Operációs rendszerek II. kidolgozott tételsor

Verzió 1.0 (Build: 1.0.2015.12.30.)

Készült:

- Dr. Fazekas Gábor Operációs rendszerek 2. diasorok és előadásjegyzetek
- Ellenőrző kérdések 2015. december 30-i állapota alapján
- OPERATING SYSTEM CONCEPTS, 8TH EDITION (Abraham Silberschatz & Peter B. Galvin)
- Boruzs Tibor féle tételsor

Készítette: Fodor Attila, Magyari László és Málik Tamás

Figyelem! Bukásért felelősséget nem vállalunk!

2015/16-es tanév, 1. félév

1. Mit jelent a preemptív processzus ütemezés?

Beavatkozó ütemezés, az operációs rendszer ha úgy dönt, elveszi a processztől a CPU-t.

2. Értelmezze a processzusütemezéshez kapcsolódó alábbi fogalmakat: végrehajtási idő, várakozási idő, válaszidő!

Végrehajtási (turnarund, fordulási) idő: várakozás a memóriába kerülésre + készenléti sorban töltött idő + CPU-n töltött idő + I/O-val töltött idő.

Várakozási idô (waiting time): készenléti sorban (ready queue) töltött idő

Válaszidô (response time): a „kérés benyújtásától" az elsô válasz megjelenéséig (nem output!)
Eltelt idő. (Interaktív rendszerekben a turnaround nem jó jellemző.)

3. Melyek az FCFS processzus ütemezés előnyei és hátrányai?

Igény bejelentési sorrend szerinti kiszolgálás (first-come, first-served =FCFS), mindenki sorra kerül.
Hátrány: az átlagos várakozási idő nagy szórása, konvoj effektus (Pl.: 4-es úton a szénásszekér)

4. Van-e optimális processzus ütemezési stratégia?

Van, a legrövidebb először. Azt a processzt engedjük a CPU-hoz, melynek a következő helyfoglalási ideje a legrövidebb. Melyik fogja a legrövidebb ideig lefoglalni a CPU-t azt előre nem tudjuk.

5. Mit jelent az exponenciális átlagolás? Mondjon példát az alkalmazására!

Megpróbáljuk megjósolni a következő helyfoglalási időt. Az eddigi viselkedések alapján kiszámoljuk a következő futás várható idejét. A mai rendszerek ezt használják. Definíciója:

Jelölje t_n az n-edik CPU foglaltsági ciklus hosszát, τ_{n+1} a következônek jósolt foglaltsági periódus hosszát.

Definiáljuk a következô foglaltság várható hosszát: Legyen $1 \geq \alpha \geq 0$ és

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$$

Alkalmazási példa: A rövidebb igény először (shortest job first, shortest job next, SJF, SJN) kiszolgálás elve.

6. Mit jelent a prioritásos processzus ütemezési stratégia?

Számokat rendelünk a processzekhez, minél kisebb ez a szám és a foglalási ciklus, annál nagyobb a prioritás. Az SJF, ill. FCFS is prioritásos kiszolgálásnak tekinthető! Belső és külső prioritás. Problémák: végtelen indefinit blokkolódás = éhezés, éhhalál. Példa: MIT 1973 IBM 7094: 1967 óta várt egy processzus a CPU-ra!! Megoldás: aging (unix példa)

7. Mit jelent a körleosztásos (Round Robin) processzus ütemezési stratégia?

Minden processzus sorban q ideig (q=10-100 millisec.) használhatja a CPU-t. n processzus esetén a maximális várakozási idő (n-1)/q. Ha q kicsi: ->FCFS, ha q nagy:-> kontextus váltási költség relatíve megnő! Egy bizonyos idő múlva az ütemező elveszi a feladattól a processzort /preemptív/ és az addig futó folyamatot a sor végére küldi és a következő folyamatot indítja egy bizonyos időre. Hátránya hogy sok az adminisztrációja.

8. Mit jelent a többsoros processzus ütemezés?

A célja a szálak egyenletes elosztása,a prioritás korrekt figyelembevétele. Többprocesszoros ütemezéskor használt paraméterek: Processzor megfelelés, ideális processzor, következő processzor

9. Mit jelent a processzus ütemezés esetén a visszacsatolásos sor?

Round Robinoknál előfordulhat, hogy pár folyamat abba a sorban ahol van „nem érzi jól magát” és akkor másik sorba kerül.

10. Mit jelent az „éhezés” (starvation)?

Prioritásos ütemezésnél a kiszolgálóhoz nem jut hozzá a sor.

11. Mi a kritikus szakasz probléma?

n folyamat verseng egy bizonyos (közös) adat használatáért · Mindegyik folyamatnak van egy kódszegmense, ahol ezt a bizonyos adatot eléri (olvassa/írja): ez az ún. kritikus szakasz. · Probléma: hogyan biztosítható, hogy amíg egy processzus a kritikus szakaszát hajtja végre, addig egyetlen más processzus se léphessen be a saját kritikus szakaszába.

