Operációs rendszerek kérdések

- ***Milyen processzor védelmi szinteket ismer, hol használjuk ezeket?***
  Intel 80286: minden utasítás egyenlő
  Intel 80386: 4 különböző védelmi szint
  - kernel mód (védett, protected): operációs rendszer magjánál használjuk
  - felhasználói mód: felhasználói felületnél használjuk
  - másik 2 módot nem használja
- ***Mi az operációs rendszer kernel módja és felhasználói módja közti különbség?***
  Az operációs rendszer védelmének érdekében kettő vagy több CPU működési mód létezik:
  - kernel mód: Minden utasítás és regiszter elérhető
  - felhasználói mód: Csak „biztonságos" utasítások és regiszterek használhatók
- ***Mi biztosítja a kernel mód – felhasználói mód közti különbséget?***
  Korszerű processzorokra jellemző, hogy legalább két futási módjuk van. A futási módok között privilégiumi, kiváltsági, fontossági különbségek vannak. A kevésbé privilegizált mód a normál mód, más néven felhasználói mód (user mode). A privilegizáltabb futási mód neve a védett mód, kernel mód (kernel mode).
  A processzornak védett módban szélesebb az utasításkészlete (azaz bizonyos gépi utasításokat csak védett módban tud hiba nélkül végrehajtani) és szélesebb a címtartománya (azaz bizonyos memória címekre csak védett módban képes hiba nélkül hivatkozni).
- ***Milyen kommunikációs típust ismerünk a perifériákkal?***
  - Lekérdezéses átvitel (I/O portok folyamatos lekérdezése)
  - megszakítás használat (nem kérdezgetjük folyamatosan, hanem a kívánt esemény bekövetkezésekor a megadott programrész kerül végrehajtásra)
  - DMA: közvetlen memória elérés
- ***Mi a POSIX?***
  POSIX = Portable Operating System Interface for uniX.
  A POSIX valójában egy minimális rendszerhívás (API) készlet, szabvány. Szabvány ANSI C-vel azonos függvénykönyvtár. Ma gyakorlatilag minden OS POSIX kompatibilis. A Windows-nak is van POSIX felülete.
- ***Mik a(z) 0., 1., 2., 3., 4. generációs operációs rendszerek jellemzői?***
  0. történelmi generáció
     - 1940-ig
     - tisztán mechanikus
     - nincs operációs rendszer
     - operátor alkalmazás
  1. első generáció
     - 1940-1955
     - sajátos kapcsolótábla, relé, vákuumcső, lyukkártyák megjelenése
     - Neumann János elvei
     - egyedi gépek
     - gépi kód, egyszerű matematikai számítások
  2. második generáció
     - 1955-1965
     - tranzisztoros rendszerek
     - megbízhatóvá váltak az elemek
     - géptermek kialakulása
     - tervezés, gyártás, programozás, üzemeltetés fázisának elkülönülése
     - lyukkártyás, szalagos egységek, kötegelt rendszer megjelenése
     - Fortran nyelv
     - operációs rendszerek
       - FMS (Fortran monitor system)
       - IBM 7094 hármasa (beolvasó, feldolgozó, megjelenítő)

3. harmadik generáció
    - 1965-1980
    - integrált áramkörök megjelenése
    - azonos rendszerek, felépítések, kompatibilitás megjelenése
    - OS/360 megjelenése (univerzális, nagyméretű, bonyolult operációs rendszer)
    - multiprogramozás, multitask megjelenése (több feladat a memóriában egyidejűleg)
    - spooling (pufferelés és sorba állítás kombinációja perifériáknál), időosztás megjelenése
    - nincs közvetlen on-line munka
4. negyedik generáció
    - 1980-tól
    - személyi számítógépek
    - MS Windows
    - LSI áramkörök, CPU fejlődés
    - Intel x86 család, IBM PC-DOS, MS DOS (parancssoros felület)
    - GUI – MS Windows, Mac OS X
    - hálózati, osztott rendszerek

❖ ***Mi a „virtuális gép" operációs rendszer struktúra lényege, honnan ered ez az elv?***

A virtuális gép a rétegelt modell általánosítása: nemcsak a fizikai hardvert, hanem az operációs rendszer magját is hardvernek tekinti. A virtuális gép a rendelkezésre álló hardverrel azonos interfészt nyújt. Az operációs rendszer azt az illúziót kelti, mintha minden processzus saját processzort és saját (virtuális) operatív memóriát használna. Erről az Exokernel gondoskodik. Az IBM-től származik az ötlet.

❖ ***Honnan származik, és mi a lényege a virtuális gépek (szerver) használatának?***

Az IBM-től származik az ötlet.

Egy hardveren egyszerre több operációs rendszer is futtatható virtuális gépként. Ma gyakorlatilag teljes a hardveres támogatás. Egy helyről vezérelhetőek a szerverek (virtuális gépek).

❖ ***Hol használtak a beágyazott rendszerek, mik azok?***

Egy beágyazott rendszer olyan speciális célú számítógép, melyet egy konkrét feladat ellátására terveztek. Fizikai méretüket tekintve a beágyazott rendszerek a hordozható eszközöktől (digitális óra, GPS, mobiltelefon, digitális kamera) az egészen nagy méretű helyhez kötött berendezésekig (közlekedési lámpa, riasztó, router, atomerőmű irányító rendszere) terjednek.

