1. **Operációs rendszerek absztrakciós szintjei**

* az absztrakciós szintek segítségével a hardverek fölé olyan rendszert helyezünk, amivel a programozó egyszerűbben tudja kezelni az erőforrásokat
  + **||Felhasználói programok**
  + **|| Fordító | Szövegszerkesztő | Parancsértelmező || Rendszerprogramok**
  + **|| Operációs rendszer || Rendszerprogramok**
  + **|| Gépi nyelv || Hardver**
  + **|| Mikroarchitektúra || Hardver**
  + **|| Fizikai eszköz || Hardver**

1. **A rendszermag és a héj szerepe valamint fontosabb feladataik**

* a kernel felelős a hardver erőforrások kezeléséért, a processzoridő szétosztása
* megkülönböztetünk: *monolitikus (egy prog. allo rendszermag), mikro- (interfeszeken komm. a programok), hibrid (kritikus funkciokat egyetlen egysegbe otvozi, masokat kulonallo modulkent kezel) és exokernelt (felhasznalora van bizva a hardver absztrakcio)*
* a kernel fő feladatai:
  + ki- és bemeneti eszközök kezelése
  + Programok, folyamatok futásának kezelése
    - Indítás, futási feltételek biztosítása, leállítása
    - Memória-hozzáférés biztosítása
    - Processzor idejének elosztása
  + Háttértárolók kezelése
  + Rendszerhívások kezelése
  + Fájlrendszer
* Shell feladatai:
  + kapcsolattartás a felhasználóval
  + alkalmazások futásának kezelése

1. **A processzus fogalma és a processzus állapotainak ismertetése**

* a folyamat az operációs rendszerben futó program példánya
* futtatáskor az OpR. több dolgot nyilvántart a programról, mint pl. a memóriaterületet, állományokat, környezeti változókat
* 3 fő állapotot különböztetünk meg a folyamatok esetén:

**Futásra kész 🡨 🡪 Futó 🡪 Blokkolt 🡪 Futásra kész**

* **Futásra kész:** ütemezésre váró folyamat
* **Futó:** az aktuális végrehajtás alatt álló folyamat
* **Blokkolt:** erőforrásra, felhasználói beavatkozásra vár

1. **Folyamatok ütemezése időosztásos és valós idejű rendszerekben**

* mikor több folyamat képe futni, akkor az operációs rendszernek el kell döntenie, melyik fusson először, illetve milyen sorrendben fussanak
* az operációs rendszer ezen részét ütemezőnek nevezzük, az erre a célra használt algoritmust pedig ütemező algoritmusnak
* **időosztásos** rendszerek (processzor kisajatitasa, gyors valasz) esetén több ütemező algoritmus is rendelkezésre áll:
  + Round-Robin
    - minden folyamatnak ki vna osztva egy időintervallum, ezalatt engedélyezett a futás
    - a folyamatok egy listában vannak tárolva, ha valamelyik elhasználta az idejét, a lista végére kerül
  + Prioritásos ütemezés
    - minden folyamathoz hozzárendelünk egy prioritást és a legmagasabb prioritású folyamatnak engedjük meg hogy fusson
  + Legrövidebb feladat fusson először
  + Garantált ütemezés
  + Sorsjáték ütemezés
* **Valos ideju rendszerek:** a folyamatok gyorsan elvegzik feladataikat, csak rovid ideig futnak
* **valós idejű** rendszerek esetén megkülönböztetünk *szigorú* és *lágy* ütemezést
  + **szigorú** 🡪 minden határidőt teljesíteni kell, kevés rendszer rendelkezik ilyennel
  + **lágy 🡪** minimalizálni kell a határidő túllépéseket. Kevés esetén nincs probléma, de ha túl sokat hagyunk ki, prblémák merülhetnek fel

1. **A rendszerhívás fogalma, rendszerhívás végrehajtásának alapelve**

* a legtöbb esetben az operációs rendszer nem engedi meg, hogy az átlagos felhasználó belepiszkáljon a rendszerbe, ehelyett úgynevezett szolgáltatásokat nyújt a programoknak
* minden felhasználói program a processzor felhasználói üzemmódjában fut, ha valami olyan műveletet akar végrehajtani, amely más futó programokat megzavarhat, akkor meg kell kérnie az oprendszert, hogy hajtsa végre a kért műveletet, melynek végrehajtását a jogtól függően az vissza is utasíthatja
* a rendszerhívás folyamata egyszerű lépésekkel leírható:
  + a felhasználói program beállítja a paramétereket
  + mikor megvan, a program jelzi valamiként a kernelnek hogy rendszerhívás történik
  + a jelzés egy ún. **exception**-t dob, mely egy speciális címre ugrik, ahonnét tovább futtatja a kódot (kb mint interrupt)
  + elmentődik a jelenlegi állapot, azonosítódik és végrehajtódik a megfelelő rendszerhívás, az állapot visszaállítódik, majd a kontroll is visszatér a felhasználói program kezébe