12. Milyen követelményeket kell kielégítenie a kritikus szakasz kezelésének?

Követelmények:

- Kölcsönös kizárás
  - nem lehet egynél több processzus a kritikus szakaszában
- Progresszió
  - a kiválasztás (futásra) nem késleltethetô határozatlan ideig
- Korlátozott várakozás
  - minden igény kiszolgálása megkezdôdik korlátozott számú kritikus szakasz végrehajtása után

13. Mit jelent a processzusok szinkronizációja kapcsán a kölcsönös kizárás elve?

Egyszerre csak egy processzus használhat egy er őforrást.

14. Mi az „entry” és „exit” kódszekciók funkciója a kritikus szakasz kezelésében?

Minden egyes fizikai laphoz (frame-hez) tartalmaz egy bejegyzést (entry). Egy bejegyzés az adott frame-ben tárolt logikai lap virtuális címét és annak a processzusnak az azonosítóját tartalmazza, amelyikhez a frame tartozik.

A folyamat végrehajtja az utolsó utasítását, majd megkéri az operációs rendszert, hogy törölje (exit). Ennek során output adatok kerülnek át a gyermektôl a szülôhöz (cf. fork), a folyamat által használt erôforrásokat az operációs rendszer felszabadítja.

15.A szemaforok mint szinkronizációs primitívek (eszközök).

magas szintû szinkronizációs eszközök (szikronizációs primitív), amelyek lehetôvé teszik egy absztrakt adattípus biztonságos megosztását konkurens processzusok között.

16.A monitor mint szinkronizációs primitív (eszköz).

Magas szintű szinkronizációs eszközök (szikronizációs primitív), amelyek lehetővé teszik egy absztrakt adattípus biztonságos megosztását konkurens processzusok között.

formálisan: a monitor eljárások, változók és adatszerkezetek együttese, amelyek egy speciális csomagba vannak integrálva. A processzusok hívhatják a monitorban levő eljárásokat, de nem érhetik el közvetlenül annak belső adatszerkezetét.

Speciális típus: condition

Speciális műveletek: x.wait, x.signal.

A monitor azt kívánja biztosítani, hogy egy időben csak egy processzus legyen aktív (rajta).

17. A kritikus régió, mint szinkronizációs primitív (eszköz).

- Magas szintű szinkronizációs eszköz.
- Egy megosztott változó v, amelynek típusa T, deklarációja

var v: shared T;

- A v változó csak az alábbi alakú utasításokon keresztül érhetô el:

region v when B do S;

ahol B egy logikai kifejezés. ·

- Amíg az S utasítás végrehajtás alatt áll, másik processzus nem érheti el a v változót.
-
- Ha a processzus megpróbálja végrehajtani a region utasítást, a B logikai kifejezés kiértékelődik. Ha B igaz, az S utasítás végrehajtódik. Ha hamis, akkor a processzus végrehajtása késleltetődik addig, amíg a kifejezés igaz nem lesz és egyetlen más processzus sincsen a v-hez kapcsolt régióban.
- Példa: korlátos puffer problémája
- Implementáció: pl. szemaforokkal

18. Ismertesse az írók és olvasók problémáját!

Egy adatot, állományt több processzus megosztva, párhuzamosan használ, egyesek csak olvassák, mások csak írják.

Hogyan biztosítható az adatok konzisztenciája? ·

Stratégiák:

1. Minden olvasó azonnal hozzáférhet az adatokhoz, hacsak egy író nem kapott már engedélyt az írásra.
2. Egy író azonnal írhat, ha erre kész és más írók éppen nem írnak.

Mindkét stratégia éhezéshez (starvation) vezethet, de van megoldás

19. Ismertesse az „vacsorázó filozófusok” problémáját. Miért/hogyan vezethet holtponthoz?

- Egy kör alakú asztal mellett öt filozófus ül, mindegyik előtt van egy tányér rizs és a szomszédos tányérok között egy-egy evő-pálcika.
- Evéshez a filozófus a saját tányérja melletti két evôeszközt használ-hatja úgy, hogy ezeket egymás után kézbe veszi.
- Ha befejezte az étkezést, vissza- teszi az eszközöket, és gondolkodni kezd.
- Majd újra megéhezik, stb.

20. Ismertesse a Bakery –algoritmust!

Minden processzus kap egy sorszámot, mielôtt belép a kritikus szekciójába (ticket). A legkisebb sorszámú processzus hajthatja végre a kritikus szekcióját.

Ha Pi és Pj sorszáma megegyezik, akkor a kisebb indexű jogosult a kritikus szekciójába belépni. · A sorszámozó rendszer mindig monoton növekvő sorszámokat generál. · Pl.: 1,2,2,3,3,3,4,4, ...

Jelölés: “<” jelentse a (ticket #, process ID) párra definiált lexikografikus rendezést.

21. Folytonos tárallokálás, fix particionálás: egyenlő- és változó hosszúságú partíciók.

Az operatív memóriát egyetlen folytonos tömbnek tekinthetjük. A memória "rendszer"- és "felhasználói program" részekre bomlik. A rendszer résznek tartalmaznia kell a memóriába beágyazott megszakítási vektort, az I/O kapukat (portok). Minden processzus egyetlen összefüggő (folytonos) részt foglal el a memóriából. Folytonos tárallokálásnál egy program számára kioszthatő tárterület folytonos (összefüggő).