❖ ***Mi a „konténer technológia"? Létezik ilyen?***

A virtuális gépeket váltotta fel a konténer technológia, amely az elvárásoknak jobban eleget tesz, gyorsabb és rugalmasabb működést biztosít alacsonyabb erőforrásigény és minimális teljesítményveszteség mellett.

A Docker egy új konténer technológia és hihetetlen népszerű.

❖ ***Mi a CHS címzés?***

Egy korai módszer a mágneslemez fizikai blokkjainak címzésére. Cilinder – Head – Sector.

❖ ***Mi a CHS-LBA címzés közi különbség? Van egyáltalán?***

Az LBA címzés a CHS (cilinder, fej, szektor) hármasa hivatkozás helyett minden szektornak egy egyedi számot ad 0-tól (N-1)-ig, ahol N a szektorok száma a lemezen.

❖ ***Írja le az FCFS ütemezés lényegét és jellemzőit!***

A sorrendi ütemezés (First Come First Served – FCFS) nem megszakítható. Egy folyamat addig fut, amíg nem végez, vagy nem blokkolódik. Egy pártatlan, egyszerű láncolt listában tartjuk a folyamatokat. Ha egy folyamat blokkolódik, akkor a sor végére kerül. Hátránya, hogy az I/O igényes folyamatok nagyon lassan végeznek. Ha az aktuális sorrendi kérés kiszolgálás helyén van egy másik kérés blokkja (mozgás nélkül elérhető), akkor szolgáljuk ki azt is.

❖ ***Írja le az SSTF ütemezés lényegét és jellemzőit!***

Shortest Seek Time First, leghamarabb elérhetőt először. Leghamarabb elérhető műveletet hajtva végre, legkisebb fejmozgást részesíti előnyben. Átlagos várakozási idő kicsi, várakozási idő szórása nagy. Átviteli sávszélesség nagy. Fennáll a kiéheztetés veszélye.

❖ ***Mi az MBR?***

A Master Boot Record (MBR) vagy más néven a partíciós szektor a merevlemez legelső szektorának elnevezése. Csak a partícionált merevlemezeknek van MBR-jük. Mérete 512 byte, két részből áll:

- rendszerindító kód (bootloader): mely elindít egy operációs rendszert a merevlemezről
- partíciós tábla: mely a merevlemez-partíciók elhelyezkedési adatait tárolja.

❖ ***Milyen fájlrendszer specifikus fájlokat ismer? Hol találhatók általában?***

Unix rendszereken karakter-specifikus és blokk-specifikus fájlok vannak a /dev könyvtárban.

❖ ***Mit jelent az „interleave" fogalma?***

Az interleave egy módszer az adattárolás hozzáférési teljesítményének javítására úgy, hogy az egymás utáni adatok tárolása nem egymást követő szektorokba történik, hanem szétszórtan.

❖ ***Mit nevezünk fájlnak és könyvtárnak?***

A fájl adatok egy logikai csoportja, névvel és egyéb paraméterekkel ellátva.
A könyvtár fájlok (könyvtárak) logikai csoportosítása.

❖ ***Mit nevezünk fájlrendszernek, mi köze van az FCFS ütemezéshez?***

A fájlrendszer módszer a fizikai lemezünkön, kötetünkön a fájlok és könyvtárak elhelyezés rendszerének kialakítására. A sorrendi ütemezéssel (FCFS) olvashatunk és írhatunk a lemezre, aminek a rendszerét a fájlrendszer adja.

❖ ***Mi az Ext2FS, van-e MBR-je?***

Az Ext2FS egy UNIX-ra tervezett fájlrendszer.
Minden partícionált merevlemeznek van MBR-je függetlenül a fájlrendszertől, így az Ext2FS-nek is.

❖ ***Mutassa be a FAT32 fájlrendszer fontosabb jellemzőit! Használják még?***

FAT (File Allocation Table)

- talán a legrégebbi, ma is élő fájlrendszer
- a partíció 0. blokkja leírja a rendszer jellemzőit
- a FAT tábla a lemez foglalási térképe, annyi eleme van ahány blokk a lemezen
- lévén egy fájl jó eséllyel több blokkon helyezkedik el (akár nem is szekvenciálisan), így a FAT a katalógusában a fájl adatok (név,…) mellett csak a fájl első blokk sorszámát tárolja el
- a FAT blokk azonosító mutatja a fájlhoz tartozó következő blokk címét, ha nincs ilyen, az értéke FFF
- a fájl utolsó módosítási idejét is tárolja
- töredezettségmentesítés szükséges, amennyiben a fájlok blokkjai nagyon szétszórva helyezkednek el a lemezen (ennek hiányában jelentősen lelassulnak az I/O műveletek)

❖ ***Mutassa be az NTFS fájlrendszer fontosabb jellemzőit!***

NTFS (New Technology File System)

- kifinomult biztonsági beállítások, mint a FAT-nél
- titkosított fájlrendszer támogatása
- naplózás
- POSIX támogatás
- fájlok és mappák tömörítése
- felhasználói kvóta kezelés (az egyes felhasználók csak a lemez bizonyos részeihez férhessenek hozzá)
- csak klasztereket tart nyilván, szektort nem
- szükséges töredezettség mentesítés
- a partíció az MFT táblázattal kezdődik

❖ ***Mi az MFT?***

Az NTFS partíció az MFT (Master File Table) táblázattal kezdődik.

