Operációs rendszerek II.

1. **Az operációs rendszer memóriakezelőjével szemben támasztott követelmények (relokáció, védelem, megosztás, logikai szervezés, fizikai szervezés)**

Megosztás (Sharing): a hatékonyságot növelhetik a több processzus által közösen használt kód és adatok.

Védelem (Protection): a processzusok alaphelyzetben csak a hozzájuk rendelt címtartományt használhatják. (A megosztás természetes velejárója.)

Relokáció: a program (logikai címtartományának) eltolása. Tömörítés és dinamikus áthelyezés. A fordítóprogram nem közvetlenül fizikai címeket használ, hanem a program elejéhez viszonyított relatív címeket. A program így áthelyezhető csak a kezdőcímet kell megváltoztatni.

Logikai és fizikai címtér

- Logikai cím = a CPU által generált cím (virtuális cím).
- Fizikai cím = a Memory Management Unit (MMU) által generált cím (reális cím).
- A fordítási és betöltési időben csak a logikai címtér (címhozzárendelés) elérhető!
- A logikai címet a MMU képezi le fizikai címmé.

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

Folytonos tárallokálásnál egy program számára kioszthatő tárterület folytonos (összefüggő).

Partícionálás: Egyszerre több folyamat is a memóriában tartózkodik. A teljes memóriát partíciókra osztjuk. 1-1 partíció úgy viselkedik a folyamatok számára mintha a teljes memória lenne.(Partíción belül alkalmazható overlay ill. swapping.

Fix partícionálás: A felhasználói tárterületet olyan különböző méretű partíciókra osztották, melyek mérete a rendszer működése során nem változhatott. Minden partícióban egyetlen folyamat tartózkodhat. A multiprogramozás fokát a partíciók száma korlátozza. Belső tördelődés: kihasználatlan memóriaterület (a partíciókon belül).

3. **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.

Külső fragmentáció: Az a jelenség, amikor a nyilvántartott lyukak száma nő, mérete csökken. Megoldható memóriatömörítéssel.

Belső fragmentáció: A fragmentáció az allokált területen belül van (nem lyuk). Keletkezhet úgy, hogy a processzushoz a kis lyukakat hozzáveszem.

Lyuk (hole): két foglalt partíció közötti szabad memória terület (blokk).

A memóriakezelés egyik módja bittérképekkel végezhető el minden allokációs egységnek egy bitet feleltetünk meg, mely 0/1-gyel jelzi annak foglaltságát. A másik megoldás láncolt lista használata. Fűzzük láncolt listába a memória szabad és foglalt területét.

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.

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

A memóriablokkok mérete 2^L és 2^U között változhat, ahol 2^L foglalható legkisebb blokkméret 2^U pedig a memória teljes mérete. 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.

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

Lapozás: a virtuális címteret lap nevű egységekre osztják, a lapnak megfelelő egység a memóriában a lapkeret, a memória és a lemez közötti átvitel laponként történik. A logikai- és fizikai címtér független blokkokra (lapokra, keretekre) bomlik. A logikai blokk (lap/page) és a fizikai blokk (keret/frame) mérete megegyezik. A méret 2 hatvány, jellemzően 512-8192. Bármelyik lap elfoglalhatja bármelyik keretet. Nyilván kell tartani a szabad és foglalt kereteket.

- minden processzusnak saját laptáblája (Page Table) van
- minden laptáblabejegyzés tartalmazza a főmemóriában található megfelelő lap keretszámát
- egy bit szükséges annak jelzésére, hogy a lap a főmemóriában van, vagy nem
- egy másik, ún. „módosító bit" szükséges annak jelzésére, hogy a lap tartalma megváltozott-e a főmemóriába való utolsó betöltődése óta
- ha nem történt változás, a lap kimentésekor a lemezre való kiírása nem szükséges

Laptáblák:

- 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

Címfordítási gyorsítótár (Translation Lookaside Buffer):

- minden virtuális memóriahivatkozás két fizikai memóriahozzáférést okoz:
- behozni a megfelelő laptáblát
- behozni az adatot
- a memóriahozzáférési idő ezen duplázásának kivédésre egy nagy sebességű cache memóriát használunk a laptábla bejegyzésekhez

TLB - Translation Lookaside Buffer

- ez tartalmazza a legutóbb használt laptábla bejegyzéseket
- úgy működik, ahogyan egy memória gyorsítótár (cache)
- ha adott egy virtuális cím, a processzor megvizsgálja a TLB-t
- ha laptábla bejegyzést talál, a keretszámot kinyeri és megalkotja a valós címet ha laptábla bejegyzést nem talál a TLB-ben, a lapszámot használja a processzus laptáblájának indexelésére először ellenőrzi, hogy a lap a főmemóriában van-e már ha nincs, egy laphiba történik
- a TLB egy újabb lapbejegyzéssel történő frissítése a következő lépés

Memória típusai:

Valós memória

- főmemória, melyben a processzusok végrehajtásra kerülnek

Virtuális memória

- a memória a merevlemezen helyezkedik el
- hatékony multiprogramozást tesz lehetővé és mentesíti a felhasználót a főmemória méretének korlátai alól, a felhasználó a valós memóriánál nagyobb memóriát érzékel

Invertált laptábla: a valós tár minden lapkeretéhez tartozik egy bejegyzés, mely jelzi, hogy az adott lapkeretet, mely program, mely lapjaként használja.

6. Szegmentáció, szegmentáció lapozással.

Szegmentálás esetén a memóriát nem egydimenziós tömbként képzeljük el, hanem – abból kiindulva, hogy jobb volna több virtuális címteret használni két dimenzióban. Több egymástól független szegmens hozható létre, melyek mérete változhat. A szegmens logikai egység, így hívása a szegmens első pontjára ugrást jelenti, és külön védelmi szintje lehet.

