1. **Mi a kiéheztetés és holtpont közti különbség (és mi a közös)? (2 pont) [161124, 170426]**

Kiéheztetés: a folyamat nem jut a számára szükséges erőforráshoz belátható időn belül az erőforrás-kezelő hibája miatt.

Holtpont: a folyamatok halmazának minden egyes eleme valamelyik másik halmazbeli folyamat által kiváltható eseményre várakozik (egy erőforrás felszabadulására, vagy számított eredményre).

1. **Mit jelent a holtpont gráfmodellje? (? pont) [100412]**

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

1. **Mik a holtpontkialakulás feltételei? (2 pont) [140430]**

A holtpont kialakulásának szükséges feltételei (Coffman)

- kölcsönös kizárás (mutex): legalább egy – többek által igényelt – erőforrás nem megosztható, azaz egyszerre csak egy processzus használhatja;

- foglalva várakozás (hold & wait): valamelyik processzus már lefoglalt egy erőforrást, és arra vár, hogy továbbiakat lefoglaljon;

- nem elvehető erőforrások (nonpreemptive): az erőforrást a foglalótól nem lehet kívülről – operációs rendszer beavatkozással – elvenni;

- körkörös várakozás (circular waiting): a körben állók folyamatok mindegyike a következő által foglalt erőforrásra vár.

1. **Mi a “strucc” algoritmus, hol használjuk? Milyen eszközöknél van (lehet) kivétel? / Ismertesse a “probléma figyelmen kívül hagyása” módszert? Hol alkalmazzák? (1 pont) [130422, 150420, 160425, 170426]**

A strucc algoritmus a „struccpolitika” elvét vallja, aminek egyszerűen a lényege az, hogy nem veszünk tudomást a holtpont létezéséről. Ha tudjuk azt, hogy egy esetleges holtpont kialakulásának valószínűsége meglehetősen kicsi, és az operációs rendszer újraindításának nincsenek kritikus következményei, akkor előfordulhat, hogy érdemes ezt a „megoldást” választani, mint holtpont „elkerülő” algoritmust.

Előnyök: Gyors holtpontelkerülés.

Hátrányok: Adatvesztés, a folyamatok végeredményeinek elvesztése fennállhat.

Holtpontok megléte: mivel az algoritmus szerint csak figyelmen kívül hagyjuk a holtpontokat, azaz a holtponthelyzeteket, azok ugyanúgy ott lesznek, és maguktól nem fognak köddé válni.

1. **Mi a virtuális rendszer fogalma? Honnan származik az elnevezés? / Honnan származik, mi a lényege a virtuális gépek (szerver) használatának? (2 pont) [111117, 131218p, 141120, 161219p]**

A "virtuális" szót általában úgy szokták magyarázni, hogy "olyan, ami van, de még sincs". Így átfogalmazva a virtuális gép egy olyan gép, ami nincs, de közben mégis van :)

Az IBM-től származik az ötlet. Egy hardveren egyszerre több operációs rendszer is futtatható virtuális gépként. Ma gyakorlatilag teljes a hardveres támogatás. Egy helyről vezérelhetők a szerverek (virtuális gépek).

1. **Mi a "virtuális gép" operációsrendszer-struktúra lényege, honnan ered ez az elv? (? pont) [100412]**

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.

1. **Mi a virtuális memóriakezelés és a virtuális gép közti különbség? (1 pont) [150420]**

A modern operációs rendszerek képesek arra, hogy látszólag több memóriát biztosítsanak a programoknak, mint amennyi fizikailag a rendelkezésükre áll. A módszert virtuális memóriakezelésnek hívják.

A virtuális gép nem csak a "nem létező" memóriát biztosítja, hanem egy egész virtuális számítógépet szolgáltat, mely bár csak virtuálisan, de rendelkezik minden szükséges hardver és szoftverelemmel.

1. **Mi a memóriakezelő feladata? Létezik egyáltalán? (1 pont) [11/12/2, 110511, 170426]**

(A folyamatok virtuális címterének címeit megfeleltesse fizikai címeknek.)