Fix partícionálás: A felhasználói tárterületet olyan különböző méretű partíciókra osztották, melyek mérete a rendszer működése során nem változhatott. Minden partícióban egyetlen folyamat tartózkodhat. A multi programozás fokát a partíciók száma korlátozza.

Egyenlő méretű partíciók kialakítása

Bármelyik olyan processzus melynek mérete kisebb vagy egyenlő a partíció méretével, betölthető egy elérhető partícióba. Ha az összes partíció tele van, az operációs rendszer kicserélheti egy partícióban levő másik processzussal
Problémák:

egy program nagyobb is lehet, mint a partíció, ekkor a programozónak az overlay technikát kell alkalmaznia

• a főmemória kihasználása nem jó: minden program, méretétől függetlenül egy egész partíciót elfoglal (belső töredezettség - internal fragmentation)Megoldás: nem egyenlő méretű partíciók

Elhelyezési algoritmus:

• Azonos méretű partíciók: azonos méret miatt bárhova mehet

• Nem azonos méretű partíciók: minden processzushoz a lehető legkisebb alkalmas partíciót választani (belső töredezettség minimalizálása)

22. Folytonos tárallokálás, dinamikus particionálás: allokációs elvek (first fit, stb.). Külső- és belső fragmentáció.

Dinamikus partícionálás: Változó méretű partíciók. Nincs belső tördelődés, viszont van külső tördelődés: memória felszabadítása után lyuk marad a partíció helyén

Allokációs elvek:

- first fit: A lista elejétől indulva az első megfelelő méretű szabad helyet keresi. Ez a leggyorsabb algoritmus, mert kevés idő megy el a szabad kely keresésével. A szabad hely ekkor két részre vágódik – kivéve, ha pont akkora, mint a program – az egyik részébe kerül a program, a másik megmarad szabadnak.
- next fit: A first fit olyan változata, mely megjegyzi az előző lyuk helyét, és innen folytatja a keresést. Nem túl hatékony.
- best fit: A listában minimumkeresést végez a legalább akkora méretű lyukak közt, melybe a program belefér. Így az egész listán végig kell mennie. Hajlamos arra, hogy kicsi, használhatatlan lyukakat csináljon.
- worst fit: A lista legnagyobb lyukát választja ki. Szintén nem túl hatékony.

Külső fragmentáció: Azt jelenti, hogy az összességében rendelkezésre álló hely elegendő, de ez apró részekben jelenik meg, ahová viszont nem fér be egyetlen folyamat sem.

Belső fragmentáció Az az eset, amikor tárkezelési eljárások egy csoportjánál a rendszer nagyobb tárterületet foglal le, mint amennyi szükséges, ezért a lefoglalt terület egy része kihasználatlan marad.

23. Folytonos tárallokálás, társblokkok módszere (buddy rendszer) előnyei.

A buddy rendszert a dinamikus partícionálás problémájára fejlesztették ki, melyben korlátozták a kioszthatómemóriaméretet 2 hatványaira. Ezáltal a szemétgyűjtögetés (garbage collection) könnyebbé vált. Kezdetben a teljes memória szabad, foglaláskor pedig a rendszer egy fát épít fel felezve a memóriablokkok méretét. Ha két egy szinten lévő blokk felszabadul, azt összevonva magasabb szintre emeljük. Viszonylag könnyen implementálható. Belső és külső elaprózódás is megjelenhet

24. Lapozás, lap és keret, laptábla. Logikai és fizikai cím, címfordítás. Invertált laptábla.

Lapozás (Paging):A külső fragmentáció problémájának egy megoldását kapjuk, ha a memóriát és a processzusokat kis, egyenlő méretű egységekre osztjuk.

- minden processzusnak saját laptáblája van
- minden laptáblabejegyzés tartalmazza a főmemóriában található megfelelő lap keretszámát

Lap: processzusdarab // A virtuális memória a lapozás során fel lesz osztva egyenlő részekre, ezek a lapok. (tehát a virtuális memóriában van)

Keret: memóriadarab // A fizikai memóriát felosztjuk a lapok méretével megegyező méretű részekre, ezek a lapkeretek (a fizikai memóriában vannak)

Laptáblák:

Az op. rendszer minden processzushoz egy ún. laptáblát tart fent

- tartalmazza a processzus lapjaihoz tarozó keretek helyzetét
- az egész laptábla túl sok főmemóriát foglalhat le
- laptáblák szintén tárolhatók a virtuális memóriában, így amikor egy processzus fut, laptáblájának csak egy része van a főmemóriában

Logikai cím: lap sorszáma + lapon belüli relatív cím

Fizikai cím: keret memóriabeli kezdőcíme + kereten belüli kezdőcím

Invertált laptábla: Egy bejegyzés itt tartalmazza, hogy mely processz mely lapja van pillanatnyilag a lapkertben

