**PA – pagrindinė atmintinė | PĮ – programinė įranga | TĮ – techninė įranga | OS – operacinė sistema**

Turinys

[Įvadas (atminties valdymas) 1](#_Toc453008760)

[Pagrindinės atmintinės valdymo sąvokos 1](#_Toc453008761)

[Paprastas atmintinės valdymas 2](#_Toc453008762)

[Fragmentacijos sąvoka. Išorinė, vidinė fragmentacija 4](#_Toc453008763)

[Virtualioji atmintinė 4](#_Toc453008764)

[Adresų tipai. Loginio adreso transliavimas į fizinį. MMU 4](#_Toc453008765)

[Atminties skirstymas, taikant paprastą puslapiavimą (proceso įkėlimas, kt.) 5](#_Toc453008766)

[Puslapių lentelė ir jos įrašo struktūra 6](#_Toc453008767)

[Segmentavimas, skirstymas segmentais ir kt. 6](#_Toc453008768)

[Puslapių mainai (keitimo strategijos, puslapių kilnojimas) 7](#_Toc453008769)

[I/O įtaisai 9](#_Toc453008770)

[I/O įtaisai (bendrai) 9](#_Toc453008771)

[I/O valdymo tikslai 9](#_Toc453008772)

[Įrenginių kontrolieriai 9](#_Toc453008773)

[I/O veiksmų valdymas 10](#_Toc453008774)

[Įrenginių tvarkyklės 10](#_Toc453008775)

[Nuo įrenginio nepriklausanti programinė I/O įranga, vartotojo lygmens I/O įranga 10](#_Toc453008776)

[Diskai 11](#_Toc453008777)

[Fizinė disko struktūra 11](#_Toc453008778)

[Disko užklausų vykdymas 11](#_Toc453008779)

[RAID architektūra 12](#_Toc453008780)

# Įvadas (atminties valdymas)

## Pagrindinės atmintinės valdymo sąvokos

Atmintinės būna 3 tipų – pagrindinė, spartinančioji (mažai apie čia kalbama) ir išorinė (arba antrinė. Tai galėtų būti, pavyzdžiui, diskai). Išorinė yra pigi, tačiau reikia daugiau laiko išrinkti duomenis. Ji taip pat skirta programoms ir duomenims ilgai saugoti, o PA skiriama tuo metu vykdomoms programoms saugoti. Duomenų perkėlimas iš vienos į kitos rūšies atmintines yra OS uždavinys.

Pagrindinės atmintinės valdymas – tai užduotis, kurią vykdo OS kartu su TĮ, siekdama, kad kuo daugiau procesų tilptų PA. OS yra priversta sudėti į PA kiek įmanoma daugiau procesų, kad neprastovėtų tuščiai. Kad procesai vienas kitam nemaišytų, OS turi fiksuoti, kurios PA sritys užimtos, kas jas užėmė, turi spręsti, kur įkelti naują procesą, kur ir kaip priskirti papildomas sritis procesams, kai jie to užsiprašo, pasibaigus – fiksuoti jų turėtas PA sritis kaip laisvas.

PA valdymui keliami 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.

Vykdymo metu (po programos sukompiliavimo) proceso įkėlimo vieta gali keistis – tai, pavyzdžiui, gali būti susiję su laikinu procesų iškėlimu į diską (swap mechanizmu). Toks laikinas dažniausiai blokuotų procesų iškėlimas leidžia OS turėti daugiau laisvos PA ir skirti ją parengtiems vykdyti procesams. Kadangi OS gali laikinai iškelti juos į išorinę atmintį, tai procesai vykdymo metu gali užimti skirtingas vietas PA. Dėl šios priežasties programos kode negali būti fiksuotų nuorodų į atmintinę.

Kai laikoma daug procesų, reikia prižiūrėti, kad jie nemaišytų vienas kitam. Reikalaujama, kad procesai negalėtų kreiptis į kitam procesui skirtą atmintinės sritį, neturint specialaus leidimo. Kompiliavimo metu negalima patikrinti programose naudojamų adresų (nes, kaip minėta, įkėlimo vieta gali keistis). Kreipimosi adresai, vykdant procesą, turi būti patikrinami TĮ priemonėmis.