A memóriahivatkozás formája: szegmensszám, eltolás

- lehet különböző méretű (dinamikusan változtatható)
- egyszerűsíti a növekvő adatszerkezetek kezelését
- lehetővé teszi programok változtatását és független újrafordítását
- alkalmas a processzusok közötti adatmegosztásra és a védelem megoldására

Szegmentációs tábla:
- minden bejegyzése tartalmazza a megfelelő szegmens főmemóriabeli kezdőcímét, illetve a szegmens hosszát
- egy bit szükséges annak eldöntésére, hogy a szegmens a főmemóriában van-e már
- egy másik bit is kell annak meghatározására, hogy a szegmens memóriába való betöltése után megváltozott-e

Szegmentáció és lapozás
- a lapozás láthatatlan, a szegmentáció látható a programozó számára
- a lapozás a külső töredezettséget csökkenti
- a szegmentáció lehetővé teszi az adatszerkezetek növelését, a moduláris
- felépítést, illetve támogatja a megosztás (és a védelem) megvalósítását
- minden szegmens több, azonos méretű lappá van tördelve

7. A virtuális memóriakezelés koncepciója, szolgáltatásai, jelentősége.

A virtuális memória koncepciója a felhasználó/programozó memóriaszemléletének teljes szeparálását jelenti a fizikai memóriától.

A lapozás és a szegmentáció előnyei:
- Több processzus is tartózkodhat egyszerre a főmemóriában
- minden processzusnak csak egy része kerül betöltésre
- a főmemóriában tartózkodó sok processzus esetén nagyon valószínű, hogy bármely időpillanatban lesz „futásra kész” processzus a főmemóriában
- Egy processzus nagyobb lehet, mint az összes főmemória mérete

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

Laphiba: Ha nincs bent a keresett lap a memóriában, akkor be kell tölteni. Kiszolgálási ideje jelentősen nagyobb, mint egy már betöltött lap esetén.
- a program, az adat és a verem együttes mérete meghaladhatja a fizikai memória mennyiségét
- az op. rendszer csak az program éppen használt részét tartja a memóriában, a többi a lemezen van

A kényszer-lapozás teljesítő képessége
Laphiba (page fault) arány: $0 \leq p \leq 1$ ($p=0$: nincs laphiba)
A „módosított” (dirty) bit szerepe a lapcserére fordított idő redukálásában

Effektív elérési idő

(EAT: Effective Access Time)

EAT= (1-p) ∗ memória elérési idő + p(a laphibával kapcsolatos póttevékenység + [az áldozat-lap kimentési ideje] + a hivatkozott lap betöltési ideje + újraindítási idő)

9. Virtuális memória stratégiák (behozás, elhelyezés, áthelyezés, tisztítás).

Behozási stratégia (Fetch policy): Azt határozza meg, mikor és mennyi lapot kell a memóriába betölteni

- Demand paging: csak akkor töltünk be egy lapot a főmemóriába, ha hivatkozás történik rá. (A processzus első indításakor sok laphiba.)
- Prepaging: a szükségesnél több lapot hoz be. (Hatékonyabb olyan lapokat behozni, melyek a diszken szomszédosak egymással.)

Elhelyezési stratégia (Placement policy): Meghatározza, hogy a valós memória mely részére tegyen egy adott processzus darabot.
- Szegmentáció esetén: best-fit, next-fit, first-fit
- Lapozás esetén: megfontolás nélkül.

Tisztítási stratégia: feladata annak eldöntése, mikor szükséges egy módosított lapot a másodlagos tárra menteni

Igényelt tisztítás (demand cleaning)
- egy lap csak akkor lesz kiírva, amikor kiválasztjuk cserére

Előtisztítás (precleaning)
- lapok kimentése még mielőtt lapkeretjeikre szükség lenne, a lapok kiírása
- kötegekben (batch) történik

A legjobb közelítés: page buffering
- a kicserélt lapokat két listában helyezzük el
- módosított lapok és nem módosított lapok
- a módosított listában lévő lapokat periodikusan kiírjuk és a nem módosított lapok listájába tesszük
- a nem módosított lapok listájában levő lapok hivatkozásuk esetén újra visszanyerhetők, vagy véglegesen elvesznek (törlődnek) ha a lapkeretüket egy másik lap kapja meg

10. Laphelyettesítési (cserélési) algoritmusok: FCFS/FIFO, Béládi anomália, optimális algoritmus.

First-in first-out (FIFO)
- a processzushoz tartozó kereteket körkörös pufferként kezeljük
- a lapokat körkörösen (round-robin) dobjuk ki
- ezt a stratégiát a legegyszerűbb megvalósítani
- a memóriában legrégebb óta tartózkodó lap kerül kidobásra

Probléma: előfordulhat, hogy ezekre a lapokra nagyon hamar szükség lesz újból
Béládi anomália: a keretek számának növelése nem csökkenti a laphibák számát

Optimális stratégia
- azt a lapot dobjuk ki, melyre való hivatkozás a jövőben a legkésőbb történne
- lehetetlen implementálni: nem ismerhetjük a jövő eseményeit...

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

Óra algoritmus (második esély algoritmus)
- bevezetünk egy jelző bitet minden laphoz {use bit)
- amikor a lap betöltődik a memóriába, a use bit 0 értéket kap
- a lapra való hivatkozás után a use bit 1 értéket kap
- amikor lapot kell cserélni, az első olyan lap lesz dobva, melynek use bit értéke 1
- a kicserélésre való keresés során minden 1 értékű use bit értékét 0-ra állítjuk

Legrégebben használt (Least Recently Used - LRU)
- laphiba esetén a legrégebben használt lapot dobjuk ki
- a lokalitási elvből következik, hogy annak valószínűsége, hogy erre a lapra a közeljövőben szükségünk lesz, a lehető legkisebb
- minden lapot meg kell címkézni az utolsó rá való hivatkozás idejével. Ennek hatékony implementálása nem egyszerű feladat!