Címfordítás: laptáblabejegyzésekhez nagysebességű cache memória a TLB (legutóbb használt laptábla bejegyzéseket tartalmazza)

- adott virtuális cím, processzor megvizsgálja a TLB-t
- ha laptábla bejegyzést talál, kinyeri a keretszámot, megalkotja a valós címet
- ha nem talál bejegyzést, lapszámot használja a processzus laptáblájának indexelésére
- lap a főmemóriában van-e (ha nincs laphiba)
- TLB frissítése egy új lapbejegyzéssel

Logikai cím fizikai címmé konvertálása

25. Szegmentáció, szegmentáció lapozással.

A szegmentálás egy memóriakezelési módszer, mely során a memóriát több címtérre, azaz szegmensekre bontjuk. Minden résznek megvan a saját, 0-tól kezdődő címtartománya. Egy memóriacím így két részből áll, egy szegmenscímből és egy eltolási- (offset) címből. Hardver és operációs rendszer szinten valósul meg. A szegmensekhez elérési jogok tartoznak: írható vagy olvasható, vagy futtatható. Így például a programunk nem írhatja felül saját magát, mert a programkódot egy olyan szegmensben tároljuk, amely csak futtatható

A lapozás és a szegmentáció:

lapozás láthatatlan, szegmentáció látható a programozó számára lapozás csökkenti a külső töredezettséget

szegmentációval lehetséges az adatszerk-ek megosztása, védelme (láthatóság, írható, olvasható), minden szegmens több azonos méretű lapra van tördelve

A szegmentáció lapozással megoldást úgy fejlesztették ki, hogy kombinálták a szegmentációt a lapozással. A DOS valós címzést használt lapozás nélkül. Az IBM OS/2 rendszere volt az első, mely tartalmazta a lapozást.

26. A virtuális memóriakezelés koncepciója, szolgáltatásai, jelentősége.

A végrahajtandó kódnak a memóriában kell lennie, de nagyon riktán van az egész program a memóriában. A teljes programra nincs szükségünk egyszerre, csupán egy részére, így az töltődik be a memóriába, a többi a virtuális memóriába kerül, a háttértárra. Koncepciója: Szabadítsuk meg a programozót a fizikai memória méretének a korlátaitól A virtuális memória koncepciója a felhasználó/programozó memória szemléletének teljes szeparálását jelenti a fizikai memóriától.

Jelentősége: az operációs rendszer végzi, tehát nem megy el a programozó ideje a memóriakezeléssel. Nem korlát többé a fizikai memória mérete. A programozó nem vesződik az overlay megtervezésével.·Egyszerre több program aktív kódrésze lehet benn a memóriában, ami jobb CPU kiszolgálást eredményezhet. (Közben nem növekszik a válaszadási és végrehajtási idő!) Kevesebb I/O tevékenység szükséges a Swapping-hez. (A futási sebesség nő!)

27. Lapozás és virtuális memóriakezelés, laphibák, a virtuális memóriakezelés teljesítőképessége.

Page Fault (laphiba)

- Az elsô hivatkozás a lapra biztosan laphibát okoz.
- Az operációs rendszer eldönti, hogy a hivatkozás maga is hibás-e (abortálás), vagy a lap nincs a memóriába betöltve.
- Üres keret (frame) keresés.
- Lap betöltése a keretbe.
- A hivatkozott (gépi) utasítás végrehajtásának folytatása (restart).

Úgy működik, hogy az operációs rendszer minden egyes folyamatnak ad a központi memóriából egy akkora részt, amelyben a folyamat még működik, és a folyamatnak csak azt a részét tartja a központi memóriában, amelyre éppen szükség van. A folyamatnak azt a részét, amelyre nincs szükség (mert például már rég nem adódott rá a vezérlés, és feltételezhetjük, hogy rövid időn belül nem is fog végrehajtódni) ki kell rakni a háttértárra (a diszken az ún. **lapozási területre).** A virtuális memóriakezelés teljesítőképessége: akkor jó a kezelés, ha minél kevesebb a laphiba.

28. Laphelyettesítési (cserélési) algoritmusok: FCFS/FIFO, Bélády anomália, optimális algoritmus.

FIFO (First In First Out) //(FCFS First Come First Served)algoritmus:

- Előny: csekély hardver támogatás szükséges!
- a processzushoz tartozó kereteket körkörös pufferként kezeljük
- a lapokat körkörösen (round-robin) dobjuk ki
- ezt a stratégiát a legegyszerűbb megvalósítani (ott, ahol nincs koncepció)
- a memóriában legrégebb óta tartózkodó lap kerül kidobásra
- Probléma: előfordulhat, hogy ezekre a lapokra nagyon hamar szükség lesz újból

Bélády anomália: a keretek számának növeléséből nem következik az, hogy csökken a laphibák száma. Tételszerűen nem lehet bebizonyítani.

Optimális algoritmus: azt a lapot kell beáldozni, melyet egy ideig nélkülözni tudunk. A referenciasztring végignézése után az aktuális pozíciótól a legmesszebbit lehet kidobni. A referenciasztringet akkor ismerjük, ha lefuttatjuk a programot.