## Paprastas atmintinės valdymas

Užkloties mechanizmas – į tą pačią atmintinės sritį įkeliama vis kita proceso dalis.

Visa atmintinė gali būti paskiriama vykdomam procesui. Vartotojo požiūriu tai būtų idealus variantas, tačiau tada OS funkcionuotų visai neekonomiškai, nes procesoriui tektų laukti, kol į atmintinę bus įkeltas tas procesas.

Efektyvus PA valdymas susijęs su tam tikros apimties PA dalies skyrimu kiekvienam vykdomam procesui. Siekiant intensyviau naudoti procesorių, PA stengiamasi laikyti kuo daugiau procesų. Optimaliai skirstant PA, sprendžiami tokie uždaviniai:

* Kurį iš procesų palikti PA, kai jos trūksta naujiems procesams?
* Kokio dydžio atmintinės sritį paskirti kiekvienam procesui?
* Į kurią PA vietą įkelti kiekvieną procesą?

Procesorius kai kuriuos duomenis gali laikyti spartinančiojoje atmintinėje (cash), iš kurios jie sparčiai išrenkami.

Valdymo strategijos yra susijusios su problemomis, kurių atsiranda, keliant procesus iš antrinės į PA. Reikia spręsti, kurią proceso dalį ir kada – ar tada, kai tos dalies prireikia, ar iš anksto, kuriuos duomenis iškelti į PA, kad būtų daugiau vietos kitiems procesams.

Procesui atmintinėje gali būti skiriama ištisinė/neištisinė sritys. Procesas, kuriam priskirta ištisinė sritis, egzistuotų kaip vientisas, nuoseklių, nepertraukiamų adresų erdvėje esantis blokas. Iš pažiūros tai labai paprastas atmintinės dalijimo būdas, tačiau dėl to kyla problemų, susijusių su tinkamo dydžio laisvo bloko suradimu bei netinkamu atmintinės naudojimu. Kai procesas įkeliamas į neištisinę atmintinės sritį, jo duomenys skaidomi tam tikro dydžio gabalais (puslapiais, segmentais), kurie įkeliami į atskiras atmintinės vietas. Šiuo metodu lengviau rasti atmintinėje procesui įkelti tinkamą vietą, padidinti procesų, vienu metu esančių PA, skaičių, tačiau sunkiau taikyti. Ištisinė sritis procesams buvo priskiriama ankstyvoje OS raidos stadijoje. Neužimtos atmintinės sritis buvo skiriama vartotojo poreikiams, tačiau atsirado problemų, pradėjus kurti programas, kurių kodas netilpo į PA. Šiai problemai buvo naudojamas užkloties mechanizmas: 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 modulio vietą. Užkloties mechanizmo trūkumai – sunku taip organizuoti užklotis, kad PA būtų naudojama efektyviai. Problemų kėlė ir vykdomos programos modifikacijos.

Skirstant PA keliems procesams, naudojami šie paprasti algoritmai:

* Atminties skirstymas fiksuoto dydžio skyriais:
* Dinaminis skyrių formavimas atmintinėje;
* Skirstymas puslapiais;
* Skirstymas segmentais.

**Fiksuoti skyriai**:

PA sudalijama į keletą nepersidengiančių skyrių (dalių). Šios dalys gali būti tiek vienodo, tiek skirtingo fiksuoto dydžio. Skyrių skaičius sistemoje yra fiksuotas. Taip sudalijus PA, bet kuris procesas, kurio dydis lygus arba mažesnis už skyriaus dydį, gali būti įkeliamas į šį skyrių. Taigi, kiekvienas aktyvus procesas gauna fiksuotą atmintinės bloką, į atmintinę įkeliami keli procesai ir procesorius gali greitai persijungti nuo vieno proceso prie kito. Jeigu visi skyriai užimti, OS gali iškelti swap procesą iš jo užimamo skyriaus į diską. Visgi yra problemų: PA naudojama neefektyviai, taip pat susiduriama su vidinės fragmentacijos problema, nes, kad ir kokia maža būtų programa, jai skiriamas visas skyrius ir jame gali būti daug nenaudojamos vietos.