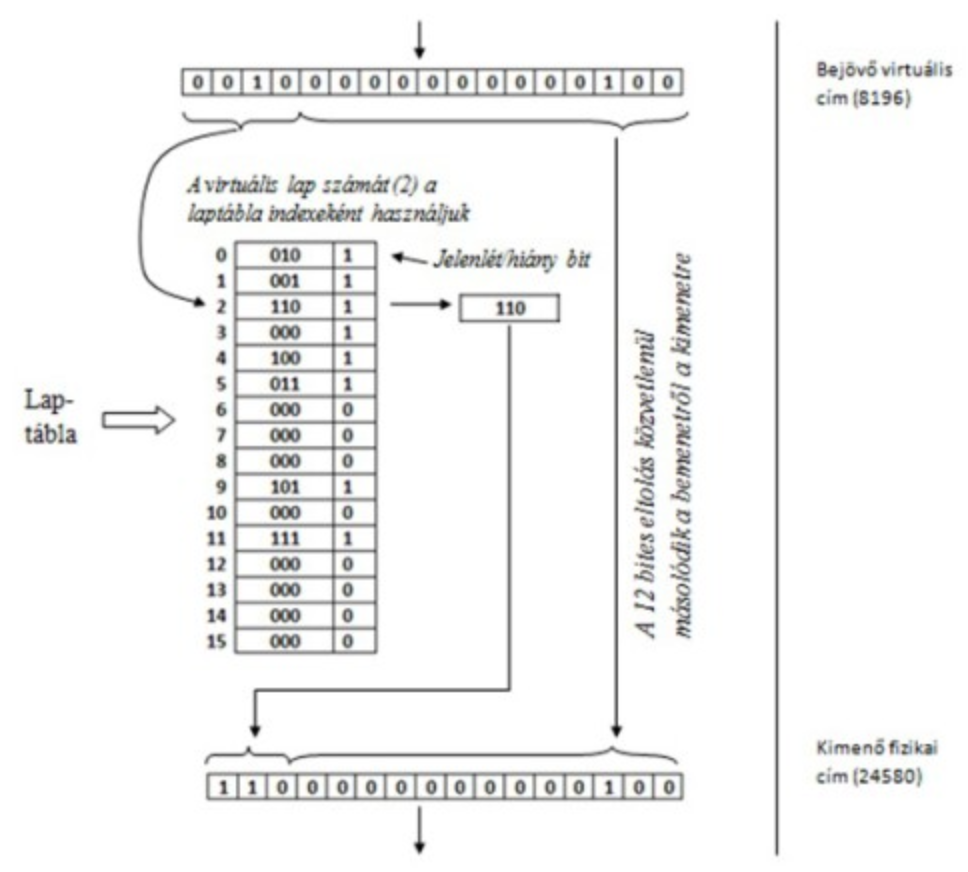
Minden program előállít valamilyen memóriacím-halmazt; ezeket virtuális címeknek nevezik. Ha nincs meg ez az eszközünk, akkor a memóriacímek közvetlenek. Ha használjuk a virtuális technikát – és a programunk nem fér be egyszerre még az egyre nagyobb és nagyobb memóriába se –, akkor a címek nem közvetlen memóriacímek, hanem egy MMU (Memory Management Unit) nevű egységbe kerülnek, ami leképezi azt egy fizikai címmé.

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)

1. **Ismertesse az MMU (memóriakezelő) működését! Mutassa be ezt 64 KB virtuális, 32 KB valós címtér esetén! (2 pont) [161124]**

A lenti ábra tetején megjelenő 16 bites virtuális címből (64 KB a virtuális címterünk, 4KB (12 bites) egy lap mérete és 32KB (15 bit) a fizikai memória nagysága), hogyan állítunk elő egy 15 bites fizikai címet? A 16 bites cím felső 4 bitjét használjuk a laptábla indexeként (0010=2), ezen indexű laptábla elem szintén 4 bites, mondjuk ennek felső 3 bitje a 12 bit kiegészítő bitjeit adja a 15 bites fizikai cím meghatározásához. A 4.bit a jelenlét bit. Ha nincs a fizikai memóriában a kért virtuális lapnak megfelelő memória, akkor jön a „lapcsere”, kitöltésre kerül a laptábla megfelelő bejegyzése, és folytatódik a program végrehajtás.



1. **Mit értünk virtuális memóriakezelésen, mi a lényege? (1 pont) [11/12/2, 110511, 130422, 160425] + Mi a lapozás? [111117, 141120]**

(A virtuális memória a számítógép RAM memóriáját a merevlemezen egy ideiglenes használatú területtel kombinálja. Ha nincs elegendő RAM memória, a virtuális memória az adatokat a RAM memóriából a lapozófájlnak nevezett területre mozgatja. Az adatok lapozófájlba mozgatása RAM memória területet szabadít fel, mellyel befejezhetők a feladatok.

[szegmentálás?] 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.

Az operációs rendszer úgy szabadít fel operatív memóriát az éppen futó program számára, hogy a memóriában tárolt, de éppen nem használt blokkokat (lapokat) kiírja a külső tárolóra, amikor pedig ismét szükség van rájuk, visszaolvassa őket.)

\*\*\*

A modern operációs rendszerek képesek arra, hogy látszólag több memóriát biztosítsanak a programoknak, mint amennyi fizikailag a rendelkezésükre áll. A módszert virtuális memóriakezelésnek hívják.

Az alapvető memóriakezelési módszerek két csoportba sorolhatók:

* végrehajtás közben mozgatják az egyes processzusokat a központi tár és a lemezterületek között (ennek módszerei a lapozás, illetve a csere);
* már a betöltéskor eldől, hogy mi hova kerül, és a futás végéig állandósul a memória, azaz a programon belüli esetleges dinamikus definíciókon kívül a memóriának a „programot” tartalmazó része állandó

Lapozás: A memóriakezelésnek az a módja, amelyben a logikai címtartományt (tehát amit a programozó lát) azonos méretű keretekre, a fizikai memóriát ugyanakkora lapokra bontjuk, és a logikai címből egy képlettel álltjuk elő a fizikai címet. Létezik két- , három- illetve négyszintű lapozás is.

1. **Honnan származik az operációsrendszer-virtualizáció, mik a jellemzői, mi köze a virtuális memóriakezeléshez? (2? pont) [121122]**
2. **Mi a virtualizáció (virtuális kiszolgáló) és a konténer technológia közti különbség? (2 pont) [171123]**

Q: What is the difference between a virtual machine and a container?

A:

Virtual machines are part of virtualization environments. A container, on the other hand, is a specific kind of virtualization strategy that offers its own benefits and disadvantages in different kinds of virtualization platforms, including hosted networks and cloud computing systems.