29. Az optimális algoritmust approximáló algoritmusok: LRU, NRU,NFU, óra/második esély algoritmus.

LRU: azt a lapot kell kidobnunk, melyet a legrégebben használtunk. Nehéz implementálni, mert nem tudni, hogy a lapra mikor történt utoljára hivatkozás. Egy vektor segítségével keressük meg a legrégebben használt lapot. Ez viszont ellentmond, tehát olyan algoritmust kell találni, mely racionális. Időigényes, és lehet hogy 1-2 szituációban nem éri meg ezt az algoritmust használni.

Legkevésbé használt (LFU: Least Frequently Used vagy NFU: Not Frequently Used) algoritmus: A bonyolult megvalósítás miatt az LRU-algoritmus helyett sokszor annak – hardvertámogatást nem, vagy alig igénylő – közelítését szokták használni (LFU). Ennél az algoritmusnál abból indulhatunk ki, hogy a közelmúltban gyakran használt lapokat a folyamatok a közeljövőben is használni fogják még, és ugyanígy, a közelmúltban ritkán, vagy nem használt lapokra a közeljövőben nem lesz szükség. Ilyenkor az operációs rendszer rendszeres időközönként végignézi a memóriában levő lapokat, és a hozzájuk rendelt számlálóhoz hozzáadja az R bit (0 vagy 1) értékét, és egyben törli az R biteket. Az algoritmus a legkisebb számláló értékkel rendelkező – vagyis a legritkábban használt – lapot választja ki kivitelre. Hátrányt jelent, hogy az algoritmus „nem felejt", vagyis az egykor gyakran használt lapok még sokáig a memóriában maradnak akkor is, ha már biztosan nem lesz rájuk hivatkozás. A problémán öregítéssel segíthetünk, például úgy, hogy az R bitet a legnagyobb helyi értékű bit helyére másoljuk, de előtte a számlálót jobbra léptetve csökkentjük a régebbi hivatkozások súlyát. A módszer másik problémája, hogy a frissen betöltött (és így biztosan kis számláló értékű) lapokat is könnyen kiteheti újra a háttértárra. Ezért általában a frissen behozott lapokat az első használatig befagyasztjuk (page locking) a tárba.

(NRU: Not Recently Used) algoritmus: az R (hivatkozott) és M (módosított) bitek használatán alapszik. A hivatkozottság egy idő elteltével elveszti a jelentőségét, ezért az operációs rendszer rendszeres időközönként törli az R bitet. Ugyanakkor az M bit értékét őrizni kell, hiszen törlése információ veszteséghez vezetne. A két bit értéke alapján az algoritmus a lapokat négy csoportba osztja, és lapkivitelnél hátranézve és a használat idejét és módját is figyelembe véve, a lehető legkisebb prioritású csoportból választ véletlenszerűen.

Óra/Második esély: Egyetlen bittel, referenciabittel mérem, hogy volt-e hivatkozás a lapra, vagy nem. Ha 0 a bit értéke, akkor nem volt, és az a lap lesz az áldozat. A lapok referenciabitjeit átnézzük ciklusban. Ha van nullás bit, akkor az lesz az áldozat. Ha volt hivatkozás a lapra, akkor a referenciabitet lenullázom, ezzel adok neki még egy esélyt. Kiküszöböli a FIFO hibáját, nem törli a gyakran használt lapokat.

30. Vergődés (trashing), a lokalitás elv, munkakészlet (working set) algoritmus lényege.

Vergődés (Trashing): lapcserélgetés folyik. Ha túl sok a laphiba, sok az I/O művelet, ennek következtében a program futása észre vehetően jóval lassabb lesz. Szembetűnő a vergődés megjelenése a gépen, az a jelenség mikor a merevlemez nagyon dolgozik, a gép meg olyan szinten be van lassulva, hogy már az egérkurzor is szaggat.

Lokalitás elv: vergődés elleni modell. Több lapot, keretet egyszerre tartunk a memóriában, melyekre lokálisan szükség van. Akár egy utasítás is megkövetelheti, hogy egyszerre több lap is benne legyen az operatív memóriában.

Munkakészlet (working set): Prediktív dolog. Minden folyamatra kiterjed, összeadva a blokkok méretét, ha a méret meghaladja a fizikai memóriáét, akkor vergődés alakulhat ki. A munkakészlet meghatározásához használható jóslási technika, amely hasonló, mint az exponenciális átlagolás módszere, tehát jóslatot lehet mondani arra, hogy a folyamatnak hány lapra lesz szüksége.

31. Az operációs rendszer állománykezelésének általános jellemzői: az állomány (fájl), mint absztrakt periféria.

Egy állomány absztrakt perifériát jelent. A fájl összekapcsolása a standard perifériával az operációs rendszer dolga. Az operációs rendszer számára egy fájl nem más, mint bájtok sorozata.