- 16 attribútum ad egy fájl bejegyzést.
- Minden attribútum maximum 1kb. Ha ez nem elég, akkor egy attribútum mutat a folytatásra.
- Az adat is egyfajta attribútum, így egy bejegyzés több adatsort tartalmazhat.
- Ha a fájl < 1kb, akkor belefér az attribútumba, közvetlen fájl.
- Elvi fájlméret 2^{64} bájt.
- Nincs maximum fájl méret.

❖ ***Mit jelent a „diszk kvóta” szolgáltatás?***

?

❖ ***Mi a RAID(1..5), mi a működésének a lényege?***

RAID – Redundant Array of Inexpensive Disks.

Ha operációs rendszer nyújtja, gyakran SoftRaid-nek nevezik. Ha intelligens (külső) vezérlőegység nyújtja, gyakran Hardver Raid-nek, vagy csak Raid diszkrendszernek nevezik.

Bár nevében olcsó, valójában inkább nem az.

Több lemezt fog össze, és egy logikai egységként látja az operációs rendszer.

Többféle „összefogási” elv létezik: RAID 0-6. Ma leggyakrabban a RAID1 és RAID5 verziókat használják.

- RAID0
  - Több lemez logikai összefűzésével egy meghajtót kapunk.
  - A lemezkapacitások összege adja az új meghajtó kapacitását.
  - A logikai meghajtó blokkjait szétrakja a lemezekre (striping), ezáltal egy fájl írása több lemezre kerül.
  - Gyorsabb I/O műveletek.
  - Nincs meghibásodás elleni védelem (nem redundáns).
- RAID1
  - Két független lemezből készít egy logikai egységet.
  - Minden adatot párhuzamosan kiír mindkét lemezre (tükrözés, mirror).
  - Tárolókapacitás felére csökken. Drága megoldás.
  - Mindkét lemez egyszerre történő meghibásodása okoz adatvesztést.
- RAID1+0: Tükrös diszkekből vonjunk össze többet.
- RAID0+1: RAID0 összevont lemezcsoportból vegyünk kettőt.
- RAID2: Adatbitek mellet hibajavító biteket is tartalmaz (ECC). Pl. 4 diszkhez 3 javító diszk.
- RAID3: Elég egy plusz „paritásdiszk”, n+1 diszk, $\sum$ n a kapacitás
- RAID4: RAID0 kiegészítése paritásdiszkkel.
- RAID5
  - Nincs paritásdiszk, ez és az adatok is el vannak osztva a tömb összes elemére (stripe set).
  - Intenzív CPU igény.
  - Redundáns tárolás, 1 lemez meghibásodása nem okoz adatvesztést.
  - 2 lemez egyidejű meghibásodása már adatvesztést okoz.
  - n lemez RAID 5 tömbben (n >= 3), n-1 lemez méretű logikai meghajtót ad.
- RAID6
  - A RAID 5 paritásblokkhoz hibajavító kód kerül tárolásra (+1 diszk).
  - Még intenzívebb CPU igény.
  - Két diszk egyidejű kiesése sem okoz adatvesztést.
  - Relatív drága.
  - n diszk RAID6-os tömbjének kapacitása n-2 diszk kapacitásával azonos.
  - Elvileg általánosítható a módszer (3 diszk kiesése…)

❖ ***Mi a RAID0+1 illetve RAID1+0 lemezek közti különbség?***

A RAID1+0 a tükrös diszkeket vonja össze, míg a RAID0+1 az összevont logikai diszket tükrözi.

❖ ***Mi a RAID5 és RAID6 közti különbség, mi a működésük lényege?***

RAID5 esetén nincs paritásdiszk, ez el van osztva a tömb összes elemére, paritásblokkokra. RAID6 ugyanez, csak van hibajavító kód is tárolva +1 diszken. Mind a kettő nagy CPU igényű, RAID6-nak nagyobb. RAID5 esetén egy lemez meghibásodása esetén nincs adatvesztés, míg RAID6 esetén két lemez meghibásodása esetén sincs.

❖ ***Ismertesse a folytonos tárkiosztás (lemez) stratégiáit, jellemzőit!***

- Egy elhelyezési stratégia.
- 3 fajtája van, mindegyik veszteséges lemezkihasználású
  - First Fit (első szabad hely, ahová befér)
  - Best Fit (arra a helyre, ahol a legkevesebb szabad hely marad)
  - Worst Fit (arra a helyre illesztjük, ahol a legtöbb szabad hely marad)

❖ ***Mi az i-node tábla / indextábla?***

A katalógus tartalmazza a fájlhoz tartozó kis tábla (úgynevezett i-node) címet. E tábla segítségével érhetőek el azok a blokkok, melyek a fájlt tartalmazzák (ugyanis egy fájl gyakran több blokkban van).

Az i-node tábla 15 rekeszből áll, melyből az első 12 a fájl blokkjaira mutat. Ha ez kevés, akkor a 13. rekesz egy újabb i-node-ra mutat, mellyel +15 rekesz érhető el. Amennyiben ez is kevés, a 14. rekeszbe újabb i-node kerülhet, és így tovább.

❖ ***Milyen partíciónak nincs i-node táblája?***

Minden nem UNIX fájlrendszerű partíciónak nincs i-node táblája.