1. **Folyamatok ütemezése időosztásos és valósidejű rendszerekben – megvolt mar a 4.kerdesnel**
2. **Atomi műveletek. Kritikus szekció. Kölcsönös kizárás. Szemaforok**

* **Atomi művelet** 🡪 olyan művelet, mely nem szakítható meg semmiként, pl. összeadás, szemaforok kezelése
* **Kritikus szekció 🡪** olyan szekció, melyben biztosítanunk kell, hogy egy időben csupán egyetlen folyamat férhet hozzá az adott erőforrásra, máskülönben versenyhelyzet alakul ki, előbb-utóbb mindegyik folyamat meg kell kapja a szükséges erőforrást
* **Kölcsönös kizárás 🡪** bizonyos erőforrásokhoz a folyamatok csakis kizárólagosan férhetnek hozz, a felhasználónak ezt biztosítania kell, másként nem garantálja semmi a megfelelő futást
* **Szemaforok 🡪** egy olyan absztrakt adattípus, melyet az erőforrásokhoz való hzzáférés szabályozásához használnak, értékének változása atomi művelet. Ha az étéke pozitív, a szemafor nyílt állapotban van, ilyenkor az első folyamat lezárhatja, majd beléphet az adott szekcióba. Ha zárva van, a folyamatnak várnia kell míg a záró folyamat felengedi. Értékétől függően több folyamat is beléphet az adott szakaszba

1. **A holtpont fogalma, kialakulásának feltételei**

* a holtpont olyan állapot, amely akkor következhet be, amikor több folyamat egyidejűleg verseng erőforrásokért, és egymást kölcsönösen blokkolják
* ha egyik sem tud továbblépni, mert éppen arra az erőforrásra lenne szüksége, amit más foglal: ez holtponthoz vezet
* kialakulásához szükséges feltételek:
  + **kölcsönös kizárás 🡪** legalább egy – többek által igényelt – erőforrás nem megosztható, azaz egyszerre csak egy folyamat használhatja
  + **foglalva várakozás 🡪** valamelyik folyamat lefoglalt egy erőforrást és arra vár, hogy továbbiakat foglaljon le
  + **nem elvehető erőforrások 🡪** az erőforrást a foglalótól nem lehet kivülről elvenni
  + **körkörös várakozás 🡪** a körben álló folyamatok mindegyike a következő által foglalt erőforrásra vár

1. **Virtuális memória. Virtuális memória megvalósítása lapozással**

* a virtuális memória lényegében a címzési tartományt jelenti. 32 bites rendszereknél 4GB, 64-eseknél sokkal több
* mivel ennyi memória általában nem áll rendelkezésre (64 bit esetén), illetve rossz cím megadása miatt az egész program leállhat, szükséges ennek kezelése
* megoldásként a virtuális címzés nem direkt a címsínre megy, hanem az MMU-ba, mely lényegében átkonvertálja fizikai címmé
* ha ez a cím nincs benne az operatív memóriában, akkor megkeresi, majd kicseréli a legrégebben nem használt adattal, majd frissíti mindkettőnek a fizikai címét
* ezáltal a címzéstartomány teljes egészét megcímezhetjük
* a **map**-elés hátulütője, hogy nagyon sok helyet foglalhat, ezért nem ajánlott a direkt használata
* megoldást jelent a lapozás
* a lapok adott méretű memóriarészek, általában 1-2-4 kB, de lehet több is
* 4 kB esetén, ez a tartomány megcímezhető 12 bittel
* a program nem virtuális címzést használ, hanem egy **lapszámot**, mely lényegében a 4kB-os részek kezdőcíme, majd egy **eltolást**, mely a pontos címet adja meg ezen belül
* így, mikor címezni akarunk, az MMU ellenőrzi, hogy az adott blokk bent van-e a memóriában, ha igen a lap kezdőcímét átalakítja, majd hozzáadja az eltolást, így megkapva a fizikai címet
* ha nem található benne, akkor az adott méretű lapot kicseréli a legrégebben használtal, majd frissíti a lapcímeket
* ezáltal sokkal több memória megcímezhető, és a **mappelés** folyamata sem vesz akkora memóriát igénybe