**Procesų įkėlimo į fiksuotus skyrius algoritmas**:

* Kai vienodo dydžio skyriai: kad procesą būtų galima įkelti į PA, reikėjo sulaukti, kol joje atsiras laisvas skyrius. Atsiradus laisvam skyriui, procesas gali būti į jį įkeliamas ir nėra svarbu, į kurį skyrių procesas įkeliamas, nes visi skyriai yra vienodo dydžio.
* Kai nevienodo dydžio skyriai: gali būti sudaroma daug eilių procesams, kurie laukia įkėlimo. Kiekviena eilė yra susieta su atskiru skyriumi; procesai pagal savo dydį yra įtraukiama į atitinkamą eilę prie skyrių.

**Dinaminis skyrių formavimas**:

Taikant šį metodą, tiek skyrių, tiek jų dydis yra kintami. Kiekvienam procesui suformuojamas tokio dydžio skyrius, kokio jis prašo. Tačiau jam pasibaigus, naujas procesas turi būti tokio pačio arba mažesnio dydžio. Naudojant šį metodą, atsiranda skylių (t.y., įterpus mažesnį procesą, lieka laisvos vietos, kurios negalima panaudoti). Tai vadinama išorine fragmentacija. Tačiau gali būti ir priešingas variantas: yra daug laisvų skyrių, tačiau kiekvieno talpa yra mažesnė negu proceso reikalavimai, todėl reikia naudoti suglaudinimo algoritmą – perstumti procesus atmintinėje taip, kad tarp jų neliktų skylių, o laisvas sritis sujungti į vieną bendrą sritį. OS turi žinoti, kurios sritys yra laisvos ir kurios užimtos. Tokio tipo informaciją OS saugo dvejetainiuose žemėlapiuose (bitmap) arba surištuose sąrašuose. Žemėlapiuose užimta sritis pažymima vienetu, o laisva – nuliu. Užimtai ir laisvai atmintinės vietai fiksuoti gali būti naudojami ir surišti sąrašai. Kiekviename sąrašo elemente pažymimas proceso (P) arba skylės (H), kurioje prasideda atitinkama sritis, adresas, srities ilgis ir nuoroda į kitą sąrašo elementą. Galimas ir dvikryptis sąrašas.

**Įkėlimo į dinaminius skyrius algoritmas:**

Procesams įkelti į pagrindinės atmintinės dinaminius skyrius atmintinės valdytojas naudoja kokį nors įkėlimo algoritmą. Procesą siekiama įkelti taip, kad kuo rečiau reikėtų glaudinti duomenis, nes tam sugaištama daug laiko. Galimi algoritmai:

* Tinkamiausio skyriaus paieška: randamas mažiausias laivas atmintinės 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. Tai nėra pats geriausias algoritmas, nes lieka daug mažų nenaudojamų skylių.
* Pirmo tinkamo skyriaus paieška: surandamas pirmas laisvas tinkamo dydžio blokas ir į jį įkeliamas procesas. Rasta skylė sudalijama į dvi dalis, į vieną įkeliamas procesas, formuojamas naujas dinaminis skyrius, o kita dalis lieka nepanaudota. Šis algoritmas gana paprastas, nes nereikia daug paieškos veiksmų, įkėlimo veiksmai atliekami atmintinės adresų pradžioje.
* Kito tinkamo skyriaus paieška: veikia taip pat, kaip ir praeitas algoritmas, tik šiuo atveju yra fiksuojama vieta, kurioje buvo rasta tinkamo dydžio skylė. Kitą kartą paieška prasideda nuo tos vietos, kur buvo sustota.

**Bičiulių sistema:**

Tai algoritmas, kuriuo bandoma išvengti tiek fiksuotų, tiek dinaminių skyrių problemų. Pradžioje visa atmintinė yra laisva, taigi pradeda, turint laisvą 2U dydžio bloką (kur U – reikšmė, priklausanti nuo konkrečios atmintinės dydžio). Procesams įkelti yra skiriami vientisi blokai, kurių dydis yra 2K. Mažiausias procesui skiriamas atmintinės blokas yra 2L, o didžiausias įmanomas išskirti blokas (visa atmintinė) yra 2U. Taigi: L <= K <= U.