Nem mostanában használt (Not Recently Used - NRU)
- hivatkozás bit (reference bit - R), módosított bit (modified bit - M)
- óramegszakítás: R bit törlése periodikusan
- 0. osztály: nem mostanában hivatkozott, nem módosított
- 1. osztály: nem mostanában hivatkozott, módosított
- 2. osztály: mostanában hivatkozott, nem módosított
- 3. osztály: mostanában hivatkozott, módosított
- véletlenszerűen veszünk ki egy lapot a legalacsonyabb nemüres osztályból

Nem gyakran használt (Not Frequently Used - NFU)
- hivatkozás bit (R) és szoftveres számlálók
- óramegszakítás: periodikusan hozzáadni az R (0 vagy 1) bitet a számlálóhoz
- a számlálók jelzik, hogy egy lap milyen gyakran kerül hivatkozásra
- laphiba esetén a legkisebb számlálóval rendelkező lapot választjuk ki cserére
- NFU és aging együtt jobb megoldás!

12. Rezidens készlet felügyelet.

Ha a rezidens lapkészlet mérete…

- **Fix kiosztású**

A processzusok fix számú lapot kapnak.
Amikor laphiba történik, ezen processzus egyik lapja lesz cserélve.

- **Változtatható kiosztású**:

A processzusokhoz rendelt lapok száma a processzus élete során változik.

- **Változtatható kiosztású, globális**:

- A legegyszerűbb megvalósítani, így a legtöbb opr. adoptálta.
- Az opr. listát tart a szabad keretekről.
- Laphiba esetén szabad keretet adunk a processzus rezidens készletének.
- Ha nincs szabad keret, egy másik processzustól cserél egyet.

- **Változtatható kiosztású, lokális**:

- Új processzus esetén a keretek lefoglalása az alkalmazás típusától illetve más kritériumoktól függ.
- Laphiba esetén az okozó processzus rezidens szetjéből választunk lapot.
- Kiosztás újraértékelése időről időre.

13. Betöltésvezérlés, processzus felfüggesztése.

Betöltésvezérlés:

- Meghatározza azon processzusok számát, melyek a főmemóriába kerülnek.
- Ha túl kevés processzus túl sok alkalommal lesz minden processzus blokkolt és túl sok idő fog elmenni cserével (swapping).
- Túl sok processzus vergődéshez vezethez.

Processzus felfüggesztése:

- Legkisebb prioritású processzust.
- Hibázó processzust (Ezen processzusnak nincs működő része a memóriában, így úgyis blokkolva van.
- Utolsó aktivált processzust.
- A legkisebb rezidens lapkészlettel rendelkező processzust (ezen processzus újratöltése igényli a jövőben a legkisebb erőfeszítést.
- Legnagyobb processzust (A legtöbb szabad keretet igényli.)
- A legnagyobb hátralévő végrehajtási idővel rendelkező processzust.

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

Vergődés(trashing):

- Ha egy folyamat olyan kevés laphoz jut, hogy folyton laphibát eredményez, futása több ezerszer lassabb lehet.
- A memóriából olyan processzus kerül ki, melyre azután azonnal szükség van.
- Előfordulhat, hogy a processzoridő nagy részét a blokkok kicserélgetése foglalja le (a felhasználói utasítás végrehajtása helyett)

Lokalitás elv:

Azon lapok halmaza, amelyekre a processzusnak nagyjából egyszerre van szüksége.

Munkakészlet (working set):

- egy processzuson belül a program és az adathivatkozások klasztereket alkotnak
- rövid idő alatt csak kevés számú processzusblokkra lehet szükség
- jóslások tehetők arra vonatkozóan, hogy a program mely blokkjaira lesz szükség a jövőben
- ezekkel együtt a virtuális memória hatásosan működhet
- Lokalitási elv: Ha egy program adott pontján vagyunk, akkor nagy valószínűséggel, viszonylag hosszú ideig ennek a pontnak egy nem túl tág környezetében fogunk maradni. Következmény: Nincs szükség arra, hogy a teljes program a fizikai memóriában maradjon.

15. Egyprocesszoros folyamatütemezés. Célja, típusai (rövid-, közép- és hosszútávú ütemezés).

Célja: Válaszidő csökkentése, processzor hatásfokának növelése.

Típusai:

Hosszútávú (Long term scheduling – Job Scheduler)
- Meghatározza, hogy mely processzusok kerülnek készenléti állapotba.
- A multiprogramozás fokát határozza meg.
- Minél több processzus van, annál kevesebb futási idő jut egy processzusra.

Középtávú (Medium)
- Csereszolgáltatás (swapping) része, felfüggesztendő processzusok kiválasztása
- A multiprogramozás felügyeletéért felelős.

Rövidtávú (Short – CPU scheduler)
- Dispatcher adja át a vezérlést a kiválasztott processzusnak.
- Leggyakrabban használt ütemezési típus.
- Hívása egy külső esemény bekövetkezésének hatására történik.

16. Ütemezési algoritmusok (kritériumok). Válaszidő, átlagos várakozási idő, átbocsátóképesség.

Rövidtávú ütemezési kritériumok:

Felhasználó szemszögéből
- a válaszidő csökkenjen (a kérelem benyújtása és az első válasz között eltelt idő)

Rendszer szemszögéből
- a CPU az idő minél nagyobb részében legyen elfoglalt

Teljesítménnyel kapcsolatos
- átbocsátó képesség (egységnyi idő alatt befejezett processzusok száma) növekedjen, illetve végrehajtási idő (memóriába kerülés ideje + várakozási idő + CPU + I/O idő) csökkenjen
- átlagos várakozási idő (készenléti sorban eltöltött idő) csökkenjen (erre próbálnak optimalizálni!)

