# **ATMINTIES VALDYMAS**

## **Pagrindinės atmintinės valdymo sąvokos**

1. **Fizinė adresų sritis** – techninės įrangos palaikoma adresų sritis.
2. **Loginė/virtuali adresų sritis** – procesui matoma atminties sritis.
3. **Fizinis (absoliutusis) adresas** – priskiriamas programos vykdymo metu.
4. **Loginis (bazinis, reliatyvus) adresas** – išreiškiamas bazinio adreso atžvilgiu.
5. **Loginių adresų sritį apibrėžiantys registrai:** 
   * Ribinis
   * Bazinis
6. **Loginių adresų formavimas**
   * Programos kompiliavimas
   * Surinkimas (gaunamas asemblerio kodas)
   * Susiejimas (tada gaunami reliatyvūs adresai)
   * Kodo vykdymas (adresai jau fiziniai)
7. **Rezidentinė (darbinė) proceso dalis** – į pagrindinę atmintinę įkelta proceso dalis.

## **Atmintinės valdymo schemos**

Sprendžia problemas, kurių atsiranda keliant procesus iš antrinės atminties į pagrindinę:

* Kurią atminties dalį perkelti?
* Kur talpinti įkeliamą procesą ar jo dalį?
* Kuriuos duomenis iškelti, kad būtų daugiau vietos kitiems, įkeliamiems procesams?

**Vietos procesui skyrimas atmintinėje:**

* **Nuoseklių (ištisinių adresų) zonos skyrimas:**
  + Procesas egzistuotų kaip vientisas, nuoseklių adresų erdvėje esantis blokas.
  + Tai labai paprastas dalinimo būdas.
  + Problemos:
    - Tinkamo dydžio laisvo bloko atradimas
    - Netinkamas atminties panaudojimas.
* **Neištisinės srities skyrimas:**
  + Proceso duomenys skaldomi tam tikro dydžio, atskirose atmintinės vietose talpinamais gabalais (puslapiais, segmentais).
  + Lengviau atrasti tinkamas proceso patalpinimui vietas atmintinėje.
  + Leidžia padidinti procesų, vienu metu esančių pagrindinėje atmintinėje kiekį.
  + Realizacija sudėtingesnė.

**Pagrindinės atminties paskirstymo strategijos (algoritmai, naudojami skirstant pagrindinę atmintinę):**

* Fiksuoto dydžio skyriai
* Dinaminis skyrių formavimas
* Paprasta segmentacija
* Paprastas puslapiavimas

----------------

Atminties paskirstymo strategijos:

* **Fiksuoto dydžio skyriai**. Pagrindinė dalis yra suskaidoma į daug atskirų skirtingo dydžio (arba vienodo dydžio) skyrių. Į juos vėliau talpinamos programos atitinkamai pagal dydį. Skyriai apsaugoti, jog nebūtų galima procesui kreiptis ne į savo skyrių. Ši strategija sukelia vidinės fragmentacijos problemą, kadangi ne visada programai reikalingas atminties dydis prilygsta gautam atminties dydis. Tad atmintinė nepilnai išnaudojama.
* **Dinaminis skyrių formavimas**. Procesams atminties skyriai yra priskiriami dinamiškai. Kiek atminties procesui reikia, tiksliai tiek jis jos ir gauna. Vėliau kažkuriam procesui pasibaigus, į jo vietą talpinamas kitas procesas ir jam priskiriama tiek vietos, kiek jam reikia. Tačiau taip susidaro tarp procesų skylės, šias skyles galima panaikinti ‚suspaudžiant‘ procesus tarpusavyje.

## **Paprastas atmintinės valdymas**

* **Fiksuoti skyriai:**
  + Pagrindinė atmintis yra sudaloma į eilę nepersidengiančių skyrių (dalių). Šios dalys gali būti tiek vienodo tiek skirtingo dydžio.
  + Procesas, kurio dydis yra mažesnis arba lygus skyriaus dydžiui, gali būti patalpinamas į šį skyrių.
  + Procesorius gali greitai persijungti tarp procesų.
  + Naudojami keli ribiniai registrai apsaugai nuo to, kad procesai negadintų vienas kito duomenų ar programos, kreipdamiesi į ne jam skirtą atmintinės bloką – tokie kreipiniai neleidžiami.
  + Jei visi skyriai yra užimti, operacinė sistema gali iškelti (swap) procesą iš jo užimamo skyriaus.
  + Procesų įkėlimo į fiksuotus skyrius algoritmas:
    - Vienodo dydžio skyriai:
      * Atsiradus laisvam skyriui, procesas gali būti įkeliamas.
      * Nesvarbu į kurį skyrių procesas bus įkeltas.
      * Jei visi skyriai užimti užblokuotų procesų, tai galima kurį nors laikinai iškelti į antrinę atmintį ir taip atlaisvinti vietą kitam procesui.
      * Kol bus įkelti į atmintinę, procesai laukia bendroje eilėje.
    - Nevienodo dydžio skyriai:
      * Gali būti sudarom daug eilių procesams, kurie laukia įkėlimo.
      * Kiekviena eilė susieta su atskiru skyriumi.
      * Procesai pagal savo dydį yra įtraukiami į atitinkamą eilę prie skyrių.
      * Bandoma sumažinti vidinės fragmentacijos problemą – procesą įtraukiant į eilę, parenkamas mažiausias skyrius, į kurį šis procesas gali būti įkeltas.
      * Atsiranda problema, kad daugelis skyrių ir eilių gali būti tušti, jei neatsiranda atitinkamo dydžio procesų. Ši problema sprendžiama sudarant bendrą laukiančiųjų procesų eilę ir atėjus laikui procesą paimti iš eilės ir įkelti, ieškomas mažiausias laisvas skyrius tinkantis šiam procesui. Tačiau naudojant šį principą padidėja vidinė fragmentacija.
* **Dinaminis skyrių formavimas**:
  + Procesams atminties skyriai yra priskiriami dinamiškai: kiek atminties procesui reikia, tiksliai tiek jis jos ir gauna.
  + Vėliau kažkuriam procesui pasibaigus, į jo vietą talpinamas kitas procesas ir jam priskiriama tiek vietos, kiek jam reikia.
  + Tačiau taip susidaro tarp procesų skylės, šias skyles galima panaikinti „suspaudžiant“ procesus tarpusavyje.
  + Talpinimo algoritmai:
* **Sekantis tinkamas** - dažnai naujai talpinamam procesui yra priskiriamas didžiausias blokas, esantis pagrindinės atmintinės pabaigoje.
* **Pirmas tinkamas** - skylės radimas sukasi apie atmintinės pradžią, jį taikant gaunama mažesnė fragmentacija nei „sekančio tinkamo“ atveju.
* **Geriausiai tinkantis - s**urandamas mažiausias tinkamas blokas esantis atmintyje.
* **Blogiausiai tinkantis -** randamas didžiausias savo apimtimi blokas, į kurį patalpinus procesą jame lieka didžiausia neišnaudota erdvė. Yra tikimasi, kad į šią neišnaudotą erdvę vėliau bus galima patalpinti kitą procesą.

