a szám végén van, pl: 123. A **lebegőpontost** (float) törtszámok, valós számok tárolására is alkalmasak. Azért lebegőpontos mert a tizedespont "lebeg", a számjegyen belül bárhova kerülhet, pl: 1.23 12.3 123. Két fixpontos vagy lebegőpontos szám között végezhető műveletek (összeadás, kivonás, szorzás, osztás) a kettes számrendszer szabályai szerint bitenként történik.

Az aritmetikai egység tartalmaz egy **akkumulátor** (AC) nevű **regisztert**, amelyben a műveletek végzése során a közbenső eredményeket tárolja.

Az ALU flagjei 1 biten 0 vagy 1 értéket vehetnek fel. Az **Overflow flag** (túlcsordulás) akkor billen 1-be ha például egy szorzás után az eremény nagyobb mint a maximális tárolható érték. A **Sign flag** (avagy Negative flag) az előjelet tárolja és akkor billen 1-be ha negatív a végrehajtás eredménye. Ezen kívül ismerünk még **Zero, Carry**, stb... flageket.

A logikai műveletek eredménye mindig vagy IGAZ vagy HAMIS. Logikai kapu például az AND, OR, NOT (negálás), XOR (kizáró vagy). Igazságtáblán egyszerűen ábrázolhatóak... Ezeknek a kapuknak hardveres megvalósítása tranzisztorokkal lehetséges manapság már elképesztően parányi, nagyságrendileg nanométeres (mm egymilliárdad része) méretekben.

6. A vezérlőegység feladata és jelei, vezérlési pontok. Huzalozott és mikroprogramozott műveleti vezérlés. CISC és RISC processzorok.

A **vezérlőegység** (CU – control unit) a processzor egyik egysége, feladata a teljes számítógép részegységeinek (aritmetikai egység, memória, kommunikációs eszközök, háttértár és perifériavezérlések) irányítása, összehangolása.

A vezérlőegység jelei gyakorlatilag négyszögjelek, lehet a **processzor belső vezérlőjele** ami a processzor részegységeit koordinálja, illetve vannak a **külső vezérlőjelek** amik a memória, I/O kezelést hangolják össze.

Huzalozott műveleti vezérlés esetén egy kombinációs hálózatból (különféle kapuk összekapcsolva egymással egy nagy hálózatban) alakítják ki fizikailag a vezérlést. Ennek az az előnye hogy **nagyon gyors** viszont gyakorlatilag **karbantarthatatlan** a fix architektúra miatt. Egy kis módosítási igény is nagy kihatással lehet az egész kombinációs hálóza logikájára, szélsőséges esetben akár újra kell tervezni az egész kapu hálózatot. Ez a vezérlés már a múlté… Helyette amit **napjainkban** is használunk az a **mikroprogramozott** műveleti vezérlés. Alacsony szintű szoftver kód felel a műveleti vezérlésért, ezt könnyebb megtervezni, implementálni és karbantartani is.

Egyes platformok **utasításkészlete** „komplex” utasításokat is tartalmaz. Egyetlen komplex utasítás elvégzi, ami más számítógépeken sok normál utasítás feladata lenne. Az utasításkészlet tehát azt határozza meg hogy mik azok az elemi szintű gépi kódú utasításoknak, amelyek végrehajtására a processzor a legalsó (hardver) szinten képes. A komplex utasításkészlettel rendelkező számítógépek (**CISC**) sok olyan, specializált utasítással bírnak, melyeket a programok csak ritkán használnak. A csökkentett utasításkészletű számítógépeket (**RISC**) úgy egyszerűsítették le, hogy csak azokat az utasításokat valósították meg bennük, melyek gyakran előfordulnak a programokban.

7. A központi tár szerepe, áramköri megvalósítása. ROM és RAM áramkörök típusai. Dinamikus RAM belső felépítése. Átlapolt memóriakezelés.

Háttértárat (mass storage) nagy mennyiségű adat és az utasítások-programok tárolására használjuk. Ez egy tartós tár, bináris formátumba tárolja az adatokat és áramkörileg úgy van kialakítva hogy akkor is megőrzi az adatot amikor az eszköz nincs áram alatt (pl. mágneses tár: merevlemez). Napjainkban tipikusan merevlemezt (HDD) vagy (SSD) meghajtót használnak erre a céla.
Ezen kívül van az **operatív tár** (más néven **főtár** vagy **központi tár**) amit egyszerűen csak **memóriának** is szoktak becézni. Azokat az utasításokat és adatokat amikkel éppen dolgozik a processzor, itt tároljuk hogy könnyen és gyorsan el tudja érni. Attól függően hogy a memóriát milyen feladat megoldására használjuk, van több fő típusa:

A **RAM** memória írható-olvasható gyors memória, a számítógép kikapcsolásakor a RAM tartalma elvész. Itt tipikusan csak az aktuálisan futó programok és az azok által feldolgozandó adat tárolódik. Altípusai a statikus **SRAM** (tipikusan cache memóriákhoz használt, processzoron belül) és a dinamikus **DRAM** ami kvázi egy tipikus számítógép DDR3 vagy DDR4 memória moduljának felel meg. A dinamikus RAM-ban 1 bit tárolására szolgáló cella 1 kondenzátorból és 1 tranzisztorból épül fel. A töltéseket apró kondenzátorok tárolják.

A **ROM** memória csak olvasható memória, például ROM memórián tárolja az alaplap a firmware-t (BIOS-t, korszerűbb gépeken UEFI-t). ROM-ból léteznak különféle altípusok: a **PROM** egy egyszer **p**rogramozható memória, utána a tartalma nem változtatható. Vagy van az **EPROM** ami egy törölhető (**e**rase) és újra**p**rogramozható memória.

Az **átlapolt memóriakezelés** ötlete a relatív **lassú DRAM-ok miatt** jelent meg. A memória függetlenül címezhető kis egységekből, **memory bank**-okból áll, ezek mindegyikét külön-külön a többitől függetlenül el lehet érni. A 0. memóriabankból

kiolvasott adat hozzáférése alatt az 1. memóriabankban lévő következő címen lévő adat már megcímezhető. Ez kissé leegyszerűsítve azt jelenti, hogy cím folytonos olvasás esetén az **adatok kiolvasása kb. kétszeres sebességgel** történhet.