Virtualization experts talk about 'container virtualization’ or 'operating system virtualization’ as an alternative to what some call traditional hypervisor-based virtualization. What’s the difference? In hypervisor-based virtualization, the system emulates the hardware; a software system creates individual computers instead of networking physical computers together. One way that experts talk about this is that hypervisor-based systems require communications to work just like they would between physical machines; in other words, these virtual hardware pieces don’t share anything that a physical machine wouldn’t share with another, except for a generalized resource environment.

With container virtualization, virtualization is done at the operating system level instead of at the hardware level. Individual instances share parts of the kernel and so data gets routed differently.

Lots of IT people are looking at container virtualization because of its ability to provide denser environments than hypervisor virtualization. Some talk about it as a 'cheap’ way of doing host environments. Engineers can choose to deploy containers in various ways, for example, through open-source platforms such as Open Stack. However, some contend that container virtualization is sometimes less scalable or flexible than traditional methods.

1. **Mi a lapozás során alkalmazott „munkahalmaz” modell? (1 pont) [161124]**

előlapozás, használtak a fizikai memóriában tartva, lapnyilvántartás (Óra javítása: WSClock)

1. **Mit értünk szegmentált memóriakezelésen, mi a lényege (legfontosabb jellemzői)? (2 pont) [121122 (1? pont), 131128 (1 pont), 150518p, 161219p (1 pont), 171123 (1 pont)]**

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 eltolási (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.

1. **Szegmentált memóriahasználatnál van-e szükség virtuális memóriakezelésre? (1 pont) [161219p]**

Nincs szükség rá, mert a szegmentálás nem csak a virtuális memóriánál használható.

1. **Mi a memóriakezelő feladata dinamikus memóriahasználat során? (1 pont) [150420]**

A processz élete során felmerült újabb memóriaigények kielégítése.

1. **Milyen dinamikusmemória-nyilvántartási módszereket ismer? (2 pont) [140430, 161124]**

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.

1. **Mi a kölcsönös kizárás, mik megvalósításának a feltételei? (2 pont) [11/12/2, 101111, 110511, 131128]**

(Kritikus programterület, szekció, az a rész mikor a közös erőforrást (memóriát) használjuk.)

Amikor egy folyamat használ egy közös erőforrást, megtiltjuk, hogy egy másik folyamat is használja ugyanazt az erőforrást. Ezt nevezzük kölcsönös kizárásnak.

A jó kölcsönös kizárásban:

- nincs két folyamat egyszerre a kritikus szekcióban

- nincs CPU paraméterfüggőség

- egyetlen kritikus szekción kívüli folyamat sem blokkolhat másik folyamatot

- egy folyamat sem vár örökké, hogy a kritikus szekcióba tudjon belépni.

1. **Mi a mutex, mikor nem használható? (1 pont) [11/12/2, 121122]**

A mutex egyik jelentése az angol mutual exclusion (kölcsönös kizárás) szóból ered. Programozástechnológiában párhuzamos folyamatok használatakor előfordulhat, hogy két folyamat ugyanazt az erőforrást (resource) egyszerre akarja használni. Ekkor jellemzően felléphet versengés. Ennek kiküszöbölésére a gyorsabb folyamat egy, az erőforráshoz tartozó mutexet zárol (ún. lock-ol).

Amíg a mutex zárolva van (ezt csak a zároló folyamat tudja feloldani - kivéve speciális eseteket), addig más folyamat nem férhet hozzá a zárolt erőforráshoz. Így az biztonságosan használható. (Például nem lenne szerencsés, ha DVD-írónkat egyszerre két folyamat használná.)

\*\*\*

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

1. **Mi a legfontosabb különbség a tevékeny várakozással, illetve szemaforral biztosított kölcsönös kizárás között? (1 pont) [170426]**

A szemaforral megvalósítható a kölcsönös kizárás tevékeny várakozással is megvalósítható, de a rendszerhívások segítségével elérhető, hogy ha a szemafor nulla, akkor a folyamat beillessze magát a várakozó processzusok sorába, és blokkolja is magát.

* Előnye: nincs aktív várakozás
* Hátránya: a másik processzuson múlik, hogy felébred-e az alvó

1. **Mit értünk tevékeny várakozás alatt? (1 pont) [11/12/2, 110511, 111117, 131128, 140430, 161124]**

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

1. **Mi az oka, hogy bizonyos helyzetben üzenetekkel biztosítjuk a kölcsönös kizárást? Miért hasznos? / Miért hasznos a kölcsönös kizárás üzenetküldéses megvalósítása? (1 pont) [131128, 150420, 170426]**

A kölcsönös kizárás teljesülése osztott memória, szemaforok segítségével nem mindig megvalósítható. Kell egy másik módszer, éspedig a jelzések, az üzenetküldés lehetősége.

Az üzenetküldés módszere akkor is jó megoldást ad, amikor egy többprocesszoros rendszerben minden processzornak saját memóriája van (ugyanis ilyenkor a TSL utasítás, Peterson-féle megoldás, szemaforhasználat nem megfelelők).

\*\*\*

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.

1. **A kölcsönös kizárás Peterson-féle megoldásának mi a lényege? (algoritmus) (2 pont) [100412, 141120, 171123]**

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();

}

1. **Ismertesse a kölcsönös kizárás “szigorú váltogatás” megvalósítását! (2 pont) [101111, 140430, 161124]**

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

1. **Mi a probléma a szigorú váltogatásos módszerrel? (2 pont) [130422, 160425]**

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.

while(1) // 0. folyamat

{

while(kovetkezo!=0);

kritikus\_szekcio();

kovetkezo=1;

nem\_kritikus\_szekcio();

}

while(1) // 1. folyamat

{

while(kovetkezo!=1);

kritikus\_szekcio();

kovetkezo=0;

nem\_kritikus\_szekcio();

}

1. **Ismertesse a kölcsönös kizárás megvalósítását TSL utasítással! (2 pont) [11/12/2, 110511, 111117, 121122, 131218p, 161219p]**

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. 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.