Prioritási sorrend szerinti kiszolgálás:
- az ütemező mindig a nagyobb prioritású processzust választja
- több készenléti sor használata (minden prioritási szinthez)
- alacsony prioritásúak éhezést, éhhalált szenvedhetnek!
- megoldás: „kora" alapján egy processzus megváltoztathatja a prioritását (aging)

17. Preemptív és nem preemptív ütemezés / döntési helyzetek.

Döntési helyzetek, módok:

Nem beavatkozó (nem preemptív)
- a processzus maga mond le a CPU-ról (futó állapotból várakozó állapotba kerül -I/O eseményre vár - vagy megáll

Beavatkozó (preemptív)
- egy futó processzust az op. rendszer megszakít és készenlétei állapotba helyez, vagy várakozó állapotból készenléti állapotba küld
- jobb szolgáltatást tesz lehetővé, hiszen egy processzus sem sajátíthatja ki a CPU-t túl sok ideig

18. Rövidtávú ütemezési stratégiák: FCFS, körleosztásos (round robin), prioritásos.

Igénybejelentési sorrend szerinti - First comefirst served (FCFS):
- minden processzus a készenléti sorba kerül
- mikor az aktuális processzus végrehajtási megszűnik, a készenléti sorban legrégebb óta váró processzus lesz kiválasztva végrehajtásra
- egy rövid processzus túl sokáig várhat végrehajtása előtt...
- az átlagos várakozási idő szórása nagy lehet! (konvoj hatás)

Körleosztásos - Round Robin:
- beavatkozás egy óra alapján: minden processzus sorban egy meghatározott ideig (q) használhatja a CPU-t (q = 10-100 millisec.)
- egyenlő időközönként óramegszakítás generálódik

- megszakítás esetén az éppen futó processzus a készenléti sorba kerül és a következő processzus kerül futó állapotba
- n processzus esetén a várakozási idő: (n-l)/q

A prioritásos ütemezés az SJF algoritmus általánosítása, a készenléti sorban minden feladathoz egy egész számmal kifejezett, a feladat fontosságát jelző prioritást is tárolunk, a CPU-t minden esetben a legmagasabb prioritású feladat kapja.

19. Az átlagos várakozási idő minimalizálása: SJF/SJN/SPF, rövidebb megmaradó idő először, magasabb válaszarány először.

Rövidebb igény először - Shortest Process Next, ShortestJob First/Next:
- alapvetően nem preemptív ütemezés
- a várható legkisebb processzor foglalási idővel rendelkező processzus kerül kiválasztásra (pontosan nem tudjuk, melyik az!)
- hosszabb processzusok háttérbe szorulhatnak, éhezés!
- ha a megjósolt foglalási idő nem helyes, az op. rendszer megszakíthatja a processzust (preemptív ütemezés)
- elméletileg minimalizálja az átlagos várakozási időt

Rövidebb megmaradó idő:
- a rövidebb igény először preemptív változata
- a feldolgozási (foglalási) idő
- becslése szükséges

Magasabb válaszarány először:
- a legnagyobb R=(w+s)/s arányú processzus választása következőnek (w: proceszorra való várakozással töltött idő; s: várható kiszolgálási idő)

20. A processzorfoglalási idő előzetes becslése: exponenciális átlagolás.

A matematikai statisztika segítségével meg lehet becsülni a következő CPU igény hosszát. A becslésre az ún. exponenciális átlagot szokták használni, ami egy alkalmasan választott w<1 esetén a következő: $b_{n+1} = w*t_n + (1-w)*b_n$, ahol t_n az utolsó CPU igény hossza, b_n az utolsó, b_{n+1} pedig a következő igény becslése. A w súly tulajdonképpen a jelen és a múlt hatásának figyelembevételét szabályozza.

21. Több soros ütemezés, visszacsatolásos ütemezés.

A több sorral dolgozó ütemező a készenléti sort több, különálló sorra osztja. Az érkező munkák valamilyen tulajdonságuk alapján kerülnek be valamelyik sorba, besorolásuk később már nem változhat. Minden sorhoz saját, az adott feladattípusnak megfelelő ütemező algoritmus tartozhat. Több sor esetén természetesen a sorok közötti választást is ütemezni kell.

Visszacsatolásos ütemezés - Feedback Scheduling:
Az alapelv az, hogy ha egy folyamat túl sok CPU időt használ, átkerül egy alacsonyabb prioritású sorba, illetve ha egy feladat túl sokat vár egy sorban, magasabb prioritású sorba juttatják. A hosszabban futó processzusok „büntetése".

22. A több processzoros folyamatütemezés architektúrái és problémái.

Szálak ütemezése:
Egy alkalmazás olyan szálak gyűjteménye lehet, melyek együttműködve illetve konkurens módon hajtódnak végre ugyanazt a címteret használva
Külön processzoron futó szálak jelentősen növelik a teljesítményt

Terhelés-megosztás (Load sharing)
- a processzusokat nem rendeljük külön-külön a processzorokhoz, globális sor alkalmazása

Csoportos ütemezés
- összefüggő szálak futásának ütemezése úgy, hogy az egymással párhuzamosan dolgozó processzorokon egyidőben fussanak

Ajánlott processzor hozzárendelés
- a szálak hozzárendelése egyedi processzorokhoz

Dinamikus ütemezés
- a szálak számának változtatása a végrehajtás folyamata közben

Terhelés-megosztás
- a terhelés egyformán van elosztva a processzorok között
- nincs szükség központi ütemezőre
- globális sor használata

Hátrányai:
- a központi sorhoz kölcsönös kizárás kell
- torlódás léphet fel, ha több, mint egy processzor néz munka után egy időben
- felfüggesztett szálak futtatásának folytatása kis valószínűséggel történik ugyanazon a processzoron
- gyorsítótár használata kevésbé hatékony
- Ha az összes szál a globális sorban van, egy program összes szála nem szerezhet hozzáférést a processzorhoz egy időben