Pavyzdys:

Tarkime, atsiranda poreikis įkelti S dydžio procesą. Jeigu 2U – 1 < s <= 2U, tai yra skiriamas visas blokas 2U. Priešingu atveju blokas sudalijamas į dvi vienodo dydžio 2U – 1 dalis (bičiulius). Jeigu 2U – 2 < s <= 2U – 1, tai procesui skiriama viena iš dalių (vienas bičiulis), o jei ne, tai viena dalis vėl dalijama į dvi dalis. Šos procesas kartojamas tol, kol ši sąlyga tampa galiojanti (t.y., gaunamas mažiausias blokas, kuris yra lygus arba didesnis už S. Jeigu du bičiuliai yra laisvi, jie sujungiami. I-asis skylių sąrašas apima skyles, kurių dydis yra 2i. Kai tik bičiulių pora atsiranda i-ajame sąraše, jie išmetami iš šio sąrašo ir sujungiami į vieną bendrą skylę (I + 1) sąraše.

## Fragmentacijos sąvoka. Išorinė, vidinė fragmentacija

Fragmentacija susidaro, kuomet atmintinėje lieka tuščių (laisvų) skyrių. Jie būna laisvi, nes nėra pilnai išnaudojama, kadangi, pavyzdžiui, gali skirtis skyriaus ir priskirto proceso dydžiai. Būna 2 tipų:

* Išorinė fragmentacija: [Skaityti čia](#Outer_fragmentacija) (**Dinaminis skyrių formavimas)**;
* Vidinė fragmentacija: [Skaityti čia](#Vid_fragmentacija) (**Fiksuoti skyriai)**;

# Virtualioji atmintinė

## Adresų tipai. Loginio adreso transliavimas į fizinį. MMU

Virtualiosios atmintinės sąvoka leidžia abstrahuoti fizinę atmintinę virtualia adresų erdve. 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 PA;
* Procesai gali vykdyti programas, kurių kodas tik iš dalies yra įkeltas į PA;
* Kiekvienam procesui leidžiama prieiga tik prie tam tikro PA poaibio;
* Programos gali būti kilnojamos;
* ...

Virtualioji atmintinė apibūdinama virtualia adresų erdve. Naudojami adresai: virtualieji (kuriuos naudoja procesai) ir fiziniai (rodantys realias objektų vietas PA). Programose naudojami adresai yra vadinami virtualiaisiais adresais. PA naudojami adresai yra fiziniai. Valdant virtualią atmintinę, tenka spręsti pagrindinę problemą: kaip suderinti procesų naudojamą virtualią adresų erdvę su adresų erdve PA (procesai negali tiesiogiai prieiti prie duomenų, saugomų išoriniuose įrenginiuose). Kompiuterio TĮ komponentas, atsakingas už atmintinės valdymą, yra MMU (Memory Management Unit). Jis vykdo virtualių adresų transliavimo į fizinius adresus, atmintinės apsaugos ir kitas funkcijas.

Adresų tipai:

* Fizinis adresas – nurodo objekto fizinę vietą PA;
* Virtualusis adresas – tai nuoroda į objekto vietą, kuria operuoja procesas, nepriklausanti nuo to, į kurią atmintinės vietą jis yra įkeltas.

Reikalingas tam tikras dinaminis adreso transliavimo mechanizmas, kuris pakeistų virtualųjį adresą fiziniu proceso vykdymo metu. Šiuo adreso transliavimo rūpinasi atmintinės valdymo įrenginys (MMU). Proceso vykdymo metu OS kilnoja proceso blokus tarp PA ir išorinės, OS priversta fiksuoti, kurie proceso blokai yra įkelti į PA, kurioje vietoje jie yra ir kurių proceso blokų PA nėra. Jeigu virtualioji adresų erdvė yra ištisinė, tai realūs adresai gali ir nebūti nuoseklūs. Pakartotinai įkeliama kuri nors proceso dalis nebūtinai pateks į prieš tai buvusią vietą, kadangi programos gali būti kilnojamos PA jų vykdymo metu. Santykiniai adresai yra dažniausia loginio adreso forma, naudojama modulinėse programose. Moduliai įkeliami į PA su visomis nuorodomis į atmintį. Santykiniai adresai į realius adresus gali būti transliuojami ir TĮ priemonėmis. Kai procesas yra perjungiamas į vykdymo būseną, į procesoriaus bazinį registrą yra įrašomas proceso pradžios fizinis adresas. Į registrą, saugantį proceso ribinį adresą, įkeliamas procesui skirtos srities galinis adresas. Programos kode aptikus santykinį, jo reikšmė 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.