8. Gyorsító (cache) tárak feladata és működési elve. Cache tárak felépítése és típusai. Helyettesítési és adataktualizálási stratégiák.

A modern (értsd: az elmúlt bő húsz évben készült) processzorok esetében komoly kihívás a végrehajtóegységek folyamatos „etetése" adattal. A rendszermemória ugyanis egyszerűen túl lassú, az adatok bekérésétől számítva mintegy 100 órajelciklusra van szükség ahhoz, hogy azok meg is érkezzenek, vagyis a processzor szinte mindig üresben járna, ha erre várna. Hogy ezt elkerüljék, a processzorgyártók komplex prediktív algoritmusokat fejlesztettek, amelyek igyekeznek a **potenciálisan szükséges adatokat** a **gyorsítótárba, avagy cache memóriába** tölteni.

Általában több szinten valósítják meg a cache-t: van **L1** azaz level 1, **L2** másodszintű cache. Az L2 cache méretében nagyobb de lassabb is mint az L1-es. A **cache memória drága**. Míg a DRAM mérete korszerű gépeken nagyságrendileg több GB, addig a cache memória mérete tipikusan **csak néhány MB méretű**. A cache memória puffert tipikusan a processzorba tokozzák bele, ez nem egy külön megvásárolható egység mint a DRAM. A **cache belső felépítését** úgy lehet elképzelni mint **sorokat**, ahol minden sor áll a **címrészből, a vezérlő részből és az adatrészből**. A processzor az adott címrész alapján az adatrészbe írja ki az adatot ami megváltozott a műveletvégzés során.

A **helyettesítési stratégiák** lényege az, hogy a **legrégebben használt** cache blokkok helyére írjuk be az új adatot Erre azért van szükség mert a cache mérete kicsi, könnyen telítődik. **Demand fetching** esetén csak a processzor igényére kér be új adatot a cachbe, **prefetching** esetén pedig egy blokk betöltése esetén előre az utána lévő blokkokat is automatikusan becacheli, hátha kelleni fog…

Az **adataktualizálási stratégiák elve** arról szól hogy ha a processzor a műveletvégzés során megváltoztat egy adatot a cacheben akkor azt mihamarabb ki kell írni a főtárba is, mielőtt az adott cache blokk felülíródna. Tipikusan a gyorsítótárban módosított adat csak akkor kerül visszamásolásra a főtárba, ha a cache-nek a módosított adatot tartalmazó sorát felül kell írni a főtárból bemásolandó újabb blokkal.

9. A virtuális tárkezelés fogalma és legfontosabb eljárásai (lapozás és szegmentálás, a virtuális cím leképezése, TLB, lapcsere stratégiák).

Egy program végrehajtásához a megfelelő programrésznek és az általa feldolgozott adatoknak a memóriában (DRAM) kell lenniük. A mai korszerű számítógépek memória mérete valahol 2-16GB között mozog. Ezeknek a gyors memória moduloknak az ára 1GB-ra leosztva sokkal drágább mint mondjuk a háttértárak árai ezért az ár-érték arány miatt nem praktikus a memória mennyiségének eszetlen növelése. A **virtualis tárkezelés** (lapozótár) lényege hogy a relatív lassú de **nagy tárhely méretű háttértár** (HDD, SSD) **egy elkülönített részén** tároljuk az **aktuálisan nem futtatott** programokat és a hozzájuk tartozó állományokat és csak akkor töltjük be a központi memóriába, ha szükség van rájuk. Pl. egy nagyon erőforrás igényes műveletnél: videó vágáskor ha elindítjuk a renderelést akkor sokszor nincs az a memória mennyiség amit ne tudna megtölteni az operációs rendszer. Ilyenkor a virtulis tárra írja ki azokat a blokkokat amik a memóriába már nem férnek bele.
A virtuális tárkezelésből a felhasználó semmit nem vesz észre, ez a háttérben teljesen transzparens módon történik hogy a memória fizikai korlátai észrevétlenek legyenek.

Lapozás eljárás estén a virtuális tár **rögzített méretű** nem átlapolható **blokkokból áll**, ezeket lapoknak nevezzük. Ezzel gyakorlatilag egy olyan struktúrát alakítunk ki mint ahogy az adatok a memóriában is tárolódnak, a fix méretű blokkokban. Mivel a virtuális táron hasonló struktúrában tároljuk az adatokat, azokat olcsóbb művelet visszamozgatni a tényleges memóriába.

A **szegmentált eljárás** esetén a **blokkok mérete nem rögzített**, ilyenkor a blokkokat szegmenseknek nevezzük. A szegmensek átlapolódóan is megadhatók, azaz ugyanaz az adat két különböző szegmensen belül is megcímezhető. Ez felveti a **tördezettség** (fregmentáció) problémáját amikor üres helyek illetve duplikációk jönnek létre a háttértáron.

A **TLB** (translation lookaside buffer) egy **memória buffer**, a legfrisebben használt lapok címfordításhoz szükséges adatait tartalmazza, tehát lényegében egy címfordító cache. A TLB a processzor és a cache tároló között helyezkedik el, mivel a programok virtuális, a cache pedig fizikai címeket tartalmaz.
A processzornak a művelet végrehajtás során a memória valódi, vagy másképpen nevezve fizikai címeire van szüksége tehát szükség van a **virtuális címek fizikai címre** való **leképzésére**. Mai gépekben ezt egy címleképezési áramköröket tartalmazó hardver egység végzi.

Különféle **lapcsere stratégiák** állnak rendelkezésre ha új lapot szeretnénk beilleszteni. A lapot beszúrhatjuk a memória **első szabad helyére**, vagy a legutoljára betöltött lap utáni **következő szabad helyre**, vagy a **legjobb helyre**, hogy a beszúrás után a legkevesebb szabad tárterület maradjon utána.

