OS EGZAMINAS.

Pauliaus Gužo duoklė OS moduliui

I. Įvadas

1. Pagrindinės atmintinės valdymo sąvokos

Pagrindinės atminties valdymas – tai užduotis, kurią vykdo operacinė sistema kartu su technine įranga, siekdama, kad kuo daugiau procesų tilptų pagrindinėje atmintinėje.

OS turi fiksuoti, kurios pagrindinės atmintinės sritys yra užimtos ir kas jas užėmė, turi spręsti, kur įkelti naują procesą, kur ir kaip priskirti papildomas sritis procesams, kai jie to užprašo, o procesams pasibaigus fiksuoti jų turėtas pagrindinės atmintinės sritis kaip laisvas.

OS branduolys užima tam tikrą fiksuotą pagrindinės atmintinės dalį. Likusi atmintis naudojama procesams.

Pagrindinės atmintinės valdymui keliami tam tikri reikalavimai:

- Galimybė keisti proceso įkėlimo vietą atmintinėje;
- Procesui skirtos atmintinės srities apsauga nuo kitų procesų poveikio;
- Galimybė bendrai naudoti kurią nors atmintinės sritį, ja dalytis;
- Loginis atmintinės organizavimas;
- Fizinis organizavimas;

Reikalaujama, kad nesant specialaus leidimo, procesai negalėtų kreiptis į kitam procesui skirtą atmintinės sritį. Tai turi būti užtikrinama techninės įrangos priemonėmis.

Jei reikia, turi būti leidžiama keliems procesams pasiekti tam tikrą bendrai naudojamą pagrindinės atmintinės sritį. To gali prireikti tais atvejais, kai besikooperuojantys procesai turi prieiti prie tų pačių duomenų arba jais keistis. Dažnai keletas vartotojų naudojasi tos pačios programos paslaugomis vienu metu. Netikslinga kiekvienam vartotojui turėti savą šios programos kopiją – racionaliau lakyti vieną tos programos kopiją ir visiems vartotojams leisti naudotis ta pačia programa.

Reikalaujama, kad proceso įkėlimas nebūtų griežtai susiejamas su tam tikra atmintinės vieta. Kadangi OS gali laikinai iškelti procesus į antrinę atmintį arba naudoti suglaudinimo algoritmą, tai procesai vykdymo metu gali užimti skirtingas vietas pagrindinėje atmintinėje. Todėl proceso programos kode negali būti fiksuotų nuorodų į atmintinę. Ši problema sprendžiama naudojant loginius ir fizinius adresus.

Programos rašomos naudojant modulinį principą. Programos moduliai ir jų charakteristikos:

- Programos kodo moduliai (skirti vykdymui)
- Duomenų moduliai (skirti tik skaityti arba skaityti ir rašyti)
- Asmeniniam naudojimui skirti moduliai
- Viešam naudojimui skirti moduliai (likusieji)

Efektyviai tvarkant vartotojų procesus, OS kartu su technine įranga privalo palaikyti bazines modulių formas ir užtikrinti reikiamą apsaugą bei galimybę bendrai naudotis moduliais.

Antrinė atmintinė yra ilgalaikės atmintinės sritis, ji yra skirta programoms ir duomenims ilgai saugoti.

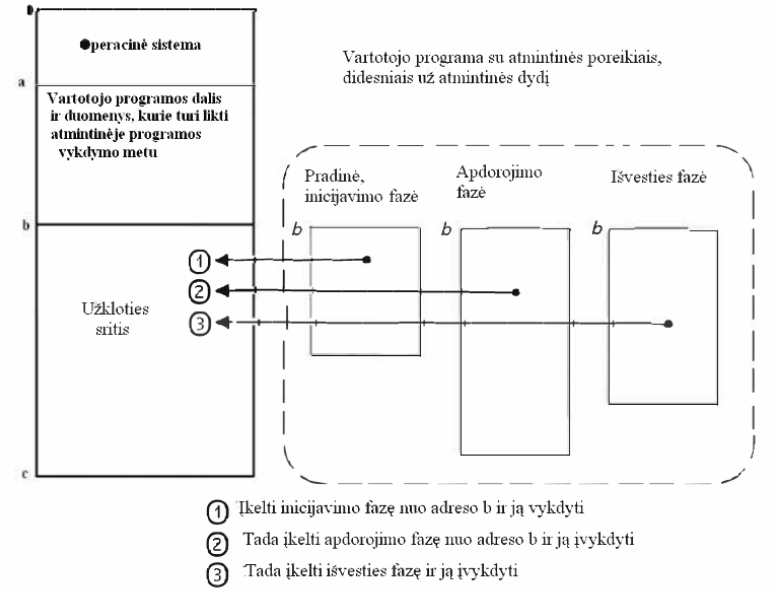
Pagrindinė atmintinė skirta tuo metu vykdomoms programoms ir jų naudojamiems duomenims saugoti.

Duomenų arba programų perkėlimas iš vienos rūšies atmintinės į kitą yra OS uždavinys, susijęs su atminties valdymu.

2. Atmintinės valdymo schemos (atmintinės paskirstymo strategijos). Kas tai, ką sprendžia, kokios būna

Procesui atmintinėje gali būti skiriama ištisinė sritis arba neištisinė sritis.

Ištisinė sritis. Procesas egzistuoja kaip vientisas, nuoseklių, nepertraukiamų adresų erdvėje esantis blokas. Tai labai paprastas atminties dalijimo būdas, tačiau sunku surasti tinkamo dydžio laisvą bloką ir susiduriama su netinkamu atminties panaudojimu.



8.3 pav. Užkloties mechanizmo pavyzdys

Ištisinė sritis procesams buvo priskiriama ankstyvoje OS raidos stadijoje. Taikant ši metodą, visa OS neužimta sritis buvo skiriama vartotojo poreikiams. Naudojant ribinį registrą, kuris rodė vartotojui skirtos srities pradžia, buvo užtikrinama OS apsauga. Problemos atsirado, pradėjus kurti programas, kurių kodas nebetilpo į pagrindinę atmintinę. Šiai problemai spręsti buvo naudojamas užkloties mechanizmas. Jo esmė tokia: programa buvo sudalijama į tam tikrus loginius skyrius – modulius. Į atmintinę buvo įkeliamas tik tuo metu aktyvus modulis. Jei, vykdant programą, prireikdavo kurio nors kito modulio, kuris nebuvo įkeltas, reikėdavo įkelti tą modulį į kurio nors kito atmintinėje buvusio programos ar duomenų modulio vietą. Užkloties mechanizmo naudojimo trūkumai buvo tie, jog buvo sunku taip organizuoti užklotis, kad pagrindinė atmintinė būtų naudojama efektyviai. Problemų kėlė ir vykdomos programos modifikacijos, kurias reikėjo atlikti.

Neištisinė sritis. Proceso duomenys skaidomi į tam tikro dydžio gabalus (puslapius, segmentus), kurie įkeliami į atskiras atmintinės vietas. Šiuo metodu lengviau rasti atmintinėje procesui įkelti tinkamą vietą, padidinti procesų atmintinėje skaičių, tačiau jį sunkiau taikyti.

3. Paprastas atminties valdymas (fiksuoti, dinamiškai formuojami skyriai. Procesų įkėlimo į šiuos skyrius algoritmai. Bičiulių sistema)

Pradžioje bus kalbama apie paprastus atminties valdymo būdus, kai vykdomo proceso visas programos kodas ir duomenys bus pagrindinėje atmintinėje.

Efektyvus pagrindinės atminties valdymas susijęs su tam tikros apimties pagrindinės atmintinės dalies skyrimu kiekvienam vykdomam procesui. Siekiant intensyviau naudoti procesorių, pagrindinėje atmintinėje stengiamasi laikyti kuo daugiau procesų.

Optimaliai skirstant pagrindinę atmintinę, sprendžiami tokie uždaviniai:

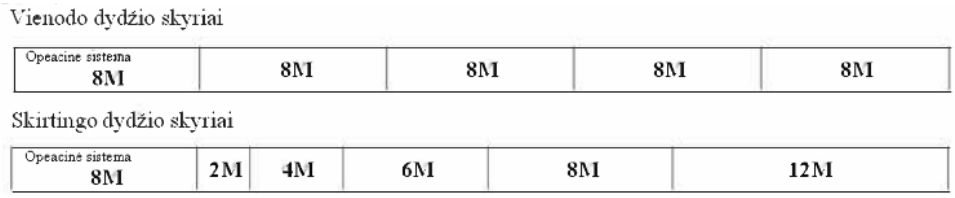
- Kurį iš procesų palikti pagrindinėje atmintinėje, kai jos pritrūksta naujiems procesams?
- Kokio dydžio atmintinės sritį paskirti kiekvienam procesui?
- Į kurią pagrindinės atmintinės vietą įkelti procesą ar jo dalį?
- Kurią proceso dalį įkelti ir kada – ar tada, kai prireiks, ar iš anksto?

Skirstant pagrindinę atmintinę keliems procesams, naudojami šie paprasti algoritmai:

- Atmintinės skirstymo fiksuoti dydžio skyriai
- Dinaminis skyrių formavimas atmintinėje
- Skirstymas puslapiais
- Skirstymas segmentais

3.1 Fiksuoti skyriai

Pagrindinė atmintinė sudalijama į keletą nepersidengiančių skyrių (dalių). Šios dalys gali būti tiek vienodo, tiek skirtingo fiksuoti dydžio. Skyrių skaičius sistemoje taip pat yra fiksuotas.



8.4 pav. Fiksuoti skyriai

Taip sudalinus pagrindinę atmintinę, bet kuris procesas, kurio dydis lygus skyriaus dydžiui arba mažesnis, gali būti įkeliamas į šį skyrių. Taigi kiekvienas aktyvus procesas gauna po tam tikrą skyrių ir procesorius gali greitai persijungti nuo vieno proceso prie kito. Procesų atminties atskirčiai užtikrinti naudojami keli ribiniai registrai. Jei visi skyriai yra užimti, OS gali iškelti (swap) procesą iš jo užimamo skyriaus į diską.

Kai skyriai yra fiksuoto dydžio, pagrindinė atmintinė naudojama neefektyviai, todėl yra susiduriama su vidinės fragmentacijos problema, nes, kad ir kokia maža būtų programa, jai skiriamas visas skyrius ir jame gali būti daug nenaudojamos vietos. Skirtingo fiksuoto dydžio skyriai kiek sušvelnina šią problemą, tačiau jos nepanaikina.

3.2 Procesų įkėlimo į fiksuotus skyrius algoritmas

Vienodo dydžio skyriai. Kad procesą būtų galima įkelti į pagrindinę atmintinę, reikėjo sulaukti, kol joje atsiras laisvas skyrius. Atsiradus tokiam skyriui, procesas gali būti į jį įkeltas. Kadangi visi skyriai yra vienodo dydžio, tai visai nesvarbu, į kurį skyrių procesas bus įkeltas. Jei visi skyriai yra užimti užblokuotų procesų, tai galima parinkti vieną iš jų, laikinai iškelti į antrinę atmintinę (swap) ir taip atlaisvinti vietą kitam procesui. Kol bus įkelti į atmintinę, procesai laukia bendroje eilėje.

Nevienodo dydžio skyriai. Kai skyriai yra skirtingo fiksuoti dydžio gali būti taikomi du algoritmai:

Gali būti sudaroma daug eilių procesams, kurie laukia įkėlimo. Kiekviena eilė yra susieta su atskiru skyriumi; procesai pagal savo dydį yra įtraukiami į atitinkamą eilę prie skyrių. Taip sumažinama vidinė fragmentacija, tačiau daugelis skyrių, o kartu ir eilių gali būti tušti, kol kitos eilės gali būti gana didelės.

Gali būti sudaroma viena bendra eilė, kurioje stovėdami procesai laukia, kol bus įkelti į pagrindinę atmintinę. Kai ateina laikas procesą paimti iš eilės ir įkelti į atmintinę, parenkamas mažiausias laisvas skyrius, į kurį procesas telpa. Taip padidinamas multiprogramavimo lygis, tačiau pasireiškia didesnė vidinė fragmentacija.

3.3 Dinaminis skyrių formavimas

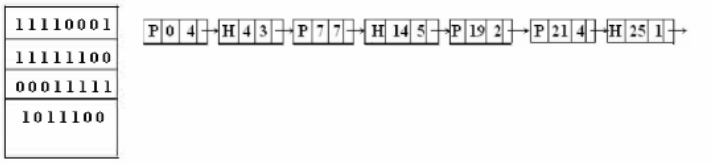
Taikant dinaminį skyrių formavimo principą tiek skyrių skaičius, tiek jų dydis yra kintami. Kiekvienam procesui, įkeliant į pagrindinė atmintinę yra sukuriamas ir suformuojamas tokio dydžio skyrius, kokio jam reikia. Tačiau kai procesas baigiasi į jo vietą galime įkelti arba mažesnės talpos procesą arba tokį patį. Taip pagrindinė atmintinėje atsiranda „skylės“ – laisvos sritys, kurios lieka didesnio skyriaus vietoje sudarius mažesnį skyrių. Tai vadinama išorine fragmentacija.

Susidaro kartais situacijų, kai, nors ir yra laisvos atminties, tačiau ji sudaryta iš tokių mažų sričių, į kurias nauji procesai jau nebetelpa. Tada naudojamas suglaudinimo algoritmas, kuris perstumia procesus taip, kad tarp jų nebelieka skylių, o laisvas sritis sujungia į vieną bendrą sritį.

Dinamiškai skirstant atmintinę, OS turi žinoti, kurios sritys šioje atmintinėje yra užimtos ir kurios yra laisvos. Tokio tipo informaciją OS gali saugoti dvejetainiuose žemėlapiuose (bitmap) arba surištuose sąrašuose.

Dvejetainiuose žemėlapiuose kiekviena užimta skyriaus sritis pažymima vienetu, o laisva nuliu. Žemėlapio apimtis priklauso nuo srities dydžio. Žemėlapiui laikyti reikia daug atminties, tačiau paprasta naudotis – tereikia rasti atitinkamą skaičių vienas po kito einančių nulių.

Surištas sąrašas rodo atmintinę suskirstytą į procesus ir skyles. Kiekviename sąrašo elemente pažymima proceso (P) arba skylės (H) sritis, jos pradžios adresas, ilgis ir nuoroda į kitą sąrašo elementą.



8.7 pav. Pagrindinė atmintinė ir jos užimtumo atvaizdavimas

3.4 Procesų įkėlimo į dinaminius skyrius algoritmas

Kai procesų ir skylių sąrašas surūšiuotas pagal adresus, tai juo galima pasinaudoti skiriant laisvą atmintinės sritį naujai įkeliamiems (sukurtiems) procesams.

Tinkamas įkėlimo algoritmas padeda nuspręsti, kurį laisvą atminties skyrių („skylę“) paskirti į pagrindinę atmintinę įkeliamam procesui, atsižvelgiant į tai kiek atmintinės jam reikia. Procesą siekiama įkelti taip, kad kuo rečiau reikėtų glaudinti duomenis, nes tam sugaištama daug laiko.

Galimi ir naudojami šie algoritmai:

- Tinkamiausio skyriaus paieškos
- Pirmo tinkamo skyriaus paieškos
- Kito tinkamo skyriaus paieškos

Taikant **tinkamiausio skyriaus paieškos** algoritmą, randamas mažiausias laisvas atminties blokas (skylė), į kurį telpa įkeliamas procesas. Taikant šį algoritmą lieka mažiausiai neišnaudotos erdvės, tačiau daug laiko sugaištama tinkamos skylės paieškai, nes reikia peržiūrėti visą sąrašą. Lieka daug mažų nenaudojamų skylių.

Taikant **pirmo tinkamo skyriaus paieškos** algoritmą surandamas pirmas laisvas (nuo atminties pradžios) tinkamo dydžio blokas ir į jį įkeliamas procesas. Rasta skylė sudalijama į dvi dalis, į vieną įkeliamas procesas, formatuojamas naujas dinaminis skyrius, o kita dalis lieka nepanaudota. Šis algoritmas gana paprastas, nereikia daug paieškos veiksmų.

Kito tinkamo skyriaus paieškos algoritmas veikia taip pat kaip ir pirmo tinkamo skyriaus paieškos algoritmas, tik šiuo atveju yra fiksuojama vieta, kurioje rasta tinkamo dydžio skylė. Kitą kartą paieška prasideda nuo tos vietos kur buvo sustota. Taigi, taikant šį algoritmą, randama pirma tinkamo dydžio skylė, einanti po paskutinio proceso įkėlimo vietos.

Kartais yra naudojamas algoritmas, kuris procesui įkelti ieško didžiausios skylės. Įkėlus procesą į tokį bloką, dar lieka gana daug neišnaudotos erdvės. Tikimasi, kad į šią neišnaudotą erdvę vėliau bus galima įkelti kitą procesą.

Visi šitie procesai gali būti paspartinti, jei informacija apie procesų užimtas vietas ir informacija apie skyles laikoma atskiruose sąrašuose. Tokiu atveju reikia peržiūrėti tik skylių sąrašą. Tačiau tada, procesui priskyrus laisvą skylę, reikia atitinkamą elementą išimti iš skylių sąrašo ir perkelti į procesų sąrašą. Taip pat naudojant atskirus sąrašus, skylių sąrašą galima surikiuoti ne pagal adresus, o pagal skylės dydį (toks sąrašas atitiktų tinkamiausio skyriaus paieškos algoritmo poreikius).

3.5 Bičiulių sistema

Bičiulių sistema tai algoritmas, kuriuo bandoma išvengti tiek fiksuotų, tiek dinaminių skyrių problemų.

Pradžioje visa atmintinė yra laisva, taigi pradedama turint laisva 2^U dydžio bloką. Procesams įkelti skiriami blokai, kurių dydis 2^K. Mažiausias procesui skiriamas blokas gali būti 2^L.