Pranašumai:

* Suteikia procesams galimybę naudoti gerokai didesnę loginių adresų erdvę nei sistemoje esanti realių adresų;
* PA galima laikyti daugiau aktyvių procesų, kiekvienam iš jų skiriant nedidelę atmintinės sritį;

## Atminties skirstymas, taikant paprastą puslapiavimą (proceso įkėlimas, kt.)

Dauguma virtualią atmintinę sudarančių sistemų remiasi skaidymo puslapiais idėja. Tokiu atveju kiekvienas procesas yra suskaidomas į vienodo dydžio blokus, vadinamus puslapiais. Kai kurios OS puslapio dydį sutapdina su disko bloko dydžiu. PA taip pat yra sudalijama į fiksuoto puslapio dydžio blokus, vadinamus rėmais (frames). Puslapiai bei rėmai yra reliatyviai mažo dydžio, palyginti su visos atmintinės dydžiu. Atlikus tokį skirstymą, viename disko sektoriuje bus vienas proceso puslapis, kuris tilps į vieną pagrindinės atmintinės rėmą. Kadangi puslapis ir rėmas yra vienodo dydžio, todėl puslapį kilnoti gana nesunku. Į PA įkeliamo proceso puslapiai gali užimti laisvus rėmus (page frames), todėl procesui nebūtina užimti ištisinės adresų erdvės PA, proceso puslapiai gali būti įkeliami į tuo metu esančius laisvus rėmus. Nelieka problemos, su kuria buvo susiduriama dinamiškai sudalijant atmintinę skyriais procesui pasibaigus. Kadangi dydis yra gana nedidelis, taip pat yra išvengiama ir vidinės fragmentacijos. Kiti pranašumai: paprasta skirstyti PA, galima užimti bet kurį rėmą, procesai gali bendrai naudotis kai kuriais puslapiais; problemos: puslapių lentelės yra didelės apimties, sunku iš anksto skirti sritį, kurios reikės duomenims ar dėklui, nes sritis gali didėti.

**Proceso įkėlimas:**

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

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

Jeigu į programą pažvelgsime kaip sudarytą iš puslapių, tai jie bus surikiuoti logine tvarka ir pirmas programos puslapis dažniausiai atitiks programos pradžią. Tačiau įkėlimo tiek į diską, tiek į PA metu šie puslapiai nebūtinai bus išdėstyti ta pačia logine tvarka. Tokio įkėlimo į PA nauda ta, kad nebelieka išorinės fragmentacijos ir bet kurį laisvą rėmą gali užimti bet kurio proceso bet kuris puslapis. Darbo su virtualiąja atmintine principai buvo sukurti palaikyti tokiai idėjai, kad lygiagrečiai vykdomos vartotojų užduotys vienu metu turi būti PA. Tokiu atveju pereinant nuo vienos vykdomos užduotis prie kitos nereikia laukti, kol bus įkelta kita užduotis,- ji jau yra atmintinėje. Kai tik viena iš užduočių yra pernešama į išorinį atmintinės įrenginį, tuoj pat į jos vietą įrašoma kita užduotis. Tačiau atsiranda nauja problema: kadangi puslapiai gali būti išdėstyti bet kurioje atmintinės vietoje, valdant atmintinę tenka registruoti, kur yra išdėstyti kiekvieno proceso puslapiai. Be to, proceso puslapių lentelėje kiekvienam puslapiui naudojamas bitas, kuris rodo, ar atitinkamas puslapis yra įkeltas į PA. Jei, vykdant programą, aptinkamas kreipinys į atmintinę ir yra nustatoma, kad kreipiamasi į neįkeltą proceso puslapį, yra sukeliama pertrauktis, susijusi su puslapio trūkumu (page fault). Dėl to OS turi stabdyti procesą ir įkelti trūkstamą puslapį.