**Bičiuliška sistema** – tai algoritmas, kuriuo bandoma apeiti tiek fiksuotų, tiek dinaminių skyrių problemas.

* Atmintinės blokai, kurie yra išskiriami procesams yra *2^{K}* dydžio.
* Pradžioje visa atmintinė yra laisva, taigi pradedama turint *2^{U}* dydžio bloką. Tarkime, atsiranda pareikalavimas patalpinti S dydžio procesą.
  + Jei *2^{U-1} < S <= 2^{U},* tai yra išskiriamas visas blokas *2^{U}.*
  + Priešingu atveju blokas sudalomas į dvi vienodo dydžio *2^{U-1}* dalis (bičiulius).
    - Jei *2^{U-2} < S <= 2^{U-1},* tai procesui išskiriama viena iš dalių (vienas bičiulis), o jei ne, tai viena iš dalių vėl yra daloma į dvi dalis.
    - Šis procesas yra kartojamas tol, kol gaunamas mažiausias blokas, kuris yra lygus arba didesnis nei S.
    - Pavyzdys: Turim 1MB = 210KB = 1024KB . Procesas prašo 100 KB
      * 29 = 512 < 100 < 210 = 1024 – netinka
      * 28 = 256 < 100 < 29 = 512 – netinka
      * 27 = 128 < 100 < 28 = 256 – netinka
      * 26 = 64 < 100 < 27 = 128 - tinka, skiriam 128 KB.
  + Jei du bičiuliai tampa laisvais , bičiuliai yra apjungiami.

------------

* Kai gale procesas nebetelpa į dvigubai mažesnę atmintinę, jis joje patalpinamas. Vėliau atsilaisvinus dviem gretimiems vienodo dydžio atminties laukams – jie apjungiami.

## **Fragmentacijos sąvoka. Išorinė, vidinė fragmentacija.**

* **Fragmentacija** –tai daugybės nedidelio ilgio nesusisiekiančių laisvos atminties skyrių (fragmentų) egzistavimas. Fragmentacija susidaro, kuomet atmintinėje lieka tuščių (laisvų) skyrių. Jie būna laisvi, nes nėra pilnai išnaudojama, kadangi, pavyzdžiui, gali skirtis skyriaus ir priskirto proceso dydžiai. Magnetiniuose diskuose fragmentacija smarkiai sumažina greitaveiką, kadangi perėjimas nuo vienos atminties vietos iki kitos užima didelę dalį laiko.
* **Išorinė** fragmentacija – kai procesų atminties fragmentai yra tarp kitų procesų atminties fragmentų.
* **Vidinė –** kai procesai nepilnai išnaudoja jiems priskirtą atmintį. Tarkim procesas užima 30kb, o jam buvo priskirti 32kb, tad atminties nėra išnaudojama efektyviai.

## **Adresų tipai. Loginio (virtualaus) adreso transliavimas į fizinį. MMU.**

**Adresų tipai:**

* **Fizinis adresas (absoliutusis)** – nurodo objekto fizinę vietą pagrindinėje atmintinėje.
* **Virtualus adresas** – tai nuoroda į objekto vietą, kuria operuoja procesas, nepriklausanti nuo to, į kurią atminties vietą jis yra įkeltas, kokia adresų forma naudojama kompiuteryje.

**?Loginio (virtualaus) adreso transliavimas į fizinį:**

* Kadangi dažniausiai programinėje aplinkoje kodas tiesiogiai nenaudoja fizinių atminties adresų, o tik virtualius, tie virtualūs adresai turi būti paversti į atitinkamus fizinius adresus.
* Tą daro puslapių lentelė.
* Perduodami puslapių, kurių viena dalis eina per puslapių lentelę, kita dalis keliauja tiesiai į fizinės atminties puslapius.
* ???Transliuojant adresą į fizinį adresą atmintinėje yra imamas bazinis segmento adresas, dedama poslinkio reikšmė ir gautas dydis lyginamas su ribiniu to segmento adresu.
* Jei gauta reikšmė yra didesnė už ribinę reikšmę gaunama segmentavimo klaida.

**MMU** – memory management unit, tai atminties valdymo įrenginys. Visi OS atminties adresai keliauja per jį ir jis virtualius adresus verčia fiziniais.

## **Atminties skirstymas taikant paprastą puslapiavimą (proceso įkėlimas, adreso transliavimas, puslapių lentelė ir jos struktūra.)**

Dauguma virtualią atmintinę sudarančių sistemų remiasi skaidymo puslapiais idėja. Tokiu atveju kiekvienas procesas yra suskaidomas į vienodo dydžio blokus, vadinamus puslapiais. Kai kurios OS puslapio dydį sutapdina su disko bloko dydžiu. PA taip pat yra sudalijama į fiksuoto puslapio dydžio blokus, vadinamus rėmais (frames). Atlikus tokį skirstymą, viename disko sektoriuje bus vienas proceso puslapis, kuris tilps į vieną pagrindinės atmintinės rėmą.

* **Proceso įkėlimas**

Prieš pradedant vykdyti kurį nors procesą, yra sprendžiami šie su atmintinės valdymu susiję uždaviniai:

* Nustatomas proceso rezidentinių puslapių skaičius;
* Surandama pakankamai laisvų rėmų PA;
* Į juos įkeliami proceso rezidentiniai puslapiai.

Jeigu į programą pažvelgsime kaip sudarytą iš puslapių, tai jie bus surikiuoti logine tvarka ir pirmas programos puslapis dažniausiai atitiks programos pradžią. Tačiau įkėlimo tiek į diską, tiek į PA metu šie puslapiai nebūtinai bus išdėstyti ta pačia logine tvarka. Tokio įkėlimo į PA nauda ta, kad nebelieka išorinės fragmentacijos ir bet kurį laisvą rėmą gali užimti bet kurio proceso bet kuris puslapis.