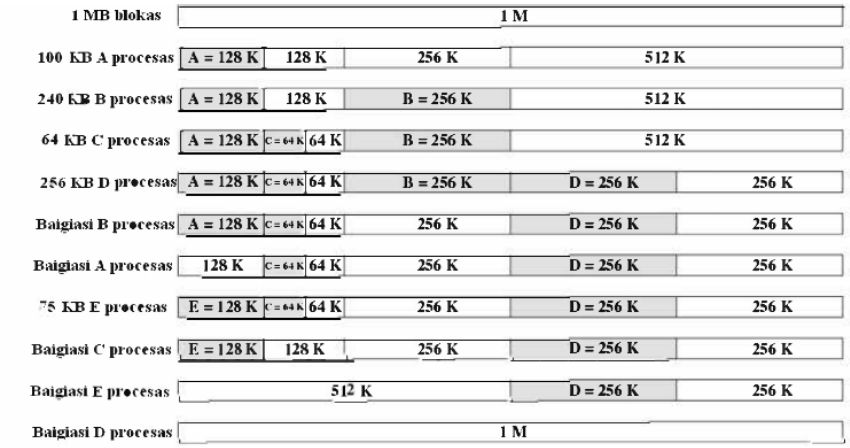
Gaunama L <= K <= U.

Tarkime, norime įkelti S dydžio procesą. Jei $2^{(U-1)} < S <= 2^U$, tai yra skiriamas visas blokas 2^U. Priešingu atveju blokas dalijamas į dvi vienodo dydžio $2^{(U-1)}$ dalis (bičiulius). Jei $2^{(U-1)} < S <= 2^U$, tai procesui skiriama viena iš dalių (vienas bičiulis), o jei ne, tai viena dalis vėl dalijama į dvi dalis. Šis procesas kartojamas tol, kol ši sąlyga tampa galiojanti, t.y. gaunamas mažiausias blokas, kuris yra lygus S arba didesnis. Jis ir skiriamas procesui įkelti.

Jei du gretimi vienodo dydžio atminties blokai (bičiuliai) tampa laisvi (procesams pasibaigus), jie sujungiami.

i -asis skylių sąrašas apima skyles, kurių dydis yra 2^i. Kai tik bičiulių pora atsiranda i -ajame sąraše, jie perkeliami į vieną bendrą skylę (i+1) sąraše.

Norint įkelti t dydžio procesą, kuriam galioja 2(i-1) < t <= 2i, patikrinamas i -asis sąrašas (į jame esančią skyle procesas būtų įdedamas su mažiausia fragmentacija). Jame surasta skylė skiriama procesui. Jei šis sąrašas yra tuščias, tikrinamas (i+1) sąrašas. Jame surasta skylė sudalijama į du bičiulius: viena dalis bus priskirta procesui, o kita bus įtraukta į i -ąjį sąrašą.



8.9 pav. Bičiulių sistemos taikymo pavyzdys

3.6 Fragmentacijos sąvoka. Išorinė, vidinė fragmentacija.

Fragmentacija - daugelio negretimų laisvų atminties sričių atsiradimas.

- **Išorinė fragmentacija –** kai didesnio skyriaus vietoje sudarius mažesnį skyrių lieka neišnaudotos atminties sričių
- **Vidinė fragmentacija –** kai procesai nepilnai išnaudoja jiems priskirtą atminties sritį.

II. Virtualioji atmintinė

Virtualiosios atmintinės sąvoka leido išspręsti problemą, susijusią su ribotu atmintinės dydžiu. Pagrindinė idėja yra ta, kad bendras programos kodo, jos duomenų ir dėklo dydis gali viršyti fizinės atmintinės tam skirtą dydį. Procesai vykdomi (pseudolygiagrečiai), jei pagrindinėje atmintinėje yra tik tos procesų dalys, kurios naudojamos tuo metu, o likusios dalys laikomos diske ir įkeliamos į pagrindinę atmintinę tik kilus poreikiui. Kai kuriam nors procesui vykdyti pritrūks į pagrindinę atmintinę neįkeltos proceso dalies, jis užsiblokuos, o procesorius tuo metu galės vykdyti kitą procesą.

Virtualioji atmintinė – tai tam tikras loginis lygmuo tarp taikomųjų programų atmintinės poreikių ir realios atmintinės.

Virtualiosios atmintinės sąvoka padeda išspręsti keletą problemų:

- Leidžia vienu metu vykdyti keletą procesų
- Leidžia vykdyti procesus, kuriems reikia didesnės už esamą pagrindinės atminties
- Procesai gali vykdyti programas, kurių kodas tik iš dalies yra įkeltas į pagrindinę atmintį
- Kiekvienam procesui leidžiama prieiga tik prie tam tikro pagrindinės atmintinės poaibio
- Procesams leidžiama dalytis viena bibliotekos ar programos kopija, įkelta į pagrindinę atmintinę
- Programos gali būti kilnojamos t.y jos gali būti įkeliamos į bet kurią fizinės atmintinės vietą
- Programuotojai gali rašyti programos kodą nesirūpindami dėl fizinės atmintinės struktūros, jos dydžio.

Virtualioji atmintinė apibūdinama virtualia adresų erdve. Naudojami dviejų tipu adresai:

- Virtualieji, kuriuos naudoja procesai
- Fiziniai, rodantys realias objektų (komandų, duomenų) vietas pagrindinėje atmintinėje

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

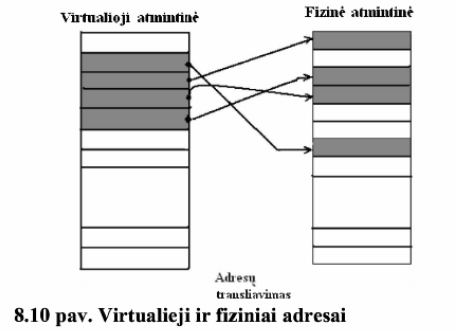
Virtualusis adresas – nuoroda į objekto vietą, kuria operuoja procesas, nepriklausanti nuo to, į kurią atmintinės vietą jis yra įkeltas, kokia adresų forma naudojama kompiuteryje.

Fizinis adresas (absoliutinis adresas) – nuoroda į objekto fizinę vietą pagrindinėje atmintinėje.

MMU (Memory Management Unit – atminties valdymo įrenginys) – tai kompiuterio techninės įrangos komponentas, atsakingas už atminties valdymą. Jis vykdo virtualiųjų adresų transliavimo į fizinius adresus, atminties apsaugos ir kitas funkcijas.

Kadangi proceso virtualiųjų adresų erdvė paprastai yra didesnė nei procesui skirta reali fizinė adresų erdvė pagrindinėje atmintinėje, tai OS pagrindinė atmintinėje gali laikyti tik dalį proceso, o kita dalis būna antrinėje atmintinėje.

OS yra priversta fiksuoti, kurie proceso blokai yra įkelti į pagrindinę atmintinę, kurioje vietoje jie yra ir kurių procesų blokų pagrindinėje atmintinėje nėra. Jei virtualioji adresų erdvė yra ištisinė, nuosekli, tai realūs adresai gali būti ir nenuoseklūs.



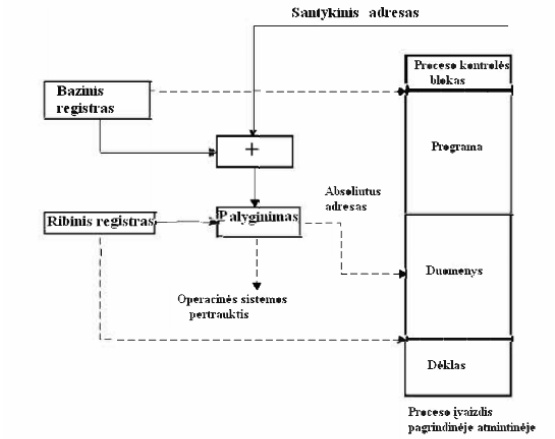
8.10 pav. Virtualieji ir fiziniai adresai

Taikant virtualiosios atmintinės idėjas, atskiros proceso dalys gali būti ne tik įkeliamos, bet ir iškeliamos iš pagrindinės atminties, bei pakartotinai įkeliamos, jei jų vėl prisireikia. Pakartotinai įkeliama kuri nors proceso dalis nebūtinai pateks į prieš ta buvusią vietą.

Santykinis adresas yra loginio adreso pavyzdys, kai adresas yra išreiškiamas kurio nors žinomo programos taško (pavyzdžiui, pradžios) atžvilgiu.

Santykiniai adresai yra dažniausia loginio adreso forma, naudojama modulinėse programose. Moduliai įkeliami į pagrindinę atmintinę su santykiniais adresais. Fiziniai adresai nustatomi vykdant komandą su santykiniais adresais.

Kai procesas yra perjungiamas į vykdymo būseną, į procesoriaus bazinį registrą yra įrašomas proceso pradžios fizinis adresas. Į ribinį registrą įkeliamas procesui skirtos srities galinis adresas. Programos kode aptikus santykinį adresą, jo reikšmė yra sudedama su bazinio registro turiniu ir taip gaunamas fizinis adresas. Prieš kreipiantis šiuo adresu, jo reikšmė yra sulyginama su ribinio adreso reikšme. Taip užtikrinama atmintinės apsauga – kiekvienas procesas gali kreiptis tik į adresus iš šiam procesui skirtos adresų erdvės.



8.11 pav. Santykinio adreso transliavimas į fizinį adresą

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

Puslapiavimo idėja:

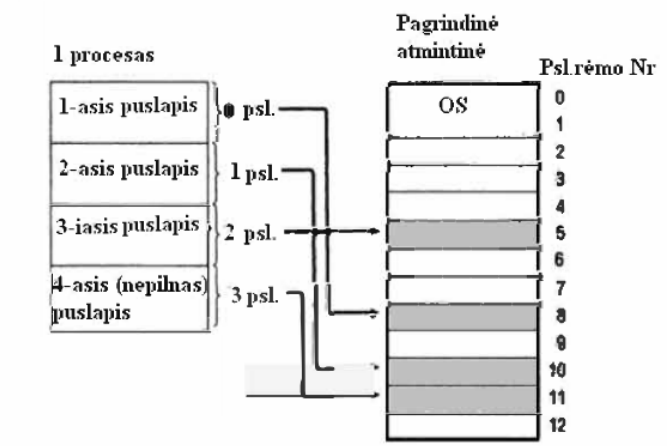
- Kiekvienas procesas yra suskaidomas į vienodo dydžio blokus, vadinamus puslapiais.
- Pagrindinė atmintinė taip pat yra sudalijama į fiksuoto puslapio dydžio blokus, vadinamus rėmais.

Kadangi puslapio (rėmo) dydis yra gana nedidelis, tai yra sumažinama vidinė fragmentacija.

Procesas nebūtinai užima ištisinę adresų erdvę pagrindinėje atmintinėje, proceso puslapiai įkeliami į tuo metu esančius laisvus rėmus. Nebelieka išorinės fragmentacijos, bet kurį laisvą rėmą gali užimti bet kurio proceso bet kuris puslapis.

2.1. Proceso įkėlimas į pagrindinę atmintinę

Iš pradžių buvo įkeliamas visas procesas (visi jos puslapiai) į pagrindinę atmitinę. Atmintinėje buvo laikomi keli procesai. Tokiu atveju pereinant nuo vieno proceso vykdymo prie kito nereikėdavo laukti, kol bus įkeltas procesas. Vėliau pradėta laikyti pagrindinėje atmintinėje tik kai kuriuos proceso puslapius. OS gali įkelti naujus puslapius prireikus poreikiui arba iš anksto. Jei programa kreipiasi į puslapį, kurio nėra pagrindinėje atmintinėje, procesas pertraukiamas, kad būtų įkeltas trūkstamas puslapis. Proceso naujo puslapio įkėlimas susijęs su tam tikrais OS bei techninės įrangos atliekamais veiksmais.



8.12 pav. Proceso puslapių įkėlimas į pagrindinę atmintinę

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

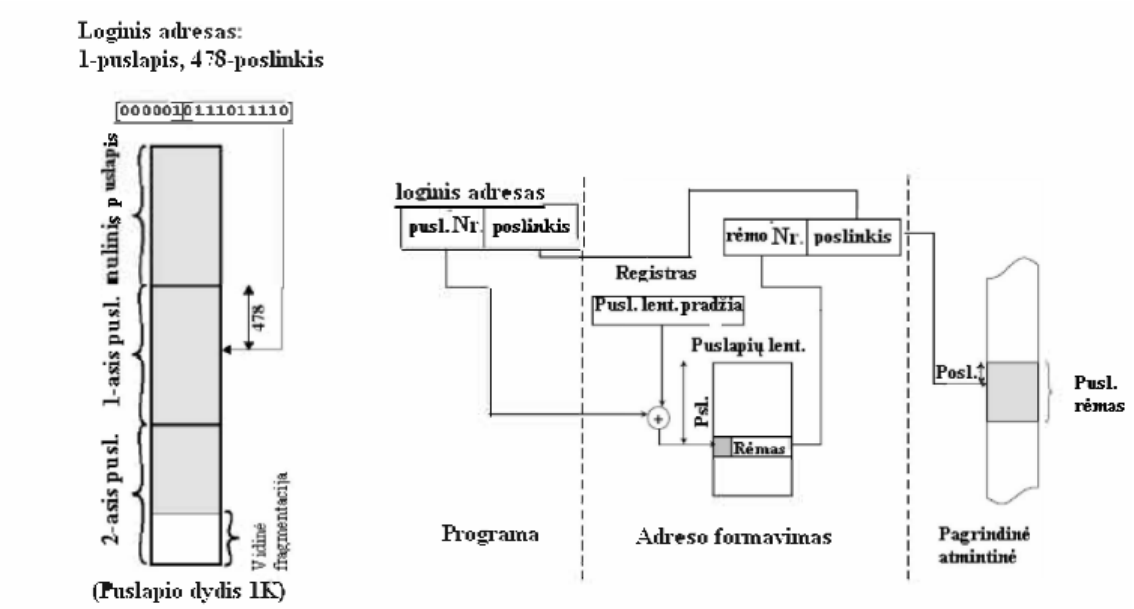
- Nustatomas proceso rezidentinių puslapių skaičius
- Surandama pakankamai laisvų rėmų pagrindinėje atmintinėje
- Į juos įkeliami proceso rezidentiniai puslapiai

3. Proceso adreso transliavimas

Proceso puslapių išdėstymas, laisvų atmintinės rėmų sąrašas registruojami proceso puslapių lentelėje. OS pagr. atmintinėje laiko kiekvieno aktyvaus proceso puslapių lentelę. Kiekvienas puslapių lentelės įrašas – tai nuoroda į fizinės atmintinės rėmo numerį.

Kiekvienoje programoje į duomenis ar komandas kreipiamasi pagal loginius adresus, kuriuos sudaro puslapio numeris p ir poslinkis tame puslapyje d. Kai programoje yra aptinkamas loginis adresas, išreikštas puslapio numeriu ir poslinkiu (p,d), tai kreipiamasi į puslapių lentelę (viename iš registrų saugomas tuo metu vykdomo proceso puslapių lentelės pradžios adresas) ir nustatomas atitinkamo rėmo numeris f, kuris kartu su poslinkio reikšme d ir nusakys realų adresą.

Aukščiausieji adreso bitai naudojami kaip puslapių lentelės indeksas, jie nurodo, į kurį puslapį yra kreipiamasi. Žemiausieji – rodo poslinkį šiame puslapyje.