Csoportos ütemezés
- egy egyszerű processzus szálainak szimultán ütemezése
- hasznos az összes olyan alkalmazásnál, ahol a teljesítmény drasztikusan csökken, ha az alkalmazás valamelyik része nem fut
- a szálakat gyakran kell egymáshoz szinkronizálni

Dinamikus ütemezés
- A szálak számának dinamikus változtatása rendszereszközök segítségével
- Az op. rendszer szervezi a processzusok betöltését
- üresen járó processzorok hozzárendelése processzusokhoz
- újonnan érkező processzusok kiosztása olyan processzorhoz, mely olyan job-ok által van használva, amelyek aktuálisan több processzort is használnak
- a kérés fenntartása, míg egy processzor elérhető nem lesz
- processzorok jobokhoz való rendelése FCFS alapon

23. Valós idejű ütemezés.

Algoritmus osztályok

Statikus, táblázat-vezérelt megközelítés:
- előzetes végrehajthatósági tervet készít, az ütemezés ennek alapján történik

Statikus, prioritás-vezérelt preemptív megoldás:
- a szituáció elemzése statikus, de az eredmények alapján az ütemezést hagyományos, prioritás alapú ütemező végzi

Dinamikus, terv-alapú megközelítés:
- új taszk indítása esetén az indítást csak akkor engedi, ha az újratervezett ütemezési terv alapján az időzítési elvárások tarthatók

Dinamikus, „best effort" megközelítés:
- nem végzünk megvalósíthatósági elemzést, a rendszer mindent megtesz, hogy a határidőket tartsa

24. I/O eszközök kategóriái, jellemzői, különbségek.

Felhasználó által olvasható:
- a felhasználóval történő kommunikációra használják
- kijelző terminálok
- kijelző, billentyűzet, egér

Gép által olvasható:
- elektronikus eszközzel történő kommunikációhoz
- lemez- (cHsk) és szalag- (tapé) meghajtók
- érzékelők
- kontrollerek

Kommunikáció:
- távoli eszközökkel történő kommunikáció eszközei
- digitális vonalvezetők
- modemek

Különbségek:

Adatátviteli sebesség (Data rate):
- több nagyságrend is lehet az

Felhasználási terület (Application):
- lemezen való fájltárolás esetén fájlkezelő szoftverre van szükség
- virtuális memória lemezen való kialakításához speciális hardver és szoftver szükséges
- rendszeradminisztrátor által használt terminálnak nagyobb prioritása lehet

Vezérlés összetettsége:
Adatátvitel egysége (Unit of transfer):
- átvitel bájtok folyamaként (pl, terminál), vagy nagyobb blokkokban (lemez)

Adatábrázolás:
- kódolási elképzelés

Hibalehetőségek (Error conditions):
- az eszközök különbözőképpen reagálnak a hibákra

25. Az I/O végrehajtása: programozott, megszakításvezérelt I/O, DMA.

Programozott I/O:
- a processzor I/O utasítást ad ki egy processzus nevében
- a művelet befejeződéséig a processzus várakozó státuszba kerül, majd végrehajtása folyatódik

Megszakításvezérelt I/O
- a processzor I/O utasítást ad ki egy processzus számára
- a processzor folytatja a rá következő utasítások végrehajtását (ha az I/O nem szükséges a folytatáshoz, ha igen, akkor egy másik processzus kerül végrehajtásra)
- az I/O egység az I/O művelet befejezésekor egy megszakítást küld a processzornak

Közvetlen memóriahozzáférés (DMA):
- a DMA egység vezérli az I/O eszköz és főmemória közötti adattranszfert
- a processzor a DMA egységnek küld egy adatblokkra vonatkozó adatmozgatási kérelmet, mely csak az egész blokk transzferje után küld megszakítást a processzornak

26. A DMA működése

- a rendszer irányítását átveszi a CPU-tól, hogy adatot mozgasson a memóriába illetve memóriából a rendszerbuszon keresztül
- a rendszerbuszon való adatmozgatáshoz ún. cikluslopást (Cycle stealing) hajt végre a DMA egység, ilyenkor a processzor működése ideiglenesen felfüggesztődik, a CPU szünetet tart egy buszciklus erejéig
- A cikluslopás miatt a CPU működése lassabb
- a szükséges buszciklusok csökkentését lehet elérni a DMA és I/O modulok egyesítésével (így köztük nincs rendszerbusz)
- továbblépés: I/O busz használata az I/O modulok és DMA modul között

27. Az I/O (operációs rendszer által történő) támogatásának követelményei.

Hatékonyság
- a legtöbb I/O eszköz a főmemóriához és CPU-hoz képest nagyon lassú
- multiprogramozás használatával lehetővé válik, hogy processzusok I/O-ra várakozzanak, miközben más processzusok végrehajtás alatt állnak
- ma már léteznek olyan gyors perifériák, amelyek kiszolgálása jelentős teljesítmény-optimalizálást igényel
- a csereszolgáltatás (swapping) lehetővá teszi, hogy további, készen álló, várakozó processzusok a processzornak munkát adjanak

Általánosság
- az I/O eszközök sokszínűségének ellenére egységes periféria-kezelési megoldás szükséges
- az eszközök I/O működésének részleteit alacsony szintű rutinokkal kell eltakarni, hogy a processzusok számára már csak általános műveletek maradjanak: olvasás, írás, megnyitás, bezárás, felfüggesztés, feloldás, stb..
- ezt az op. rendszer szintjén kell megtenni

28. A pufferelés szükségessége, technikái.

A pufferelés okai
- a processzusoknak meg kell várniuk az I/O befejeződését ahhoz, hogy folytatódjanak
- bizonyos lapoknak a főmemóriában kell maradni az I/O alatt
- Blokkos eszközök
- az információ adott méretű blokkban van tárolva
- az egyes blokkok írhatók, olvashatók, az összes többi blokktól függetlenül
- egy blokk egyszerre kerül átvitelre
- lemezeknél és szalagoknál használatos

Karakteres eszközök
- információ átvitele, mint bájtok folyama, nincs blokkszerkezet
- terminálokhoz, printerekhez, kommunikációs portokhoz, egérhez, és a másodlagos tárakon kivüli eszközökhöz használatos

Egyszeres pufferelés:
Az op. rendszer a főmemóriában egy puffert rendel az I/O kéréshez.