* **Loginio adreso transliavimas:**

Kiekvienas loginis adresas turi būti išreikštas puslapio numeriu P ir poslinkiu tame puslapyje D. Viename iš proecsoriaus registrų visada yra tuo metu vykdomo proceso puslapių lentelės pradžios fizinis adresas. Kai programoje aptinkamas loginis adresas, išreikštas puslapio numeri ir poslinkiu (P, D), tai kreipiamasi į puslapių lentelę ir nustatomas atitinkamo rėmo numeris F, kuris, kartu su poslinkio reikšme D, ir nustakys realų adresą. Paprastai aukščiausieji adreso bitai naudojami kaip puslapių lentelės indeksas. Jis rodo, į kurį puslapį yra kreipiamasi. Žemiausieji adreso bitai rodo poslinkį šiame puslapyje. Puslapio ir atmintinės rėmo dydis yra 2N, o loginis bet kurio objekto adresas nusakomas pora (P, D). Pavyzdžii, jei adresas nurodomas 16 bitų seka (0000010111011110), tai pirmi 6 bitai (000001) gali nurodyti puslapio numerį, o kiti 10 bitų rodytų poslinkį puslapyje, kuris gali būti nuo 0 iki 1023.

* **Puslapių lentelė**

Puslapių lentelė – atmintinėje esanti duomenų struktūra, kurioje saugomas virtualių adresų atvaizdavimas į fizinius adresus.

Svarbiausias įrašo laukas yra virtualųjį proceso puslapį atitinkančio PA rėmo numeris, kurio reikšmė naudojama transliuojant adresą – ja pakeičiamas virtualusis puslapio numeris. Puslapio galiojimo bitas rodo, ar atitinkamas virtualusis puslapis yra įkeltas į atmintinę. Jei šio bito reikšmė yra nulinė, tai reiškia, kad puslapis dar neįkeltas. Puslapių apsaugos bitai rodo prieigos prie puslapio galimybes. Jeigu apsaugai skiriamas tik vienas bitas, tai nulinė bito reikšmė rodytų, kad puslapį galima tiek skaityti, tiek rašyti. Galima naudoti ir tris bitas, kuriais nusakomos skaitymo, rašymo ir vykdymo prieigos prie puslapio galimybės. Modifikavimo ir kreipimosi į puslapį bitai nuolat seka, kaip naudojamas puslapis. Šie bitai nagrinėjami, kai OS sprendžia, kurį iš PA esančių puslapių reikia iškelti į diską. Kreipimosi į puslapį bitas skirtas uždrausti įrašyti į spartinančiąją atmintinę, jis naudojamas tuo atveju, kai norime neleisti kreiptis į spartinančiąją atmintinę.

--------------------------

Puslapiavimas yra atminties dalinimas į puslapius, kurie gali būti laikomi kitame duomenų kaupiklyje. Pvz. dalis duomenų kuri netelpa RAM gali puslapiais būti perkelta į kietąjį diską. Taip dirbtinai padidinamas RAM kiekis programoms.

Fizinė atmintis yra sudaloma į fiksuoto, puslapio dydžio blokus, vadinamus rėmais (frames).

Atminties adresas yra rinkinys (f, o):

* f – rėmo numeris (fmax rėmų)
* o – poslinkis nuo rėmo pradžios (omax baitai/rėmai)

**Fizinis adresas gaunamas** – max poslinkis \* rėmo numeris + poslinkis

Kiekvienas procesas yra suskaidomas į vienodo dydžio puslapius.  
Virtualaus puslapio adresas yra rinkinys:

* p – psl numeris,
* o – poslinkis nuo puslapio pradžios,

**Virtualus adresas gaunamas** - max poslinkis \* puslapio numeris + poslinkis

**Proceso įkėlimas –** programa puslapiais įkeliama į antrinę atmintį ir tie puslapiai, kuriuos reikia vykdyti, yra įkraunami į pagrindinę atmintį.

**Puslapių lentelę sudaro** puslapio adresas, validumo bitas, pakitimų bitas ir prieigos galimybių bitai.

## **Atminties skirstymas taikant segmentacijos principus (skirstymo segmentais principas, bendrai naudojami segmentai, segmentų apsauga ir prieigos kontrolė)**

**Skirstymo segmentais principas**

Procesas išskaidomas loginiais segmentais. Kiekvienas segmentas apibrėžiamas adresu, nurodančiu segmento pradžią bei ribiniu poslinkiu. Atmintis dažniausiai segmentuojama pagal atminties pobūdį (programos kodas, duomenys, stekas ir t.t.).  
Kiekvienas elementas segmente apibrėžiamas segmentu numeriu ir poslinkiu segmente.

**Bendrai naudojami segmentai**

* Bendrai naudojantis segmentais, kiekvieno proceso segmentų lentelėse yra įrašai, kurie rodo į tas pačias pagrindinės atmintinės vietas.
* Bendrai naudojamų segmentų dažniausiai neleidžiama modifikuoti.
* Segmentai yra skirti padalinti atmintį į loginius segmentus. Vėliau šiuose segmentus gali dalintis keli skirtingi procesai arba tik vienas, priklauso nuo poreikio. Segmentai būna gan dideli ir skirtingų dydžių, kai tuo tarpu puslapiai labai maži ir visi vienodo dydžio. Puslapiavimas vyksta automatiškai, o segmentaciją turi dažnai valdyti pats programuotojas

**Segmentų apsauga ir prieigos kontrolė**

Segmentų apsauga ir prieigos kontrolė nusakoma apsaugos bitais. Jie nurodo ar procesas gali atlikti: skaitymo, rašymo, kodo vykdymo arba pridėjimo (append) veiksmus su tam tikru segmentu.

Kombinuojant įvairias skaitymo, rašymo, vykdymo bitų reikšmes, galima gauti įvairias prieigos kontrolės modas.

Skirtingi proceso segmentai gali reikalauti skirtingos apsaugos, tai ir atspindi atitinkami apsaugos bitai, nusakydami skirtingą prieigos modos reikšmę.

Segmentų apsaugos bitai įrašomi į segmentų lentelę.

## **Puslapių mainai (puslapių keitimo strategijos, puslapių kilnojimas)**

OS puslapius į pagrindinę atmintinę gali kelti:

* Atsiradus kažkurio puslapio poreikiui. Pradžioje gali būti įkeliamas tik pirmas reikalingas puslapis, po to keliamas tas puslapis, į kurį daromas kreipinys.
* OS gali bandyti nustatyti kokių puslapių procesui prireiks ir įkelti juos iš anksto.