8.14 pav. Loginio adreso transliavimas į fizinį adresą

Atminties adresas yra rinkinys (f, o):

- f – rėmo numeris
- o – poslinkis nuo rėmo pradžios

Fizinis adresas gaunamas – max poslinkis * rėmo numeris + poslinkis

Virtualaus puslapio adresas yra rinkinys (p, o):

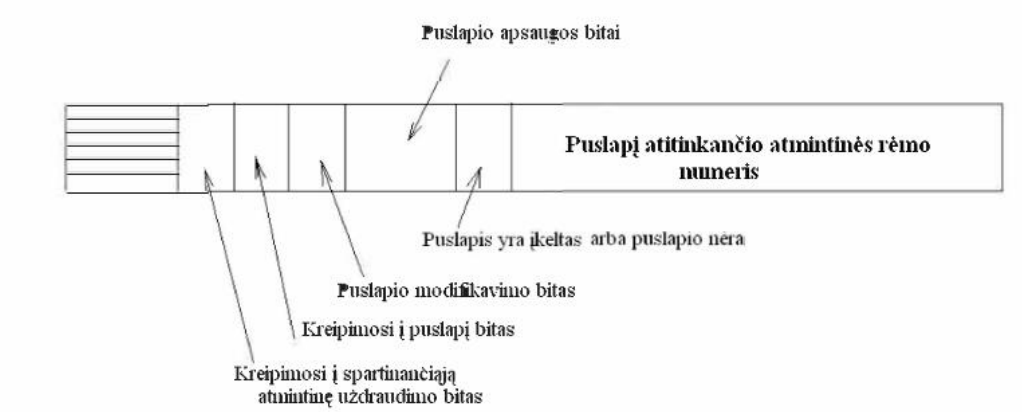
- p – psl numeris
- o – poslinkis nuo puslapio pradžios

Virtualus adresas gaunamas - max poslinkis * puslapio numeris + poslinkis

3.1. Puslapių lentelė ir jos struktūra

Puslapių lentelės įrašą sudaro (nuo dešinės į kairę):

1. Virtualųjį proceso puslapį atitinkančio atmintinės rėmo numeris
2. Puslapio galiojimo bitas (1-įkeltas, 0-neįkeltas)
3. Puslapio apsaugos bitai (0-tik skaityti, 1-skaityti ir rašyti)
4. Puslapio modifikavimo bitas
5. Kreipimosi į puslapį bitas
6. Kreipimosi į spart. Atmintinę uždraudimo bitas (naudojama lentelėms, kai saugoma nuoroda į įrenginių registrus, o ne į atmintinę)

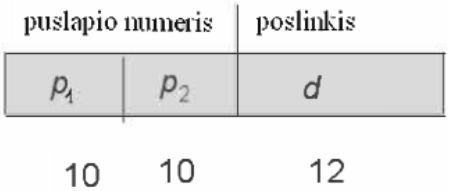


8.15 pav. Puslapių lentelės įrašo struktūra

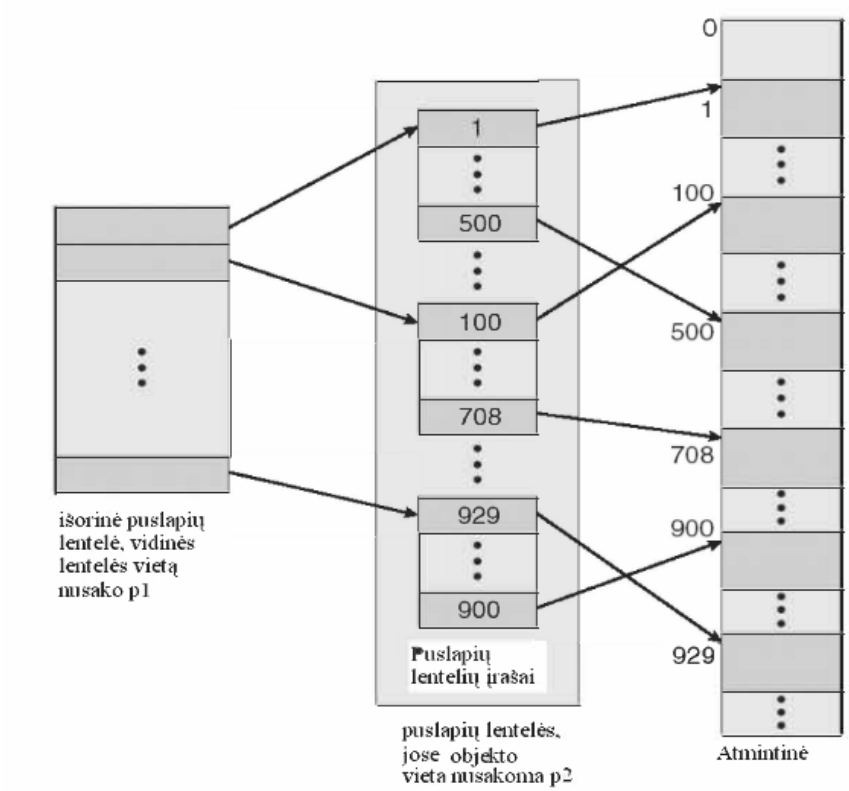
Proceso puslapių lentelės gali būti laikomos procesoriaus registruose arba pagrindinėje atmintinėje. Laikant registruose sutrumpinamas adreso transliavimo laikas, tačiau registruose nėra pakankamai atminties saugoti puslapių lentelėms. Dažnesnis variantas kai visa proceso puslapių lentelė yra laikoma pagr. atmintinėje ir vienas iš procesoriaus registrų rodo į puslapių lentelės pradžią. Perjungiant procesorių vykdyti kita procesą reikia pakeisti to registro turinį naujo proceso puslapių lentele. Tačiau kiekvieną kartą į atmintinę reikia kreiptis dviem kreipiniais.

Proceso puslapių lentelės gali būti per didelės apimties net saugojimui pagr. atmintinėje. Tokiu atveju pati puslapių lentelė gali būti saugoma virtualiojoje atmintinėje (skirstoma puslapiais ir tik tuo metu reikalinga dalis jos laikoma pagr. atmintinėje). Ši problema sprendžiama naudojant hierarchines, TLB, maišos ar invertuotas puslapių lenteles.

Naudojant **hierarchines lenteles**, visa loginė adresų erdvė sudalinama į kelių lygių puslapių lenteles. Loginis 32 b adresas dalinamas į išorinės lentelės puslapio numerį p1 (10 b), vidinės lentelės puslapio numerį p2 (10 b), vidinės lentelės poslinkį d (12 b), kuris nurodo tikslų fizinį adresą.



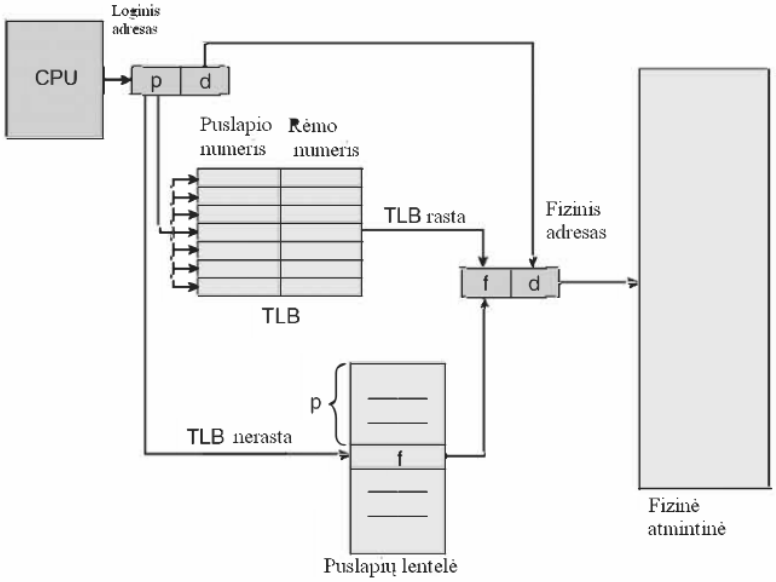
8.16 pav. Hierarchinis adresas



8.17 pav. Dviejų lygių puslapių lentelių schema

Naudojant **TLB lenteles**, paspartinamas adreso transliavimas, jos saugo dalį puslapių lentelės informacijos. Saugoma informacija apie tuos puslapius, į kuriuos kreipiamasi dažniausiai. TLB lentelės laikomos spartinančioje atmintinėje ir jas naudoja MMU.

TLB lentelės saugo virtualiųjų puslapių numerių bei jų fizinių adresų tarpusavio atitikimus. Jei reikalaujamas adresas randamas TLB lentelėje, jis transliuojamas greitai. Jei adreso TLB lentelėje nėra, puslapio adreso ieškoma proceso puslapių lentelėje. Nustačius fizinį adresą, atitinkamas įrašas įtraukiamas į TLB lentelę (tai gali būti daroma pakeičiant vieną iš buvusių įrašų). Jei puslapio lentelėje nurodytam puslapiui galiojimo bitas nenustatytas, tai generuojama puslapio trūkumo pertrauktis, kurią apdoroja OS. Atitinkamas virtualusis puslapis įkeliamas į pagr. atmintinę ir puslapių lentelės įrašas atnaujinamas.

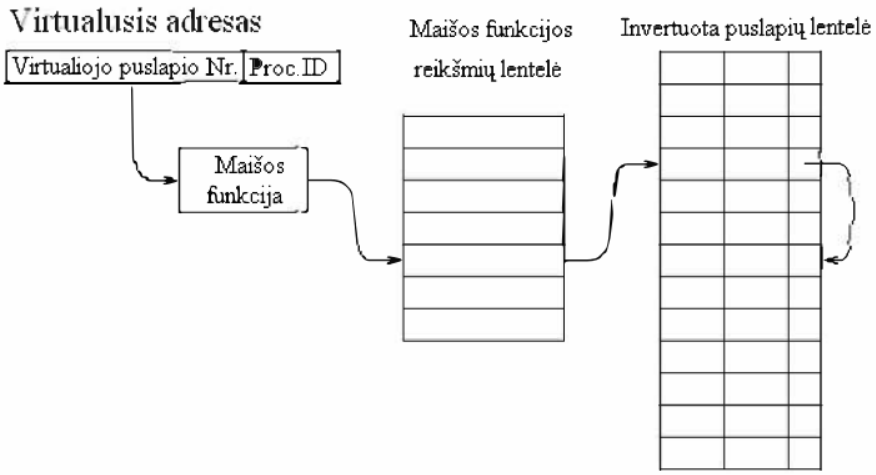


8.18 pav. Adreso transliavimas naudojant TLB įrašus

Naudojant **invertuotas lenteles**, smarkiai sumažinamas puslapių lentelėms saugoti reikalingas atmintinės dydis. Užuot laikius sistemoje kiekvieną procesą atitinkančias puslapių lenteles, kuriose yra saugomas jo virtualiųjų puslapių ir juos atitinkančių fizinių rėmų tarpusavio atitikimas, sistemoje yra saugoma viena invertuotų puslapių lentelė, surikiuota fizinių rėmų didėjimo tvarka. Šiuo atveju puslapių lentelė yra sudaryta iš įėjimų, kurie saugo informaciją apie tai, kurio proceso (proceso ID) ir kuris virtualusis puslapis yra šiame rėme. Taigi tokios lentelės dydis priklauso tik nuo pagrindinės atmintinės dydžio.

Linijinė invertuota puslapių lentelė yra paprasčiausia invertuotų lentelių forma. Kadangi visi procesai turi dalyti šia lentele, tai jos įėjimuose saugomi procesų ID bei virtualiųjų puslapių numeriai. Fizinių rėmų numeriai nėra saugomi, jie tiesiog atitinka šios lentelės indekso reikšmes. Nors šios lentelės ir sutaupo atminties, tačiau virtualiojo adreso transliavimas į fizinį adresą tampa sudėtingesnis. Transliuojant virtualųjį adresą, virtualiojo puslapio numeris bei proceso ID reikšmė turi būti nuosekliai lyginama su visais šios lentelės įėjimais. Radus sutapimą adresas transliuojamas, o jei sutapimo nerandama, tai reiškia, kad virtualiojo puslapio trūksta, ir dėl to kyla pertrauktis, susijusi su puslapio trūkumu. Tokia paieška atliekama greitai tik esant nedidelei lentelei.

Tam, kad adresą būtų galima greitai transliuoti reikalingas efektyvus paieškos invertuotoje lentelėje algoritmas. Todėl naudojamos maišos funkcijos ir maišos (hash) lentelės.



8.19 pav. Invertuotos puslapių lentelės

Iš maišos funkcijos reikšmių lentelės, kurioje saugomos invertuotos puslapių lentelės indeksų reikšmės, surandamas įėjimas invertuotoje lentelėje, t. y. randamas atmintinės rėmo numeris. Kadangi invertuotoje lentelėje yra saugomas rėmo numerio bei proceso ID ir jo virtualiojo puslapio numerio tarpusavio atitikimas, tai tikrinama, ar surastas virtualusis puslapis tikrai yra to proceso ieškomas puslapis. Šis patikrinimas reikalingas todėl, kad tą pačią maišos reikšmę galima gauti naudojant kelias skirtingas procesų ID ir virtualiojo puslapio numerio kombinacijas. Kadangi esant daugiau nei vienai maišos funkcijos įėjimo reikšmei gali būti gaunama ta pati išėjimo (maišos) reikšmė, tai invertuotoje lentelėje saugoma grandinėlė įrašų, kuriuos reikia peržiūrėti ieškant tinkamo puslapio. Grandinėlės paprastai nėra ilgos – vienas ar du įėjimai (8.19 pav.). Invertuotoje puslapių lentelėje kiekvienas įrašas atitinka vieną puslapio rėmą, o ne virtualųjį puslapį. Taigi fiksuotos atmintinės dydžio reikia invertuotai puslapių lentelei saugoti. Jis nepriklauso nuo aptarnaujamų procesų ar jų virtualiųjų puslapių skaičiaus.

Suradus sutapimą invertuotoje lentelėje, randamas dydžio fizinis adresas, t. y. transliuojamas adresas. Jei sutapimo nerandama, tai reiškia, kad atitinkamas virtualusis proceso puslapis nėra įkeltas į pagrindinę atmintinę, todėl atitinkamas procesas pertraukiamas dėl puslapio trūkumo ir atitinkamas virtualusis puslapis įkeliamas į pagrindinę atmintinę.

Skirstymas puslapiais pranašumai:

- Paprasta skirstyti pagrindinę atmintinę
- Galima užimti bet kurį laisvą atmintinės rėmą
- Procesai gali bendrai naudotis kai kuriais puslapiais

Skirstymas puslapiais problemos:

- Puslapių lentelės yra didelės apimties
- Sunku iš anksto skirti sritį (ji gali didėti)

Tinkamo puslapio dydžio parinkimas. Parinkus mažą puslapio dydį, sumažėja vidinė fragmentacija, tačiau padidėja puslapių lentelė, nes padidėja bendras puslapių skaičius. Padidinus puslapį, sumažėja puslapių lentelė, patogesni darosi informacijos mainai su diskiniu įrenginiu, reikia mažiau puslapių mainų. Paprastai naudojami 4 K, 8 K, ar 16 K dydžio puslapiai.

Tinkamo kiekio proceso puslapių laikymas pagrindinėje atmintinėje. Jei į pagr. atmintinę bus įkelta nedaug puslapių, reikės dažnai juos keisti. Jei kiekvienam procesui skiriama daug puslapių, tai dėl riboto atmintinės dydžio gali būti ribojamas multiprogramavimo lygis. Minimalus puslapių skaičius gali būti nusakomas architektūra (į kiek puslapių gali būti nuorodos vienoje komandoje). Maksimalų puslapių skaičių gali nusakyti vartotojui skirtos atmintinės rėmų skaičius. Parenkant tinkamą skaičių gali būti atsižvelgiama į proceso dydį.

3.2. Bendrai naudojamos atmintie sritys

Dažnai procesai tuo pačiu metu naudojasi tuo pačiu kurių nors programų vykdomuoju kodu: tekstų redaktoriais, kompiliatoriais. Šio bendro kodo procesai neturėtų keisti, jis tūrėtų būti laikomas srityje, kuri būtų žinoma visiems procesams. Toks kodas gali būti programos kodo puslapiai, o kiekvieno proceso duomenų puslapiai turėtų būti atskiri. OS turi įvertinti puslapius kaip bendrai naudojamus arba kaip privačiai naudojamus. Tokiu atveju tik kiekvieno proceso privatusis kodas ir duomenys turėtų būti laikomi skirtingose atmintinės srityse.

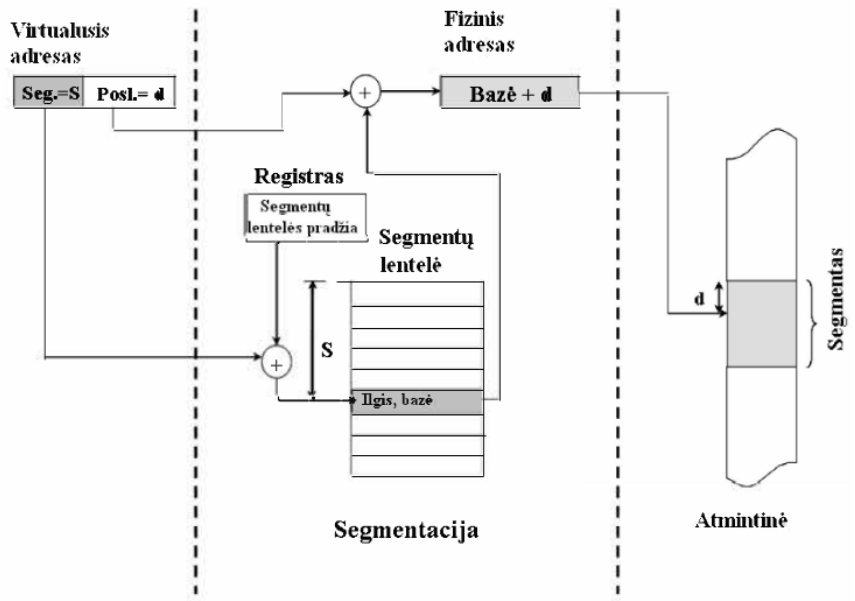
Atsiranda problemų bendrai naudojant puslapius. Procesui pasibaigus, atminties valdytojas gali nuspręsti visus su tuo procesu susijusius puslapius atlaisvinti, nepaisant to kad kai kurie puslapiai reikalingi kitiems procesams. Šiai problemai spręsti kiekvienam procesui gali būti naudojamos dvi procesų lentelės: viena, susijusi su vykdomuoju kodu, kita, susijusi su duomenų puslapiais.

4. Atminties skirstymas taikant segmentacijos principus (skirstymo segmentais principas, bendrai naudojami segmentai, segmento apsauga ir prieigos kontrolė)

Skirstymo segmentais principas

Bazinė atmintinės skirstymo segmentais idėja yra ta, kad naudojama ne viena su procesu susiejama virtualiosios atmintinės sąvoka, o daug virtualių erdvių sričių – segmentų, pavyzdžiui, viena virtualioji adresų erdvė susiejama su programos kodu, kita su duomenis, trečia su dėklu ir t.t. Kadangi kiekvienas segmentas egzistuoja atskiroje adresų erdvėje, jie gali didėti ar mažėti neveikdami vienas kito. Kiekvienam segmentui, įkeliamam į fizinę atmintinę, skiriama nuoseklių adresų erdvė. Skirstant proceso erdvę segmentais, papildoma bazinio ir ribinio adreso idėja – atsiranda visa lentelė, sauganti bazinių ir ribinių adresų poras kiekvienam proceso segmentui. Kadangi segmentai siejasi tik su programos sudėtinėmis dalimis, tai segmentų yra gerokai mažiau nei puslapių, todėl segmentų lentelės informaciją galima saugoti registruose. Tai leidžia sutrumpinti kreipimosi į atmintinę trukmę.

Kiekvieno segmento virtualusis adresas nusakomas segmento numerio ir poslinkiu segmente. Segmento numeris naudojamas kaip indeksas segmentų lentelėje, kuri sudaryta iš bazinio ir ribinio adreso porų. Transliuojant adresą imamas bazinis segmento adresas, pridedama poslinkio reikšmė ir gautas dydis lyginamas su ribiniu segmento adresu. Jei gauta reikšmė yra didesnė už ribinę, gaunama segmentavimo klaida.