❖ ***Milyen I/O eszközkategóriákat ismer? Mi a kivétel?***

- Blokkos eszközök
  - adott méretű blokkban tároljuk az információt
  - blokkméret 512 byte – 32768 byte között
  - egymástól függetlenül írhatók vagy olvashatók a blokkok
  - blokkonként címezhető
  - ilyen eszköz: HDD, CD, szalagos egysége, …
- Karakteres eszközök
  - nem címezhető
  - csak jönnek-mennek sorban a „karakterek” (bájtok)
  - ilyen eszköz: egér, billentyűzet, …
- Kivételek
  - időzítő: nem blokkos és nem karakteres, meghatározott időintervallumonként megszakításokat hoznak létre
  - memória leképezésű képernyők

❖ ***I/O szoftver modellje, milyen eszközkategóriákat ismer?***

A szoftverrendszere rétegesen épül fel (tipikusan 4 réteg).

1. megszakítást kezelő réteg – legalsó kernel szinten kezelt, szemafor blokkolással védve
2. eszközmeghajtó programok
3. eszköz független operációs rendszer program
4. felhasználói I/O eszközt használó program

❖ ***Mire szolgál az I/O port? Létezik egyáltalán?***

?

❖ ***Mire valók a megszakítások?***

A megszakítások nagyon fontos elemei a számítógépek működésének. Amikor a mikroprocesszornak egy eszközt vagy folyamatot ki kell szolgálnia, akkor annak eredeti tevékenységét felfüggesztve, megszakítások lépnek életbe. Létezik szoftveres és hardveres megszakítás.

❖ ***Mi a szoftveres és a hardveres megszakítás közi különbség? Van egyáltalán?***

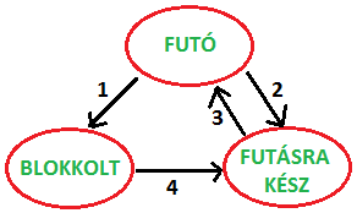
A szoftveres megszakítás kezelése azonos a hardveres megszakítás kezelésével. Különbség a megszakítás forrása.

A hardveres megszakítás esetén a processzor erőforrást (processzoridő) igénylő eszköz megszakításkérést küld a mikroprocesszor megszakítás engedélyezés kérés vezetékéhez. A megszakításvezérlő áramkör szabályozza a megszakítások kiszolgálását. Amennyiben a megszakítás lehetséges, a kérést kezdeményező eszköz használhatja a mikroprocesszort.

A szoftveres megszakítás esetén a főprogram futását egy alprogram szakítja meg. Ebben az esetben a főprogram futásállapota elmentésre kerül, majd miután a megszakítást kérő program befejezte a műveletet a főprogram folytatja a futását a megszakítás előtti pozícióból. Példa erre, amikor a folyamat egy rendszerhívást tesz meg, ekkor a futása félbeszakad, végbemegy a hívás, és a folyamat futása folytatódik.

❖ ***Az operációs rendszerek folyamatainak milyen állapotait, állapotátmeneteit ismerjük?***

- futó
- futásra kész: ideiglenesen leállították, arra vár, hogy az ütemező CPU időt adjon a folyamatnak.
- blokkolt: ha logikailag nem lehet folytatni a tevékenységet, mert pl. egy másik eredményére vár.



1. futó → blokkolt : várni kell valamire
2. futó → futásra kész : ütemező jelenleg mást futtat
3. futásra kész → futó : az ütemező most ezt a folyamatot futtatja
4. blokkolt → futásra kész : a várt adat megérkezett

❖ ***Mi a különbség folyamatok és szálak között?***

Egy folyamat egy utasítássorozatnak (szálnak) felel meg. Néha szükséges lehet, hogy egy folyamaton belül „több utasítássorozat" legyen. A szál egy folyamaton belüli különálló utasítás sor. Folyamatnak önálló címtartománya van, szálnak nincs. Továbbá csak a folyamat rendelkezik a következőkkel: globális változók, megnyitott fájl leírók, gyermek folyamatok, szignálkezelők és ébresztők.

❖ ***Mit nevezünk kritikus tevékenységnek?***

Kritikus programterület, szekció az a rész, amikor a közös erőforrást használjuk.
Kritikus tevékenységnek nevezzük azt, amikor két vagy több folyamat keresztezi egymást a kritikus területen.

❖ ***Mit értünk monopol módú eszköz alatt?***

Egyedi (monopol) használatú erőforrások például a nyomtató, CD-író, szalagegység.
Egy időben csak egy folyamat használhatja az erőforrást.

❖ ***Hány folyamat lehet aktuálisan „running" (futó) állapotban?***

Egy időben csak egy folyamat lehet aktív. A folyamatok gyors váltogatásával „teremtetünk" párhuzamos végrehajtás érzetet.

❖ ***Mit takar a folyamat leíró táblázat?***

Folyamatleíró táblázat (Process Control Block, PCB)
A rendszer inicializálásakor jön létre. Már indításkor is tartalmaz 1 elemet, a rendszerleíró már a rendszer indulásakor bekerül. Tömbszerű szerkezete van (PID alapján). Egy sorában egy összetett processzus adatokat tartalmazó struktúra található. Egy folyamat fontosabb adatai: azonosítója, neve, tulajdonosa, csoportja, stb.

❖ ***Mi a kölcsönös kizárás, mik a megvalósítás feltételei?***

Kritikus programterület, szekció az a rész, amikor a közös erőforrást használjuk.
A jó kölcsönös kizárás az alábbi feltételeknek felel meg:

- nincs két folyamat egyszerre a kritikus szekcióban
- nincs sebesség, CPU paraméter függőség
- egyetlen kritikus szekción kívüli folyamat sem blokkolhat másik folyamatot
- egy folyamat sem vár örökké, hogy a kritikus szekcióba tudjon belépni

❖ ***Ismertesse a kölcsönös kizárás „szigorú váltogatás" megvalósítását!***

A kölcsönös kizárás feltételeit teljesíti, kivéve azt, hogy egyetlen kritikus szekción kívüli folyamat sem blokkolhat másik folyamatot. Több folyamatra is általánosítható.