**Puslapių keitimo politika:**

* **Lokali politika:** keičiamas tam pačiam procesui priklausantis puslapis nauju puslapiu.
* **Globali politika:** keičiamas bet kokiam procesui priklausantis puslapis nauju puslapiu.

**Puslapių keitimo strategijos**:

* Keičiamas puslapis, kurio prireiks vėliausiai,
* Keičiamas atsitiktinis parinktas puslapis,
* Keičiamas paskutiniu metu nenaudotas puslapis (NRU),
* Puslapiai keičiami laikantis FIFO disciplinos (gali būti taikomas antras šansas),
* Keičiamas mažiausiai paskutiniu metu naudotas puslapis (LRU),
* Keičiamas ne dažnai naudotas puslapis,
* Laikrodžio algoritmas

Detaliau:

* Keičiamas puslapis, kurio prireiks vėliausiai;

Panašus į procesams planuoti naudojamą algoritmą – trumpiausias procesas vykdomas pirmas.

Šis algoritmas būtų optimalus, bet jis nėra praktiškai taikomas, nes sunku nustatyti, kokiais kreipiniais, į kurį puslapį ir kokia tvarka bus kreipiamasi, vykdant procesą, OS paprastai to iš anksto nežino.

* Keičiamas atsitiktinai parinktas puslapis;

Siūlo leistiną puslapį parinkti atsitiktinai.

Tokį algoritmą nesunku įgyvendinti, tačiau jis nėra labai funkcionalus, nes, kaip jau minėta, gali būti iškeltas puslapis, kurio procesui tuoj vėl gali prireikti.

* Keičiamas pastaruoju metu nenaudotas puslapis (NRU);

Keičiant pastaruoju metu nenaudotus puslapius, stengiamasi atmintinėje išlaikyti tuos puslapius, į kuriuos neseniai buvo kreiptasi.

Šiuo tikslu nagrinėjami du bitai, kurie yra susiję su kiekvienu puslapiu: bitas R, kurio vienetinė reikšmė rodo, kad į puslapį yra kreiptasi, ir bitas M, kuris rodo, kad puslapis yra modifikuotas. Jie atnaujinami, vykstant kreipiniams į puslapį, juos įjungia TĮ.

Taikant NRU algoritmą, pasirenkamas atsitiktinis puslapis iš žemiausios netuščios klasės.

NRU yra gana paprastas, nors ir nėra optimalus, tačiau gana gerai funkcionuoja.

* Puslapiai keičiami, laikantis FIFO disciplinos;

Keičiamas tas puslapis, kuris yra įkeltas seniausia.

Nors iš pažiūros tai ir teisinga taktika, tačiau rezultatai nebūna geri, nes vienodu dažniu yra išmetami tiek tie puslapiai, kurie nėra dažnai naudojami, tiek tie, į kuriuos nuolat kreipiamasi. Į fizinę atmintinę sudaroma įkeltų puslapių grandinėlė, ties kurios pradžiai yra seniausiai įkelti puslapiai.

Tai taip pat gana paprastas algoritmas, tačiau retai taikomas.

Geresnis yra antrojo šanso algoritmas. Priešingai nei pirmu FIFO atveju, jei bitas R įjungtas, tai jo reikšmė verčiama nuliu, o pats puslapis nukeliamas į eilės galą tarsi tai būtų naujai įkeltas puslapis, o procesas kartojamas.

* Taikomas laikrodžio algoritmas;

Šis algoritmas yra efektyvesnis nei antrojo šanso algoritmas, nes puslapių nereikia nuolat permesti į grandinėlės galą, bet iš esmės naudojami tie patys principai, kaip ir antrojo šanso algoritme.

Tik naudojamas žiedinis įkeltųjų puslapių sąrašas, kuriame laikrodžio „rodyklė“ rodo į seniausią puslapį.

Jeigu šio puslapio bito R reikšmė lygi vienetui, tai ji keičiama į nulį, o rodyklė pasislenka prie kito puslapio ir procesas kartojamas, kol randamas puslapis, kurio R reikšmė lygi nuliui.

Jeigu ji lygi nuliui, tai naujas puslapis įkeliamas vietoj šio ir „rodyklė“ taip pat pasislenka link kito puslapio.

* Keičiamas mažiausiai pastaruoju metu naudotas puslapis (LRU – Least Recently Used);

Algoritmas yra tam tikra optimalaus algoritmo aproksimacija.

Taikant laikoma, kad dažnai pastaruoju metu naudoti puslapiai bus naudojami ir ateityje, todėl iškeliamas tas puslapis, kuris buvo naudotas mažiausiai.

Tiksliai įdiegti tokį algoritmą nelengva.

Reikėtų turėti visų PA esančių puslapių nuoseklų sąrašą, kurio priekyje būtų dažniausiai naudoti puslapiai.

Daugiausiai sąnaudų reikėtų nuosekliam puslapių sąrašui palaikyti.

Tačiau yra alternatyvūs sprendimai. Vienas iš jų: naudoti N \* N dydžio matricą, kur N – puslapių rėmų skaičius PA.

Pradžioje visa matrica užpildoma 0. Kai kreipiamasi į K-ąjį rėmą, TĮ priemonėmis matricos K-osios eilutės elementai nustatomi į vienetinę padėtį, o paskui K-ojo stulpelio elementai – į 0.

Eilutė, kurios dvejetainė reikšmė mažiausia, bet kuriuo laiko momentu rodo numerį rėmo, kuriame esantis puslapis turi būti pakeičiamas, taikant LRU algoritmą.

* Keičiamas nedažnai naudotas puslapis (NFU – Not Frequently Used).

Kadangi ne visos mašinos gali pasinaudoti tokio tipo skaitikliu todėl naudojamas NFU algoritmas.

Jis naudoja programinius skaitiklius, kurių reikšmės pradžioje yra 0.

Po kiekvienos laikrodžio mechanizmo generuojamos pertraukties, visų puslapių, į kuriuos paskutiniame laiko intervale buvo kreiptasi, skaitikliai padidinami vienetu Taigi, skaitikliai rodo kreipimųsi į puslapius dažnį.

Puslapis su mažiausia reikšme gali būti keičiamas nauju puslapiu.

Trūkumas, kad matuojamas tik kreipinių dažnis ir visai neatsižvelgiama į kreipinių laiką.

Taip pat, skaitiklio reikšmė gali būti didelė dar kurį laiką, tačiau puslapiu niekas nesinaudoja, todėl pasirenkami naudingi puslapiai, o nenaudingi lieka.