8.21 pav. Adreso transliavimas skirstant segmentais

Šis skirstymo būdas kompiliatoriams yra paprastesnis. Kiekviena programos kode esanti procedūra užima atskirą segmentą, kuris turi pradinį bazinį adresą. N-oji procedūra yra kviečiama kviečiant n-ąjį segmentą.

Problema yra ta, kad sunku rasti pakankamai didelę nuoseklių adresų erdvę atmintinėje, kad tilptų visas segmentas. Todėl neišvengiamai susiduriama su išorine fragmentacija.

Bendrai naudojami segmentai

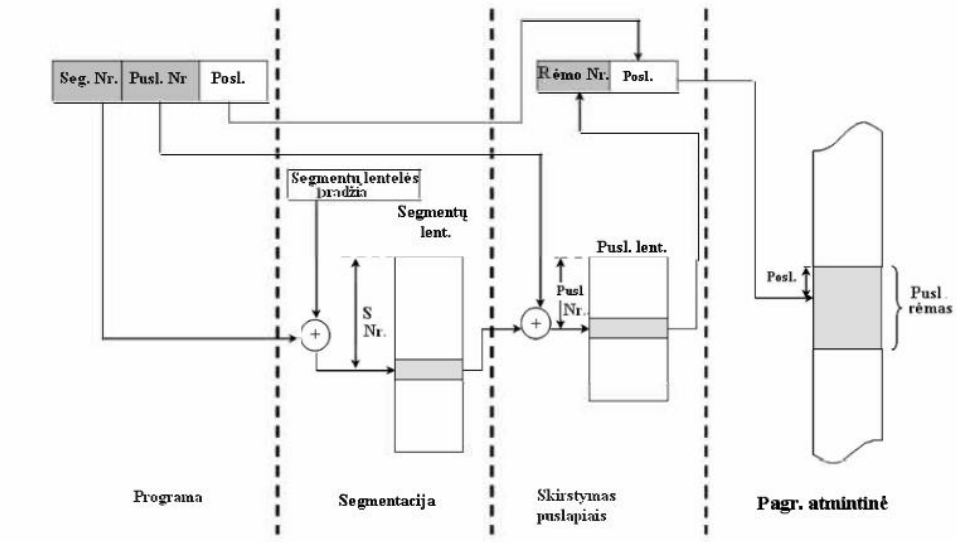
Procesams gali būti leista bendrai naudotis tam tikrais segmentais. Tokiu atveju, kiekvieno proceso segmentų lentelėse yra įrašai, kurie rodo į tas pačias pagrindinės atmintinės vietas. Bendrai naudojamų segmentų dažniausiai neleidžiama modifikuoti.

Segmento apsauga ir prieigos kontrolė

Naudojami apsaugos bitai, kuriais nusakoma ar procesas gali atlikti skaitymo, rašymo, vykdymo veiksmus su šiuo segmentu. Kombinuojant įvairias skaitymo, rašymo bei vykdymo reikšmes, galima gauti įvairias prieigos kontrolės modas (jų yra 7). Segmentų apsaugos bitai yra įrašomi į segmentų lentelę ir , procesui kreipiantis į atitinkamą segmentą, atliekama šių bitų kontrolė. Segmentų lentelėje atskiru bitu gali būti pažymima kokiame režime: privilegijuotame ar vartotojo, galima vykdyti segmento kodą.

Segmentų skirstymas puslapiais

Segmentus galima suskirstyti puslapiais. Taikant šią idėja objekto adresas nusakomas 3 dedamosiomis: segmento numeriu, puslapio numeriu segmente ir poslinkio puslapyje reikšme. Segmentų lentelėse esančiuose įrašuose yra atitinkamo segmento puslapių lentelės adresas. Segmento numeris ir puslapio numeris nusako fizinės atmintinės rėmą, o paties objekto adresas rėme nusakomas poslinkio reikšme.



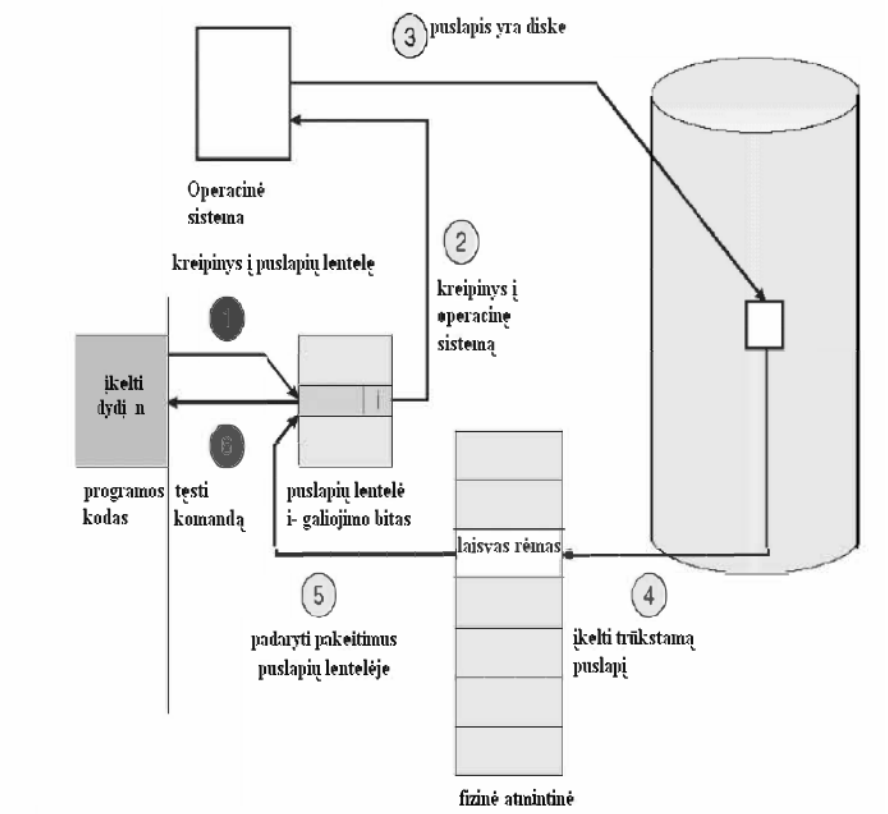
8.23 pav. Adreso transliavimas skirstant segmentus puslapiais

Tokio skirstymo pranašumas tas, kad išlaikoma loginė proceso struktūra ir taikomas paprastas atminties valdymas. Sumažinama išorinė fragmentacija. Segmentų skirstymas puslapiais leidžia ne visą procesą laikyti pagrindinėje atmintinėje. Tačiau, laikant segmentų ar puslapių lenteles pagr. atmintinėje, objektui išrinkti gali prireikti net trijų kreipinių į atmitinę.

5. Puslapių mainai (puslapių keitimo strategija, puslapių kilnojimas)

Puslapių mainai

OS, norėdama įkelti trūkstamą puslapį, turi surasti laisvą vietą (laisvą rėmą) pagr. atmintinėje. Jei to laisvo rėmo nėra, ji turi išlaisvinti vieną iš užimtų rėmų, t.y. šiame rėme esantį puslapį reikės iškelti į diską. Tačiau, jei rėme esantis puslapis nėra modifikuotas, tai jo įrašyti į diską nereikia, nes jis jau ten yra. Todėl tikrinama ar šis rėmas buvo modifikuotas. Iškeliant puslapį to puslapio galiojimo bitas yra išjungiamas, o įkeliant naują puslapį įjungiamas. Stengiamasi iškelti puslapį, į kurį pastaruoju metu nebuvo kreipiamasi.



8.24 pav. Trūkstamo puslapio įkėlimo veiksmai

Puslapių keitimo strategijos

Keičiant puslapius, reikia nuspręsti vietoj kurio pagr. atmintinėje esančio puslapio bus įkeliamas naujas puslapis iš disko. Gali būti taikoma **vietinė politika arba visuotinė**.

Kiekvienam procesui gali būti skiriamas **fiksuotas rezidentinių puslapių skaičius** (priklauso nuo veikiančių procesų skaičiaus ir bendro procesams skirto rėmų skaičiaus). Tokiu atveju naudojama vietinė puslapių keitimo politika, vienas iš to paties proceso puslapių keičiamas nauju puslapiu.

Kiekvienam procesui gali būti skiriamas **kintamas, dinaminis rezidentinių puslapių skaičius** (proceso puslapių kiekis gali ir didėti, ir mažėti). Tokiu atveju gali būti naudojama tiek vietinė, tiek visuotinė keitimo politika. Vykdant visuotinę keitimo politiką, parenkamas keisti puslapis gali priklausyti kitam procesui, todėl proceso rezidentinių puslapių skaičius gali ir padidėti, ir sumažėti.

Puslapių mainams naudojama keletas algoritmų, kurie padeda efektyviai parinkti, kurį puslapį iškelti:

- **Keičiamas puslapis, kurio prireiks vėliausiai**
  Panašus į procesams planuoti naudojamą algoritmą – trumpiausias procesas vykdomas pirmas. Šis algoritmas būtų optimalus, lengvai nusakomas žodžiais, bet jis nėra praktiškai taikomas, nes sunku nustatyti kokia tvarka bus kreipiamasi į puslapius. Šis algoritmas taikomas kitų algoritmų efektyvumui palyginti.
- **Keičiamas atsitiktinai parinktas puslapis**
  Tokį algoritmą nesunku įgyvendinti, tačiau jis nėra funkcionalus, nes gali būti iškeltas puslapis, kurio procesui tuoj pat vėl reikės. Vykdomi bereikalingi OS veiksmai.
- **Keičiamas pastaruoju metu nenaudotas puslapis (NRU)**
  Keičiant pastaruoju metu nenaudotus puslapius (NRU) stengiamasi atmintinėje išlaikyti tuos puslapius, į kuriuos neseniai buvo kreiptasi. Šiuo tikslu nagrinėjami 2 bitai : bitas R, kuris rodo, kad yra kreiptasi į puslapį ir bitas – M, kuris rodo, kad puslapis yra pakeistas. Esant laikrodžio mechanizmo pertrauktims, kreipimosi bitai R nustatomi į nulį, taigi vienetinė bito R reikšmė rodo, kad į atitinkamą puslapį buvo kreiptasi šiame trumpame laiko intervale. Įvykus pertraukčiai dėl puslapio trūkumo, visi pagr. atmintinėje esantys puslapiai suskirstomi į keturias klases:
    - 1 klasė: R=0, M=0
    - 2 klasė: R=0, M=1
    - 3 klasė: R=1, M=0
    - 4 klasė: R=1, M=1

  Pasirenkamas atsitiktinis puslapis iš žemiausios netuščios klasės. NRU yra gana paprastas, nesunkiai diegiamas algoritmas, kuris gana neblogai funkcionuoja.
- **Puslapiai keičiami laikanti FIFO disciplinos**
  Taikant FIFO algoritmą, keičiamas tas puslapis, kuris yra įkeltas seniausiai. Šis algoritmas iš pažiūros yra geras, bet rezultatai nebūna geri, nes yra išmetami ir tie puslapiai, kurie nėra dažnai naudojami ir tie į kuriuos nuolat kreipiamasi.
  Sudaroma puslapių grandinėlė, kurios pradžioje yra seniausi, o gale naujausi puslapiai.
  Algoritmas paprastas, tačiau retai taikomas dėl nedidelio efektyvumo. Be to algoritmas kenčia nuo Belady anomalijos, kai puslapių trūkumas priklauso ne nuo procesui skirtų puslapių rėmų skaičiaus, o nuo to, kokia tvarka kreipiamasi į puslapius.
  Plačiau naudojamas antrojo šanso FIFO algoritmas. Šis algoritmas yra efektyvesnis. Jei pradžioje esančio puslapio R bitas išjungtas, tai puslapis keičiamas. Jei R bitas įjungtas, tai R bitas išjungiamas ir keliamas į grandinėlės galą.
- **Taikomas laikrodžio algoritmas**
  Naudojamas žiedinis įkeltųjų puslapių sąrašas, kuriame laikrodžio rodyklė rodo į seniausią puslapį. Jei norimo keisti puslapio R bito reikšmė lygi vienetui, tai ji keičiama į nulį ir laikrodžio rodyklė pasisuka prie kito puslapio ir viskas kartojama. Kai randama R bito reikšmė lygu nuliui, tai tas puslapis keičiamas nauju ir laikrodžio rodyklė pasisuka prie kito puslapio.
  Praktikoje naudojama modifikacija WSclock, kuris įvertina kada paskutinį kartą buvo kreiptasi į puslapį. Taip pat naudojamos M bito reikšmės.
  Efektyvesnis nei antrojo šanso FIFO, nes nereikia perkelti puslapių iš grandinėlės pradžios į galą.
- **Keičiamas mažiausiai pastaruoju metu naudotas puslapis (LRU)**
  Tai tam tikra optimalaus algoritmo aproksimacija. Taikant šį algoritmą laikoma, kad dažnai pastaruoju metu naudoti puslapiai bus naudojami ir ateityje, todėl iškeliamas tas puslapis, kuris buvo naudotas mažiausiai pastaruoju metu. Reikėtų turėti nuoseklų puslapių sąrašą, kurio

priekyje būtų dažniausiai naudojami puslapiai, o gale rečiausiai naudojami puslapiai. Nediegiamas, nes daug sąnaudų reikia šiam sąrašui palaikyti.

Praktikoje naudojama modifikacija įjungiant papildomus bitus, kai kreipiamasi į puslapį, ir tam tikrais laiko tarpais juos nunulinant. Gali būti naudojami ir skaitikliai.

Vienas iš tokių sprendimų naudoti (n*n) dydžio matricą (n-puslapių skaičius). Pradžioje visa matrica užpildoma nuliais. Kai kreipiamasi į k-ąjį rėmą, k-osios eilutės elementai nustatomi į vienetinę padėtį, o k-ojo stulpelio elementai į nulinę padėtį. Eilutė, kurios dvejetainė reikšmė mažiausio rodo numerį rėmo, kuris bus keičiamas.

- **Keičiamas nedažnai naudotas puslapis (NFU)**

  Naudoja programinius skaitiklius, kurių reikšmės pradžioje nustatomos į nulines padėtis. Po kiekvienos laikrodžio mechanizmo pertraukties visų puslapių, į kuriuos paskutiniame laiko intervale buvo kreiptasi, skaitikliai padidinami vienetu. Taigi skaitikliai rodo kreipimųsi į puslapius dažnį. Puslapis su mažiausia skaitiklio reikšme gali būti keičiamas nauju.

  Trūkumas tas, kad matuojamas tik kreipinių dažnis, neatsižvelgiama kada buvo kreiptasi. Jei kuriame nors laiko intervale dažnai buvo naudotas puslapis A, tai šio puslapio skaitiklio reikšmė dar ilgai gali išlikti didelė, nors jis nebenaudojamas.

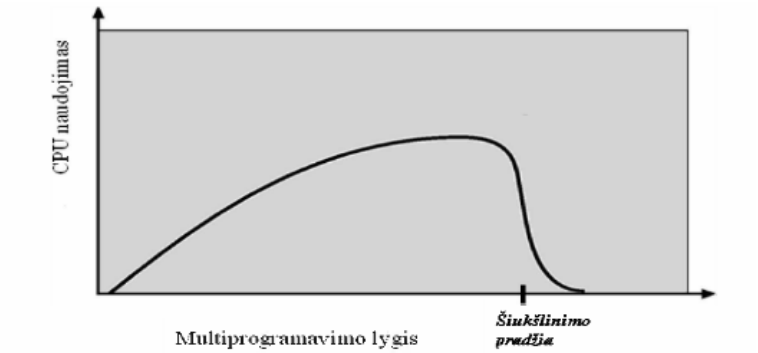
  Sendinimo algoritmas yra viena iš NFU modifikacijų. Bandoma įvertinti kreipimosi laiką. Kiekvieną kartą, prieš keičiant puslapių skaitiklių reikšmes, skaitiklių reikšmės perstumiamos vienu bitu į dešinę, o paskui prie pirmojo bito iš kairės pridedamos naujos bitų R reikšmės. Kreipiniai, kurie vykdomi artimesniu laiku turi didesnę įtaką skaitiklio reikšmei nei anksčiau buvę kreipiniai. Sendinimo algoritmas gali atsekti tik vėliausių 16 (32) kreipinių eigą (sveikam skaičiui vaizduoti skirtas bitų skaičius), tačiau paprastai to pakanka. Pagal ši algoritmą iškeliamas puslapis, kurio skaitiklio reikšmė mažiausia.

8-2 lentelė. Keitimo algoritmų apibendrinimas

Algoritmas	Komentaras
Optimalus	Praktiškai taikomas tik kitiems algoritmams palyginti
NRU – pastaruoju metu nenaudotas	Labai apytikris (*very crude*) algoritmas
FIFO	Gali iškelti naudojamus puslapius
Antrojo šanso	Smarkiai pagerintas FIFO
Laikrodžio	Realistinis
LRU	Puikus, bet tiksliai toks yra sunkiai įdiegiamas
NFU	Tai tam tikra LRU aproksimacija
Sendinimo	Efektyvus algoritmas, gerai aproksimuojantis LRU
„WSclock“	Geras, efektyvus algoritmas

Puslapių kilnojimas:

Rezidentinė (darbinė) proceso puslapių grupė – tai toks į pagrindinę atmintinę įkeltų puslapių skaičius, kuriam esant procesas gali būti efektyviai vykdomas. Jei pagrindinėje atmintyje laikoma per mažai puslapių atsiranda dažni pertraukimai dėl puslapio trūkumo. Dažnas puslapių kilnojimas vadinamas šiukšlinimu (trashing), nes OS didelę laiko dalį skiria ne procesų vykdymo valdymui, o puslapių kilnojimui.