**Loginio adreso transliavimas:**

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

## Puslapių lentelė ir jos įrašo struktūra

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

[Praleista: puslapių lentelės. Per daug info, reikia paveiksliukų]

## Segmentavimas, skirstymas segmentais ir kt.

Bazinė atmintinės skirstymo segmentais idėja yra ta, kad naudojama ne viena su procesu susiejama virtualiosios atmintinės sąvoka, o daug virtualiųjų erdvių sričių – segmentų, pavyzdžiui, viena virtualioji adresų erdvė susiejama su programos kodu, kita su duomenimis, trečia su dėklu ir t.t. Kadangi kiekvienas segmentas egzistuoja atskiroje adresų erdvėje, jie gali didėti arba mažėti, neveikdami vienas kito. Skirstant proceso erdvę segmentais, išplečiama bazinio ir ribinio adreso idėja – atsiranda visa lentelė, sauganti bazinių ir ribinių adresų poras kiekvienam proceso segmentui. Segmentų yra gerokai mažiau nei puslapių, todėl segmentų lentelės informaciją galima saugoti registruose.

**Skirstymas segmentais:**

Segmentai siejasi su proceso sudėtinėmis dalimis. Kiekvienam segmentui, įkeliamam į fizinę atmintinę, skiriama nuoseklių adresų erdvė, o kiekvieno objekto adresas segmente yra nusakomas segmento numeriu bei poslinkiu segmente. Segmento numeris naudojamas kaip indeksas segmentų lentelėje, kuri sudaryta iš bazinio ir ribinio adreso porų. Transliuojant adresą į fizinį adresą atmintinėje, imamas bazinis segmento adresas, pridedama poslinkio reikšmė ir gautas dydis lyginamas su ribiniu to segmento adresu. Jeigu gauta reikšmė yra didesnė už ribinę reikšmę, gaunama segmentavimo klaida. Šis skirstymo būdas kompiliatoriams yra paprastesnis. Kompiliuota programa pateikiama kaip segmentų rinkinys. Kiekviena programos kode esanti procedūra užima atskirą segmentą, kuris turi pradinį nulinį adresą, o kiti adresai yra nurodomi segmento pradžios atžvilgiu. N-oji procedūra kviečiama, kviečiant N-ąjį segmentą. Didelė problema yra ta, kad visą segmentą reikia įkelti į nuosekliai einančius PA adresus, o tokią tinkamą sritį surasti sunku ir jį taikant neišvengiamai susiduriama su išorine fragmentacija.

**Bendrai naudojami segmentai:**

[Reikia paveiksliuko ☹]

**Segmentų apsauga ir prieigos kontrolė:**

Dažniausiai naudojami apsaugos bitai, kuriais nusakoma, ar procesas gali atlikti skaitymo, rašymo, kodo vykdymo arba pridėjimo (append) veiksmus su šiuo segmentu. Tam galima gauti įvairias prieigas kontrolę:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Moda** | **Skaitymo bitas** | **Rašymo bitas** | **Vykdymo bitas** | **Aprašas** | **Taikymas** |
| Nulinė moda | 0 | 0 | 0 | Prieiga neleistina | Saugus |
| 1-oji moda | 0 | 0 | 1 | Leidžiama tik vykdyti | Leidžiama procesams, kurie negali keisti ar kopijuoti šio segmento, bet gali vykdyti |
| 2 moda | 0 | 1 | 0 | Leidžiama tik rašyti | Nenaudinga galimybė, nes negali skaityti |
| 3 moda | 0 | 1 | 1 | Leidžiama rašyti ir vykdyti |
| 4 moda | 1 | 0 | 0 | Leidžiama tik skaityti | Informacijos skaitymas |
| 5 moda | 1 | 0 | 1 | Leidžiama skaityti ir vykdyti | Programa gali būti kopijuojama, vykdoma, bet ne modifikuojama |
| 6 moda | 1 | 1 | 0 | Leidžiama skaityti ir rašyti | Apsaugo duomenis nuo klaidingo bandymo juos vykdyti |
| 7 moda | 1 | 1 | 1 | Neribota prieiga | Garantuojama patikimiems vartotojams |