10. Az adatrögzítés elve a mágneses háttértárolókon. A merevlemez fizikai felépítése (szektor, sáv, cilinder) és logikai felépítése (klaszter, FAT, bootszektor). A merevlemezes egység teljesítményjellemzői (elérési idő, adatátviteli sebesség).

A merevlemez **mágneses elven működő tárolóeszköz**. Tartós tár, tehát a számítógép kikapcsolása, áramtalanítása után is megőrzi a rá mentett adatot. A számítástechnikában a merevlemez (HDD) az egyik legolcsóbb tár ha az 1GB-ra eső árát nézzük. A winchester fémházának belsejében korong alakú lemezeket találunk egy tengelyre fűzve. Ezeken a lemezeken egységnyi mágnesezhető részecskék vannak. **A tengelyre fűzütt lemezeket**

egy motor forgatja nagy sebességgel (asztali gépekben tipikusan 7200 fordulat percenként), miközben egy **író/olvasó fej** az egyes mágnesezhető részegységek fölé közel mozogva elektromágneses gerjesztéssel megváltoztatja az egységek mágnesességének irányát vagy egyik vagy másikféle irányban. Ez a kétféle mágnes állapot (true/false) megalapozza a **bináris adattárolás** lehetőségét.

A HDD-beli lemezeket azonos központú, különböző sugarú körök tagolják, ezeket **sávoknak** nevezzük. Azokat a sávokat, melyek egymás alatt helyezkednek el, **cilindernek** nevezzük. A kör-sávokat tovább lehet bontani ún. **szektorokra**. A sávokat, cilindereket és szektorokat számozzuk, ez alapján vannak nyilvántartva. A HDD meghajtó formázása (particionálása) közben csoportosíthatjuk a szektorokat, így alakíthatunk ki helyfoglalási egységeket, azaz **clustereket**. Ezek a legkisebb címezhető egységek a HDD-ben, legkisebb mérete 512byte lehet (1db szektor) a legnagyobb pedig 64kb (128db szektor).

A **FAT** (file allocation table) egy szabványos fájlrendszercsaládnak a neve. a FAT16 és a FAT32 is elavult manapság, szűkül a felhasználási köre de ennek ellenére széles körben támogatott az operációs rendszerek között. Manapság az NTFS, exFat, HFS+, APFS tekinthető korszerű fájlrendszerek. FAT32-re formázott pendrivera pl nem lehet rámásolni 4GB-nál nagyobb méretű fájlt, pl. filmeket mert képtelen tárolni ekkora fájlt.

Egy bootolható (rendszerinditó) partíción belül az első szektor(oka)t rendszerbetöltő, másnéven **bootszektornak** nevezik.

A merevlemez mivel egy **mechanikus eszköz**, különböző **késleltetésekkel** lehet számolni. Ezek a fejmozgatási idő, forgatási idő, adatátviteli idő, stb… Nagyságrendileg néhány ms hosszúságúak ezek a késleltetések. Modern merevlemezek nagyjából 80-130MB/s sebsséggel képesek adatokat írni-olvasni SATA 3-as csatlakozón. **A jövő egyértelműen az SSD meghajtóké**, ahogy egyre olcsóbbak lesznek...

11. A megszakítási rendszer (megszakítások típusai, a megszakítás kiszolgálása, vektortáblázat) és alkalmazásai. A megszakítás-vezérlő feladatai.

A programmegszakítás (interrupt) azt jelenti, hogy a program futása egy külső vagy belső esemény bekövetkezte miatt megszakad, és majd csak a progammegszakítás kiszolgálását végző kód lefutása után tér vissza a CPU az eredeti program folytatásához.

Megszakítás történhet akkor amikor egy **periféria jelzi hogy egy input/output műveletet befejezett**, tehát egy meghatározott művelet befejezésekor. De megszakítás az is amikor valamilyen **géphiba miatt egy áramköri jelzést küld** a gép vagy ha **külső forrásként** megnyomjuk az **újraindtás gombot** a számítógépen.
A megszakítás kezelés **aszinkron** a feldolgozás, mert időben nem megjósolható, hogy egy megszakítást kiváltó esemény (például egy billentyű leütése) mikor fog megtörténni.

A **vektottáblázat** a megszakításokat kiszolgáló rutinok kezdőcímeit tartalmazza. A megszakítás kiszolgálása úgy zajlik hogy a megszakításkérelemre a processzor a vektortáblázatból megkapja a megszakítási elem sorszámát, majd elmenti az utasításszámláló és állapotregiszterek aktuális tartalmát. Végrehajtja a a megszakítást kiszolgáló rutint majd a processzor visszatölti az elmentett regiszter tartalmakat és folytatja a megszakított program végrehajtását.

A **megszakítás vezérlő** feladatai többek között hogy **fogadja** a megszakításkérő kérelmét, vizsgálja hogy nincs-e maszkolással tiltva, **priorizálja** a megszakítás fontosságát, valamint **közli a megszakítás kérést a processzorral** az **INT** vezetéken. Ha a processzor kész a kérés fogadására (**IACK** - Interrupt Acknowlegment) akkor a megszakításvezérlő átadja a processzornak a megszakításvektor címét.

12. Az I/O adatátvitel típusai. A közvetlen memória-hozzáférés (DMA) lényege és végrehajtása. A DMA-vezérlő regiszterei és működése.

Az I/O (input/output) adatátvitelnél megkülönböztetünk több típust az alapján hogy az adott input/output kérést hogyan dolgozza fel a számítógép.

Polling esetén a processzor minden meghatározott órajelciklusban (lehetőleg néhány ms-onként) ellenőrzi hogy volt-e I/O művelet. Ez költséges megoldás mert állandóan, sok esetben feleslegesen terheli a processzort.

Megszakításos I/O átvitel esetén egy megszakítás esemény váltódik ki, például arról hogy lenyomtam egy billentyűt, és ezt a megszakítás vezérlő jelzi a processzornak. Azért hívjuk megszakításnak, mert ilyenkor a processzor megszakítja az éppen végrehajtott programot a megszakítás feldolgozásának idejére, majd tovább folytatja a megszakított programot.