8.30 pav. Puslapių kilnojimas ir šiukšlinimo situacija

Kaip sprendžiama problema, susijusi su šiukšlinimo situacija?

Venas iš sprendimo būdų būtų procesams reikalingo rezidentinio puslapių skaičiaus nustatymas. Atliekant ilgalaikį planavimą parenkami tik tie procesai, kurių atmintinės srities poreikiai tenkinami.

Ką daryti su procesais, kurių rezidentinės srities poreikiai yra dideli?

Sprendžiant šią problema siūloma taikyti rezidentinės aibės modelį, pagal kurį procesas gali būti laikomas pagrindinėje atmintinėje tik tokiu atveju, jei visi puslapiai, kuriais procesas tuo momentu naudojasi, gali būti pagrindinė atmintinėje. Jei puslapių poreikiai išauga, o pagr. atmintinėje nėra vietos, tai procesas iškeliamas iš pagr. atmintinės. Išmetus kelis procesus, joje likę procesai baigiami greičiau ir gali būti atlaisvinama atmintis kitiems procesams.

Problema yra sekti rezidentinius puslapius, procesų kreipinius į puslapius, keisti šių rezidentinių puslapių aibę. Tam, kad nereikėtų saugoti sąrašo puslapių, į kuriuos yra kreiptasi pastaruoju metu, fiksuojamas paskutinio kreipimosi į puslapį laikas ir rezidentiniais puslapiais laikomi tie puslapiai, į kuriuos nebuvo kreiptasi tam tikru periodu.

Puslapių keitimo algoritmai gali būti projektuojami taip, kad būtų išmetami tik tie puslapiai, kurie nėra tam tikro proceso rezidentiniai puslapiai. Vienas iš tokiu algoritmų WSclock.

III. Įvesties ir išvesties sistema

OS valdo daug įvairių I/O įtaisų. Kai kurie iš jų yra skirti bendrauti su vartotoju (žmogumi), tokių įtaisų pvz.: spausdintuvai, klaviatūra, pelė, vaizduokliai. Kiti įtaisai, pavyzdžiui, diskai, magnetinės juostos, sensoriai ir t.t, skirti tik komunikuoti su elektroniniais įrenginiais. Be to yra įtaisų, kurie naudojami komunikacijose su nutolusiais įrenginiais, - tai modemai ir tinklinės kortos.

Įtaisai skiriasi duomenų perdavimo greičiais, jų kontrolės sudėtingumu, duomenų kodavimu, gali skirtingai reaguoti į klaidų situacijas. Pagal duomenų perdavimo vienetus įtaisai skirstomi į blokinius ir simbolinius.

Blokiniuose įtaisuose duomenys yra saugomi ir į juos/iš jų yra perduodami tarpusavyje nepriklausomai fiksuoto ilgio blokais (nuo 512 b iki 32 kb). Tai diskai, magnetinės juostos.

Simboliniai įtaisai pasižymi tuo, kad duomenys į juos arba iš jų yra perduodami kaip baitų (simbolių) srautas. Tai spausdintuvai, tinklo įrenginiai (modemai), klaviatūra, pelė.

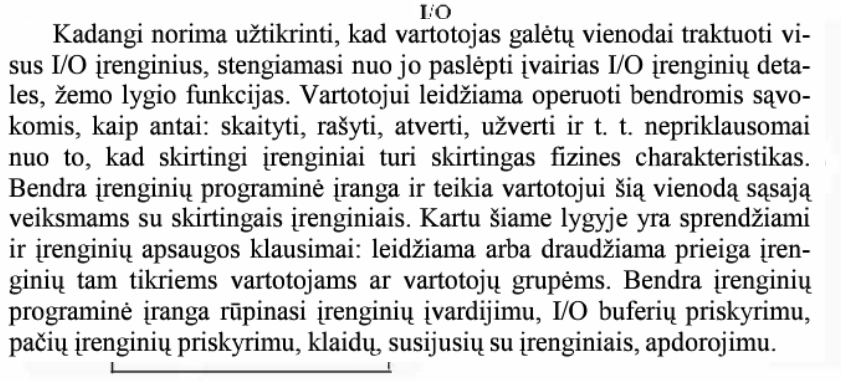
Laikrodžio mechanizmas – jo paskirtis yra tiesiog generuoti pertrauktis pagal laiką

1. I/O valdymo tikslai

I/O valdymo uždaviniai, kuriuos sprendžia OS yra susiję su dviem pagrindiniais tikslais:

1. **Siekiama užtikrinti efektyvų sistemos funkcionavimą.** Dauguma I/O įrenginių yra gerokai lėtesni nei procesorius, I/O operacijų greitis nesuderinamas su procesoriaus atliekamų veiksmų greičiu.
2. **Siekiama pateikti vartotojui bendrą, nepriklausantį nuo įrenginio, sąveikos su įrenginiais mechanizmą.** Vartotojo lygmenyje sąveika su I/O įrenginiais daroma nepriklausoma nuo fizinių įrenginio charakteristikų. Su nepriklausymu nuo įrenginių susijęs ir universalių vardų naudojimas: loginis įrenginio vardas mažai priklauso nuo paties įrenginio.

Įrenginio generuojamų klaidų apdorojimas. Paprastai stengiamasi šio tipo klaidas apdoroti kaip galima arčiau pačių įrenginių. Įrenginio kontroleris, aptikęs klaidą, bando pataisyti ją pats. Jei tai nepavyksta, klaidą apdoroti perduodama įrenginio tvarkyklei. Jei su klaida susidorojama žemame lygyje, tai paprastai aukštesnis lygis net nežino apie ją.



9.6 pav. I/O valdymo lygiai

2. Įrenginių kontroleriai

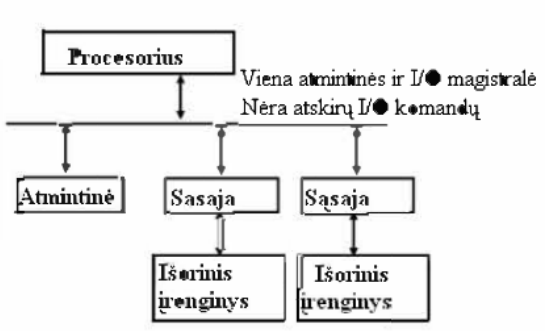
Įrenginių kontroleris – tai sąsajos korta arba integruotas modulis, kuris „kontroliuoja“ išorinius įrenginius (pvz.: standžiojo disko kontroleris).

Kontroleris turi gana primityvų procesorių, keletą registrų ir nedidelio dydžio atmintinę.

OS, rašydama į šiuos registrus, gali perduoti tam tikras komandas kontrolerio valdomam įrenginiui, pvz:. atsiųsti duomenis, priimti duomenis. Skaitydama iš šių registrų, OS gali sužinoti ir įrenginio būseną, pvz:. ar įrenginys yra pasiruošęs priimti arba perduoti duomenis.

Registrų adresacijai gali būti naudojami I/O prievado numeriai. Tokiu atveju, veiksmai su įrenginiais nurodomi naudojant specialas, dažniausiai asemblerio, komandas.

Registrų adresacijai gali būti skiriama vieta žemiausiuose pagrindinės atmintinės adresuose. Tokiu atveju, įrenginių tvarkyklių programos gali būti rašomos C kalba, nes operuojama adresuojamais dydžiais, esančiais atmintinėje.



9.2 pav. Į atmintinę atvaizduojamas I/O

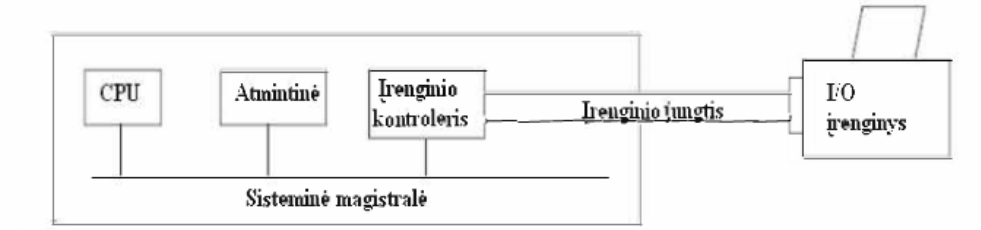
Kiekvienas I/O įrenginys turi unikalius savo duomenų ir būsenos registrų adresus, kurie gali būti traktuojami kaip ir bet kuri kita atminties vieta. Kai procesorius nori perskaityti dydį iš atmintinės arba iš įrenginio, į magistralės adresų linijas perduodamas adresas, į vieną iš komandų linijų perduodamas skaitymo signalas, o į kitą perduodamas signalas, nurodantis, iš kur turi būti skaitoma.

OS neturi tiesioginio ryšio su įrenginiu, paprastai ji turi ryšį su įrenginio kontroleriu, o šis valdo įrenginį. **Todėl I/O sistemą sąlygiškai galima suskirstyti į dvi sudedamąsias dalis:**

- Kontrolerį ir įrenginio sąsają;
- Kontrolerį ir kompiuterių sistemos sąsają

Kontroleris su įrenginiu jungiamas per įrenginio jungtį. Įrenginio jungtis – tai ryšių kabelis, jungiantis įrenginį su kontroleriu, pvz:. SCSI/IDE kabelio sąsaja diskams, RS-232 modemams.

I/O adresas – tai pavienis adresas ar adresų grupė duomenims siųsti per sisteminę magistralę į įrenginio kontrolerį ir iš jo priimti.



9.3 pav. I/O sistema

Pertrauktys iš kontrolerio siunčiamos techninės įrangos **IRQ pertraukčių siuntimo linija.**

Kai įrenginys baigia nurodytą veiksmą, pertraukčių siuntimo linija siunčiamas pertraukties signalas. Adresas linijos, kurioje pasirodo pertraukties signalas, panaudojamas kaip indeksas pertraukčių vektorių lentelėje, kurioje yra nuorodos į pertrauktis apdorojančias programas.

Tikslioji pertrauktis – tokio tipo pertrauktyje visų pirma, programos skaitiklis yra išsaugomas žinomoje vietoje, antra visos komandos iki komandos, kurią rodo programos skaitiklio reikšmė yra įvykdytos, trečia, jokia komanda, esanti už skaitiklio rodomos reikšmės, nėra įvykdyta, ketvirta, komandos, kurią rodo programos skaitiklis, vykdymo būsena yra žinoma.

Šiuolaikinėse sistemose procesoriai gali būti sujungti vamzdžio (pipeline) principu. Todėl, veiksmai, susiję su pertraukčių apdorojimu, nėra tokie paprasti. Vienas procesorius gali išrinkti komandą, kitas ją iškoduoti, trečias vykdyti. Keletas komandų gali būti įvairiose vykdymo stadijose, todėl, įvykus pertraukčiai, programos skaitiklio reikšmė gali neatspindėti tikslios ribos tarp baigtų ir dar nevykdytų komandų. Veiksmai tampa dar sudėtingesni, kai procesorius yra superskaliarinis. Gali susidaryti tokia situacija, kai komandos nebus vykdomos iš eilės, todėl tarp įvykdytų komandų gali būti kelios nevykdytos. Tokiu atveju pertraukties situacija apibūdinama **netiksliai apibrėžta pertrauktimi.** Sprendimas – pertraukties metu didelis kiekis informacijos apie būseną išsaugomas dėkle. Dėl to sulėtėja pertraukčių apdorojimas.

3. I/O veiksmų vykdymas

Įvesties ir išvesties veiksmai gali būti atliekami įvairiais I/O metodais:

- **Programuojamosios I/O**
  OS perduoda proceso I/O komandą įrenginio kontroleriui. Procesas, iškvietęs I/O veiksmus, paliekamas aktyviojo laukimo būsenos. OS periodiškai apklausia kontrolerį, norėdama nustatyti, ar I/O operacija jau baigta. Pagrindinė problema, kad procesorius priverstas laukti, kol I/O modulis pasiruošia priimti ar perduodi duomenis. Tuo metu procesorius galėtų vykdyti kitus veiksmus.
- **Pertrauktimis grindžiamos I/O**
  Procesoriui sudaroma galimybė vykdyti kitus veiksmus, kol vykdomi I/O veiksmai. Procesas sisteminiu kvietiniu paprašo I/O veiksmų ir yra blokuojamas, kol bus baigti šie veiksmai. Kai duomenys perduodami kontrolerio duomenų registrui, procesorius kviečia planuoklę, kuri išrenka vykdyti kitą procesą. Kontroleris praneša apie I/O veiksmų pabaigą siųsdamas pertraukties signalą. OS vykdo pertraukties apdorojimą: vykdomas procesas sustabdomas išsaugant jo būseną, iškviečiama pertraukties apdorojimo programa.
  Gali tekti dažnai apdoroti pertrauktis, kas užima šiek tiek laiko. Taigi procesorius naudojamas neefektyviai

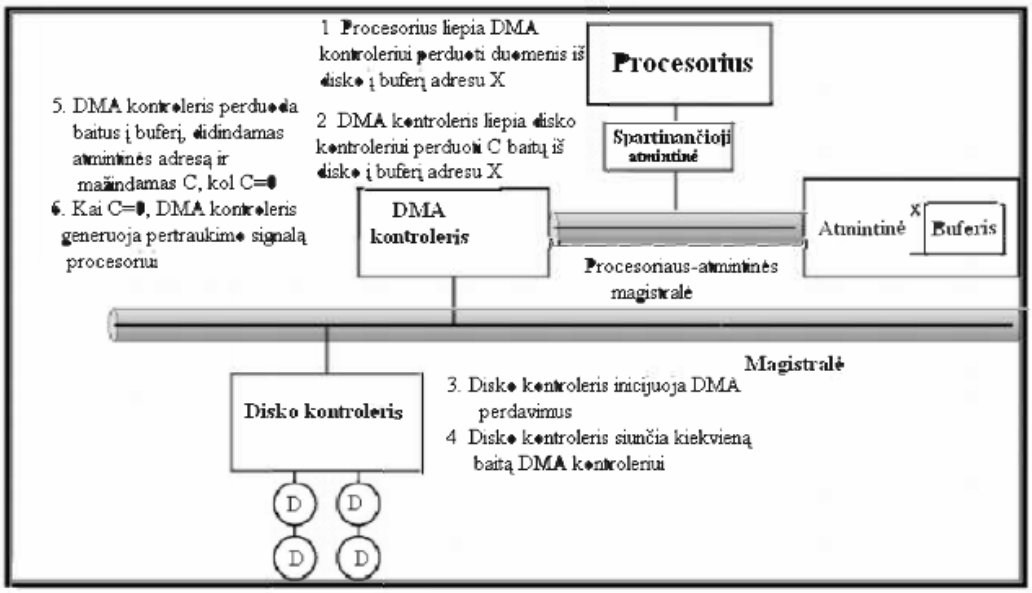
- **Tiesioginės prieigos prie atmintinės (DMA)**

  Esant tiesioginiai prieigai prie atminties OS pateikia I/O komandą DMA kontroleriui ir normaliai funkcionuoja toliau. DMA kontroleris inicijuoja skaitymo arba rašymo komandą, pateikdamas per magistralę atitinkamą užklausą, pvz., disko kontroleriui. Pabaigęs skaityti duomenis iš disko, disko kontroleris inicijuoja duomenų perdavimą DMA kontroleriui. DMA kontroleris perduoda priimamus baitus tiesiai į pagrindinę atmintinę be procesoriaus įsikišimo. **DMA modulis kontroliuoja keitimąsi duomenimis tarp pagrindinės atminties ir I/O įrenginio.** Perdavus visą informacijos bloką (3-5 punktai) DMA kontroleris generuoja procesoriaus pertrauktį.

  DMA moduliui gali tekti perimti procesoriaus ciklus, kad pasinaudotų sistemine magistrale. DMA reikia buferio duomenims saugoti, kol laukia prieigos prie sisteminės magistralės. DMA buferio tipo I/O pvz.: diskai, realaus laiko I/O (garso, vaizdo) įrenginiai.

  DMA pranašumai: taikant šį metodą, procesoriui nereikia skaityti ar rašyti iš kontrolerio į pagrindinę atmintinę, nereikia persitraukti po kiekvieno įvesto ar išvesto simbolio – sumažėja pertraukčių skaičius.

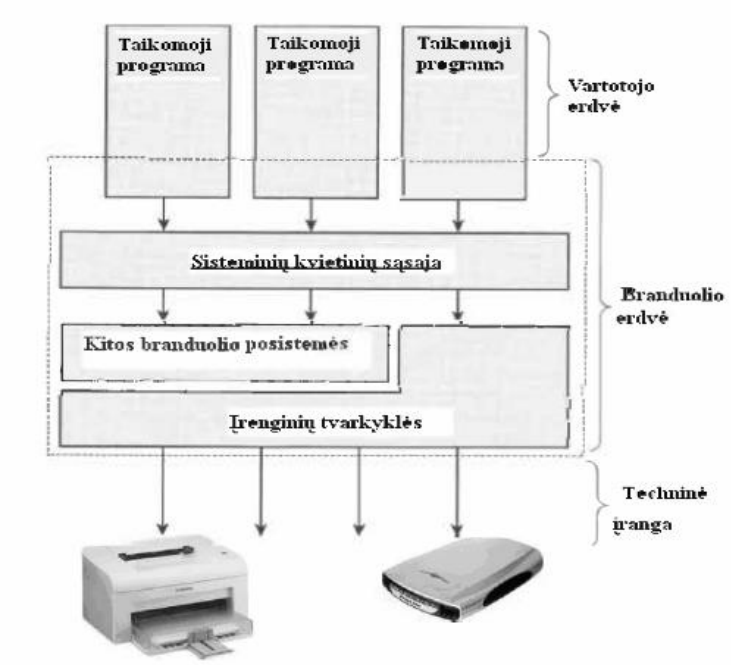
  DMA trūkumai: procesorius paprastai yra greitesnis nei DMA logika ir gali perduoti duomenis gerokai greičiau, todėl ne visi kompiuteriai naudoja DMA modulius.