Sendinimo algoritmas yra modifikacija, kuri bando įvertinti kreipimosi laiką.

Esmė, kad kiekvieną kartą, prieš keičiant puslapių skaitiklių reikšmes, skaitiklių reikšmės perstumiamos vienu bitu į dešinę, o paskui prie pirmojo iš kairės pusės esančio bito pridedamos naujos bitų R reikšmės.

Nuo LRU skiriasi, jog sendinimo algoritmas gali atsekti tik vėliausių 16 kreipinių eigą.

**Puslapių kilnojimas**

Dažnas kilnojimas vadinamas šiukšlinimu, nes OS didelę laiko dalį nevykdo procesų.

Vienas iš sprendimo būdų būtų procesams reikalingo rezidentinio puslapių skaičiaus nustatymas.

Jeigu jį būtų galima nustatyti, tai atliekant ilgalaikį planavimą, galima būtų paleisti tik tuos procesus, kurių atmintinės srities poreikiai galėtų būti tenkinami.

O jeigu dideli poreikiai?

Siūloma taikyti rezidentinės aibės modelį (working set model), pagal kurį procesas gali būti laikomas PA tik tokiu atveju, jeigu visi puslapiai, kuriais procesas tuo metu naudojasi, gali būti PA.

Jeigu puslapių poreikiai auga, o PA nėra vietos, procesas iškeliamas iš PA.

Išmetus kiti užbaigiami greičiau.

Dažniausia kliūtis – reikalavimas sekti rezidentinius puslapius, procesų kreipinius į puslapius, keisti šių puslapių aibę, įkeliant naujus ir išmetant senus puslapius.

# **ĮVESTIES IR IŠVESTIES (I/O) SISTEMA**

## **I/O valdymo tikslai**

* Efektyvus sistemos funkcionavimas,
* Pagreitintas duomenų perdavimas,
* Saveika su I/O įrenginiais nepriklauso nuo jų fizinių charakteristikų
* Programos neturi būti pritaikomos konkretiems įrenginiams

## **Įrenginių kontroleriai. I/O veiksmų vykdymas**

Įrenginių kontroleriai - tai sąsajos korta arba integruotas modulis, kuris „kontroliuoja“ išorinius įrenginius. Visi įrenginiai į OS kreipiasi per įrenginių kontrolerį, kuris generuoja pertrauktis procesoriui. Įrenginių kontroleris yra magistrale sujungtas su procesoriumi.

Kontroleriai dažniausiai turi silpną procesorių ir šiek tiek atminties, pavyzdžiui spausdintuvų kontroleriai būna galingi.

**I/O veiksmų vykdymas:**

**Metodai:**

* **Programuojamas I/O:** OS pateikia I/O komandą įrenginio kontroleriui, tada pastoviai tikrina kontrolerį kol šis užbaigs operaciją.
* **Pertrauktimis grindžiamas I/O:** I/O komanda perduodama kontroleriui, procesorius toliau tęsia darbą, kontroleris užbaigia, nusiunčia pertrauktį į procesorių.
* **Tiesioginė prieiga prie atminties:** OS pateikia I/O komandą DMA moduliui ir normaliai funkcionuoja toliau. DMA kontroleris inicijuoja skaitymo arba rašymo komandą. DMA kontroleris perduoda priimamus baitus tiesiai į PA be procesoriaus įsikišimo. Procesoriaus darbas nutraukiamas tik tada, kai visas informacijos blokas yra perduotas į PA.

## **Įrenginių tvarkyklės**

* Kiekvienas prie kompiuterio prijungtas įrenginys reikalauja tam tikros to įrenginio valdymo programos, kuri yra vadinama įrenginių tvarkykle.
* OS paprastai skirsto visas įrenginių tvarkykles į 2 kategorijas.
* Tai tvarkyklės, skirtos blokiniams įrenginiams (pvz., diskams) ir simboliniams įrenginiams (pelei, klaviatūrai).
* Dauguma OS apibrėžia 2 sąsajas, kurias turi tenkinti blokinių ir simbolinių įrenginių tvarkyklės.
* Šias sąsajas sudaro rinkinys procedūrų, kurias gali naudoti likusi OS dalies, kad priverstų įrenginių tvarkyklę atlikti tam tikrus veiksmus.
* Jos atsakingos už keletą funkcijų: jos turi priimti skaitymo/rašymo užklausas; turi sugebėti patikrinti įrenginį, sugebėti jį inicijuoti.

## **Nuo įrenginio nepriklausanti programinė I/O įranga, vartotojo lygmens I/O įranga**

**Nuo įrenginio nepriklausanti programinė I/O įranga**

Tikslios ribos tarp įrenginių tvarkyklių ir nuo įrenginio nepriklausančios PĮ nėra, tai gali priklausyti nuo OS, tačiau daug funkcijų yra bendrų įvairiems įrenginiams. Šios funkcijos apima:

* Bendros sąsajos su įvairiomis įrenginių tvarkyklėmis sudarymą;
* Buferių naudojimą I/O veiksmams;
* Įrenginių priskyrimą ir priskyrimo atšaukimą;
* Nepriklausomo nuo įrenginio bloko dydžio palaikymą.

**Vartotojo lygmens įranga:**

* Pagrindinė PĮ dalis, susijusi su I/O veiksmais, yra sudėta į OS.
* Vartotojo lygmenyje yra bibliotekos, kuriose esančiais I/O sisteminiais kvietiniais gali pasinaudoti vartotojas savosiose programose.
* Vartotojo lygmenyje taip pat vykdomos ir kito tipo funkcijos – kaupimo sistema (spausdintuvams).
* Šiam tikslui sukuriamas specialus kaupimo aplankas, į kurį procesai įkelia savo spausdinti siunčiamus failus.

## **Fizinė disko struktūra**

* Paprastai diskas turi keletą besisukančių plokštelių, kurių abiejose pusėse magnetinės skaitymo ir rašymo galvutės gali rašyti ir skaityti duomenis.
* Sektoriaus dydis lemia minimalų skaitomų ir rašomų duomenų dydį – bloką.
* Failas yra saugomas viename ar keliuose disko blokuose.
* Blokas yra perdavimo vienetas, vykstant mainams tarp disko ir PA.
* Nuoseklus duomenų skaitymas vyksta greičiau nei atsitiktinis duomenų išrinkimas – tai susiję su disko svirtelių judesiu.
* Modernūs diskai, visgi, yra dar sudėtingesni, nes ne visi jų sektoriai yra vienodo dydžio.
* Šiuolaikiniai diskai teikia aukštesnio lygmens sąsają (SCSI).
* Taigi, norint atlikti skaitymo ar rašymo operaciją su disku, reikia nusakyti tik loginio bloko, su kuriuo bus atliekami veiksmai, numerį.