1. **Mi a szemafor, mikor nem használható? (1 pont) [100412, 110511, 111117, 130422, 131218p, 160425]**

(Egy olyan speciális változó, ami a kölcsönös kizárás megvalósításához szükséges.)

Egyfajta „kritikus szakasz védelemre” 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.

1. **Mi a különbség a szemafor és a mutex között? (1 pont) [100412, 141120]**

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.

1. **Mit nevezünk RAID rendszernek, mi a RAID5, mi a működésének lényege? (2 pont) [101111, 110511, 130422/a, 160425/a, 170426] / Mi a RAID(1..5)?**

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, ∑ 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…)

1. **Mi a RAID5 és RAID6 közti különbség, mi a működésük lényege? (2 pont) [111117, 130422/b, 131218p, 160425/b]**

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.

1. **Mi a hardver RAID és a szoftver RAID közti különbség, mi a működésük lényege? (2 pont) [141120, 150420, 171123]**

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.

[???]

1. **Mi a RAID 0+1 és a RAID1+0 közötti különbség? (1? pont) [121122]**

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

1. **Milyen I/O eszköztípusokat (eszközkategóriákat?) ismer? Mi a kivétel? (???) / Az operációs rendszer milyen periféria eszközkategóriákat használ? / Milyen I/O kommunikációs módokat ismer? (1 pont) [11/12/2, 110511, 121122, 140430, 150420, 170426]**

blokkos eszközök:

* az információt adott méretű blokkban tárolja, mindegyiket saját címmel
* az egyes blokkok írhatók és olvashatók az összes többi blokktól függetlenül
* pl. lemez

karakteres eszközök:

* vagy kibocsájtja vagy fogadja a karaktersorozatot
* nincs blokk szerkezet
* nem címezhető, nincs keresés
* pl. nyomtató, egér, ...

Kivételek:

órák:

* nem címezhetőek blokkonként
* nem generálnak, nem fogadnak karaktersorozatot
* csak jól meghatározott időintervallumonként megszakításokat hoznak létre
* memória leképezésű képernyők

1. **Milyen I/O eszközöknél nem a strucc algoritmust használja az os.inf.elte.hu? (1 pont) [171123]**
2. **Ismertesse az I/O szoftverrendszer réteges felépítését! (I/O szoftver modellje?) [140430]**

A szoftverrendszere rétegesen épül fel (tipikusan 4 réteg).

* megszakítást kezelő réteg: legalsó kernel szinten kezelt, szemafor blokkolással védve
* eszközmeghajtó programok
* eszköz független operációs rendszer program
* felhasználói I/O eszközt használó program

1. **Mire szolgál az input/output port? Létezik-e? (1 pont) [161124]**

// idk Létezik fizikai I/O port, a perifériákhoz való csatlakozásra szolgál.

1. **Mit takar az LBA címzés? Hol használatos? (1 pont) [170426]**

LBA (Linear Block Address) - a cilinder, fej, szektor szám hivatkozás helyett minden szektornak egy egyedi számot ad 0-tól (N-1)-ig, ahol N a szektorok száma a lemezen, vagyis olyan, mintha a megye, város, utca, házszám postai címzés helyett minden háznak egyedi számot adnánk.

Az LBA-t úgy a BIOS, mint az operációs rendszer, valamint a lemez kell támogassa. A különbség az LBA és az ECHS között az, hogy az ECHS esetén a BIOS a megszakítások paramétereit az átalakított geometriából a lemez logikai geometriájává alakítja, míg LBA-val az átalakított geometriát direkt logikai blokk (szektor) számmá alakítja.

[???]

1. **Mit nevezünk kritikus tevékenységnek? (1 pont) [150420, 170426]**

Kritikus tevékenységnek nevezzük azt, amikor két vagy több folyamat keresztezi egymást a kritikus területen.

1. **Mit nevezünk kritikus programterületnek? (1 pont) [141120, 171123]**

Kritikus programterület, szekció az a rész, amikor a közös erőforrást használjuk.

1. **Mi a DDK? Miért van szükség a használatára? (2 pont) [170426]**

Definition - What does Driver Development Kit (DDK) mean?

A driver development kit (DDK) is a software product offered by a software vendor or third-party development firm. It allows hardware vendors to develop software drivers for their hardware products. A DDK is intended to make the development process easy and typically includes detailed documentation and sample projects. It may include a build environment with testing tools for driver developers. This type of tool kit is often released by an operating system (OS) vendor to help developers build a software driver for a certain hardware product, or to update an existing software application driver to suit a newly released OS.