9.7 pav. I/O naudojant DMA

4. Įrenginių tvarkyklės

Kiekvienas prie kompiuterio prijungtas įrenginys reikalauja tam tikros įrenginio valdymo programos, kuri yra vadinama **įrenginių tvarkykle.**



9.8 pav. Įrenginių tvarkyklės

Tam, kad būtų galima susisiekti su įrenginio kontrolerio registrais, to įrenginio tvarkyklė turi įeiti į OS branduolio sudėtį.

OS paprastai skirsto visas įrenginių tvarkykles į 2 pagrindines kategorijas:

1. Tvarkyklės skirtos blokiniams įrenginiams (pvz., diskams)
2. Tvarkyklės skirtos simboliniams įrenginiams (pvz., pelei, klaviatūrai)

Įrenginių tvarkyklės yra atsakingos už keletą funkcijų. Visų pirma jos turi priimti skaitymo arba rašymo užklausas, atėjusias iš aukštesnio, nuo įrenginio nepriklausomo lygmens, ir jas įvykdyti, patikrinusi pateiktus užklausos parametrus. Antra, tvarkyklės turi sugebėti patikrinti įrenginį, sugebėti jį inicijuoti.

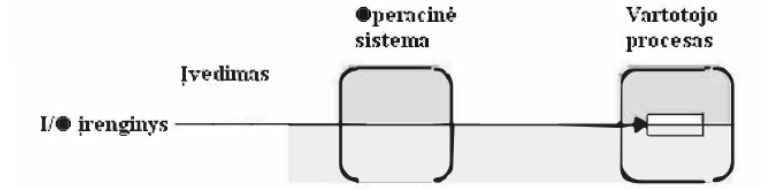
5. Nuo įrenginio nepriklausanti programinė I/O įranga

Bendros sąsajos su įvairiomis įrenginių tvarkyklės sudarymas.

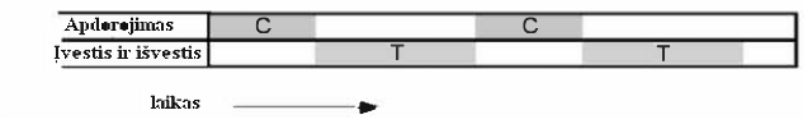
Pagrindinė šios, nuo įrenginio nepriklausančios įrangos paskirtis – atlikti tas I/O funkcijas, kurios yra bendros visiems įrenginiams, bei užtikrinti vienodą sąsają vartotojo lygio programoms.

Buferių naudojimas I/O veiksmams.

Mainai su I/O vyksta labai lėtai. **Nenaudojant buferių** procesai yra blokuojami su kiekviena skaitymo ar rašymo komanda ir turi laukti, pvz, vieno nuskaityto simbolio. Įrenginys irgi turi sustoti ir laukti, kol procesas duos kitą skaitymo ar rašymo komandą. Proceso (kodo ar duomenų) atmintinės sritis, susijusi su I/O veiksmais, turi būti blokuojama, tam, kad būtų galima atlikti įvesties ar išvesties veiksmus, kai I/O įrenginys bus tam pasiruošęs (tai trukdo valdyti atmintinę).

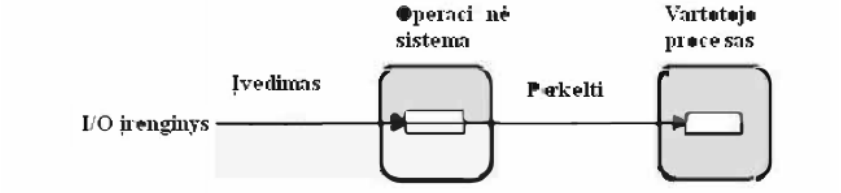


9.9 pav. Buferio nenaudojant I/O sistema

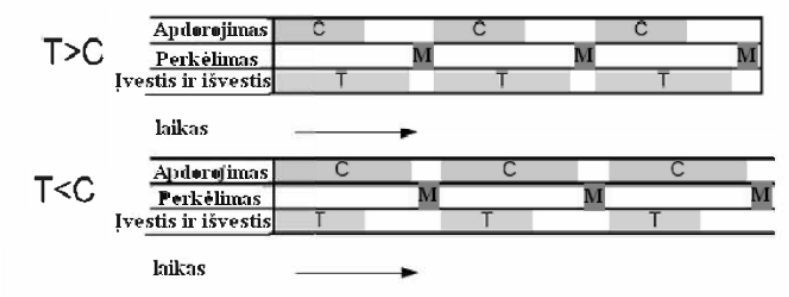


9.10 pav. Užklausos vykdymo, nenaudojant buferio, trukmės

Operacinė sistema I/O užklausoms **gali skirti buferį** branduolio srityje ir liepti pertrauktis apdorojančiai programai dėti informaciją čia. Jei informacijos mainai vyksta blokais, tai įvedamas blokas yra nukreipiamas į buferį, o buferio turinys perkeliamas į vartotojo sritį. Jei informacijos mainai su įrenginiu yra srautiniai (simbolinio tipo), tai šis srautas gali būti sudalijamas fiksuoto arba kintamo ilgio gabalais. Kai yra vienas buferis, galima teigti, kad OS gali įvesti duomenis iš anksto. Jei T>C, procesas turi laukti, kol buferis bus užpildytas. Jei T<C, OS laukia, kol buferis atsilaisvins. Užklausos vykdymo trukmė lygi max{T, C} + M.

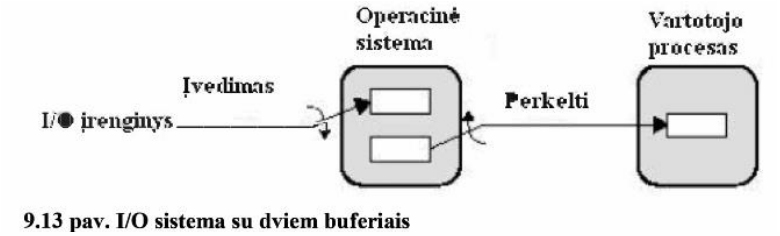


9.11 pav. I/O sistema, naudojanti buferį

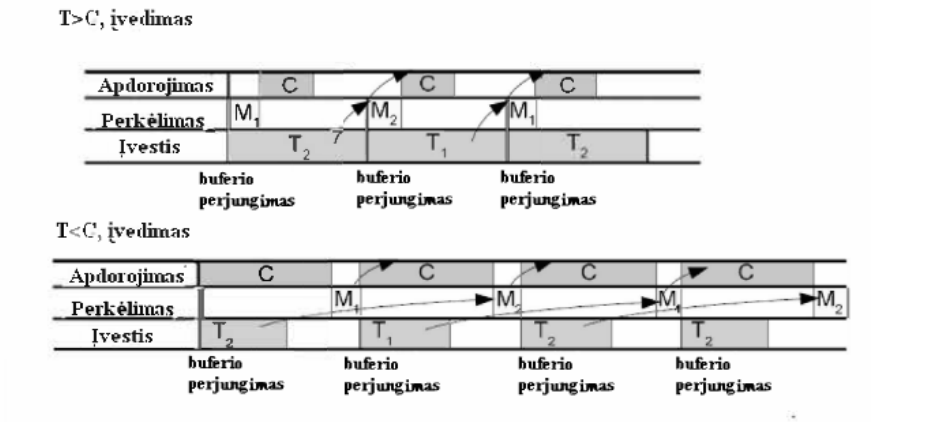


9.12 pav. Užklausos vykdymo, naudojant vieną buferį, trukmės

Naudojant 2 buferius, duomenys yra perkeliami iš kurio nors vieno sisteminio buferio į vartotojo adresų erdvę (arba atvirkščiai). Į kitą buferį tuo pačiu metu gali būti keliami duomenys iš įrenginio (arba į įrenginį). Paskui buferiai perjungiami. Jei T>C, tai įrenginys nuolat vykdo duomenų mainus, o procesorius blokuojamas dėl I/O. Jei T<C, procesorius neblokuojamas dėl I/O u-klausų, tačiau įrenginys turi sustoti ir laukti. Užklausos vykdymo trukmė lygi max{T + M, C}.

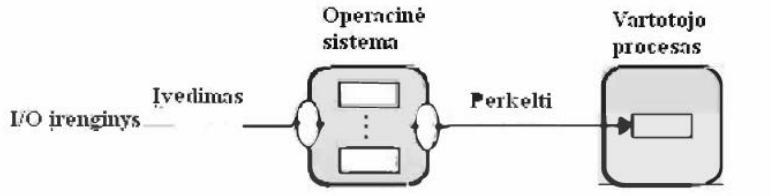


9.13 pav. I/O sistema su dviem buferiais



9.14 pav. Užklausos vykdymo, naudojant du buferius, trukmės

OS gali **naudoti ir žiedinį buferį**, tokiu atveju naudojama daugiau nei 2 buferiai. Šio tipo buferiai naudojami siekiant suderinti I/O veiksmų ir duomenų apdorojimo proceso laikus arba susidoroti su I/O pliūpsniais (kas būdinga tinkliniam I/O). Privalumai: buferiai leidžia procesams nesiblokuoti ties I/O užklausa, o įrenginių nereikia stabdyti perduodant duomenis. Trūkumai: I/O veiksmai darosi sudėtingesni, daugėja duomenų perdavimų, tarp įrenginio, buferio ir vartotojo atmintinės, naudojama daugiau pagrindinės atmintinės tiems patiems duomenims saugoti (buferyje ir vartotojo srityje).



9.15 pav. I/O sistema su žiediniu buferiu

Pranešimas apie klaidas.

Su I/O veiksmais susijusios klaidos yra dažnas reiškinys. Vienas iš šių klaidų šaltinių – programavimo klaidos, su jomis susiduriama kai procesas užprašo neįmanomų atlikti veiksmų, pvz., paprašo rašyti į tik skaitymui skirtą įrenginį (klaviatūrą). Klaidos pranešimas perduodamas I/O užklausos šaltiniui.

Kitokio tipo klaida gali atsirasti jei, pvz, bandoma rašyti į sugadintą disko bloką. Pirmiausia bandys su problemų susitvarkyti įrenginio tvarkyklė, tačiau jei ji nepajėgs, klaidos apdorojimą perduos nuo įrenginio nepriklausančiai programinei įrangai. Klaida gali būti perduodama ir vartotojui, grąžinant klaidos kodą atitinkantį sisteminį kvietinį.

Įrenginių priskyrimas.

Kai kuriuos įrenginius vienu metu gali naudoti tik vienas procesas. OS tokiu atveju turi priimti arba atmesti procesų užklausas į šį įrenginį priklausomai nuo to, ar jis tuo momentu yra prieinamas. Dažnai tokio tipo užklausos yra aprašomos kaip sisteminis kvietinys, prašantis atverti įrenginį atitinkantį specialųjį failą. Įrenginys atlaisvinamas užveriant šį failą. Jei įrenginys tuo metu neprieinamas, failo atverti nepavyksta. Galimi ir kiti variantai, kai, užklausai nepavykus, procesai kuriam laikui yra blokuojami

Nuo įrenginio nepriklausantis bloko dydis.

Mainuose su diskais informacija perduodama blokais, kurie yra tapatinami su disko sektoriaus dydžiu, tačiau skirtingi diskai gali turėti skirtingo dydžio sektorius. Nuo įrenginio nepriklausančios PĮ užduotis yra paslėpti tai ir aukštesniems lygmenims pateikti vienodo dydžio blokus.

6. Vartotojo lygmens I/O įranga

Pagrindinė programinės įrangos dalis, susijusi su I/O veiksmai, yra sudėta į OS. Vartotojo lygmenyje yra bibliotekos, kurios esančiais I/O sisteminiais kvietiniais gali pasinaudoti vartotojas savosiose programose.

Vartotojo lygmenyje taip pat yra vykdomos ir kito tipo funkcijos – tai kaupimo (spooling) sistema, kuri naudojama spausdintuvuose ar tinkliniuose įrenginiuose. Šiam tikslui sukuriamas specialus kaupimo aplankas.

IV. Diskai

Diskai yra fiziniai įrenginiai, turintys tam tikrą fizinę struktūrą.

Išrenkant ar įrašant informaciją į diską vykdomi tam tikri specifiniai veiksmai.

OS slepia įvairias technines detales nuo aukštesnio lygmens programinės įrangos, ji rūpinasi šio įrenginio kontrole, inicijuoja veiksmus su šiuo įrenginiu, aukštesniems programiniams lygmenims užtikrina abstraktų požiūrį į diską kaip į failų rinkinį.

Antrinė atmintinė gali būti sudaryta iš keleto diskų, kurie yra tiesiogiai prijungti prie kompiuterio arba per sparčiosios prieigos tinklą.

OS skirtingiems klientams gali teikti skirtingo lygio prieigą prie disko:

- Fizinio disko (plokštelę, cilindrą, sektorių)
- Loginio disko (disko bloko numerį)
- Loginio failo (failo bloką, įrašą ar baitus)

1. Fizinė disko struktūra

Paprastai diskas turi keletą besisukančių plokštelių, kurių abejose 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 arba keliuose disko blokuose.

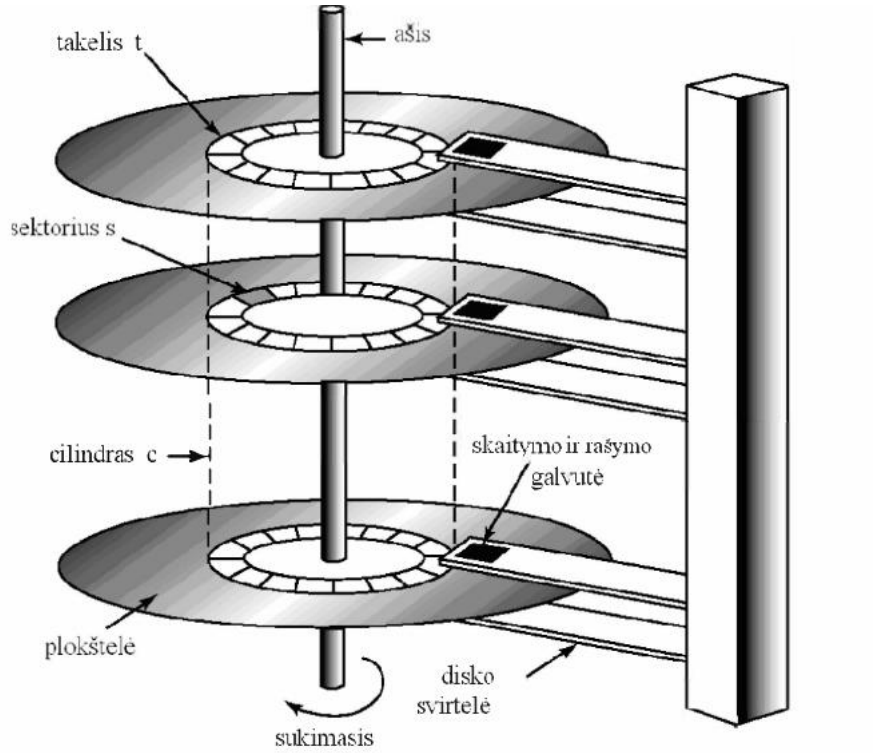
Blokas yra perdavimo vienetas vykstant mainams tarp disko ir pagrindinės atmintinės.

Norint pasiekti duomenis standžiajame diske, turi būti nurodoma, kurioje plokštelėje, kuriame takelyje ir kuriame sektoriuje yra reikalingi duomenys. Taip pat turi būti nurodoma ir kokį informacijos kiekį reikia nuskaityti arba įrašyti.

Nuoseklus duomenų skaitymas vyksta greičiau nei atsitiktinis duomenų išrinkimas – tai susiję su disko svirtelių judesiu.

Senesni diskai reikalaudavo, kad OS nurodytų visus parametrus, reikalingus duomenims išrinkti: cilindro, takelio, sektoriaus, plokštelės numerius.

Modernūs diskai reikalauja tik loginio bloko numerio.



9.16 pav. Disko struktūra

2. Disko užklausų vykdymas

Skaitymo iš disko ar rašymo į diską užklausos greitis priklauso nuo paieškos, sukimosi bei perdavimo trukmių.

Paieška – tai laikas, per kurį disko svirtelės nustatomos ties reikiamu cilindru. Priklauso nuo svirtelės judėjimo greičio.

Perdavimo laikas – tai laikas, per kurį duomenys nuo disko paviršiaus perduodami į disko kontrolerį, kuris persiunčia juos į pagrindinę atmintinę. Priklauso nuo įrašų diske tankio.

Sukimasis – tai laikas, per kurį reikiamas sektorius atsiranda ties nuskaitymo galvute. Priklauso nuo disko sukimosi greičio.

OS naudodama diską, stengiasi minimizuoti šiuos laikus, mažindama paieškos bei sukimosi trukmes.