❖ ***Mi a probléma a kölcsönös kizárás „szigorú váltogatás”-os megvalósítással?***

A kölcsönös kizárás egyik feltételét nem teljesíti (egyetlen kritikus szekción kívüli folyamat sem blokkolhat másik folyamatot). Például ha az 1. folyamat a lassú, nem kritikus szekcióban van, és a 0. folyamat gyorsan belép a kritikus szekcióba, majd befejezi a nem kritikus szekciót is, akkor az 1. folyamat saját magát blokkolja.

0. folyamat

```
while(1)
 {
   while(kovetkezo!=0) ;
   kritikus_szekcio();
   kovetkezo=1;
   nem_kritikus_szekcio();
 }
```

1.folyamat

```
while(1)
 {
   while(kovetkezo!=1) ;
   kritikus_szekcio();
   kovetkezo=0;
   nem_kritikus_szekcio();
 }
```

❖ ***Mit takar az alábbi algoritmus részlet, mi a baj vele, ha van?***

```
while(1)
{
   while(kovetkezo!=1);
   kritikus_szekcio();
   kovetkezo = 0;
   nem_kritikus_szekcio();
}
```

Az algoritmus a kölcsönös kizárás „szigorú váltogatás”-os megvalósítása. Az a baj vele, hogy a kölcsönös kizárás egyik feltételét nem teljesíti (egyetlen kritikus szekción kívüli folyamat sem blokkolhat másik folyamatot), így például egy folyamat blokkolhatja saját magát.

❖ ***A kölcsönös kizárás Peterson féle megoldásának mi a lényege? (algoritmus)***

A szigorú váltogatás javítása. A kritikus szekció előtt minden folyamat meghívja a belépés, majd utána a kilépés függvényt.

```
while(1)
 {
  belepes(processz);
  kritikus_szekcio();
  kilepes(processz);
  nem_kritikus_szekcio();
 }
```

❖ ***Mit takar az alábbi algoritmus részlet, mi a jellemzője?***

```
…
void belepes(int proc)
{
   int masik;
   masik=1-proc; //mivel N=2
   kovetkezo=proc;
   akarja[proc]=1; //proc futni akar
   while (kovetkezo==proc && akarja[masik]);
}
…
```

A kölcsönös kizárás Peterson féle megoldásának belepes() eljárásának a javítása. A javítás során csak az eljárás 3. és 4. sora lett megcserélve, mást nem módosítottunk.

❖ ***Ismertesse a kölcsönös kizárás megvalósítását TSL utasítással!***

A mai rendszerekben a processzornak van egy „TSL reg, lock” formájú utasítása (TSL – Test and Set Lock). Ez az utasítás beolvassa a „lock” memóriaszó tartalmát a „reg” regiszterbe, majd egy nem nulla (pl. az 1) értéket írja a „lock” memóriacímre. Ez egy megszakíthatatlan atomi művelet.

Lényeges, hogy ezen művelet során a memóriasín zárolva van, tehát más CPU-k (vagy processzormagok) nem érhetik el ebben az időszakban a memóriát. Ez egy nagyon egyszerű megvalósítási lehetőséget ad arra, hogy készítsünk a TSL utasítás segítségével egy „belép”, illetve „kilép” eljárást, amelyek közé foglalhatjuk a kritikus szekciót.

❖ ***Mit értünk tevékeny várakozás alatt?***

Mikor a CPU-t „üres” ciklusban járatjuk a várakozás során, CPU időt pazaroljuk.

❖ ***Mit takar, mire jó az alábbi algoritmus részlet, mi a jellemzője?***

```
zöld-fehér:
        TSL regiszter, széf
        cmp regiszter, 0
        jne zöld-fehér
        ret
```

Tevékeny várakozás (CPU idő pazarlás) gépi kódban. TSL utasítást használ, atomi művelet (megszakíthatatlan). Kritikus szekciónál használjuk.

❖ ***Mit értünk folyamatok erőforrás görbéjén, mire használható?***

Folyamatosan figyeljük az erőforrás igényeket, elengedéseket. Figyeljük, hogy biztonságos állapotba kerülünk-e az újabb erőforrás kiosztással, ahol az erőforrások vannak a központban. Ezt az erőforrás pályagörbe vizsgálatával tesszük.

❖ ***Mi a szemafor?***

Egyfajta „kritikus szakasz védelem”-re szolgáló egész változó. Dijkstra javasolta a bevezetését. A szemafor tilosat mutat, ha értéke 0. Ha értéke >0, akkor az adott folyamat beléphet a kritikus területre. Két művelet tartozik hozzá:

- belépéskor a szemafor értékének csökkentése (down)
- kilépéskor az érték növelése (up)
- Ezeket Dijkstra P és V műveleteknek nevezte.

❖ ***Mit takar az alábbi algoritmus, mi a jellemzője?***

```
void valami(int i)
{
  while(1)
  {
    ülök();
    kell_ágy(i); //ball ágy
    kell_ágy((i+1)%N); //jobb ágy
    fekszem_az_ágyon();
    nemkell_ágy(i);
    nemkell_ágy((i+1)%N);
  }
}
```

Étkező filozófusok probléma megvalósítása szemaforral. Ennél a megvalósításnál holtpont alakulhat ki.