A számítógépek az adatokat különböző fizikai háttértárakon tárolhatják. Egy egységes logikai szemléletet vezettek be az adattárolásra és adattárakra: az operációs rendszer elvonatkoztatva a tároló eszköz és a tárolt adat fizikai tulajdonságaitól, egy logikai tároló egységet (adatállomány/fájl/file) használ. A fájlokat az operációs rendszer képezi le a konkrét fizikai tároló berendezésre.

Felhasználói szemszögből: a fájl összetartozó adatok egy kollekciója, amelyeket egy másodlagos tárban tárolunk. A fájl a felhasználó számára az adattárolás legkisebb allokációs egysége: felhasználói adatot a háttértáron csak valamilyen fájlban tárolhatunk.· Az operációsrendszer támogatást nyújthat a fájl tartalmának kezelésében, a fájl szerkezetének (adatszerkezet) létrehozásában.

32. Az állományszerkezet fogalmai (mező, rekord, állomány, adatbázis) és az állományokkal kapcsolatos alapvető műveletek.

Mező: az adat alapvető egysége, egy értéket tartalmaz, hosszával és típusával jellemezhető Rekord: összetartozó mezők gyűjteménye, egy egységként kezelhető

Állomány: hasonló rekordok gyűjteménye, önálló egység, egyedi fájlnevek, hozzáférés korlátozható Adatbázis: összetartozó adatok gyűjteménye, az elemek között kapcsolatok léteznek Alapműveletek:

1. Létrehozás (create) – területallokálás + új bejegyzés a directoryba.
2. Írás (write) – write pointer szerepe
3. Olvasás (read) – kurrens pozíció szerepe
4. Újrapozicionálás (repositioning)
5. Törlés (deleting)
6. Csonkítás (truncate)

További műveletek: bővítés (append), átnevezés (rename), másolás (copy), az attribútumok megváltoztatása.

33. Operációs rendszer és futtató rendszer támogatás az állomány kezeléshez: a fájlrendszer architektúrája (rétegei). Az állomány megnyitás és lezárás szerepe.

Fájlkezelő rendszer

a fájlokhoz való hozzáférést biztosítja a felhasználók számára

a programozónak nem szükséges fájlkezelő szoftvert fejlesztenie, ez az op. rendszer egyik szolgáltatása Fájlrendszer architektúra

- Eszközkezelők:
  - legalacsonyabb szint
  - perifériákkal való közvetlen kommunikáció (eszközfüggő)
  - I/O műveletek megkezdéséért felelős az adott eszközön
  - I/O kérelmeket dolgoz fel
- Fizikai I/O:
  - alacsony (blokk) szintű műveleteket végez
  - a blokkok elsődleges memóriában való elhelyezésével foglalkozik

- I/O felügyelő:
  - a fájl I/O elkezdéséért és bejezéséért felelős
  - a hozzáférés ütemezésével foglalkozik (telj esítményfokozás)
  - az operációs rendszer része Logikai I/O
  - lehetővé teszi az alkalmazások és a felhasználó számára a rekordokhoz való hozzáférést
  - általános célú rekord I/O műveleteket szolgáltat
  - a fájlokat jellemző alapvető adatokat tartja karban

Megnyitáskor először a rendszer beolvassa az attribútumokat, ezután a munka fájlal felgyorsul a feldolgozás. Először kiderül, hogy létezik-e a fájl. Ha outputállomány, ezzel hozza létre az iniciális attribútumait. Ezzel valósul meg a fájlvédelem, az osztott felhasználás, vagyis ha többen megnyitották, szabad-e nekem használni. Lezáráskor közlöm, hogy én hoztam létre, hogy más is használta, és a rendszer visszaírja az attribútumokat a háttértárra. Mentésnél a rendszer lezárja és újranyitja az állományt.

34. Fájl szervezési módok és támogatásuk (pile, szekvenciális, indexelt, indexszekvenciális, direkt/hashing).

pile

- adatgyűjtés érkezési sorrendben (struktúrálatlanul)
- a cél: nagy mennyiségű adatot felhalmozni és elmenteni
- rekordoknak különböző mezőik lehetnek
- nincs szerkezete
- a rekordhoz való hozzáférés fárasztó kereséssel jár....

szekvenciális

- a rekordokat egyetlen sorrendben, a fájl első rekordjától az utolsó felé haladva éri el, mely sorrend megegyezik a rekordok létrehozásának sorrendjével
- a rekordok mérete és formátuma azonos,
- kulcsmező használata
- egyértelműen meghatározza a rekordot
- a rekordok fizikailag egymás után következnek, vagy rekordmutatók használatával egy láncolt lista határozza meg a rekordok sorrendjét.
- akkor alkalmazzuk, ha a fájlt használó program a rekordok összességének feldolgozását igényli

indexelt

- a különböző kulcsmezőkhöz többszintű indexet használunk
- új rekord hozzáadása esetén az összes indexfájlt frissíteni kell
- olyan alkalmazásoknál használatos, ahol az információ időzítése kritikus