Kadangi vienu metu gali būti apdorojama daug procesų, tai daug gali būti ir disko užklausų. Užklausoms tvarkyti gali būti naudojamos šios disciplinos:

- FCFS disciplina – pirma atėjusi užklausa ir bus pirma aptarnauta. Ji tinka tais atvejais, kai apkrova nėra didelė, nes esant ilgoms užklausų eilėms, laukimo trukmė ilgėja. Garantuoja, kad visos atėjusios užklausos bus aptarnautos
- Prioritetinė disciplina – užklausoms suteikiami tam tikri prioritetai (pvz. pagal proceso ilgį)
- SSTF (Shortest Seek Time First) disciplina – pagal šį algoritmą pirma aptarnaujama ta užklausa, kurios paieškos laikus bus trumpiausias. Išlošia tos užklausos, kurios yra orientuotos į vidurinius disko blokus. Kai kurių užklausų aptarnavimas gali būti atidėtas neapibrėžtam laikui. Tikslas – sumažinti paieškos laiką.
- SCAN disciplina – pradžioje aptarnaujamos užklausos, kurios yra susijusios su galvutės svirtelės judesiu ta pačia kryptimi, kol tokių nebelieka; paskui užklausos aptarnaujamos priešinga kryptimi.
- C-SCAN disciplina – apsiriboja judesiu tik viena kryptimi. Kai pasiekiamas paskutinis takelis, galvutės keliauja į kitą disko galą ir paieška vėl kartojama.
- SLTF (Shortest Latensy Time First) disciplina – pirmiausia yra aptarnauta ta paraiška, kuriai duotame cilindre sukimosi laiko dedamoji bus mažiausia. Ši disciplina nesunkiai pritaikoma, sukimosi vėlinimo atžvilgiu pasiekiamas beveik optimalus užklausos aptarnavimo laikas. Tikslas – sumažinti sukimosi laiko įtaką užklausos aptarnavimui.
- SPTF – įvertiną visą galvutei pozicionuoti reikalingą laiką (paieškos plius sukimosi) ir pirmiausia aptarnauja paraiškas, kurių pozicionavimo laikas trumpiausias. Gaunamos geros disko funkcionavimo charakteristikos, tačiau kai kurių užklausų aptarnavimas gali būti atidėliojamas.

3. RAID architektūra

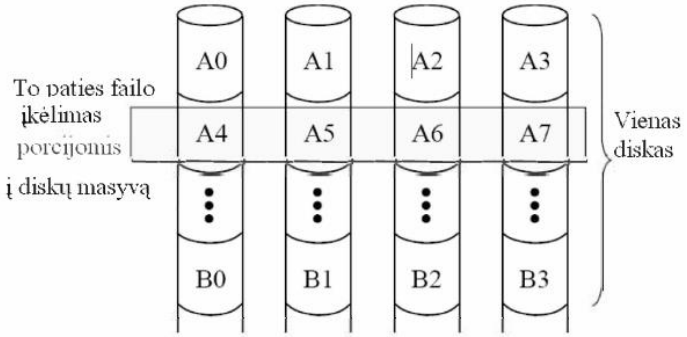
Diskų funkcionalumui padidinti ir duomenų praradimams minimizuoti naudojama RAID architektūra. Ši technologija apima RAID 0 – RAID 7 ir keletą mišrių architektūrų.

Norint įdiegti šią technologiją, reikia bent dviejų standžiųjų diskų ir RAID valdiklio, kuris keičia įprastą disko kontrolerį. Kadangi SCSI diskai yra gana funkcionalūs, dauguma RAID architektūrų susideda iš RAID SCSI valdiklio ir keleto SCSI diskų, kuriuos OS interpretuoja kaip vieną didelį diską.

Norint sutrumpinti informacijos mainų laikus bei padidinti saugomos informacijos patikimumą ir atsparumą klaidoms yra taikomi šie principai:

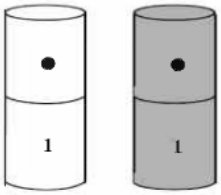
- **Veidrodinio atspindžio principas** – duomenys vienu metu rašomi į kelis diskus; tai leidžia išsaugoti informaciją nustojus veikti vienam iš diskų;
- **Skaidymo principas** – duomenys suskaidomi į dalis ir dalimis lygiagrečiai rašomi į skirtingus diskus. Tai leidžia paspartinti informacijos įrašymą ir nuskaitymą, nes galima naudoti lygiagrečias operacijas.
- **Klaidų korekcijos principas** – rašant informaciją į diskus, kartu yra įrašoma ir papildoma informacija, susijusi su galimų klaidų nustatymu. Skaitant iš disko, tai leidžia patikrinti informacijos korektiškumą.

RAID 0 architektūra skirta diskų spartai padidinti. Jai nebūdingas joks duomenų perteklumas. Įrašymo metu taikant skaidymo principą failo duomenų blokai yra siunčiami į skirtingus diskus. Trūkumas – yra nelabai patikima, nes kuriam nors iš diskų išėjus iš rikiuotės, duomenys prarandami. Spartai padidinti galima naudoti daugiau diskų.



9.22 pav. RAID 0 technologija

RAID 1 architektūroje taikomas veidrodinis principas. Įrašymo metu duomenys dubliuojami į du (arba keletą) diskus, o nuskaityti galima iš bet kurio disko. Užtikrinamas duomenų saugumas, nes leidžiama turėti kelias identiškas duomenų kopijas. Sparta negreitėja, duomenims dubliuoti naudojama visa papildomo disko talpa. Sugedus vienam diskui darbas nesustoja. Naudojama sistemose, kur saugomi itin svarbūs duomenys.

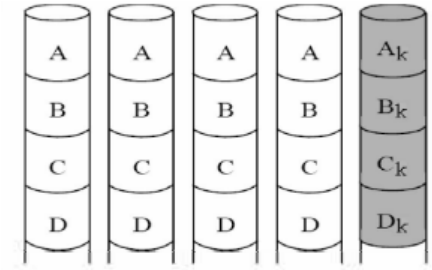


9.23 pav. RAID 1 technologija

RAID 2 architektūroje, rašant į diskus, informacija dalijama ne sektoriais, o baitais. Saugomi ir klaidų korekcijos kodai jiems skirtuose atskiruose diskuose. Ši architektūra reikalauja geros diskų sukimosi sinchronizacijos ir turi prasmę tik esant gana dideliam diskų skaičiui. Be to, reikia nemažai vietos korekcijos kodams saugoti (pvz., esant 32 duomenų diskams, 6 diskų reikės korekcijos kodams saugoti)

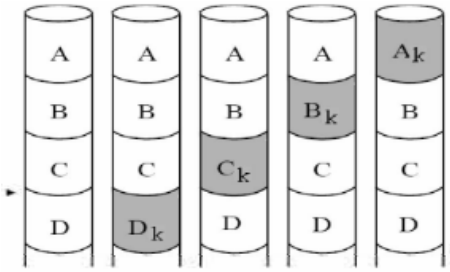
RAID 3 yra supaprastintas RAID 2 atvejis. Naudojamas paprastesnis klaidų korekcijos principas – kiekvienam duomenų žodžiui skaičiuojamas vienas lyginumo bitas, žodžio bitai išdėstomi po vieną skirtinguose diskuose, o šie lyginumo bitai rašomi į atskirą diską.

RAID 4 atveju informacijos vėl dalijama taip kaip ir RAID 0 atveju, o lyginimo kontrolei naudojamas atskiras diskas. Nuskaitymo metu duomenys sutikrinami su kontroliniu disku.



9.24 pav. RAID 3 4 technologija

RAID 5 architektūroje duomenys skaidomi ir rašomi į skirtingus diskus. Kiekvienas diskas turi sekciją kontroliniams kitų diskų duomenims saugoti (apie 20% disko talpos). Sudegus vienam diskui, sistema gali toliau dirbti, duomenys atkuriami iš kitų diskų.



9.25 pav. RAID 5 technologija

Yra ir daugiau klasikinių RAID tipų, tačiau jie kur kas sudėtingesni, reikalauja itin brangių RAID valdiklių.

V. Failų sistema

1. Failai ir failų sistema, failų sistemos sprendžiami uždaviniai, failų sistemos abstrakcijos lygiai

Failais vadinamas bet koks vardą turintis duomenų rinkinys, kuriuos yra manipuliuojama kaip atskiru vienetu.

Failų vardams taikomos taisyklės priklauso nuo konkrečios OS.

Failuose informacija saugoma tol, kol failas nėra išmetamas. Procesai gali juos skaityti, rašyti į juos ar kurti naujus failus.

OS gali sukurti vartotojui sąsają, kuri palengvina jam navigacijos veiksmus su failais. Tai vadinama **failų sistema**.

Failų sistema – tai OS dalis, kuri vykdo veiksmus su failais, užtikrina jų ilgalaikį saugojimą. Tai antrinės atmintinės abstraktaus atvaizdavimo vartotojams paslauga.

Failų sistema turi užtikrinti prieigos kontrolę bei failų apsaugą.

Viena svarbių OS užduočių yra tvarkyti failus pagal naudojamos failų sistemos principus. Failų sistema slepia įrenginių technines detales, leisdama vartotojams operuoti failų ir aplankų sąvokomis. Failų sistema teikia failų saugojimo ir organizavimo metodą.

Failų sistema saugo papildomus duomenis, kurie padeda lengviau rasti ir pasiekti failus ir juose esančius duomenis.

Failų sistema padeda efektyviai išnaudoti išorinės atmintinės įrenginius: tai pasiekiama taikant optimalius failų išdėstymo atmintinėje metodus, kurie leidžia optimaliai išnaudoti disko blokus; iki minimumo sutrumpinti prieigos prie failo trukmę.

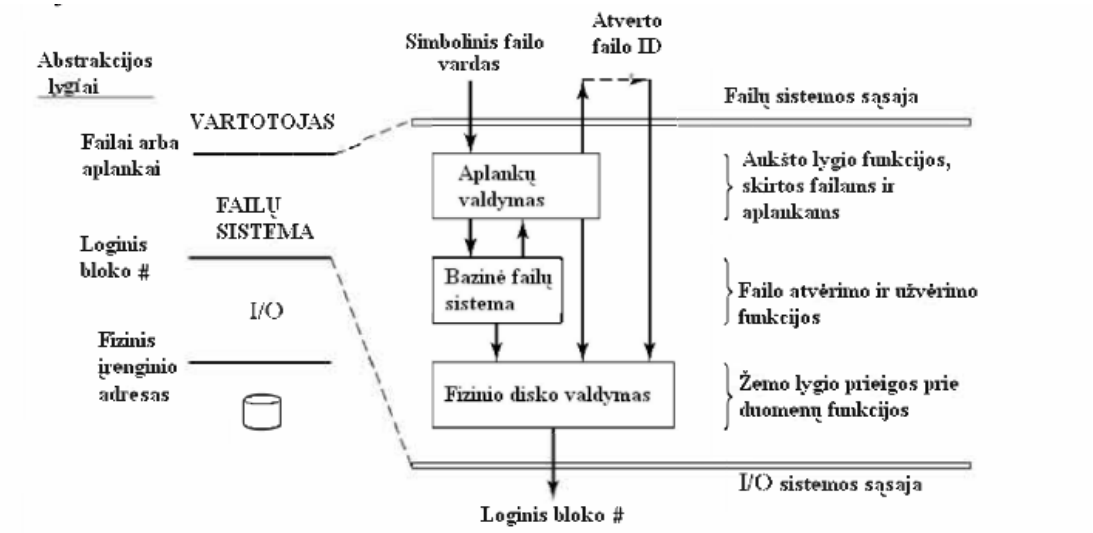
Failų sistema užtikrina loginę failų organizaciją, kuri atspindima kelių lygių aplankų struktūra.

Failų sistema apima diske saugomų failų visumą, duomenų struktūras, reikalingas failams tvarkyti (aplankus, failų deskriptorius, informaciją apie vietą diske), failų tvarkymo programas (skirtas failams kurti, naikinti, kopijuoti, perkelti, įrašyti).

Failų sistemos struktūra hierarchinė. Vartotojui pateikiama vartotojo lygio sąsaja, kuria naudodamasis jis gali naudotis failų bei aplankų sąvokomis. **Žemesniame lygyje yra aplankų valdymas, kurio paskirtis pagal loginį failo vardą pateikti ši failą atitinkantį unikalų failo identifikatorių, atrasti failo deskriptorių**.

Bazinė failų sistema vykdo failo atvėrimo ir užvėrimo funkcijas.

Fiziniai failų sistemos organizavimo metodai leidžia nusakyti, kokie realūs disko blokai atitinka failo blokus.



10.1 pav. Hierarchinė failų sistemos struktūra

Failas – tai dvejetainių dydžių rinkinys. Failas gali atitikti programą ar dokumentą, o kai kuriais atvejais ir pačios failų sistemos dalį.

Failai gali turėti labai skirtingas duomenų struktūras, tačiau gali būti pasiekiami tais pačiais failų sistemoje naudojamais metodais.

Failas vartotojo požiūriu – tai įrašų grupė, turintis savo vardą ir laikoma ar apdorojama kaip visuma.

Procesui pasibaigus jo vykdymo metu sukurtas failas gali išlikti sistemoje ir į jį gali kreiptis kiti procesai.

„Unix“ sistemos interpretuoja failus kaip tiesiog nestruktūrizuotą baitų seką. Vartotojo programos gali įdėti į failus bet ką ir taip suteikti jiems didelį lankstumą.

Struktūrizuoti failai palengvina ieškoti informacijos ir ją saugoti, todėl yra plačiai taikomi duomenų bazėse.

„Unix“ sistemoje failas yra sudarytas iš penkių dalių : antraštės, teksto, duomenų, kilnojimo bitų ir simbolių lentelės.

Su failais galima atlikti įvairius veiksmus, pvz:. galima sukurti failą, jį išmesti, ieškoti failo, apverti, jį užverti, nustatyti jo atributą, sužinoti failo atributus, modifikuoti jo turinį.

OS saugo informaciją, kuriuose disko blokuose yra saugomas failas. Kai kuriose failų sistemose failai gali būti saugomi tik ištisiniuose (nuosekliuose) disko blokuose. Taip saugant pakanka žinoti, kur diske yra failo pradžia ir kokios jis dydžio. Tai sukelia problemų jei failas yra keičiamas ar plečiamas. Moderniose failų sistemose failai išdėstomi diske ne į ištisinę sritį, o į tam tikru būdu susietą disko blokų sąrašą.

Įkeliant failą į diską įsimenama, į kuriuos loginius disko blokus tas failas įrašomas. Atliekant veiksmus su failais loginių blokų numeriai verčiami į fizinius disko adresus, kuriuos nusako takeliai, cilindrai, plokštelės, sektoriai. Adresus verčia diskų tvarkyklės, kurios perduoda disko kontroleriui atitinkamas žemo lygio komandas.

2. Failų atributai

Failo atributai arba failo metaduomenys – tai failui būdingos tam tikros charakteristikos kaip failo vardas, savininkas, savininko grupė, dydis, leidimai prieiti, paskutinio modifikavimo, sukūrimo, paskutinės prieigos laikai, dydis, tipas, slaptažodis, failo vieta diske ir t.t.

Skirtingose OS atributų skaičius gali būti skirtingas.

Failo vardas – tai failą identifikuojanti simbolių seka.

Failo tipas dažnai apibūdina failo turinį

Failo saugojimo vieta nurodo, kuriame įrenginyje ir kur tame įrenginyje yra saugomas failas.

Failo dydis rodo einamąjį failo dydį baitais, žodžiais ar blokais,

Prieigos prie failo leidimai nusako kokius veiksmus ir kokiems vartotojams leidžiama atlikti su failu.

3. Veiksmai su failais

Vartotojui leidžiama sukurti (create), išmesti (delete) failą. Failą galima sukurti tuščią, be duomenų ir nustatyti pradinius failo atributus.

Leidžiama atverti (open), užverti (close) failą. Atveriant failą, į pagr. atmintinę įkeliami failo atributai bei informacija apie jo išdėstymą diske. Atvertas failas įkeliamas į atvertų failų lentelę.

Leidžiama rašyti (read), skaityti (write) failą. Sistema išlaiko skaitymo arba rašymo rodyklę, rodančią esamą poziciją faile. Pozicijai pakeisti naudojamas seek

Leidžiama gauti atributus (get attributes), nustatyti atributus (set attributes). Nustatant atributus galima keisti failo priklausomybę, apsaugos modą, paskutinio modifikavimo datą.

Leidžiama papildyti (append), kopijuoti failą (copy), pervadinti (rename).

4. Prieigos prie failų metodai

- **Nuoseklioji prieiga** – pati paprasčiausia prieiga. Šiuo atveju failo baitai yra nuskaitomi nuosekliai, eilės tvarka, po vieną baitą nuo failo pradžios. Neefektyvus sudėtingoms užduotims, naudojamas magnetinėse juostose, rečiau diskuose.
- **Tiesioginė prieiga** – modernesnis prieigos metodas. Tiesioginė prieiga turi leisti tiesioginiai prieiti prie bet kurio užprašyto informacijos įrašo. Taikant šį metodą, fialas yra interpretuojamas kaip aibė numeruotų blokų ar įrašų.
- **Atsitiktinė prieiga –** individualūs adresai tiksliai nusako duomenų poziciją. Ji dažniau naudojama kai skaitoma iš RAM atmintinės.
- **Asociatyvioji prieiga** – ši prieiga taikoma spartinančiojoje atmintinėje. Duomenų adresai nusakomi jų turiniu, o ne vieta.

5. Aplankai

Aplankai turi dvi paskirtis:

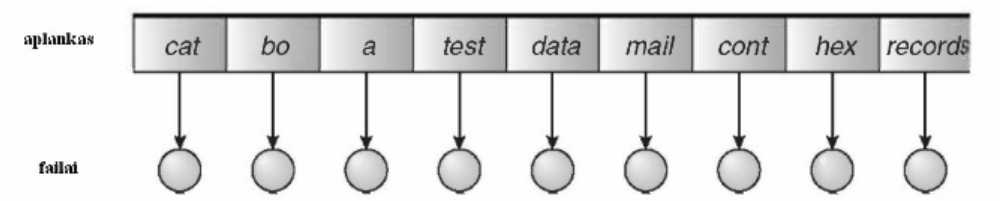
1. Vartotojams jie leidžia struktūrizuoti failų laikymą failų sistemoje, grupuojant juos pagal tam tikras savybes, leidžia skirtinguose aplankuose saugomiems failams naudoti vienodus vardus.
2. Leidžia atskirti loginę failų sistemos struktūrą nuo fizinio failų saugojimo diske

Aplankuose saugomas failų sąrašas ir metaduomenys apie šiuos failus.