Blokkos eszközök esetében a lépései:
- a bevitel egy pufferba történik
- az adatblokk a pufferból akkor kerül a „felhasználóhoz", amikor szükség van rá
- ezután egy másik blokk pufferba mozgatása következik (Read ahead)
- a felhasználói processzus feldolgoz egy blokk adatot, míg a következő blokk beolvasásra kerül
- a swapping megoldható, mert az input az operációs rendszer saját pufferébe kerül, nem pedig a felhasználó pufferébe
- az operációs rendszer nyilvántartja a rendszer pufferek felhasználói folyamatokhoz történő hozzárendelését

Karakteres eszközök esetében:
- egyszerre egy sor használata
- felhasználói adatbevitel: egyszerre egy sor, kocsi-vissza (carriage return) jellel a sor végén
- a terminálra való kivitel: egyszerre egy sor

Dupla pufferelés:
- két puffert használunk, az egyiket az op. rendszer, a másikat a felhasználói processzus
- a processzus adatot írhat illetve olvashat az egyikbe(ből), míg az op. rendszer üríti és tölti a másikat

Körkörös pufferelés:
- n puffer rendelése egy processzushoz
- minden egyes puffer egy körkörös puffer egyik egysége
- a leggyorsabb, akkor használatos, amikor az I/O műveleteknek a processzussal lépést kell tartani

29. Merevlemezek ütemezése: az adatelérés szakaszai, hatásuk a teljesítményre.

A lemezmûvelet mindig valamelyik sáv valamelyik szektorára történő pozicionálással kezdődik.

Lemez teljesítményét jellemző paraméterek:
- íráshoz és olvasáshoz a lemezfejnek a kívánt sávba és a kívánt szektor elejére kell helyeződni

Keresési idő
- az író/olvasó fej kívánt sávpozícióba mozgatásának ideje

Forgási késleltetés (rotational latency) ideje
- várakozás, amíg a kérdéses szektor az író/olvasó fej elé fordul

Hozzáférési idő
- a keresési idő és forgási késleltetés összege, a fej írásra/olvasásra kész

Az adatátvitel megkezdődik, ahogy a szektor a fej alá kerül.
A keresési idő arányos a keresési távolsággal (jó közelítéssel).

30. Lemezütemezési elvek és értékelésük, FIFO, prioritásos, LIFO, SSTF, SCAN, CSCAN, LOOK, N-step-SCAN, FSCAN. Az optimális algoritmus létezése. Külső és belső lemezvezérlés: LBA.

A keresési idő függvényében változik a teljesítmény.

Egy lemez esetében általában sok I/O kérelem történik, ha a kérelmeket véletlen sorrendben szolgáljuk ki, a legrosszabb teljesítményt érjük el

First-in, first-out (FIFO)
- kiszolgálás a kérelmek beérkezésének sorrendjében
- korrekt ütemezés, kevés számú folyamatnál hatékony is lehet
- sok processzus esetén a véletlen ütemezés teljesítményéhez közelít

Prioritásos
- a cél nem a lemezhasználat optimalizálása, a processzusok prioritásától függ....
- a rövidebb, kötegelt jobok prioritása nagyobb, mindig a nagyobb prioritású kérést szolgálja ki először

Last-in, first-out
- mindig a legfrissebb kérést szolgálja ki először
- éhezés lehetősége: egy job soha nem térhet vissza a sor elejére

Legkisebb elérési idő először (Shortest Seek Time First - SSTF)
- mindig a legrövidebb kiszolgálási időt igénylő (amihez a legkevesebb fej mozgás fej mozgás szükséges) kérést szolgálja ki

Pásztázás (SCAN)
- cél a hatékonyság növelése éhezés elkerülése mellett
- a fej egy irányba mozog csak, kielégítve az összes esedékes kérelmet, amíg eléri az utolsó track-et abba az irányban
- ezután megfordul és ellentétes irányba is pásztázik
- a lemez középső részeit favorizálja, ill. nagy mennyiségű kérés „leragaszthatja"

Egyirányú pásztázás (Circular SCAN)
- a szkennelést csak egy irányra korlátozza
- az utolsó szektor elérése után a fej a diszk ellenkező végére ugrik és a szkennelés újból megkezdődik

N-step-SCAN
- a leragadás megoldása
- a diszk kérelmi sort (queué) N nagyságú részekre (subqueue) osztjuk
- egyszerre egy ilyen résznek a feldolgozása történik pásztázással
- ha N nagy, akkor ez nem más, mint a SCAN, ha N=l, akkor pedig FIFO

FSCAN
- a leragadás megoldása
- két sor (queue), amíg az egyikből dolgozik, a kérések a másikba gyűlnek

31. A lemezmegbízhatóság növelése: RAID, szintjei, csíkozás.

Diszkkel kapcsolatos problémák
- CPU lényegesen gyorsabb a diszknél
- nagy kapacitású diszkek hibájának magas kockázata
- diszkek mérete sosem elég nagy...