❖ ***Mikor nem használható a szemafor?***

?

❖ ***Mi a mutex?***

Ha a szemafor tipikus vasutas helyzetet jelöl, azaz 1 vonat mehet át csak a jelzőn, akkor a szemafor értéke 0 vagy 1 lehet. Ez a bináris szemafor, más nevén mutex. Kölcsönös kizárásra használjuk.

❖ ***Mi a különbség a szemafor és a mutex között?***

Az a különbség, hogy míg az utóbbi csak kölcsönös kizárást tesz lehetővé, azaz egyszerre mindig pontosan csakis egyetlen folyamat számára biztosít hozzáférést az osztott erőforráshoz, addig a szemafort olyan esetekben használják, ahol egynél több, de korlátos számú folyamat számára engedélyezett a párhuzamos hozzáférés.

❖ ***Mi a monitor?***

Magasabb szintű nyelvi konstrukció, ami megoldja a szemaforok „könnyen elrontható” problémáját. Monitorban eljárások, adatszerkezetek lehetnek. Egy időben csak egy folyamat lehet aktív a monitoron belül. Ezt a fordítóprogram automatikusan biztosítja. Ha egy folyamat meghív egy monitor eljárást, akkor először ellenőrzi, hogy másik folyamat aktív-e? Ha igen, akkor felfüggesztésre kerül. Ha nem, akkor beléphet, végrehajthatja a kívánt monitor eljárást.

❖ ***Mit jelent a monitor „condition” típusa?***

Mi van, ha egy folyamat nem tud továbbmenni a monitoron belül? Ennek megoldására használjuk az állapot változókat (condition). Rajtuk két művelet végezhető: wait, signal.

wait(): Az állapot bekövetkeztére vár, eközben más folyamatok is beléphetnek a monitorba.

signal(): Jelzi az állapot bekövetkeztét. Ha nincs az állapotra váró folyamat, nem csinál semmit. Egyébként kiválaszt egy várakozó folyamatot, és felébreszti. (A signal()-t hívó folyamat a felébresztett monitorhívás befejeződéséig várakozik.)

❖ ***Mire használható a monitor?***

Kölcsönös kizárás megvalósítására használjuk és ez a magasabb szintű konstrukció sokkal biztonságosabb, mint a szemaforos megoldás.

❖ ***Mit takar az alábbi algoritmus részlet, mi a jellemzője?***

```
…
condition tele, üres;
int darab;
főz(hal ponty)
{
  if (darab==N) wait(tele);
  süt(ponty);
  darab++;
  if (darab==1) signal(üres);
}
…
```

Gyártó-Fogyasztó probléma megvalósítása monitorral. Kölcsönös kizárást tesz lehetővé.

❖ ***Mi a különbség a monitor és a mutex között?***

A mutex egy speciális bináris szemafor, ami 0 vagy 1 értékű (egyszerű változótípus). Tehát a gyorsabb folyamatnak lezárja az erőforrást. A monitor egy magasabb szintű nyelvi konstrukció a kölcsönös kizárásra (eljárások és adatszerkezetek lehetnek benne).

❖ ***Miért hasznos a kölcsönös kizárás üzenetküldéses megvalósítása?***

Ha küldő-fogadó nem azonos gépen van, akkor szükséges úgynevezett nyugtázó üzenetet küldeni. Ha küldő nem kapja meg a nyugtát, ismét elküldi az üzenetet. Ha a nyugta veszik el, a küldő újra küld. Ismételt üzenetek megkülönböztetése sorszám segítségével.

❖ ***Mit takar az alábbi algoritmus rész, mi a jellemzője?***

```
void munkás() //munkás folyamata
{
   int termék; //termék tárolási hely
   message m; //üzenet tároló helye
   while(1) //folyamatosan dolgozunk
   {
      termék = készít();
      fogad(vásárló,m); //vásárlói
      m = üzenet_készítés(termék);
      küld(vásárló,m); //küldés
   }
}
```

Gyártó-fogyasztó probléma megoldása üzenetküldéssel. Ha küldő-fogadó nem azonos gépen van, akkor szükséges úgynevezett nyugtázó üzenet. Az üzenetküldés a párhuzamos rendszerek általános technikája.

❖ ***Mi a randevú stratégia?***

Egy üzenetküldési stratégia. Az üzenetek tárolására ideiglenes tároló helyet (levelesláda) hozzunk létre mindkét helyen. A randevú stratégia esetén ezeket elhagyjuk. Ilyenkor ha send előtt van receive, akkor a küldő blokkolódik, illetve fordítva.

❖ ***Mit értünk folyamatok ütemezésén?***

Az ütemező a folyamatok gyors váltogatásával teremt párhuzamos végrehajtás érzetet.

Egy ütemezési algoritmus alapján eldönti az ütemező, hogy mikor melyik folyamat futhat.

❖ ***Mi a garantált ütemezés lényege?***

Minden aktív folyamat arányos CPU időt kap. Nyilván kell tartani, hogy egy folyamat már mennyi időt kapott. Ha egy folyamat arányosan kevesebb időt kapott, akkor az kerül előre.

❖ ***Mi a „sorsjáték ütemezés" lényege, hol használják?***

A folyamatok között „sorsjegyeket" osztunk szét, az kapja a vezérlést, akinél a húzott jegy van. Arányos CPU időt könnyű biztosítani, hasznos pl. videó szervereknél.