1. **Melyik állítás lehet igaz? Miért? (2 pont) [150420, 170426]**
2. **A hardvermegszakítás szoftveresen kezelhető.**
3. **A szoftvermegszakítás hardveresen kezelhető.**

// minden hardvermegszakítást muszáj kezelni az OS-nek vagy bárminek szóval a jel az "feljön" szoftver szintre, fordítva meg nem lehetséges bár az is lehet hogy de

1. **Mi a szoftveres és hardveres megszakítás közti különbség? Van egyáltalán? (1 pont) [11/12/2, 110511, 111117, 121122, 131128, 161124]**

A szoftveres megszakítás kezelése azonos a hardveres megszakítás kezelésével. Különbség a megszakítás forrása. (Amikor a mikroprocesszornak egy eszközt, vagy folyamatot ki kell szolgálni, annak eredeti tevékenységét felfüggesztve, megszakítások lépnek életbe. Létezik szoftveres és hardveres megszakítás.)

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.

1. **Mit értünk folyamatok ütemezésén? (? pont) [100412]**

Az ütemező a folyamatok gyors váltogatásával teremt párhuzamos végrehajtás érzetet. Egy ütemezési algoritmus alapján eldönti az ütemező, hogy mikor melyik folyamat futhat.

1. **Mi a garantált ütemezés?**

Minden aktív folyamat arányos CPU időt kap. Nyilván kell tartani, hogy egy folyamat már mennyi időt kapott, ha valaki arányosan kevesebb időt kapott az kerül előbbre.

1. **Mi az arányos ütemezés, és mitől arányos? (2 pont) [101111 (1 pont), 141120, 150420, 170426, 171123]**

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

1. **Írja le az SSTF/SS(T)F ütemezés lényegét és jellemzőit! (2 pont) [11/12/2, 100412, 110511, 121122 (2? pont), 130422 (1 pont), 131128, 160425 (1 pont), 161219p]**

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.

1. **Mi a “sorsjáték-ütemezés” lényege, hol használják? (? pont) [100412]**

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

1. **Mi a Round-Robin-ütemezés lényege? (2 pont) [111117, 131218p, 150518p]**

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.

1. **Jellemezze a CFS ütemezést! Mire használható? Mi a lényege? (2 pont) [160425, 161124]**

* 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

1. **Milyen valós idejű folyamatütemezést biztosít a kiszolgálónk? Mikor használhatjuk? (2 pont) [161219p]**
2. **Milyen valós idejű rendszerjellemzőket ismer a tantárgy kiszolgálóján? (1 pont) [171123]**
3. **Mi a valós idejű ütemezés lényege? (1? pont) [121122]**

Garantálni kell adott határidőre a tevékenység, válasz megadását.

1. **Milyen kommunikáció-típust ismerünk a perifériákkal? / Milyen periféria kommunikációs módot ismer? / Az operációs rendszer milyen periféria kommunikációt használhat? (1 pont) [100412, 111117, 131218p, 150420]**

* 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

1. **Mi a CHS címzés? (? pont) [100412]**

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

* Példa: 1.44 MB FD
* Sávok száma: 80 (0-79)
* Fejek (cilinder) száma: 2 (0-1)
* Szektorok száma egy sávon: 18 (1-18)
* Összes méret: 80\*2\*18=2880 szektor \* 512 byte

1. **Mi az i-node tábla / indextábla? (1 pont) [100412, 150420]**

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.

1. **Milyen partíciónak nincs i-node táblája? (1? pont) [121122]**

Minden nem UNIX fájlrendszerű partíciónak nincs i-node táblája.

1. **Az operációs rendszerek folyamatainak milyen állapotait, állapotátmeneteit ismerjük? (? pont) [100412]**

* 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

1. **Ismertesd a bankár algoritmust! Mi a lényege? (2 pont) [11/12/2, 100412, 110511, 121122, 131128, 141120, 150518p, 171123]**

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.

1. **Mi a POSIX? (1 pont) [11/12/2, 101111, 131128, 161219p]**

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.

1. **Mik a 0. generációs operációs rendszerek jellemzői? (2 pont)**

- 1940-ig

- tisztán mechanikus

- nincs operációs rendszer

- operátor alkalmazás

1. **Mik az 1. generációs operációs rendszerek jellemzői? (2 pont)**

* 1940-1955
* sajátos kapcsolótábla, relé, vákuumcső, lyukkártyák megjelenése
* Neumann János elvei
* egyedi gépek
* gépi kód, egyszerű matematikai számítások

1. **Mik a 2. generációs operációs rendszerek jellemzői? (2 pont) [11/12/2, 101111, 130422, 131128]**

* 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)

1. **Mik a 3. generációs operációs rendszerek jellemzői? (2 pont) [140430]**

* 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

1. **Mik a 4. generációs operációs rendszerek jellemzői? (2 pont) [150518p]**

* 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

1. **Mi az MBR? (1 pont) [110511]**

A Master Boot Record (MBR) vagy más néven a partíciós szektor a merevlemez legelső szektorának elnevezése. Csak a particionált merevlemezeknek van MBR-jük. A merevlemez legelején, az első partíció előtt található meg. 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.

1. **Mi a FAT, van-e MBR-je? (1 pont) [101111]**

File Allocation Table. Fájlrendszer típus. Láncolt listás nyilvántartás. MBR (Master Boot Record) egy külön része a merevlemeznek, nincs összefüggésben a filerendszerekkel.

1. **Mutassa be a FAT32 fájlrendszer fontosabb jellemzőit! Használják-e még? (2 pont) [160425]**

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)

1. **Mi az Ext2FS, van-e MBR-je? (1 pont) [111117, 131218p, 141120, 150518p]**

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

1. **Mire jó az NTFS, van-e MBR-je? (1 pont) [171123]**

Fájlrendszer. Minden particionált merevlemeznek van MBR-je függetlenül a fájlrendszertől, így az NTFS-nek is.