DMA (direct memory access), **közvetlen memória hozzáférés** esetén az I/O eszköz és a memória (DRAM) közötti **adatátvitelbe a processzor nem vesz részt**. Ezt egy külön egység, a DMA vezérlő végzi. Ennek az az előnye hogy a processzor felszabul az I/O műveletek terhe alól. A DMA adatátvitel **végrehajtása** abból áll hogy ellenőrizzük a perifériát, jelezzük a buszfoglalási kérelmet, majd a DMA vezérlő masterként lefoglalja a buszt. Ha kész az adatátvitel akkor a DMA vezérlő jelzi a processzornak, majd megszűnik a buszengedélyezés.

A DMA vezérlő regiszterei a **címregiszter** ami az aktuális forrás/cél memóriacímet tárolja, a **számlálóregiszter** a még átvitelre váró adatokat tárolja, a **módregiszter** az átvitel irányát tárolja (írás/olvasás), a **maszkregiszter** értelemszerűen a maszkot tárolja (ha van), valamint az **állapotregiszter** az átvitel státuszát tárolja például hogy befejeződött-e az átvitel.

13. A sín (busz) feladata, logikai felépítése, típusai. Sínvezérlés (szinkron, aszinkron). Master és slave eszközök. Buszarbitráció (soros és párhuzamos sínfoglalás).

A számítógép részegységei közötti kapcsolatokat a sínrendszer biztosítja. Ez gyakorlatilag egy **összetett mikrovezeték hálózat**, ami különböző sínvezérlő áramköröket és aktív (pl. tranzisztor) és passzív (pl. ellenállás) áramköri elemeket is tartalmaz. A sínen címek, adatok és vezérlőjelek utaznak, ezek jellemzően **négyszögjelek** formájában.

Logikailag szétbontható a **címsínre**, az **adatsínre** és a **vezérlősínre** ahol például a megszakítások vezérlőjelei is végighaladnak.

A **belső sínrendszer** a **processzoron belül** annak különböző részeit kapcsolja össze, a **külső sínrendszer** a processzort köti össze a különböző részegységekkel, jellemzően az **alaplapon** alakítják ki ezt a típusú sínrendszert.

Szinkron sínvezérlés onnan kapta a nevét hogy a sínen kommunikáló eszközök egyszerre, azonos órajelen azaz szinkronban ütemezettek. Az adás-vétel mindig azonos sebességgel történik. Problémája az hogy **közös órajelet** kell biztosítani az összes sínre kapcsolt eszköz számára.

Aszinkron sínvezérlés esetén az események **tetszőleges időben** bekövetkezhetnek, itt nincs szükség közös órajelre, emiatt nagyon eltérő sebességű eszközök kiszolgálását is lehetővé teszi. Az átvitel során szükség van egy úgynevezett **handshake**-re hogy az adás-vétel megtörtént-e (visszaigazolás).

A sínt egy időben csak egy eszközpár használhatja, az aktív **kezdeményező fél** a **master**, a másik fél pedig a **slave** (egyenes fordításban "szolga") tehát a kommunikációt

semmiképpen nem irányítja. A master tipikusan egy DMA vezőrlő vagy valamilyen processzor szokott lenni a slave pedig például a memória vagy egy periféria.
Amikor a sín használatát egyszerre több master eszköz is igényli, akkor el kell dönteni hogy melyik eszköz kapja meg a használatot. Ez az eljárás a **buszarbitráció**.

Soros sínfoglalás esetén egy "várólista" alapján egészen egyszerűen **sorba állítjuk** a sínfoglalási igényeket.

Párhuzamos sínfoglalás esetén minden sínhasználatért váró eszköz önálló kérő és engedélyező vezérlővonallal rendelkezik, és a busz arbiter **prioritás szerint** engedélyezi a sín igénybevételét.

14. Az I/O eszközvezérlők, interfészek feladata, regiszterei, címzése. Soros és párhuzamos port és adatátvitel. Az adó és vevő szinkronizálása.

Az I/O eszközvezérlők **kapcsolják össze a háttértárakat és perifériákat a számítógép I/O sínrendszerével**. Ha egy program egy adatot akar pl. beolvasni, akkor az erre vonatkozó parancs az eszközvezérlőn megy keresztül majd ez vezérli az I/O eszköz például a winchester tevékenységét.

AZ I/O interfészeket legtöbbször az I/O sínbe behelyezett például PCI kártyaként (hangkártya, tuner kártya, stb..) valósítják meg. Emellett elhelyezhetők az alaplapon különféle alapvető funkciót betöltő integrált I/O interfészek (ethernet, hang, stb...) amik annyira alapvető dolgok hogy ezeket az interfészeket már eleve minden PC alaplapra integrálják legyen az asztali gép vagy laptop.

Az I/O interfészek regiszterei **a parancs regiszter**, az **állapot regiszter** és a **puffer regiszter**. Ezeknek a beszédes neve elárulja hogy mit tárolnak a műveletvégzés során...

Soros adatátvitel (pl. USB - Universal Serial Bus) esetén az interfész és a periféria között az adatokat **bitenként sorba egymás után visszük át**. Mindig két eszköz vesz részt, egyik az **adó**, a másik a **vevő** szerepkörét tölti be.
A soros adatfolyam értelmezéséhez a vevőnek fel kell ismernie az adatbitek határait, ehhez előírhatunk egy speciális bitsorozatot amit határként ismer fel a vevő. Így biztosítható az **adó és a vevő összehangolt működése.**

Párhuzamos adatátvitel (pl. LPT - Linear Print Terminal) esetén az interfész és a periféria között az **adatokat bitcsoportonként egyszerre visszük át**, ezt párhuzamos adatátvitelnek nevezzük.
Külön vezeték(ek) szükségesek az adó-vevő szinkronizmus megvalósítására is.

15. Monitorok típusai, paraméterei, működési elve. A monitorvezérlő interfész feladata, felépítése, jellemzői (felbontás, színmélység, képmemória mérete) és működése.