❖ ***Mi az arányos ütemezés lényege?***

Az ütemezés során figyelembe vesszük a folyamatok kapott ideje mellett a felhasználókat is.
Olyan, mint a garantált ütemezés, csak itt a felhasználókra vonatkoztatva.

❖ ***Jellemezze a CFS ütemezést!***

- hasonlít a garantált ütemezésre
- a CPU idők nyilvántartása egy fa struktúrában, balra kisebb, jobbra nagyobb idejű folyamatok
- nincs direkt prioritás
- mindenki azonos, fair módon részesül a CPU erőforrásokból, de
  - a nagyobb prioritású folyamat kisebb idő csökkentést szenved el
  - az alacsonyabb prioritású nagyobbat
- így érvényesül a prioritási elv, nincs kiéheztetés
- nem kell prioritásonként folyamat nyilvántartás

❖ ***Mi a Round-Robin ütemezés lényege?***

Körben járó ütemezés, mindenkinek van időszelete, aminek a végén, vagy blokkolás esetén jön a következő folyamat. Pártatlan, egyszerű ütemezés. Egy listában tárolhatjuk a folyamatokat (jellemzőit), és ezen megyünk körbe-körbe. Időszelet végén a körkörös listában következő lesz az aktuális folyamat.

❖ ***Mi a valós idejű ütemezés lényege?***

Garantálni kell adott határidőre a tevékenység, válasz megadását.

❖ ***Mi a hard real time / soft real time rendszer?***

Valós idejű rendszerek megvalósításai.
Az idő kulcsszereplő. Garantálni kell adott határidőre a tevékenység, válasz megadását.
Hard Real Time: szigorú, abszolút, nem módosítható határidők.
Soft Real Time: toleráns, léteznek a határidők, de ezek kismértékű elmulasztása tolerálható.
A programokat több kisebb folyamatokra bontják.

❖ ***Mit nevezünk valós idejű operációs rendszernek?***

Egy olyan operációs rendszer, amelyben minden egyes rendszerhívás egy előre meghatározott időn belül garantáltan végrehajtásra kerül, a konkrét körülményektől függetlenül.

❖ ***Mit jelent a holtpont? Adja meg a holtpont kialakulásának feltételeit!***

Két vagy több folyamat egy erőforrás megszerzése során olyan helyzetbe kerül, hogy egymást blokkolják a további végrehajtásban.

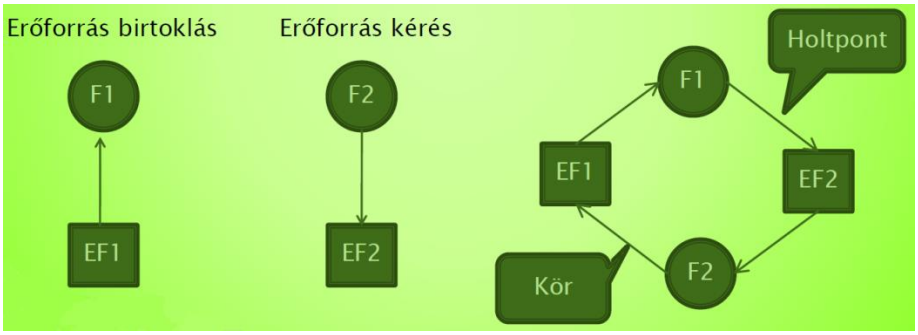
- kölcsönös kizárás feltétel: minden erőforrás hozzá van rendelve 1 folyamathoz vagy szabad
- birtoklás és várakozás feltétel: korábban kapott erőforrást birtokló folyamat kérhet újabbat
- megszakíthatatlanság feltétel: nem lehet egy folyamattól elvenni az erőforrást, csak a folyamat engedheti el
- ciklikus várakozás feltétel: két vagy több folyamatlánc kialakulása, amiben minden folyamat olyan erőforrásra vár, amit egy másik tart fogva

❖ ***Ismertesse a „probléma figyelmen kívül hagyása" módszert! Hol alkalmazzák?***

Ezt a módszert gyakran strucc algoritmus néven is ismerjük. Vizsgálatok szerint a holtpont probléma és egyéb összeomlások aránya 1:250, így nem törődünk vele, és reménykedünk, hogy nem okoz nekünk kárt. A Unix és a Windows világ is ezt a módszert használja (túl nagy az ár a várható haszonért cserébe).

❖ ***Mit jelent a holtpont gráfmodellje?***

Holtpont feltételek modellezése irányított gráfokkal, ahol a folyamatot körrel, az erőforrást négyzettel jelöljük. Ha az erőforrások, folyamatok irányított gráfjában kört találunk, az holtpontot jelent.



❖ ***Mit jelent a kiéheztetés?***

A rendszernek biztosítani kell a folyamatok erőforrás-igényének korlátos időn belüli kiszolgálását. Ha egy folyamat az igényelt erőforrást sohasem kapja meg (végtelen ideig vár), akkor mondjuk, hogy a folyamat „éhezik az erőforrásra”. Ez nagyon könnyen előfordulhat például prioritással rendelkező igények kiszolgálásakor. Ha magas prioritású igények gyakrabban érkeznek, mint az igények kiszolgálására fordított átlagos idő, akkor az esetleg befutó alacsonyabb prioritású igények semmi garanciát nem kapnak arra nézve, hogy meddig kell várniuk a kiszolgálásra.