RAID koncepciója: nagy kapacitású és teljesítményű drága diszkek helyett kisebb (olcsóbb) diszkeket használva érjük el célunkat, azaz:
- kapacitás, teljesítmény és megbízhatóság növelése

RAID jellemzői:
- lényegében több diszk összekapcsolása úgy, hogy azok egymástól függetlenül és parallel működnek:
- az operációs rendszer számára egy diszknek látszanak
- az adatot szétosztjuk a diszkek között
- a diszk hibák ellen paritás információ tárolásával védekezzünk
- a szabvány 5+1 szintet definiál

A különböző megoldások a szükséges tárolóterület overhead-ben, a megoldás teljesítményigényében és a biztonság szintjében térnek el.
RAID 0 csíkozás:adatok azonos méretű szegmensekre darabolása | RAID 1 tükrözés | RAID 3 2 lemez között csíkozás, paritáslemez | RAID 5 3 lemez paritás is csíkozva |

32. Lemez cache (gyorsítótár) működése.

- központi memória puffer a diszk szektorainak
- a diszk néhány szektorának másolatát tartalmazza
- amikor egy I/O kérelem történik, először ellenőrzésre kerül, vajon a kívánt szektor benne van-e a gyorsítótárban

Blokkcsere algoritmusok:

Legrégebben használt (Least Recently Used - LRU)
- az a blokk lesz cserélve, amelyik a legrégebb idő óta a gyorsítótárban van és nem történt rá hivatkozás
- a gyorsítótár blokkok halmazából épül fel
- a legutoljára hivatkozott blokk (illetve egy új blokk) a halom tetejére kerül
- a halom alján levő blokk lesz eltávolítva új blokk behozatalakor

Legritkábban használt (Least Frequently Used - LFU):
- az a blokk lesz cserélve, amelyikre a legkevesebb hivatkozás történt
- minden blokkhoz egy számláló tartozik, értéke minden hozzáférés alkalmával eggyel nő, a legkisebb számhoz tartozó blokk lesz cserélve.

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

- A számítógépek az adatokat különböző fizikai háttértárakon tárolhatják, a számítógép kényelmes használhatósága érdekében az operációs rendszerek egységes logikai szemléletet vezetnek be az adattárolásra és adattárakra: az operációs rendszer - elvonatkoztatva a tároló eszköz és a tárolt adat fizikai tulajdonságaitól - egy logikai tároló egységet (adatállomány - fájl - filé) használ.
- A fájlokat az operációs rendszer képezi le a konkrét fizikai tároló berendezésre. A fájlokat tartalmazó fizikai tároló berendezések általában nem törlődnek.
- 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 operációs rendszer támogatást nyújthat a fájl tartalmának kezelésében, a fájl szerkezetének (adatszerkezet) létrehozásában.

34. 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

Fájl:
- hasonló rekordok gyűjteménye
- önálló egység
- egyedi fájlnevek
- hozzáférés korlátozható

Adatbázis:
- összetartozó adatok gyűjteménye
- az elemek között kapcsolatok léteznek

Alapvető műveletek fájlokkal
Retrieve All, Retrieve One, Retrieve Next, Retrieve Previous, InsertOne, DeleteOne, UpdateOne, RetrieveFew

35. 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

Célok, elvárások:
- felhasználók (alkalmazások) adattárolási -kezelési igényeinek kielégítése
- a fájlban levő adat érvényességének garantálása
- a teljesítmény optimalizálása a rendszer és a felhasználó szemszögéből is
- I/O támogatás biztosítása különböző tárolóeszközök számára
- adatvesztés és sérülés lehetőségének minimalizálása ill. kizárása
- egységes programozói I/O interfész biztosítása
- I/O támogatás biztosítása többfelhasználós rendszeren

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

36. Fájl szervezési módok és támogatásuk (pile, szekvenciális, indexelt, index-szekvenciális, direkt/hashing).

Pile
- adatgyűjtés érkezési sorrendben (strukturálatlanul)
- 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 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 talárjuk, 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

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

Direkt hasításos (hash) fájlok
- direkt hozzáférési eljárás, melynek során egy kulcs értékéből az ún. hasítófuggvény 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

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

Egy directory-ban (fájl-jegyzékben) tárolt információ:
(pl.) név, típus, cím, aktuális hossz, maximális hossz, legutóbbi elérés (access) időpontja, legutóbbi módosítás (update) időpontja, tulajdonos azonosító (owner ID), védelemmel kapcsolatos adatok.

Egy directory-val kapcsolatban végrehajtható (absztrakt alap-) mûveletek:
Fájl keresés (file search), fájl létrehozás (create), fájl törlés (delete), directory listázás (dir, list, ls), fájl átnevezés (rename), a fájl rendszer pásztázása (traverse), bejárása.

A könyvtárszervezéssel szemben támasztott elvárások:
- Hatékonyság: minden fájl könnyen visszakereshető legyen.
- Névadás (naming):
  - Két user használhassa ugyanazt a nevet más fájlokhoz.
  - Ugyanannak a fájlnak lehessenek különböző azonosítói.
- Csoportosítás (grouping)
  - Fájlok csoportosítása tulajdonságaik alapján. (com, bat, pss, nfs ntfs, ... dtv)

Egyszintû directory
- Az összes felhasználó állományai egyetlen jegyzéket (directory) alkotnak.
- Névadási- (névütközési-) és csoportosítási problémák

Kétszintû directory
- Egy-egy felhasználó állományai elkülönített jegyzéket (directory) alkotnak.
- Szintek: MFD - UFD (master/user file directory)
- Megoldja a névütközés problémáját, de nincs csoportosítás.
- A felhasználók nem tudnak kooperálni (nehéz egymás állományait elérni).
- Új fogalom jelenik meg: elérési út (path)
- A rendszer fájlok használatának problémája
dedikált kópia?

Egy (több) speciális (kitüntetett) UFD-ben lehet (pl.) elhelyezni. (Primitív csoportosítás.)
- Keresési út (search path) fogalma.

38. Fa szerkezetű és aciklikus fájljegyzékek.

Fa-szerkezetű könyvtár
- főkönyvtár, alatta felhasználói könyvtárak
- egy fájljegyzék bizonyos elemei lehetnek újabb fájljegyzékek (alfajÍjegyzék), 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 hivatkozhatok (relatíve path)

Aciklikus gráf - szerkezetû 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!