indexelt szekvenciális

- direkt hozzáférési eljárás, amely a kulcs szerinti kereséshez indexeket használ
- index: kulcsértékeket és rekordmutatókat tartalmazó táblázat. Az index lehet egyszintű vagy többszintű. Az indexek külön fájlba, ún. indexfájlba kerülnek.
- az egyszintű indexben illetve a többszintű index legalsó szintjén a kulcsértékek mellett
- a rekordmutatókat találjuk, míg a többszintű index felsőbb szintjein a kulcsértékek mellett az alattuk lévő szint táblázataira találunk utalásokat.
- új rekordok hozzáadása egy overflow fájlhoz, amit firssítéskor hozzáfűzünk a fő fájlhoz
- a teljesítmény növeléséhez többszintű indexeket lehet használni ugyanahhoz a kulcsmezőhöz
- olyan adatbázisokhoz is alkalmazzuk, ahol gyakoriak az összetett feltételű keresések

direkt hasításos (hash) fájlok

- direkt hozzáférési eljárás, melynek során egy kulcs értékéből az ún. hasítófüggvény határozza meg a rekordmutatót. Ha az így kijelölt helyen nincs a keresett rekord, az eljárás szekvenciális kereséssel folytatódik.
- kulcsmező szükséges minden rekordhoz
- alkalmazás: ha a tárolandó adatmennyiséghez képest legalább 3-4- szeres terület áll rendelkezésre

35. Könyvtár (fájljegyzék/directory) szerkezetek fejlődése, fájljegyzékekkel kapcsolatos operációs rendszer műveletek.

Tartalom: fájlokkal kapcsolatos információkat tartalmaz (kiterjesztés, hely, tulajdonos)
– a könyvtár maga is egy fájl, melynek tulajdonosa az operációs
rendszer a fájlnevek és fájlok közötti kapcsolatot biztosítja

Könyvtárszerkezet

- Egyszintű könyvtár
  - bejegyzések listája, minden fájlhoz egy
  - szekvenciális fájl, ahol a fájlnevek szolgálnak kulcsként
  - nem nyújt segítséget a fájlok rendezéséhez (csoportosítási problémák)
  - nem lehet két különböző fájlnak ugyanaz neve! (elnevezési problémák)
- Kétszintű könyvtár
  - egy-egy jegyzék minden felhasználónak és egy főkönyvtár (user/master dir.)
  - a főkönyvtár minden felhasználóhoz tartalmaz bejegyzést (hozzáférési jogok)
  - minden felhasználói jegyzék egy egyszerű listája a felhasználó fájljainak
  - névadási probléma megoldva, de csoportosítás továbbra sem lehetséges

36. Fa szerkezetű és aciklikus fájljegyzékek.

- Fa-szerkezetű könyvtár
  - főkönyvtár, alatta felhasználói könyvtárak
  - egy fájljegyzék bizonyos elemei lehetnek újabb fájljegyzékek (alfájljegyzék), így fájljegyzékeknek egy hierarchikus rendszere jön létre
  - a fájlok a főkönyvtárból kiindulva különböző ágakon haladva találhatók meg
  - ez lesz a fájl elérési útja (pathname)
  - több fájlnak is lehet azonos neve, amíg az elérési útjuk eltérő
  - munkakönyvtár (current directory) váltása cd()
  - a fájlok a munkakönyvtárhoz képest is hivatkozhatók (relative path)
- **Aciklikus fájljegyzék (fájljegyzék=directory:**
  - Egy aljegyzék (fájl) osztott használatának problémája: több felhasználó/alkalmazás szeretné a
  - saját directory rendszerében látni.
  - Megoldás: egy típusú directory bejegyzésnek, valamely fájlra, vagy aljegyzékre mutató kapcsolónak (*symbolic **link***) a bevezetése. Logikailag: alias-képzés!
  - Technikailag lehetséges lenne a mutató (link) helyett a teljes directory bejegyzést

37. Fájl hozzáférési módok és fájlvédelem, UNIX fájlvédelem.

Fájlmegosztás

- Többfelhasználós rendszerben a fájlok megoszthatók a felhasználók között

Hozzáférési jogok

- nincs
- a felhasználó még a fájl létezéséről sem tud
- a felhasználó számára nem engedélyezett azon könyvtár olvasása, mely tartalmazza a fájlt
- ismeret
- a felhasználó csak a fájl létezéséről tud, illetve hogy ki a fájl tulajdonosa
- végrehajtás
- a felhasználó betöltheti és futtathatja a programot, de nem másolhatja
- olvasás
- a fájl minden célból olvasható, így futtatható és másolható is
- hozzáfűzés
- a fájlhoz adat hozzáfűzhető, de a fájl eredeti tartalma nem törölhető és módosítható
- frissítés
- a fájl módosítható, törölhető, létrehozható, újraírható, stb.
- védelem megváltoztatása
- a felhasználó a hozzáférési jogokat megváltoztathatja
- törlés
- a felhasználó törölheti a fájlt

    - tulajdonos
- az összes elöbbi joggal rendelkezik
- jogokat határozhat meg más felhasználó számára a következő csoportosítással
    - egy bizonyos felhasználó
    - felhasználók egy csoportja (user group)
    - mindenki (publikus fájlok esetén)
- Szimultán hozzáférés
    - a felhasználó lezárhatja a fájlt frissítés megakadályozása céljából
    - a felhasználó lezárhat egyedi rekordokat frissítés közben
    - a megosztott hozzáférés problémái: kölcsönös kizárás és holtpont

38.Fájl allokáció: folytonos, láncolt, indexelt.