## **Disko užklausų vykdymas**

Skaitymo iš disko ar rašymo į diską užklausos atlikimo greitis priklauso nuo:

* Paieškos - laikas, per kurį disko svirtelės nustatomos ties reikiamu cilindru. Jis priklauso nuo to, kaip greitai gali judėti svirtelės.
* Sukimosi - laikas, per kurį reikiamas sektorius atsiranda ties nuskaitymo galvute.
* Perdavimo laiko - laikas, per kurį duomenys nuo disko paviršiaus perduodami į disko kontrolerį, kuris persiunčia juos į PA.

OS, naudodama diską, stengiasi minimizuoti šiuos laikus. Kadangi paieška trunka ilgai, tai OS stengiasi tvarkyti disko užklausas. Užklausoms tvarkyti gali būti naudojamos šios disciplinos:

* **FCFS** – pirma atėjusi ir bus pirma aptarnaujama;
* **Prioritetinė** – užklausoms gali būti suteikiami tam tikri prioritetai;
* **SSTF** – pirmiausiai aptarnaujama užklausa, kurios paieškos laikas bus trumpiausias;
* **SCAN** – pirmiausiai aptarnaujamos užklausos, susijusios su galvutės svirtelės judesiu ta pačia kryptimi;
* **C-SCAN** – apsiribojama judesiu tik viena kryptimi;

## **RAID architektūra**

Diskų funkcionalumui padidinti ir duomenų praradimams minimizuoti naudojama RAID architektūra.

RAID technologija apima RAID 0 – RAID 7.

Naudojant architektūrą, siekiama sutrumpinti informacijos mainų laikus bei tam tikru perteklumu padidinti saugomos informacijos patikimumą.

Naudojami tam tikri principai:

* Veidrodinio atspindžio – duomenys rašomi į kelis disku vienu metu;
* Skaidymo – duomenys suskaidomi į dalis ir dalimis rašomi į skirtingus diskus;
* Klaidų korekcijos – rašant informaciją į diskus, kartu įrašoma ir papildoma informaciją su galimų klaidų nustatymu.

**RAID architektūros:**

* RAID 0 – architektūra, skirta diskų spartai padidinti, nebūdingas duomenų perteklumas;
* RAID 1 – taikomas veidrodinis principas;
* RAID 2 – informacija dalinama ne sektoriais, o baitais. Saugomi ir klaidų tolerancijos kodai;
* RAID 3 – supaprastintas RAID 1 atvejis. Paprastesni klaidų korekcijos principai;
* RAID 4 – vėl dalijama taip, kaip ir RAID 0 atveju.
* RAID 5 – duomenys skaidomi ir rašomi į skirtingus diskus;
* RAID 6 ir RAID 7 – *pakeitimai knygoje neaprašyti*.

# **SVEČIŲ PASKAITŲ MEDŽIAGA**

## **Soketo sąvoka. Berkeley soketų API. Kliento-serverio darbo algoritmai TCP, UDP atvejais**

**Soketo sąvoka:**

* Soketas yra tam tikra abstrakcija, atitinkanti galinį komunikavimo tašką.
* Jį apibrėžia IP adresas ir prievado (porto) numeris.
* Komunikavimas vyksta tarp dviejų procesų, taigi yra nusakomas dviem galiniais taškais.
* Pavyzdžiui kliento-serverio komunikacijose duomenys keliauja tarp dviejų soketų.

Visuma operacijų, kurios gali būti vykdomos su soketais sudaro Soketų API .

**Kontrolės operacijos** - tai prievado numerio surišimas su soketu, ryšio inicijavimas arba priėmimas sokete arba soketo sukūrimas, suardymas.

**Duomenų perdavimo operacijos** - duomenų rašymas / skaitymas per soketą.

**Statusą nusakančios operacijos** - tokios kaip kad IP adreso susijusio su soketu suradimas.

**Taikomosios programos veiksmai vykdomi su soketais:**

* Sukurti soketą ir surišti jį su tam tikru adresu.
* Sudaryti sujungimus ir priimti susijungimus naudojant sukurtą soketą.
* Siųsti ir priimti duomenis per sukurtą soketą.
* Nutraukti soketų operacijas.

**Berkeley soketų API**

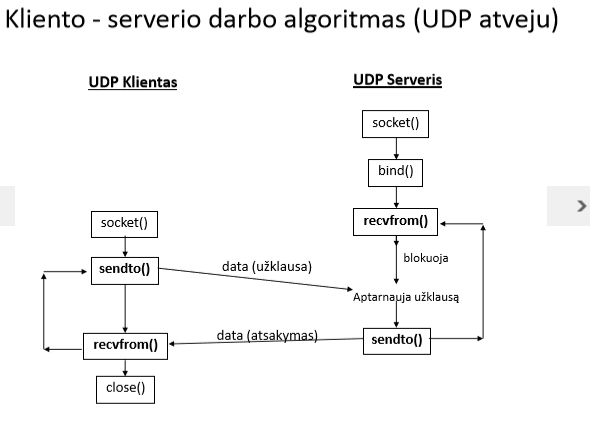
Berkeley soketai leido programuotojams panaudoti interneto galimybes savo produktuose.

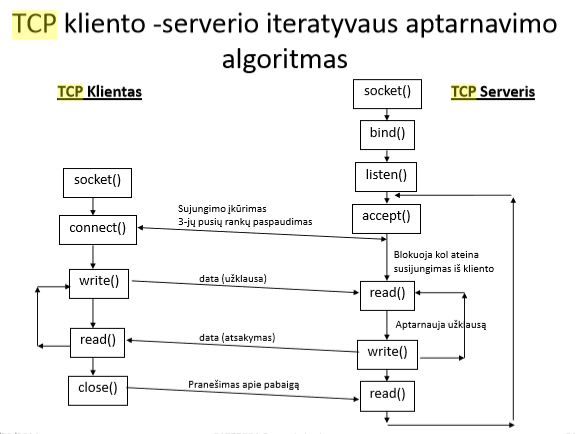
Šios API tikslas – leisti vartotojams kurti aplikacijas, kurios bendrauja tarpusavyje komunikuodamos tinkle.