## Puslapių mainai (keitimo strategijos, puslapių kilnojimas)

**Patys puslapių mainai:**

OS, įkeldama puslapius į PA, gali elgtis įvairiai. Tai gali būti atliekama atsiradus kreipiniui į tam tikrą, PA dar nesantį, proceso puslapį, todėl iškyla poreikis įkelti puslapį (demand paging) arba OS gali bandyti nustatyti, kurių puslapių procesui reikės, ir iš anksto įkelti keletą puslapių. Kiekvienam procesui priskiriamos atmintinėje esančių puslapių skaičiaus minimumo bei maksimumo reikšmės (kiek mažiausiai ir daugiausiai galės turėti puslapių skaičių procesas PA). Kai laisvos PA yra mažiau nei leidžiama, OS iškelia tuos proceso puslapius, kurie viršija minimalią reikšmę. Kai ne visi proceso puslapiai yra įkeliami į PA, puslapių lentelėje reikia pažymėti, ar puslapis yra PA, ar diske. Tam naudojamas bitas, vadinamas galiojimo bitu. Jeigu kreipiamasi į puslapį, kurio galiojimas bitas neįjungtas, tai galėtų reikšti, kad atitinkamą puslapį reikia įkelti. Susidaro puslapio klaidos situacija ir kreipiamasi į OS. OS, norėdama įkelti šį trūkstamą puslapį, turi surasti laisvą vietą (laisvą rėmą) PA. Jeigu nėra, ji turi išlaisvinti vieną iš užimtų rėmų. Atlaisvinimas reiškia, kad šiame rėme esantį puslapį reikės iškelti į diską. Jeigu puslapis nebuvo modifikuotas, tai jo įrašyti į diską nereikia. Todėl tikrinama, ar šiame rėme esantys dydžiai buvo keisti. Jeigu buvo modifikuotas, o tai rodo atitinkamas modifikacijos bitas, tai reikia šį puslapį perrašyti į diską: aktyvinama rašymo į diską užduotis, padaroma pakeitimų puslapių lentelėje.

**Keitimo strategijos:**

Keičiant puslapius, reikia nuspręsti, vietoj kurio PA esančio puslapio bus įkeliamas naujas puslapis iš disko. Gali būti taikoma tiek vietinė politika, tiek visuotinė keitimo politika. Kiekvienam sistemoje vykdomam procesui gali būti skiriamas fiksuotas rezidentinių puslapių skaičius. Šis skaičius priklauso nuo veikiančių procesų skaičiaus bei bendro, visiems procesams skirto PA rėmų skaičiaus. Jei naudojama vietinė puslapių keitimo politika, tai joks procesas negali atimti kitam procesui priklausančio fizinės atmintinės rėmo. Kai vykdomas kurio nors proceso puslapių keitimas, susijęs su puslapio trūkumu, iškelti parenkamas vienas iš to paties proceso puslapių, įkeltų į PA. Taikant vietinę puslapių keitimo politiką, vienas iš to paties proceso puslapių keičiamas nauju puslapiu. Jeigu skiriamas kintamas puslapių skaičius, tai, vykdant procesą, šis puslapių kiekis gali kisti. Kintamas rezidentinių puslapių skaičius įgalina OS vykdyti tiek vietinę, tiek visuotinę keitimo politiką. Vykdant visuotinę keitimo politiką, puslapis, kuris parenkamas keisti, nebūtinai turi priklausyti šiam procesui. Puslapių mainams naudojama keletas algoritmų, kurie padeda efektyviai parinkti, kurį puslapį iškelti:

* Keičiamas puslapis, kurio prireiks vėliausiai;

Pirmasis algoritmas, pagal kurį reikėtų keisti tą puslapį, kurio prireiks vėliausiai, yra panašus į procesams planuoti naudojamą algoritmą – trumpiausias procesas vykdomas pirmas. Šis algoritmas būtų optimalus, bet jis nėra praktiškai taikomas, nes sunku nustatyti, kokiais kreipiniais, į kurį puslapį ir kokia tvarka bus kreipiamasi, vykdant procesą, OS paprastai to iš anksto nežino.