Másodlagostár-kezelés

- fájl allokáció: másodlagos tárhely fájloknak való kiosztása
- szabad tárhely kezelés: nyomon követi a kiosztásra alkalmas tárhelyet

Előfoglalás

- a fájl létrehozásakor szükség van a lehető legnagyobb várható fájlméretre
- nehéz elég pontosan megjósolni a potenciális maximális fájlméretet
- fájlméret túlbecslése célravezető

Háttértár kiosztási módszerek

- **folytonos kiosztás**
    - minden fájl egymást követő blokkok sorozatát foglalja el
    - a helyfoglalás katalógusbejegyzése: kezdő blokk és elfoglalt blokkok száma
    - algoritmusok szükségesek a megfelelő méretű szabad helyek megkeresésére
    - algoritmusok közös hibája: külső töredezettség veszélye
    - állományok általában nem bővíthetők!!
- **láncolt kiosztás**
    - minden állomány blokkok lácolt listája, ezek a lemezen tetszőleges helyen helyezkednek el
    - minden blokk tartalmaz egy mutatót a lánc következő blokkjára
    - a fájl allokációs tábla bejegyzése az első és az utolsó blokkra mutat
    - nincs külső töredezettség, és a fájlok egyszerűen bővíthetők
    - szekvenciális fájlok esetén biztosít nagy hatékonyságot
- **indexelt kiosztás**
    - mutatókat indexblokkokba tömöríti, az indexblokk i-edik eleme az állomány i. blokkjára
    - mutat, a fájlallokációs tábla az indexblokk címét tárolja

39.A FAT.

Az indexelt állományszervezés általánosítása.

Elv: 1 indextábla, minden blokkhoz 1 bejegyzés,olyan univerzális indexelés, mikor a pointereket a memóriában tartom, minden blokkhoz 1 pointer, ha a szóban forgó blokk fájlhoz tartozik, megmutatja melyik a fájlnak a következő blokkja. Rendszertöltéskor a memóriába be lehet vinni a fat táblát. ha nagy a háttértár, akkor blokkok helyett foglalási egységet használunk (cluster). Tehát a FAT 2 helyen van jelen, a memóriában és a háttértáron, amit az újabb rendszerek updatelnek. A FAT 32 annyiban különbözik a FAT-tól, hogy a clusterek 32 bitesek. **A FAT nem tartalmazza a fájlnevet, azt a directory tartalmazza.**

40.A UNIX rendszerek indexelt allokációjának sémája. (INODE)

Az inode-ok egy-egy fájl minden adatát tartalmazzák a nevén kívül. A név ugyanis a könyvtárban tárolódik az inode sorszámával együtt. Az inode több adatblokk sorszámát tartalmazza, melyek a fájl adatait tárolják. Az inode írja le egy fájl lemezen való elhelyezkedését, a fájl tulajdonosát, a hozzáférési jogosultságokat és időket. Minden fájl egy és csak is egy inode-dal rendelkezik, míg neve több is lehet. A néven keresztül lehet kapcsolatot teremteni az inode-dal, amely azután elvezet a fájlban tárolt információhoz. az inode tartalmazza: a fájl tulajdonosainak azonosítóját, a fájl típusát, hozzáférési jogokat, utolsó hozzáférés illetve módosítás idejét, fájlra mutató linkek számát, fájlméretet, a fájl által elfoglalt lemezblokkok táblázatát

41.Szabadhely kezelés.

- Bit tábla használata: diszk minden blokkjához egy bitet rendelünk, a bit értéke mutatja az adott blokk foglaltságát
- Láncolás: láncolt lista a szabad blokkokról
- Indexelés: indextábla a szabad blokkokról
- Szabad blokkok listája: külön területen, a diszken tárolva

42.Directory implementáció.

A jegyzék maga is egy fájl, ami bejegyzéseket tartalmaz más fájlokról. A bejegyzésekben
a fájlok attribútumait tárolhatjuk. Miután a jegyzék is fájl, blokkok tartoznak hozzá is. Blokkjain belül vannak a bejegyzések, ezek lehetnek állandó, vagy változó hosszúságúak. A bejegyzésekben az attribútumok között a legfontosabb a fájlnév. A bejegyzések struktúrája lehet:

- lineáris,
- nem rendezett bejegyzéseken, amik között nem foglalt bejegyzések is lehetnek (a törölt fájlokra);
- rendezett, „hézagok” nélküli bejegyzéseken, ahol is gyorsabb keresési módszerek is megvalósíthatók;
- hash táblás: a szokásos szekvenciális bejegyzések mellett egy hash táblát is implementálnak, ami a gyorsabb keresést segíti.