Vö: link, ln rendszerhívás (UNIX), create link (NT).
Technikailag lehetséges lenne a mutató (link) helyett a teljes directory bejegyzést duplikálni:(DE: konzisztencia!)
- Hard link és soft link.
- Ha a kapcsolókat követve nem juthatunk vissza egy olyan bejegyzéshez, amelyet már korábban elértünk, akkor a directory szerkezete aciklikus gráf.
- Nehéz detektálni a ciklust.
- **Problémák**: pásztázás, törlés.
- Törlési politikák: Mi történjék a fájlra mutató linkekkel, ha töröljük a fájlt?
illetve mi legyen a fájllal, ha töröljük a linket?
"Függő" link megengedett: megmarad a link, ha töröljük a fájlt (vagy a fájl nem érhető el).
Pl. UNIX soft link, lokális lemezek mountolása egy fájl rendszerbe. (Ez jól is jöhet!)
"Függő" link nem megengedett.

Megoldások:
- fájl-referencia listák
- referencia számlálók

39. Fájl hozzáférési módok és fájlvédelem, UNIX fájlvédelem.

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

Fájlmegosztás
- 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 előbbi 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

40. Fájl allokáció: folytonos, láncolt, indexelt.

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

41. A FAT.

A táblában a lemez minden blokkjához tartozik egy bejegyzés (pointer), amelyik – amennyiben a blokk fájlhoz tartozik –- a fájl következő blokkjára mutat. Az egyes állományokhoz csak a lánc első blokkjának sorszámát kell tárolni az állományleíróban, a következő blokkok a FAT-ból megtalálhatók. Ezt az allokációs táblát egyidejűleg a szabad blokkok tárolására is fel lehet használni. A tárolt információ közvetlen elérése is egyszerűbb, nem kell az egyes blokkokat végigjárni, elég e FAT-ban „lépkedni”. A FAT, vagy annak jelentős része a tárban tartható, így ez a keresés gyors.

42. 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

43. 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

44. Directory implementáció.

A rendszerben két tartomány van (primitív)
- user
- supervisor

UNIX
Tartomány = USER-ID
Tartomány váltást a fájl rendszeren keresztül lehet végrehajtani.

Minden állományhoz tartozik egy tartomány-bit (setuid bit).
Ha az állomány végrehajtásra kerül és a setuid bit 1, akkor a USER-ID beállítódik a végrehajtás alatt levő fájl tulajdonosának megfelelően. Ha a végrehajtás befejeződött, a USER-ID visszaáll eredeti állapotába.

45. Az adatvédelem és adatbiztonság alapfogalmai, veszélyforrások

„Bizalmasság"
- a számítógépen tárolt információt csak az arra jogosultak férjenek tudják olvasni

Integritás
- eszközöket módosítását csak az arra jogosultak végezhetik (írás, olvasás, törlés...)

Elérhetőség
- eszközök elérhetőek az arra jogosultak számára

Azonosítás
- a rendszer képes megerősíteni/ellenőrizni a felhasználó azonosságát azonosítani

Veszélyforrások

Megszakítás (Interruption)
- egy rendszereszköz elérhetetlenné vagy használhatatlanná válik
- támadás az elérhetőség ellen
- hardverrombolás
- kommunikációs vonal elvágása
- a fájlkezelő rendszer megbénítása

Lehallgatás (Interception)
- illetéktelen csoportok hozzáférést szereznek egy eszközhöz
- támadás a bizalmasság ellen
- „kábelcsapolás" a hálózatban adatlopási céllal
- tiltott fájl és programmásolás

Módosítás (Modification)
- illetéktelenek hozzáférést szereznek és befolyásolják is a rendszert
- támadás az integritás ellen
- egy programkód megváltoztatása
- hálózaton átvitt üzenet tartalmának módosítása

Gyártás (Fabrication)
- illetéktelenek hamis objektumokat helyeznek a rendszerbe
- támadás az azonosítás ellen
- hamis üzenetek küldése hálózaton
- rekordok hozzáadása fájlhoz

46. Védelmi tartomány modell. Hozzáférési mátrixok.

Tartomány szerkezet
Hozzáférési jog = <objektum név, joghalmaz>
A joghalmaz a rendszerben implementált (érvényes) operációknak az a részhalmaza, amelynek elemei az objektumra végrehajthatók.
Tartomány = hozzáférési jogok halmaza

Hozzáférési mátrixok
Sorok: tartományok
Oszlopok: tartományok + objektumok
Bejegyzések: hozzáférési jogok, operátor nevek

Alany
- felhasználók illetve alkalmazások

Objektum
- fájlok, programok, memóriaszegmensek

Hozzáférési jogok
- olvasás, írás, futtatás

Hozzáférést vezérlő lista:
- oszlopokból álló mátrix
- minden objektumhoz megadja a felhasználókat illetve, hogy az egyes felhasználóknak milyen hozzáférési jogaik vannak

A hozzáférési mátrix világosan elválasztja a mechanizmust a politikától

Mechanizmus: az operációs rendszer szolgáltatja a hozzáférési mátrixot + megfelelő szabályokat. Biztosítja, hogy a mátrix csak arra jogosult agensek által manipulálható és a szabályokat szigorúan betartatja.

Politika: a felhasználó diktálja.
Ki érhet el egyes objektumokat és milyen módban teheti ezt.

A hozzáférési mátrixok implementációja
Minden oszlop: = egy ACL (Access Control List, hozzáférési lista) egyetlen objektumhoz.
Definiálja, hogy ki és mit "operálhat" az objektumon.
Minden oszlop: = egy CL (Capability List, jogosultsági lista) egyetlen tartományhoz.
Definiálja, hogy a különböző objektumokon mit "operálhat" a tartományban levő processzus.