1. **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

1. **Melyik fájlrendszernek van MBR-je? (1 pont) [161219p]**

Minden particionált merevlemeznek van MBR-je függetlenül a fájlrendszertől.

1. **Mi a TLB, mikor használjuk, mi a szerepe? (1 pont) [111117, 131218p, 150518p, 161219p]**

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

1. **Mi a laptáblák szerepe? Van közük a TLB-hez? (2 pont) [131128] / Mi a laptábla? Hol használjuk? (1 pont) [171123]**

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.

1. **Mi az invertált laptábla? (1 pont) [161124]**

A valós tár (fizikai memória) minden lapkeretéhez tartozik egy bejegyzés ahelyett, hogy a virtuális címtér minden lapjához legyen egy bejegyzés. Egy bejegyzés azt tartalmazza, hogy az adott lapkeretet melyik program használja. Előnye: sok helyet takarít meg. Hátrány: nehezebb a címfordítás, lassabb.

1. **Ismertesse a folytonos tárkiosztás (lemez) stratégiáit, jellemzőit! (2 pont) [101111, 111117 (1 pont), 131218p (1 pont)]**

* 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)

1. **Mi a randevú stratégia? (1 pont) [101111, 141120]**

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

1. **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.

1. **Mire használható a monitor? (1 pont) [131128]**

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

1. **Mi a különbség a monitor és a mutex között? (1 pot) [101111]**

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).

1. **Mi a mutex és az egyszerű egész közti különbség? Van egyáltalán? (1 pont) [150518p]**
2. **Mit jelent a monitor “condition” típusa? (1 pont) [101111, 141120]**

Mi van, ha egy folyamat nem tud tovább menni 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.)

1. **Mi a soft real time rendszer? (1 pont) [101111] / Mi a hard 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.

1. **Mi a real time multitask rendszer? (1 pont) [141120]**

Oximoron?

1. **Mit jelent az interleave fogalma? (1 pont) [130422]**

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

1. **Milyen processzorvédelmi szinteket ismer, hol használjuk ezeket? (1 pont) [130422, 131128, 160425]**

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

1. **Mi az operációs rendszer kernelmódja és felhasználói módja közti különbség? (1 pont) [100412, 101111]**

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

1. **Mi biztosítja az operáció rendszer kernel és felhasználói módja között a különbséget? (1 pont) [131218p, 150518p]**

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).

1. **Mi az MFT? (1 pont) [130422, 160425]**

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.

1. **Mit nevezünk valós idejű operációs rendszernek? (1 pont) [130422, 160425, 161124]**

Minden egyes rendszerhívás egy előre meghatározott időn belül garantáltan végrehajtásra kerül, a konkrét körülményektől függetlenül.

1. **Mit értünk preemptív operációs rendszer alatt? (1 pont) [140430]**

Az aktuális processztől a kernel bizonyos idő után elveszi a vezérlést, és a következő várakozó folyamatnak adja. (Ma tipikusan ilyen rendszereket használunk.)

1. **Mi az LRU algoritmus? (1 pont) [140430]**

Lapozási stratégia. LRU – Least Recently Used. Legutoljára használt algoritmus. A memóriában tárolt lapokat a FIFO-hoz hasonlóan egy listában tároljuk el. A lista végére kerül a legutoljára használt lap, a elejére pedig a legrégebben használt. Ennek megfelelően laphiba esetén az új lapot a rendszer a lista végére szúrja be.

Ez az algoritmus azon a megfigyelésen alapszik, hogy a legutóbb használt lapokra valószínűleg még szükség lesz. Az algoritmus hátránya az, hogy a listát minden egyes memóriahivatkozás után frissíteni kell.

1. **Mi a fájlrendszer és a könyvtárszerkezet közti különbség? (1 pont) [170426] / Mit nevezünk fájlnak és könyvtárnak?**

A fájl adatok egy logikai csoportja, névvel és egyéb paraméterekkel ellátva.

A könyvtár fájlok (könyvtárak) logikai csoportosítása.

1. **Mit nevezünk fájlrendszernek, mi köze van az FCFS ütemezéshez? (2 pont) [111117, 131218p]**

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.

1. **Milyen fájlrendszer-specifikus fájlokat ismer? Hol találhatóak általában? (2 pont) [130422]**

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

1. **Mondj példát láncolt szerkezetű fájlrendszerre! (1 pont) [141120]**

FAT12, FAT16, FAT32

1. **Milyen fájlrendszer-elhelyezési stratégiákat ismer? Melyik veszteséges? (2 pont) [150420]**

- Folytonos tárkiosztás (azaz folyamatos elhelyezésű)

- 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.

- Mindegyik veszteséges.

- Láncolt elhelyezkedés (azaz láncolt tárolás)

Nincs veszteség (csak a blokkméret).

- A fájl adatai egy láncolt blokk listában vannak. (Így az utolsó blokk elérése lassú.)

- Szabad-foglalt blokkok: File Allocation Table, FAT – Nagy méretű lehet, és a FAT mindig a memóriában van.

- Indextáblás elhelyezés

- Katalógus tartalmazza a fájlhoz tartozó kis tábla (un. inode) címét. E tábla segítségével érhetőek el azok a blokkok, melyek a fájlt tartalmazzák (ugyanis egy fájl gyakran több blokkban van).