❖ ***Mi a közös és különböző a holtpont és a kiéheztetés fogalmában?***

A holtpont és a kiéheztetés is az erőforrások megszerzése miatt jön létre. A kiéheztetés speciális esete a holtpont, amikor az éhezést maguk a fennálló igények okozzák, és valamelyik igény visszavonásával a többi továbbléphet.

❖ ***Ismertesse a bankár algoritmust! Lényege!***

A bankár algoritmus minden kérés megjelenésekor azt nézi, hogy a kérés teljesítése biztonságos állapothoz vezet-e. Biztonságos állapot egy olyan helyzet, melyből létezik olyan kezdődő állapotsorozat, melynek eredményeként mindegyik folyamat megkapja a kívánt erőforrásokat és befejeződik. Ha az algoritmus ilyen állapothoz vezet, akkor jóváhagyja, ha nem, akkor a kérést elhalasztja. Eredetileg 1 erőforrásra tervezett.

❖ ***Mit értünk virtuális memóriakezelésen, mi a lényege? Mi a lapozás?***

Egy program több memóriát is használhat, mint amennyi rendelkezésre áll, ezt a virtuális memória segítségével érjük el. A virtuális címtér azonos méretű egységekre, „lapokra” van osztva. Az ennek megfelelő egység a fizikai memóriában a lapkeret. A lapok és lapkeretek mérete mindig megegyező.

A lapozás egy virtuális memóriát használó memóriakezelő módszer, melyet használva a számítógép az elsődleges memória adatait tárolja el a virtuális memóriában (többnyire merevlemezen), majd szükség esetén ezt visszaolvassa.

❖ ***Mire szolgálnak a lapcserélési algoritmusok / lapcserélési stratégiák / lapozási algoritmusok?***

A virtuális tárkezelési módszer során is elkerülhetetlenül elérkezik az a pillanat, hogy a fizikai memória megtelik, azaz nincs üres hely benne, ahova behozható lenne az új memórialap. Ilyenkor lépnek életbe a lapcserélési stratégiák (lapozási algoritmusok), amelyek meghatározzák, hogy melyik lap helyére töltődjön majd be az új memóriatartalom.

❖ ***Mi a TLB, mi a szerepe?***

TLB – Translation Lookaside Buffer egy asszociatív cache, amit a memóriakezelő hardver használ, hogy gyorsítson a virtuális címfordítás sebességén.

❖ ***Ismertesse a laptáblák szerepét! Van köze a TLB-hez?***

A virtuális címet fizikai címre a laptábla segítségével lehet fordítani. Ez lassú, plusz egy memória-hozzáférést jelent. Ezért a lapkezdőcímek egy részét egy asszociatív cache-ben eltárolják, ez a TLB. Címfordításkor párhuzamosan indul a keresés a laptáblában és a TLB-ben. Ha az egyikben megtalálja, akkor befejeződik a keresés.

❖ ***Mit jelent az invertált laptábla fogalma?***

?

- ***Mire jó a „Dirty-bit”, hol használják?***
  Módosítás nyilvántartása a laptáblában. Minden memórialaphoz tartozik egy hardver által (CPU) automatikusan kezelt bit, amit betöltéskor törlik, módosításkor beállítják. Ha a dirty bit 1, akkor erre a lapra történt írási művelet mióta a fizikai memóriába került. 0 érték esetén a lapokat nem kell a diszkre kiírni, ha kiszorulnak a fizikai memóriából (hiszen nem lett módosítva a tartalma).
- ***Mi a lapozás során alkalmazott „munkahalmaz” modell?***
  ?
- ***Mi a memóriakezelő (MMU) feladata?***
  MMU – Memory Management Unit
  - CPU által kiadott memóriahozzáférési kérelmeket kezeli
  - a virtuális címek fizikai címekre való fordítása, azaz a virtuális memória kezelése
  - a memóriavédelem
  - CPU gyorsítótár vezérlése
  - sínütemezési funkciók
  - memória-bankváltás (lapozás)
- ***Ismertesse az MMU működését! Mutassa be ezt 64KB virtuális, 32KB valós címtér esetén!***
  ?
- ***Mit nevezünk szegmentált memóriakezelésnek?***
  A szegmentálás egy memóriakezelési módszer. Célja a memória több címtérre bontása. A memóriát logikai részekre úgynevezett szegmensekre osztják, és minden résznek megvan a saját, 0-tól kezdődő címtartománya. Egy memóriacím így két részből áll, egy szegmenscímből és egy offset címből, azaz a memória kétdimenziós. Két szinten valósul meg, hardver és operációs rendszer szinten. A lapozással ellentétben ez nem marad rejtve a felhasználó (programozó) előtt.
- ***Szegmentált memória használatnál szükség van-e virtuális memóriakezelésre?***
  ?
- ***Mi köze az operációs rendszer virtualizációnak a virtuális memóriakezeléshez?***
  ?
- ***Milyen dinamikus memória nyilvántartási módszereket ismer?***
  Amikor nem ismert egy program memória igénye, akkor dinamikus memória foglalást használunk. Kód fix szeletet, az adat és a verem változó méretet kapnak a memóriából. A foglalásra két allokációs módszer van, kicsi (nagy memória igény, kicsi lyukak a memóriában), nagy (nagy memória veszteség az egységek miatt). A dinamikus memória nyilvántartása bittérképpel vagy láncolt listával (külön lyuk és folyamat lista) történik.