Aplankai teikia informaciją, kurios reikia norint greitai surasti failo duomenų blokus diske. Pats aplankas taip pat yra failas, tik specialios struktūros. Kiekvienas įėjimas aplanke – tai simbolinis failo vardas kartu su nuoroda į tai, kur diske saugomas tas failas. Taip pat gali būti saugoma informacija apie failo dydį, tipą, prieigos galimybes, modifikavimo ir sukūrimo laikai.

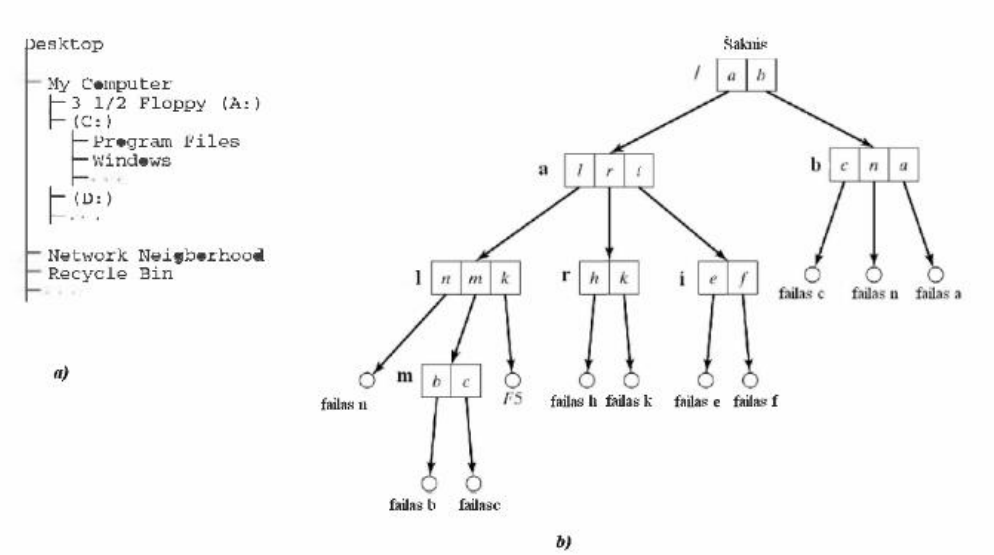
Aplankų struktūros:

1. Vieno lygio. Negalima naudoti tų pačių vardų, negalima naudoti grupavimo, kilus reikalui surasti kurį nors failą



10.2 pav. Vieno lygio aplankų struktūra

2. Medžio tipo arba hierarchinė. Tokia struktūra turi daug lygių. Failo vieta tokioje struktūroje nusakoma absoliučiais arba santykiniais kelio vardais.



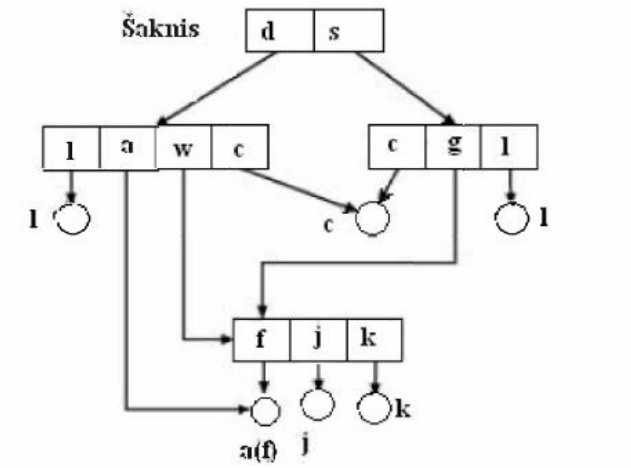
10.3 pav. Hierarchinė aplankų struktūra: *a)* „Windows“, *b)* „Unix“

3. Neciklinio grafo tipo. Susidaro leidžiant keliems vartotojams dalytis kuriais nors failais ar katalogais. Naudojamos nuorodos.

   Nuoroda – tai aplanko įėjimas, kuris siejasi failu ar aplanku, esančiu kitame aplanke.

   Simbolinė nuoroda – tai aplanko įrašas, nurodantis kelią iki kito failo ar aplanko.

   Kieta nuoroda – tai aplanko įrašas, kuriuo nusakoma failo vieta jį saugančiame įrenginyje.



10.4 pav. Neciklinio grafo tipo aplankų struktūra

Su aplankais galima atlikti panašias operacijas kaip ir su failais: juos galima sukurti, išmesti, galima peržiūrėti aplankų turinį, juose galima atlikti failų paiešką, galiam pereiti į kitą aplanką naudojant kelius (santykinius arba absoliučius), galima pervardyti, perkelti aplankus, pakeisti leidimus prieigai, sukurti kietus ar simbolinius ryšius į kitus aplankus ar failus.

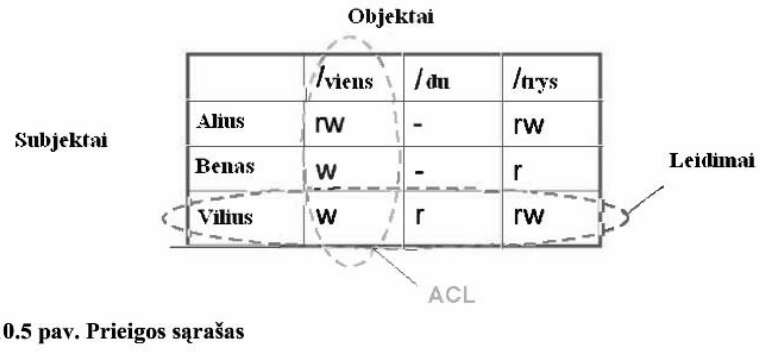
6. Bendras naudojimasis failais ir failų apsauga

OS leidžia keliems vartotojams naudotis tais pačiais failais tiek viename kompiuteryje, tiek tinklo sistemoje.

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 tikra 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.



10.5 pav. Prieigos sąrašas

7. Failų lentelė

Bazinė failų sistema valdo atvertųjų failų lentelę. Bazinė failų sistema tikrina vartotojo prieiga prie atveriamojo failo. Jei leidimai numatyti, šis failas įeina į atvertųjų failų lentelę, failui suteikiamas buferis įvedimui ir išvedimui.

Failų sistema kiekvienam procesui saugo to proceso atvertų failų lentelę dar **vadinamą aprašų (deskriptorių) lentele**. Kiekviename šio sąrašo įėjime saugoma nuoroda į sistemos atvertųjų failų lentelę. Procesui, atveriančiam failą grąžinamas jo atvertų failų lentelės indeksas, atitinkantis šio failo įrašo vietą jo lentelėje. Naudodamasis šiuo indeksu, kuris yra vadinamas failo deskriptoriu (arba aprašu) vartotojas toliau gali atlikti visus veiksmus su šiuo failu.

8. Fiziniai failų sistemos organizavimo metodai

Fizinė failų sistemos organizacija aprašo failo išdėstymo išorinėje atmintinėje, pvz, diske, taisykles.

Diskai paprastai yra dalinami į kelis skaidinius (partitions), kurių kiekviename saugoma nepriklausoma failų sistema. Kiekvieno disko nuliniame sektoriuje paprastai saugomas sistemos įkrovimo įrašas, o šio sektoriaus gale saugoma disko skaidinių lentelė, su kiekvieno skaidinio pradžios ir pabaigos adresais.

Failo blokus diske siekiama išdėstyti taip, kad failo paieškos laikas būtų kiek galima trumpesnis, o vidinė ir išorinė fragmentacija – kiek galima mažesnė.

Failai jie gali būti įkeliami į disko :

- Nuosekliai išdėstytus blokus
- Blokus, sutraukiant juos į sąrašą
- Blokus, sudarant indeksų sąrašą

9. Laisvos disko atmintinės valdymas

Laisvos disko atmintinės valdymas yra panašus į pagrindinės atmintinės valdymą.

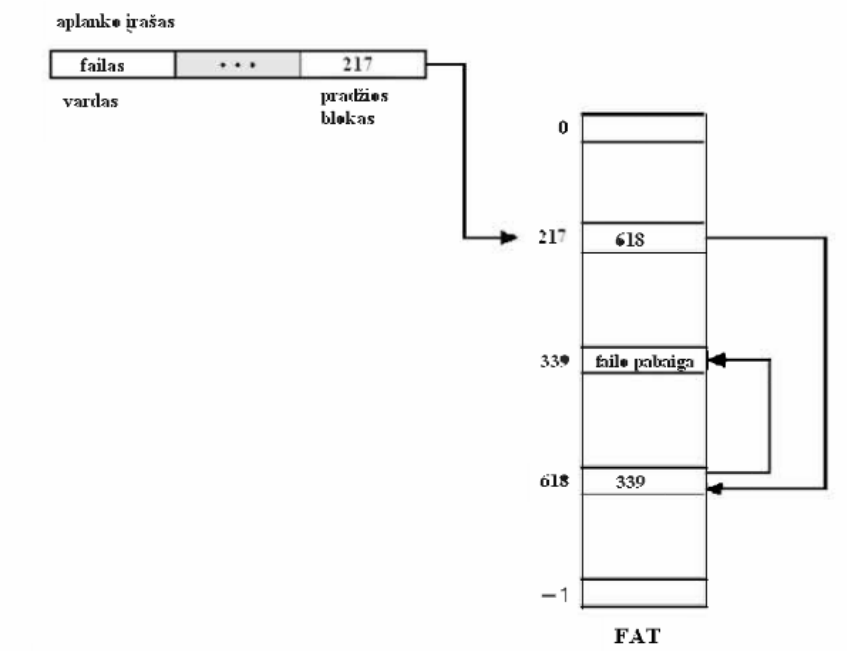
Failų sistema, norėdama priskirti laisvą bloką failui, turi turėti laisvų blokų sąrašą. Laisvų blokų sąrašas yra formuojamas sutraukiant į sąrašą laisvų blokų grupes, o ne pavienius blokus.

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 ir naudojant dvejetainį žemėlapį, kuriame vienetais pažymimi užimti disko blokai, o nuliais – laisvi blokai. Dvejetainio žemėlapio trūkumas – ieškant laisvo bloko, failų sistemai gali tekti peržiūrėti visą dvejetainį žemėlapį. Be to, dvejetainį žemėlapį tenka saugoti pagr. atmintinėje, o tam reikia skirti vietos.

10. Skirtingose OS naudojamos failų sistemos

Windows tipo OS sistemos ilgą laiką naudojo FAT tipo failų sistemas. Jos informaciją apie failus, jų vietą diske laiko FAT lentelėje. FAT lentelė formuojama kaip masyvas, o aplankuose yra nurodoma nuoroda į pirmą kiekvieno failo bloką. Tolimesni failo blokai susiejami nuorodomis.



10.16 pav. FAT lentelės įrašas

Siekiant sumažinti FAT lentelės dydį ir padidinti failams skiriamų blokų kiekį vieno priskyrimo metu, kiekvienas disko skaidinys (particija) yra sudalomas į vienodo dydžio blokų junginius (cluster). Kiekvienas failas gali užimti vieną ar daugiau junginių, kurie nebūtinai yra šalia vienas kito.

FAT lentelė sudaryta iš įrašų, kurie aprašo kiekvieną skaidinio blokų junginį. Galimi lentelės įrašo tipai:

- Grandinėlės sekančio junginio numeris
- Speciali atžyma apie grandinėlės pabaigą
- Speciali atžyma apie blogą junginį

- Speciali atžyma apie rezervuotą junginį
- Nulis, pažymintis, kad junginys yra nenaudojamas.

FAT lentelės kiekvienas įrašas atitinka kiekvienam disko blokų junginiui. FAT lentelė yra saugoma diske, kiekvienam skaidiniui yra atskira FAT lentelė. Ši lentelė OS nusako kurie disko blokai (jų junginiai) yra laisvi, o kurie užimti. Siekant sutrumpinti paiešką, FAT lentelės dalį galima laikyti spartinančioje atmintinėje.

FAT16 – naudojami 16 bitų įrašai

FAT32 – naudojami 32 bitų įrašai

FAT32 leidžia nurodyti didesnį blokų junginių skaičių nei FAT16, todėl FAT16 sistemose maksimalus failo dydis gali siekti 2 GB, FAT32 – iki 4 TB. Taip pat ir disko skaidinio dydis gali būti didesnis. FAT32 leidžia efektyviau išnaudoti disko erdvę, naudodama mažesnius blokų junginius.

HPFS (labai funkcionali failų sistema) – tai failų sistema, sukurta OS/2 operacinei sistemai, siekiant pagerinti kai kuriuos FAT failų sistemos apribojimus. Ji toleruoja ilgus failų vardus, efektyviau naudoja disko erdvę, nes leidžia priskirti failams vietą diske ne blokų junginiais, o blokais (po 512 B), užtikrina spartesnę prieigą prie didesnių standžiųjų diskų nei FAT failų sistema. Naudojant HPFS aplanko įėjime saugoma daugiau informacijos nei FAT sistemose.

NTFS sistemos, pvz NTFS5 sistema, kurią naudoja „Windows 2000“ teikia daug naujų privalumų:

- Šifravimas, kuris leidžia užkoduoti diske saugomis duomenis
- Disko kvotos, kuriuos įgalina nurodyti vartotojui leidžiamas ribas diske
- Defragmentacija – laisvos vietos sujungimas į vientisą erdvę atliekant perkėlimus
- Montavimo taškai kurie leidžia montuoti fialų sistemą prijungiant papildomus diskus, nepriskiriant jiems raidės.

NFTS sistema leidžia turėti didelių dydžių failus (iki 2^{64} B, yra labai patikima, nes registruojami veiksmai, atliekami su failų sistema).

11. Žurnalus naudojančios failų sistemos

Failų sistemos, kurios naudoja žurnalus, visi atliekami pakeitimai yra fiksuojami žurnaluose. Šie įrašai paprastai yra laikomi atskirame fialų sistemos skyriuje.

Įrašas žurnale daromas prieš atliekant realų failų sistemos duomenų struktūros pakeitimą. Po to atliekami pakeitimai realiose duomenų struktūrose ir tai užbaigus padaromas įrašas apie veiksmų pabaigą.

Failų sistemos neprieštaringumas garantuojamas tuo, kad po trikio failų sistemai atkurti galima pasinaudoti žurnalo įrašais. Jei žurnale nėra įrašo apie sėkmingą pakeitimo įvykdymo pabaigą, tai tokius įrašus galima atmesti.

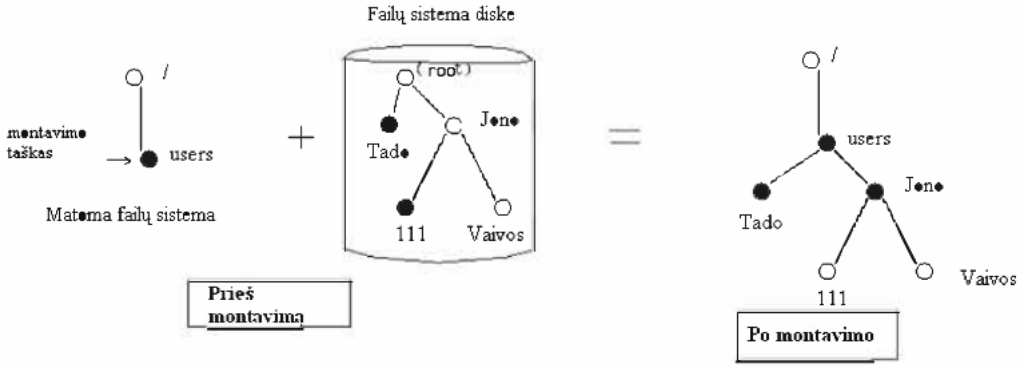
Kokie duomenys gali būti įrašomi į žurnalus? Yra 2 galimybės – visad duomenų įrašas arba metaduomenų įrašas. Pirmuoju atveju į žurnalą yra įrašomi visi duomenys.

12. Failų sistemos montavimas

Atskirame diske ar atskirame disko skaidinyje esančios failų sistemos įtraukiamos į bendrą failų sistemos struktūrą yra žinomos kaip failų sistemos montavimas. Failų sistema turi būti montuojama prieš tai, kol ji bus prieinama sistemos procesams, taip kaip ir failas turi būti atveriamas prieš tai, kai juo pradedama naudotis.

Montuojant failų sistemą, parenkamas įrenginys, kuris bus montuojamas, ir montavimo taškas (tai vieta failų sistemoje, kurioje montuojamo įrenginio failų sistema bus prijungiama prie esamos). Failų sistema patikrina, ar montuojama failų sistema yra tinkama. OS 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. Iš jos taip pat bus leidžiama „peršokti“ į kitas failų sistemos šakas.



10.17 pav. Failų sistemos montavimas

VI. Svečių paskaita

1. 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 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.
- **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

2. Mobilios OS

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 bevieliai prisijungimo būdai.

Programų steką sudaro 4 sluoksniai:

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

Android OS apima grupę specialiai operacinei sistema sukurtų sisteminių C/C++ bibliotekų, kuriomis vartotojai – programų kūrėjai gali naudotis per Programų karkasą.

Kai kurios bibliotekos turi savo šaknis atviro kodo projektuose. Dėl licencijavimo konfliktų buvo nuspręsta Android OS įdiegti savą C biblioteką („Bionic“) ir sukurti specifinę Java programų vykdymo aplinką – Dalvik virtualią mašiną

Java programos yra rašomos Java programavimo kalba (tačiau nebūtinai). Tos programos būna kompiliuojamos į .dex(dalvik) programas ir vykdomos dalvik VM.
4.4 Android versija leido vartotojams pakeisti Dalvik VM į ART (Android RunTime), kuri interpretuoja kodą į OS suprantamas instrukcijas. Android 5.0 versijoje Dalvik nebebuvo naudojamas ir liko tik ART.

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.