Katódsugárcsöves monitorok (CRT) mára abszolút elavult technológia. A képernyőre eső **elektronsugár** felvillantja a katódsugárcső előtt elhelyezkedő **foszforréteggel bevont** felület egy pontját. Ezekből a fénypontokból áll össze a kép.

Folyadékkristályos monitor (LCD) esetén két átlátszó lap közé szorítanak **folyadékkristályt**. A feszültség a kijelző minden cellájára külön vezérelhető, aminek hatására a folyadékkristály-molekulák egy bizonyos irányba állnak be, és így több vagy kevesebb fényt eresztenek át.

TFT monitornál az aktív mátrixos technológiát használják. Minden egyes képpontja egy saját tranzisztorból áll ami a cella ki- és bekapcsolását gyorsítja.

A **plazmamonitor** nem más, mint **piciny fénycsövek alkotta mátrix**. A fénycsöveg közé semleges gázt (neon, argon, xenon) töltenek. Ha egy cellához tartozó elektródákra magas feszültséget kapcsolnak, akkor a kisülés hatására **a cellában lévő gáz átmegy plazmába**.

A fenti monitor típusok javarészt elavultnak számítanak, a legújabb technológiák a **LED-háttérvilágítású LCD**, illetve az **OLED**-es monitorok. Gyakorlatilag már csak ilyen technológiájú TV-készülékeket forgalmaznak... LED technológia hosszabb élettartamot és kis fogyasztást biztosít. Az OLED kijelzők jellemzően csak kisebb, például okostelefonokban kapnak helyet, monitornak vagy TV-nek egyelőre közel megfizethetetlen az áruk. Az OLED kijelző rendkívül **vékony** (akár fél milliméter is lehet), **könnyű** és **hajlékony** változatai is léteznek ami **viselhető eszközöknél** lehet praktikus. Nem használ háttérvilágítást ezért a fekete szín itt tényleg vaksötét! Emellett nagyobb fényerőt tud kibocsátani mint egy LCD-s monitor és energiatakarékosabb is annál.

A **monitorvezérlő interfész** (magyarul videókártya) feladata a monitoron megjelenítendő kép előállítása. Ez a kép pontokból tevődik össze, amelyeket pixeleknek neveznek. A megjelenítésre előkészített kép digitális változatát a kártya RAM memóriájában tárolják, amely így egy **framebuffer** szerepét tölti be.
A videómemóriában minden pixelhez tároljuk a színmélységét is.

A videókártya lényeges paramétere a max. kimeneti felbontás, amely a maximális megjeleníthető pixelek számát adja meg (pl: 1920x1080 Full HD).
A videókártya összetevői maga a **grafikus processzor** (GPU), a **videómemória**, a ROM-BIOS (**VBIOS**), valamint ha a kártyán van analóg (VGA) kimenet akkor az ennek kezeléséhez szükséges **digitális->analóg konverter** (DAC) egység. Mára egyre jobban tűnik el a régi analóg tűs VGA csatlakozó mert a DVI, HDMI és DP **fizikai csatolófelületek** digitális jelátvitelt biztosítanak így nincs szükség jel konverzióra.

16. Hálózati átviteli közegek. Vonalak megosztásának módszerei. Digitális jelek kódolása. A paritásbit és a CRC. Modemek feladata. ISDN, ATM, DSL technológiák.

Vezetéknélküli hálózati átviteli közegekből létezik az infravörös (lézeres), rádióhullámos és műholdas átvitel.

Az **infravörös** (lézeres) átvitelnél a kommunikáció teljesen digitális. Jól irányítható, viszont kültéren a köd és eső zavarokat okozhat.
A **rádióhullám** nagyobb távolságok áthidalására szolgál. A frekvenciasávok kiosztását a hatóság felügyeli hogy egy frekvenciasávon csak egy pl: rádióadó sugározzon.
A műholdas átvitel esetén az űrbe, alacsony pályára állított műholdaknak felküldik a mikrohullámú jelet, majd azok felerősítve visszasugározzák. Óriási területen tudnak így szórni amit a Földön antennával lehet fogni.

A hírközlési **vonalak megosztására** a **multiplexelést** találták ki, azaz a vonalat felosztják csatornákra.
Lehet **frekvencia osztásos multiplexelés**, ilyenkor az adó oldalon a csatornák jeleit egy-egy vivőfrekvenciára ültetik.
Időosztásos multiplexelés esetén a vonalat időben osztják fel, több elemi adatcsatornára.

A **digitális jelkezelés** esetében minden információ továbbítása **bitek formájában** történik, függetlenül az információ fajtájától.
2 érték fordulhat elő pl. a 0-át és az 1-et reprezentáló feszültség vagy áram érték, és ez viszonylag nagy zaj mellet is jól megkülönböztethető megfelelő áramköri megoldásokkal. Másrész megfelelő kódolással az előforduló hibákat jelezni, sőt javítani is lehet.

A **hibakezelésnél** használatos fogalom a **paritásbit** és a CRC. A vevő oldalon a **pariásvizsgálat** abból áll hogy a kódszóban lévő 1-esek számát megnézzük hogy páros vagy páratlan. A **CRC** gyakorlatilag egy **hiba detektáló kód**. A küldő oldalon a küldendő információhoz hozzáfűznek egy rövid, az üzenetből számított ellenőrző értéket. Amikor a fogadó fél megkapja az üzenetet akkor ő is elvégzi az ellenőrző érték számítását a fogadott adatok alapján. Ha nem ugyanaz az ellenőrző kód jön ki a kapott adatokból akkor az azt jelenti hogy az üzenet nem jött át pontosan tehát valamilyen hiba volt az átvitelben.

A **modem** (a modulátor és demodulátor szavakból összetett szó) egy olyan berendezés, ami a **digitális jelet analóg információvá**, illetve a másik oldalon az **analóg jelet digitális információvá** alakítja. Az eljárás célja, hogy a digitális adatot analóg módon átvihetővé tegye. Például rádiós és mikrohullámú modem, kábelmodem, telefonos modem, ADSL.