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

1. **Milyen fájlrendszernél nem találunk ma lemez foglaltsági térképet? (1 pont) [171123]**

// van: FAT, NTFS, ext2

1. **Mik a program(kód)-folyamat-szál közti különbségek? (2 pont) [130422, 140430 (1 pont), 160425] / Mi a különbség a folyamat és a szál között? Van egyáltalán? (1 pont) [131128]**

Folyamat: önálló programegység, utasításszámlálóval, veremmel stb. Általában nem független folyamatok. Három állapotban lehet: futó, futásra kész, vagy blokkolt.

Szál: egy folyamaton belül több egymástól „független” végrehajtási sor. A folyamatnak önálló címtartománya van, szálnak viszont nincs.

\*\*\*

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.

Csak szálnak van: utasításszámláló, regiszterek, verem

1. **Mire jó a dirty bit, hol használják? (1 pont) [131218p, 150518p]**

Módosítás nyilvántartása (modified/dirty bit): minden memórialaphoz tartozik egy HW által kezelt bit (pl. a laptáblában) - betöltéskor törlik, módosításkor beállítják.

* Ha a dirty bit 1, akkor erre a lapra történt írási művelet, mióta a ﬁzikai memóriába került. Ezt a bitet a CPU tartja karban (automatikusan), ezzel az operációs rendszer munkáját

segítve.

* 0 érték esetén a lapokat nem kell a diszkre kiírni, ha kiszorulnak a ﬁzikai memóriából (hiszen a tartalmuk a ﬁzikai memóriába helyezés óta változatlan, a diszken tárolt változat tehát továbbra is aktuális).

1. **Mi a CHS és az LBA címzés közti különbség? (2 pont) [131218p (1 pont), 150420]**

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.

1. **Hány folyamat lehet aktuálisan futási (running) állapotban? (1 pont) [160425]**

Valódi párhuzamosságról akkor beszélhetünk, amikor valóban több folyamat fut egy időben. Ez akkor lehetséges, ha a számítógépünkben több processzor van. Közös órajelre működhetnek, akár közös memóriát is használhatnak. Ezek a megoldások gyors kommunikációt tesznek lehetővé a folyamatok között. Nagy teljesítményű úgynevezett „tudományos” számítógépek akár több száz processzort is tartalmazhatnak. Másik elterjedt megoldás, amikor önálló számítógépeket hálózati kapcsolat segítségével üzemeltetnek úgy, mint egy többprocesszoros gépet. Ebben az esetben a számítógépek egymással hálózaton keresztül kommunikálnak, aminek következménye a sokkal lassabb kommunikáció. Viszont egy egyprocesszoros (néhány processzoros) gép esetén is megteremthetjük a párhuzamosság illúzióját, ha a folyamatokat gyorsan váltogatjuk. Ekkor a folyamatok egy kis ideig futhatnak (egy kis időszeletet kapnak), majd az operációs rendszer keres egy másik folyamatot, és azt futtatja egy kis ideig. Az interaktív felhasználók ezt úgy látják, mintha több folyamat futna párhuzamosan. Ezt a technikát látszatpárhuzamosságnak nevezik.

1. **Ismertesse a gyártó-fogyasztó probléma üzenetküldéses megvalósítását! (1 pont) [161219p] / Miért érdekes a gyártó-fogyasztó probléma üzenetküldéses megvalósítása? (1 pont) [161219p]**

Vegyünk két folyamatot, amelyek egy közös tárhelyen dolgoznak: az egyikük adatot helyez el ebben (gyártó), a másik adatokat olvas, és ki is veszi azokat a közös tárhelyből (fogyasztó).

A kölcsönös kizárás biztosítása osztott memória és szemaforoksegítségével nem mindig megvalósítható. Ebben segít egy másik módszer: a jelzések és az üzenet küldésének a lehetősége.

Két rendszerhívást definiálunk a szolgáltatás biztosítására: a „send”, illetve a „receive” eljárást. (Ezek elvi definíciók, a gyakorlatban, egyes környezetekben más-más névre is hallgatnak, mint például a „kill”, a „msg\_snd” vagy a „msg\_rcv”. Később majd mutatunk a használatukra példákat.) Mindkét eljárásnak két paramétert definiálunk: az első a címzett vagy a feladó, míg a második paraméter maga az üzenet.

Lássuk a probléma egy megoldását pék-vásárló esetére. A kész kenyerek tárolására akkora polc van a pékségben, amin száz kenyeret tárolhatunk (N = 100). Ha megtelt az összes hely, a pék várakozik, amíg a vásárlók elvisznek a teli polcról néhány kenyeret, ezután újra süt. Ha sok vásárló jött a pékségbe, akkor esetleg elfogyott a kenyér; ilyenkor a vásárlók szép sorban várakoznak, amíg az újabb sütés befejeződik. Elindul mindkét folyamat, a „pék” is, és a „vásárló” is. Lássuk először a vásárlót! Még üres a bolt, ezért elküldjük az N darab üres üzenetet, majd várjuk a kenyeret. Ha nincs kenyeres üzenet, a vásárlófolyamat blokkolódik. Amint megjön a kenyeres üzenet, a folyamat folytatja a működését. Amint megjön a kenyeres üzenet, azt kicsomagoljuk, majd visszaküldjük az üres üzenetet. A „pék” folyamat rögtön süt egy kenyeret, majd várja, hogy azt egy üzenetben kérjék tőle. Ha nincs üres „kenyér-szállító” üzenet, a pékfolyamat blokkol. Amint megkapja az üzenetet, abban visszaküldi az elkészült „kenyeret”.