Vartotojui nereikia rūpintis tokiais klausimais:

* Kaip veikia tinklai,
* Kaip siunčiami duomenys tinklu,
* Kaip duomenys paruošiami persiuntimui tinklu.

**Kliento-serverio darbo algoritmai TCP, UDP atvejais**





**TCP** protokolas - SOCK\_STREAM.

* + Patikimas pristatymas
  + Garantuota pristatymo tvarka
  + Orientuotas į susijungimą
  + Dvipusis

**UDP** protoklas - SOCK\_DGRAM

* + Nepatikimas pristatymas
  + Negarantuota eilės tvarka
  + Neorientuotas į susijungimą
  + Gali siųsti arba priimti

## **Mobilios OS** **S: mobilieji įtaisai, jų savybės. Android OS programų steko architektūra, Dalvik virtuali mašina. Apribojimai programinei įrangai**

**Mobiliųjų įtaisų savybės:**

* Platforma su SDK,
* galima vienu metu naudoti keletą programų (multitasking),
* lanksčiai konfigūruojamas,
* didelis jungiamumas,
* dažniausiai paprastesnės, nei įprastos OS,
* dažnai naudojami tik mobilieji ar bevie
* liai prisijungimo būdai.

**Programų steką sudaro 4 sluoksniai:**

* Programos (applications).
* Programų karkasas (framework).
* Programų vykdymo terpė (Bibliotekos ir DVM).
* Linux branduolys

**DVM:**

Dalvik VM – Programos rašomos Java kalba, tačiau nėra JVM.

Java klasės kompiliuojamos į Dalvik (.dex) vykdomuosius failus ir vykdomi Dalvik VM, kuri yra sukurta specialiai Android.

DVM specialiai optimizuota įrenginiams su ribotu energijos šaltiniu, atmintimi ir CPU.

Bibliotekos – Java 7 edition, Kolekcijos, I/O, ...

* Dalvik virtuali mašina (DVM) – registrais paremta virtuali mašina, kuri buvo optimizuota siekiant užsitikrinti, kad mobilusis įrenginys galėtų efektyviai vykdyti daugelį mašinos egzempliorių.
* Lyginant su Java VM, dėl \*dex failų ir Dalvik VM struktūros, programų vykdymui naudojama daug mažiau atminties.
* Android OS kiekviena Android taikomoji programa veikia kaip atskiras, vieną giją turintis procesas savo Dalvik virtualioje mašinoje, perduodama visą atminties ir procesų valdymo atsakomybę programų vykdymo aplinkai.

**Apribojimai programinei įrangai:**

* Programos neturi būti uždaromos, kai jomis tuo metu yra naudojamasi,
* Mobilieji įrenginiai neturi swap atminties,
* Persijungimas tarp programų turi būti labai greitas (<1s)
* Programos turi būti kuriamos naudojantis API, kuris turi turėti pakankamai funkcijų ir būtų prieinamas trečiųjų šalių programoms.

# **FAILŲ SISTEMA**

## **Failai ir failų sistema (sąvokų apibrėžimai), failų sistemos sprendžiami uždaviniai, failų sistemos abstrakcijos lygiai**

**Failas** - bet koks, turintis vardą duomenų rinkinys, kuriuo yra manipuliuojama kaip atskiru vienetu.  
Operacinė sistema gali sukurti vartotojui sąsają, kuri palengvina jam navigacijos veiksmus su failais. Ši sąsaja yra **failų sistema**.

**Failų sistema apima:**

* diske saugomų failų visumą;
* duomenų struktūras, reikalingas failams tvarkyti (aplankai, failų deskriptoriai, informacija apie vietą diske);
* failus tvarkančias programas (skirtas failų kūrimui, naikinimui, kopijavimui, perkėlimui, įrašymui ir t.t.).

**Failų sistemos uždaviniai:**

* katalogų valdymas,
* vartotojo prieigos kontrolė,
* failų vardų valdymas,
* manipuliavimo failais funkcijos,
* failų struktūros valdymas

**Abstrakcijos lygiai:**

* failai/aplankai
* loginis bloko #
* fizinis įrenginio adresas

## **Failų atributai, veiksmai su failais, prieigos prie failų metodai. Aplankai**

**Failų atributai**: read only, archived, hidden, system

**Veiksmai su failais:**

* sukurti failą,
* išmesti failą,
* ieškoti failo,
* atverti failą,
* uždaryti failą,
* nustatyti failo charakteristikas,
* modifikuoti failo turinį.

**Prieigos prie failų metodai**:

* **Nuosekli** **prieiga** – visi įrašai skaitomi nuosekliai nuo pradžios, negalima šokti atgal skaitymo metu. Taikoma magnetinių juostų atveju.
* **Atsitiktinis** **išrinkimas** – baitai arba įrašai skaitomi bet kuria tvarka. Svarbu duomenų bazių sistemose.
* **Asociatyvi** **prieiga** – atmintinė adresuojama pagal turinį, ne pagal adresą. Naudojama labai greitos paieškos taikomosioms programoms.
* **Tiesioginė** **prieiga**

**Aplankas**(katalogas) – failas su specialia struktūra.

**Paskirtys:**

* vartotojams leidžia grupuoti failus pagal tam tikras savybes.
* leidžia skirtinguose kataloguose esančius failus užvadinti vienodais vardais.
* failų sistemos požiūriu, leidžia atskirti loginę failų struktūrą nuo fizinio failų talpinimo diske.

## **Bendras naudojimasis failais ir jų apsauga.**

OS turi numatyti galimybę keliems vartotojams naudoti tuos pačius failus. Bendrai naudojant failą, sprendžiama veiksmų sinchronizacijos problema: kas atsitiks, jei vienas procesas skaitys failą, o kitas ims rašyti į jį.

Bendras failų naudojimas yra neatsiejamas nuo tam tikro apsaugos mechanizmo.

Failų sistema paprastai kuria tam tikrą apsaugos mechanizmą, kuris sprendžia tokius klausimus:

* Kas turi prieigos prie failo teisę?
* Kokius veiksmus vartotojas gali atlikti su failu?

Apsaugos mechanizmas leidžia patikrinti, ar tam tikro vartotojo vykdomas tam tikras veiksmas su failu yra teisėtas ir leistinas.

Šiai apsaugai įgyvendinti gali būti kuriami prieigos kontrolės sąrašai, kurie gali nurodyti, kokius veiksmus ir kokie subjektai gali atlikti su tam tikru objektu.