Az **ISDN** telefonrendszer **az összes analóg szolgáltatást digitálisra váltja fel**. Az ISDN-ben a jelek digitálisak az otthoni telefontól vagy számítógéptől kezdve az egész hálózaton át. Az ISDN-nek nincs szüksége modemre.

Az **ADSL** (**aszimmetrikus** digitális előfizetői vonal) egy kommunikációs technológia, ami a hagyományos modemeknél gyorsabb digitális adatátvitelt tesz lehetővé a csavart rézérpárú telefonkábelen. Az ADSL jellemzője a DSL megoldásokon belül, hogy a letöltési és a feltöltési sávszélesség aránya nem egyenlő (vagyis a vonal aszimmetrikus), amely az otthoni felhasználóknak kedvezve a letöltés sebességét helyezi előnybe a feltöltéssel szemben, általában 8:1 arányban.

Az **ATM** egy **aszinkron** telekommunikációs átviteli mód. Az adatot kis, fix csomagméretű cellákban továbbítja, emiatt gyors.

17. A számítógép-hálózatok architektúrája, az OSI-modell (rétegek, rétegszolgálatok). A TCP/IP modell (feladata, rétegei, protokollok, információ-áramlás, címzés, útválasztás).

Az OSI modell a számítógépek kommunikációjához szükséges **hálózati protokollt** határozza meg. A különböző protokollok által nyújtott funkciókat egymásra épülő rétegekbe sorolja.

Minden réteg csak és kizárólag az **alatta lévő rétegek** által nyújtott funkciókra **támaszkodhat**, és az általa megvalósított funkciókat pedig csak **felette lévő réteg számára nyújthatja**. Ma a teljes OSI modell egy részhalmazát használják csak.
Lentről felfelé haladva a 7 réteg:

Fizikai réteg felel a fizikai átviteli közegen keresztül a **bitfolyam átviteléért**.
A fizikai réteg feladata **a bitek kommunikációs csatornára való juttatása**. Meghatározza az eszközökkel kapcsolatos fizikai és elektromos specifikációt, az **érintkezők kiosztását**, a használatos **feszültség szinteket** és a kábel specifikációkat.

Adatkapcsolati réteg felelős a **fizikai címzésért** (Mac). Csomópontból csomópontba való megbízható adatátviteli szolgáltatásért felel, védi a felsőbb rétegeket a fizikai rétegtől. Adatkeretekkel dolgozik.

Hálózati réteg feladata az **útvonalválasztás és IP** (**logikai címzés).** Felelős két hálózati felhasználó közötti hálózati összeköttetés létesítéséért, megszüntetéséért és az összeköttetésen keresztüli adatátvitelért. Csomagokkal dolgozik.

Szállítási réteg felel a **végpontok közötti kapcsolatért** és a megbízhatóságért. Feladata hogy a viszonyrétegtől kapott adatfolyamot **kisebb darabokra vágja szét**, ha szükséges, akkor a túloldalon ezeket sorrendhelyesen visszaállítsa. Biztosítania kell a **hibamentes adatátvitelt**.

Viszonyréteg lehetővé teszi a **csomóponti kommunikációt**. Egy viszony például alkalmas arra **hogy állományokat továbbítson két gép között**. A viszonyok egyidőben egy- és kétirányú adatáramlást is lehetővé tehetnek. A viszonyréteg egy másik szolgáltatása a szinkronizáció.

Megjelenítési réteg az információk **megjelenítéséért felel**. Ide tartozik például a kódátalakítás, fordítás, **tömörítés**, stb.

Alkalmazási réteg széles körben igényelt protokollokat tartalmaz. Például több száz inkompatibilis terminháltípus létezik ma a világon ezért definiálni kell egy hálózati virtuális terminált, és a többi programot ezt felhasználva kell megírni. Az alkalmazásréteg tipikus feladatai az **állománytovábbítás, elektronikus levelezés, távoli bejelentkezés**.

A rétegek közötti kommunikáció úgynevezett **rétegszolgálatok** segítségével valósul meg. Ezek mindig két szomszédos réteg között találhatók. Lényegében a két réteg közötti kommunikáció ténylegesen ezeken a pontokon keresztül valósul meg.

A TCP/IP protokoll nem követi az OSI modellt. Unixos gépekre dolgozták ki eredetileg. Lentről felfelé haladva az 5 réteg:

Fizikai rétegben a keretátvitel történik.

Adatkapcsolati réteg az adatot keretekre bontja. Ha a kapott adat túl nagy ahhoz, hogy egy keretbe kerüljön, feldarabolja.

Internet réteg (IP) feladata, hogy a felsőbb rétegektől kapott csomagokat, az Interneten alkalmazott címzés, az IP cím alapján továbbküldje a cél felé, vagyis csomagokra bontsa a TCP-szegmenseket, és elküldje őket bármely hálózatról. Megvizsgálja, hogy a csomagot milyen útvonalon kell továbbítani.

Szállítási réteg (TCP) az egymásnak üzenetet küldő két végpontot összekötő réteg. Nem vizsgálja a végpontok közötti állomásokat, csak azzal foglalkozik, hogy a végpontok között megvalósuljon az adatátvitel.

Alkalmazási réteg a felhasználó által indított program és a szállítási réteg között teremt kapcsolatot. Alkalmazási protokollok: **FTP, HTTP**, SMTP, DHCP, stb…

TCP/IP címrendszerében az IP címek szigorú szabályok alapján vannak kialakítva. A hálózat minden gépéhez hozzá kell rendelni (legalább) egy **egyedi** számot a hálózati környezet számára. Az IP címeket négy darab, ponttal elválasztott számmal írjuk le, pl: 192.168.0.10

Az egyes elemek **0 és 255 közötti értéket** vehetnek fel, tehát 4db 8 bites számról van szó. A hálózaton alhálózatokat hozhatunk létre **címmaszkok** használatával, pl: 255.255.255.0
A 127.0.0.1 = **localhost**, loopback egy spéci cím ami az adott gép saját IP címére mutat.