Összegzésképpen elmondhatjuk, hogy a pék-vásárló probléma megoldása az üzenetküldés módszerével akkor is jó megoldást ad, amikor egy többprocesszoros rendszerben minden processzornak saját memóriája van. Láttuk korábban, hogy az ilyen rendszerekben a közös memóriát használó megoldások (TSL utasítás, Peterson-féle megoldás, szemaforhasználat) nem megfelelők.

1. **Mi a konténer technológia? Létezik-e ilyen? (1 pont) [161219p]**

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.)

1. **Mit jelent a diszk kvóta szolgáltatás? (1 pont) [161219p]**
2. **Hol használatosak a beágyazott rendszerek, mik azok? (1 pont) [161124]**

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.

1. **Mit takar a folyamatleíró táblázat? (1 pont) [161124]**

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.

1. **Ismertesse egy partíció felépítését (általánosan)? (1 pont) [150518p]**
2. **Írja le az operációs rendszer boot folyamatát? (2 pont) [150518p]**

ROM-BIOS megvizsgálja, lehet-e operációs rendszert betölteni, ha igen, betölti a lemez MBR programját a 7c00h címre (konvenció szerint ez mindig erre a címre történik). Ez után az MBR programja vizsgálja meg mi az elsődleges partíció, majd azt betölti a memóriába.

1. **Mi az eszközmeghajtó program? (1 pont) [150518p]**

Az a program, amely a közvetlen kommunikációt végzi az eszközzel. Ez a kernelnek (operációs rendszer magja) a része. Lemezek írása, olvasása során DMA-t használnak. Réteges felépítés.

1. **Mi az oka a 2TB HDD címzési határnak? Van ilyen egyáltalán? (1 pont) [150518p]**
2. **Mit értünk folyamatok erőforrás görbéjén, mire használható? (1 pont) [11/12/2, 110511, 121122, 140430]**

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.

1. **Mit értünk monopol módú eszköz alatt? (1? pont) [121122]**

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

1. **Mit jelent a GDT? Hol használjuk? (1 pont) [140430]**
2. **Használhat-e virtualizált környezetben futó operációs rendszer virtuális memóriát? (1 pont) [140430]**
3. **Mi (ki) kapcsolja védett módba a CPU-t a Windows 10 betöltése során? [171123] (1 pont)**

**<KÓDRÉSZLETESEK LEJJEBB>**

1. **Mit takar az alábbi kódrészlet, mi jellemzi? (2 pont) [131218p, 150518p]**

**zöld-fehér:**

**TSL regiszter, széf** // megnézi hogy a széf lock szabad-e. ha igen a regiszterbe 0-t rak, amúgy 1-et

**cmp regiszter, 0** // összehasonlítja a regiszter 0-val (kb egy if)

**jne zöld-fehér** // ha nem egyenlő 0-val, akkor visszaugrik a programkód elejére

**ret** // szal ez asszem az aktív várakozás, mert folyamatosan megy ez a ciklus amíg fel nem szabadul a széf lock

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

1. **Mit takar az alábbi algoritmus? (2 pont) [101111, 141120, 150420]**

**void valami(int i)** // pihen(i) [141120] kell2ágy(int i) [150420]

**{**

**while(1)**

**{**

**ülök();**

**kell\_ágy(i); //bal ágy** kell1ágy(i); [150420]

**kell\_ágy((i + 1) % N); //jobb ágy** kell1ágy((i+1)%N); [150420]

**fekszem\_az\_ágyon();**

**nemkell\_ágy(i);**

**nemkell\_ágy((i + 1) % N);**

**}**

**}**

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

1. **Mit takar az alábbi kódrészlet, mi jellemzi? (2 pont) [11/12/2, 110511, 111117, 140430, 161124, 171123]**

**condition tele, üres;**

**int darab;**

**termel(autó elem)** // kenyér elem [11/12/2, 110511, 171123] | sör korsó [140430] | főz (hal ponty) [111117]

**{**

**if (darab == N) wait(tele);**

**polcra(elem);** // süt(ponty) [111117]

**darab++;**

**if(darab == 1) signal(üres);**

**}**

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

1. **Mit takar az alábbi kódrészlet? Helyes-e, ha nem, miért nem? (2 pont) [130422, 160425]**

**void belepes(int meccs)**

**{**

**int masik;**

**masik = 1 - meccs; //mivel N = 2…**

**akarja[meccs] = 1;**

**kovetkezo = meccs;**

**while (kovetkezo == meccs && akarja[masik]);**

**}**

Peterson-féle kölcsönös kizárás belepes() megvalósítása.

1. **Mit takar az alábbi kódrészlet, mi jellemzi? (2 pont) [131128, 161219p]**

**void belepes(int proc)**

**{**

**int masik;**

**masik = 1 - proc; // mivel N = 2**

**kovetkezo = proc; // két sor csere**

**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.

1. **Mit takar az alábbi algoritmusrészlet, mi a jellemzője? (2 pont) [170426]**

**void munkás() //munkás folyamata**

**{**

**int termék; //termék tárolási helye**

**message m; //üzenet tárolási 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.

1. **Mit takar az alábbi algoritmusrészlet, mi a baj vele, ha van? (2 pont) [121122]**

**while(1)**

**{**

**while(kovetkezo!=1);**

**kritikus\_szekcio();**

**kovetkezo = 0;**

**nem\_kritikus\_szekcio();**

**}**

Az algoritmus a kölcsönös kizárás „szigorú váltogatásos” 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.