Nustatant prieigos kontrolę kiekvienam objektui yra nurodomi subjektai ir jiems leistini veiksmai, o kiekvienam subjektui yra nurodomi objektai ir veiksmai, kuriuos su jais galima atlikti.

----------Not mine---------

Linux sistemos atveju yra galimybė sujungti skirtingo tipo failų sistemas į vieną bendrą failų sistemą: FAT, FAT32, NTFS, ir HPFS.

Techninės įrangos sukeltos problemos

* Failų sistema ir įvykiai, dėl kurių prarandami duomenys:
* Programinės įrangos klaidos
* Atsitiktiniai praradimai
* Priešiškai nusiteikę vartotojai ar įsilaužėliai
* Natūralios ar žmogaus sukeltos nelaimės

Vartotojas turi suprasti kokie jo duomenys reikalauja apsaugos, rūpintis tuo, kad reikės atstatyti duomenis, jei to prireiks, koordinuoti backup planus su sistemos administratoriumi, jei daromi backupai, patikrinti ar iš jų tikrai pavyksta atkurti duomenis.

## **Failų lentelė**

* Failų sistema kiekvienam procesui saugo to proceso atvertų failų lentelę.
* Proceso atvertų failų lentelėje saugomas visų to proceso atvertų failų sąrašas.
* Kiekviename šio sąrašo įėjime saugoma nuoroda į sistemos atvertų failų lentelę, failo atidarymo moda bei einamoji pozicija faile.
* Procesui, atverčiančiam failą grąžinamas jo atvertų failų lentelės indeksas, atitinkantis šio failo įrašo vietą jo lentelėje.
* Naudodamasis šiuo indeksu vartotojas toliau gali atlikti veiksmus su šiuo failu.
* Įėjimas atvertųjų failų lentelėje užpildomas šia informacija: failo dydis, vieta diske, nuorodos į buferius, nuorodų į šį failą skaičius.

## **Fiziniai failų sistemos organizavimo būdai**

**Failų fizinė organizacija** aprašo failo išdėstymo išorinėje atmintyje, pvz., diske, taisykles.

* Failas susideda iš fizinių įrašų – blokų.
* Blokas – tai mažiausias duomenų vienetas, kuriuo išorinis įrenginys apsikeičia su pagrindine atmintine.
* Priskiriant disko blokus failams yra siekiama efektyviai išnaudoti disko erdvę.

Problema – paskirti failams vietą diske: Failai – tai blokų eilė.

* Disko erdvė turi būti išnaudota efektyviai
* Failus turi būti lengva išrinkti iš disko

**Failo talpinimo strategijos**:

* Nuoseklus failų išdėstymas diske
  + Failui priskiriama tiek nuosekliai einančių blokų, kiek reikalauja failo dydis.
  + Gali būti sunku nuspręsti kiek vietos skirti pradžioje.
  + Paprastai įgyvendinama, greita nuosekli prieiga.
* Susietas sąrašas.
  + Aplanke nurodomas failo pradžios blokas, o blokuose yra nuoroda į sekantį bloką.
  + Nėra fragmentacijos, paprasta įterpti/išmesti, nėra galimas tiesioginis kažkurio bloko išmetimas.
* Indeksų sąrašo formavimas.
  + Formuojama indeksų lentelė, rodanti kuriame fiziniame bloke yra patalpintas atskiras failo blokas.

## **Laisvos disko atminties valdymas**

* Sudaromas laisvų blokų sąrašas.
* Jei yra keli iš eilės einantys laisvi blokai, tai laisvų blokų sąrašas formuojamas sutraukiant į sąrašą laisvų blokų grupes.
* Atlaisvintas kuris nors blokas įtraukiamas į laisvų blokų sąrašą: jei jam iš dešinės ar kairės esantis blokas laisvas, padidinamas laisvų, nuosekliai išdėstytų blokų skaičius ir pakeičiama nuoroda.
* Laisvi blokai priskiriami failams, imant juos iš sąrašo pradžios, o atsilaisvinę blokai gali būti nukeliami į šio sąrašo pabaigą.
* Laisvų blokų sąrašas gali būti atžymimas naudojant dvejetainį žemėlapį, kuriame vienetais pažymimi užimti disko blokai, o nuliais – laisvi blokai. 

## **Skirtingose OS naudojamos failų sistemos: Windows failų sistema (FAT, FAT16, FAT32), Žurnalinės failų sistemos, jose taikomi principai.**

**FAT:**

* Informacija apie failus, jų vietą diske laiko FAT lentelėse.
* FAT lentelė formuojama kaip masyvas, o aplankuose yra nurodoma nuoroda į primą kiekvieno failo bloką.
* Tolimesni failo blokai susiejami nuorodomis.
* FAT lentelė sudaryta iš įrašų, kurie aprašo kiekvieną skaidinio blokų junginį.

Kiekviena FAT versija naudoja skirtingo ilgio įrašus.

Įrašo ilgis yra atspindimas ir FAT versijos vardu – FAT16 lentelė atveju naudojami 16 bitų įrašai, o FAT32 – 32 bitų.

FAT32 leidžia nurodyti didesnį blokų junginių skaičių nei FAT16, todėl FAT16 sistemose failo maksimalus dydis gali siekti 2GB, FAT32 – iki 4TB.

**Žurnalinės failų sistemos**

* Visi atliekami veiksmai fiksuojami žurnaluose.
* Šie įrašai paprastai laikomi atskirame failų sistemos skyriuje.
* Įrašas žurnale daromas prieš atliekant realų failų sistemos duomenų struktūros pakeitimą.
* Po to atliekami pakeitimai realiose duomenų struktūrose ir tai padarius padaromas įrašas apie veiksmų pabaigą.
* Į žurnalus gali būti rašomi dviejų tipų duomenys: visas duomenų įrašas arba metaduomenų įrašas.

## **Failų sistemos montavimas**

* Tai atskirame diske esančios failų sistemos įjungimas į egzistuojančią katalogų struktūrą.
* Montuojant failų sistemą, parenkamas įrenginys, kuris bus montuojamas, ir montavimo taškas. Paprastai montavimo taškas būna tuščias aplankas.
* Failų sistema patikrina ar montuojama failų sistema yra tinkama.
* Operacinė sistema padaro atitinkamą įrašą, kuriuo įsimenama, kad įrenginio failų sistema yra primontuota tam tikrame taške.
* Nuo šio momento bet kokia prieiga prie montavimo taško automatiškai peršoks prie primontuotos failų sistemos.