A csomagkapcsolt rendszerekben az **útválasztás** (routing) azt a folyamatot jelöli, amivel kiválasztjuk az útvonalat, amin a csomagot továbbküldjük. Egy csomag egy rakás hálózati eszközön (routerek, switchek, bridegek, stb…) is áthaladhat amíg elér a fogadó félhez. Az útvonalat az **útválasztási tábla** (routing table) alapján határozza meg a router.

18. Lokális hálózatok szabványos megvalósítása (Ethernet, vezérjeles sín, vezérjeles gyűrű): protokollok, közeg-hozzáférési módszerek, átviteli közegek, fizikai egységek.

Napjaink lokális hálózatai között egyértelműen az Ethernet a legelterjedtebb technológia a fizikai és az adatkapcsolati rétegben. Átviteli sebessége alapján van 10 Mb/s sebességű, a 100 Mb/s sebességű az 1 Gb/s sebességű **Gigabit-es Ethernetet**, valamint a **10 Gigabit-es Ethernetet**.

A **vezérjeles sín** valójában egy vezérjeles gyűrű, egy **koaxiális kábelen** megvalósított **virtuális gyűrű**. Mindenki csak a szomszéd címét ismeri. A legmagasabb sorszámú küld vezérjelet, azt egy állomás veszi és ha a jel szabad hozzáfűzi az adatokat és foglaltra állítja.

Vezérjeles gyűrűnél a gyűrűben vezérjelet (tokent) adnak körbe az állomások, aki ezt birtokolja, bizonyos ideig adhat. Amíg az adat kering a gyűrűben addig nincs újabb vezérjel, így hát nincs ütközés sem. Előre lehet tudni mennyi idő telik el max. míg 1 állomás újra adhat.

Közeghozzáférés fajtái:
CSMA/CD protokoll: többszörös hozzáférésű, ütközés-érzékelő protokoll. Az állomás figyeli a hálózatot. Ha nincs rajta csomag, akkor kezdeményezi az adást. Ha két állomás egyszerre próbálkozik, akkor mindkét állomás egy ideig vár, majd újra megpróbálja elküldeni a csomagot. Nagy hálózat esetén ez a módszer lelassítja a működést.

Token Passing protokoll: adási jog továbbítási protokoll**.** Egy időben csak egy állomás adhat, ez a jog halad körbe. Ezt láttuk vezérjeles gyűrűnél...

Központosított vezérlés: Egy kitüntetett állomás vezérli a hálózatot és engedélyezi az állomásokat - jogokat ad és vesz el.

Átviteli közegek:

Sodrott érpár két rézhuzalból áll, amely spirálisan feltekerve így kiküszöböli a zaj jelentős részét. Erősítés nélkül is több km-ig használható. Ha több érpár fut együtt, akkor azokat kötegelik. Analóg és digitális átvitelre is alkalmas. Pl: UTP, (Cat 1-6) kábelek.

Koaxiális kábel: Digitális jelátvitelre alkalmas. A kábel felépítése úgy néz ki hogy közepén fut a központi vezeték, ekörül vastag műanyag szigetelés van és ezt még réz fonattal borítjuk körbe az árnyékolás miatt. Remek sávszélesség, és kitűnő zajvédelem jellemzi.

Optikai szál: az adatok átvitele fényimpulzussal történik (van fény - 1, nincs fény - 0). Három része: fényforrás - **lézerdióda** -, átviteli közeg - **üvegszál** -, **fényérzékelő** - fotodióda -. Nagy távolságokat képes áthidalni (pl. tengerek alatti kábelek), gyakorlatilag zaj immunis mert nem elektromágneses impulzusokat küldünk.

Fizikai egységek a hálózatban lehet az:

Ismétlő ami **erősíti** és újraidőzíti a jelet. A **hub** többfelé **osztja a jelet**, de akár erősítheti is! A **switch is a jelet osztja,** minden portján teljes sávszélességet biztosít. A **router**-ekben is megtalálható általában egy switch, tehát a router egyben switch is. Feladata a hálózati címek (IP címek) alapján a hálózati eszközök összekapcsolása és azok közötti adatforgalom irányítása.

19. Az operációs rendszer erőforrás-kezelőjének feladata. A holtpont és kezelésének stratégiái. Biztonságos állapot. A szemafor használata a termelő-fogyasztó folyamatok esetében.

Az erőforrás-kezelő a rendszermag azon része, amely az **erőforrások elosztásáért és lefoglalásáért felelős**. Ha egy folyamat erőforrást igényel, az erőforrás kezelő dönti el, hogy a kérés kielégíthető-e.
Az erőforrás kezelő gondoskodik a számítógép erőforrásainak hatékony gazdaságos elosztásáról, illetve az erőforrások használatáért vívott versenyhelyzetek kezeléséről.

Hardver erőforrások: pl. processzor, memória, nyomtató és az egyéb perifériák.
Szoftver erőforrások: a különböző közösen használható programok, adatállományok, adatbázisok.

Ha bármely folyamat, amely erőforrást igényel feltétel nélkül megkapja azt, hamar **holtpont** alakul ki. Holtpontnak nevezzük, amikor **több folyamat ugyanannak az erőforrásnak a felszabadulására vár**, amit csak egy ugyancsak várakozó folyamat tudna előidézni. Elkerüléséhez megelőzésre vagy folyamatos figyelésre majd felszámolásra van szükség.

Ha megtiltjuk, hogy egyszerre több folyamat is rendelkezzen erőforrással, **elszaporodhatnak a várokozó folyamatok** és **kiéheztetés** lesz a végeredmény. A kiéheztetés az erőforrás kezelő stratégia miatt jöhet létre. Ilyenkor összesen ugyan **van elegendő erőforrás**, de szerencsétlen esetben egyes folyamatok **mégis "éheznek".** Az erőforrás-kezelő dönt, hogy melyik folyamat jut erőforráshoz, és lehet, hogy egy folyamat elé mindig bekerül egy másik, így az a folyamat beláthatatlan ideig nem jut erőforráshoz.

Egyetlen foglalási lehetőség stratégiája: Csak az a folyamat foglalhat erőforrást, amelyik még egyetleneggyel sem rendelkezik.

Rangsor szerinti foglalás stratégiája: Az erőforrásokhoz sorszámot rendelünk és a leggyakoribbak kapják a legkisebbet. Igény esetén egy folyamat csak a birtokoltnál magasabb sorszámú erőforrásokat igényelhet.

Biztonságos állapot: Egy rendszer állapota akkor biztonságos, ha létezik egy olyan sorrend, amely szerint a folyamatok erőforrás igényei kielégíthetőek.

A szemafor használata a termelő-fogyasztó folyamatok esetében azt jelenti hogy a termelő és a fogyasztó közös memóriaterülethez fér hozzá, de azt nem használhatják egyszerre. Hogy kizárjuk az egyidejű erőforrás-használatot, használhatunk szemafort, mivel a szemafor megmutatja, ha egy másik folyamat éppen használja a kívánt erőforrást.

20. A magas, közbenső és alacsony szintű ütemezők feladata egy operációs rendszerben. A folyamatok állapotai. Ütemezési algoritmusok.

Az idővel való gazdálkodást ütemezésnek (scheduling) nevezzük. Az ütemezés során a folyamatok állapota megváltozik. Attól függően, hogy milyen állapotok között történik váltás, az ütemezők több szintjét definiálhatjuk.

A **magas szintű ütemező** választja ki a háttértár programok közül azt, amelyik az operációs rendszer felügyelete alá kerülhet.

Közbenső ütemező folyamatosan figyeli a rendszer állapotát (terhelését) és ha túlságosan sok folyamat kerül futásra kész állapotba és egyiknek sem jut elég processzoridő, akkor egyes folyamatokat felfüggeszt, illetve prioritásukat átrendezi a rendszer hatékony működésének érdekében.

Az **alacsony ütemező** feladata, hogy a processzort a futásra kész folyamatok között igazságosan és hatékonyan ossza el. Legfőbb követelmény vele szemben a gyorsaság.

A folyamatok állapotai lehet**:** **Futásra kész**: CPU-n kívül az erőforrások rendelkezésre állnak. **Fut**: Végrehajtás alatt. **Várakozik**: Foglalt az erőforrás amire szüksége van. **(Megszakad)**: Végrehajtása megszakítva egy interrupt miatt.

Ütemezési algoritmusok:

FCFS (First Come First Served): A folyamatok érkezési sorrendben kapják meg a processzoridőt lefutásukig,.

SJF (Shortest Job First): A legrövidebb processzoridőt igénylő folyamatot részesíti előnyben. A legrövidebb várakozási időt adja, viszont a hosszabb futást igénylő folyamatokkal „mostohán" bánik.

RR (Round Robin): Minden egyes folyamatnak egy meghatározott processzoridőt biztosít, és azután megszakítja és a várakozási sor végére teszi. Előnye, hogy a legrövidebb válaszidőt produkálja és a folyamatok között demokratikusan osztja el a CPU-t, viszont jelentős adminisztrációt igényel.

21. Többfeladatos (multitasking) operációs rendszerek feladatai, felépítése. A tárvédelem feladata és megvalósítása (privilégiumi szintek, jogosultságok, szegmensek, deszkriptorok, kapuk).

Többfeladatos egy OS ha egy időben több folyamat végrehajtását végzi. Ilyen folyamatok pl.: processzor-idő, a különböző erőforrások, és a képernyő el-/felosztása, stb...

Eszközkezelés: perifériák különbözőségeinek elfedése a programok elől.

Megszakításkezelés

Rendszerhívás: az OS magjának úgy kell kiszolgálnia a felhasználói alkalmazások igényét, hogy azok ne vegyék észre, hogy nem közvetlenül használják a perifériákat.

Erőforrás-kezelés: közös erőforrás-használat megelőzése vagy bekövetkezéskor feloldása.
CPU-ütemezés: CPU idejének elosztása valamilyen stratégia alapján és munkák közti átkapcsolási folyamatok vezérlése.
Memóriakezelés: felosztja a RAM-ot úgy, hogy a folyamatok se egymást se az OS-t ne zavarják.
Állomány- és háttértárkezelés: Rendet tart az állományok között.
Felhasználói felület

A tárvédelem feladata feladata a programok és adatok védelme. Ez azért különösen fontos, mert az OS és a felhasználói programok fizikailag ugyanabban a memóriában vannak tárolva. Meg kell megvalósítania az **operációs rendszer védelmét** a felhasználói programoktól. A **felhasználói folyamatokat védeni kell egymástól** - de biztosítani kell kommunikációjukat. Biztosítani kell az adatokhoz, programokhoz való **hozzáférési jogok ellenőrzését**.

Privilégiumok és jogosultságok figyelembe vétele esetén egy program csak a vele azonos, vagy magasabb szintű programot hívhatja meg és nála alacsonyabb szinten lévő adatot nem nem érhet el... Pl. iOS esetén a **sandboxing**: minden app a saját homokozójában, a saját adataiban „matathat” csak. A rendszerhez vagy más appokhoz nem fér hozzá.

Két privilegizált szintet különböztetünk meg:

- **Magas** jogosultsági szint az OS programok számára
- **Alacsony** szintet a felhasználói programoknak

Az **azonos** privilegizálási **szinten futó taszkok memóriaterületét védeni kell** egymástól, ezért minden taszkhoz védelmi táblák kerülnek felépítésre. A táblákban található **deszkriptorok** (infók) határozzák meg a taszkok hozzáférési jogosultságait (olvasási/írási/végrehajtási).

A program és a memória logikai egységekre, **szegmensekre** van osztva. A szegmensek egymás memóriaterületeit nem zavarhatják. A folyamatokat úgy védjük, hogy minden egyes folyamathoz szegmensleíró táblát rendelünk.

A vezérlés privilegizálási szintje mindig ellenőrzésre kerül a **kapukon** történő áthaladása során. Kapu típusok: **call kapu** (taszkok közötti paraméter átadásnál), **megszakítási kapu, trap kapu, task kapu.**