* Keičiamas atsitiktinai parinktas puslapis;

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

* Keičiamas pastaruoju metu nenaudotas puslapis (NRU);

Keičiant pastaruoju metu nenaudotus puslapius, stengiamasi atmintinėje išlaikyti tuos puslapius, į kuriuos neseniai buvo kreiptasi. Šiuo tikslu nagrinėjami du bitai, kurie yra susiję su kiekvienu puslapiu: bitas R, kurio vienetinė reikšmė rodo, kad į puslapį yra kreiptasi, ir bitas M, kuris rodo, kad puslapis yra modifikuotas. Jie atnaujinami, vykstant kreipiniams į puslapį, juos įjungia TĮ. Taikant NRU algoritmą, pasirenkamas atsitiktinis puslapis iš žemiausias netuščios klasės. NRU yra gana paprastas, nors ir nėra optimalus, tačiau gana gerai funkcionuoja.

* Puslapiai keičiami, laikantis FIFO disciplinos;

Keičiamas tas puslapis, kuris yra įkeltas seniausiai. Nors iš pažiūros tai ir teisinga taktika, tačiau rezultatai nebūna geri, nes vienodu dažniu yra išmetami tiek tie puslapiai, kurie nėra dažnai naudojami, tiek tie, į kuriuos nuolat kreipiamasi. Į fizinę atmintinę sudaroma įkeltų puslapių grandinėlė, ties kurios pradžiai yra seniausiai įkelti puslapiai. Tai taip pat gana paprastas algoritmas, tačiau retai taikomas.

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

* Taikomas laikrodžio algoritmas;

Šis algoritmas yra efektyvesnis nei antrojo šanso algoritmas, nes puslapių nereikia nuolat permesti į grandinėlės galą, bet iš esmės naudojami tie patys principai, kaip ir antrojo šanso algoritme. Tik naudojamas žiedinės įkeltųjų puslapių sąrašas, kuriame laikrodžio „rodyklė“ rodo į seniausią puslapį. Jeigu šio puslapio bito R reikšmė lygi vienetui, tai ji keičiama į nulį, o rodyklė pasislenka prie kito puslapio ir procesas kartojamas, kol randamas puslapis, kurio R reikšmė lygi nuliui. Jeigu ji lygi nuliui, tai naujas puslapis įkeliamas vietoj šio ir „rodyklė“ taip pat pasislenka link kito puslapio.

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

Algoritmas yra tam tikra optimalaus algoritmo aproksimacija. Taikant laikoma, kad dažnai pastaruoju metu naudoti puslapiai bus naudojami ir ateityje, todėl iškeliamas tas puslapis, kuris buvo naudotas mažiausiai. Tiksliai įdiegti tokį algoritmą nelengva. Reikėtų turėti visų PA esančių puslapių nuoseklų sąrašą, kurio priekyje būtų dažniausiai naudoti puslapiai. Daugiausiai sąnaudų reikėtų nuosekliam puslapių sąrašui palaikyti. Tačiau yra alternatyvūs sprendimai. Vienas iš jų: naudoti N \* N dydžio matricą, kur N – puslapių rėmų skaičius PA. Pradžioje visa matrica užpildoma 0. Kai kreipiamasi į K-ąjį rėmą, TĮ priemonėmis matricos K-osios eilutės elementai nustatomi į vienetinę padėtį, o paskui K-ojo stulpelio elementai – į 0. Eilutė, kurios dvejetainė reikšmė mažiausia, bet kuriuo laiko momentu rodo numerį rėmo, kuriame esantis puslapis turi būti pakeičiamas, taikant LRU algoritmą.

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

Kadangi ne visos mašinos gali pasinaudoti tokio tipo skaitikliu todėl naudojamas NFU algoritmas. Jis naudoja programinius skaitiklius, kurių reikšmės pradžioje yra 0. Po kiekvienos laikrodžio mechanizmo generuojamos pertraukties, visų puslapių, į kuriuos paskutiniame laiko intervale buvo kreiptasi, skaitikliai padidinami vienetu. Taigi skaitikliai rodo kreipimųsi į puslapius dažnį. Puslapis su mažiausia reikšme gali būti keičiamas nauju puslapiu. Trūkumas, kad matuojamas tik kreipinių dažnis ir visai neatsižvelgiama į kreipinių laiką. Taip pat, skaitiklio reikšmė gali būti didelė dar kurį laiką, tačiau puslapiu niekas nesinaudoja, todėl pasirenkami naudingi puslapiai, o nenaudingi lieka. Sendinimo algoritmas yra modifikacija, kuri bando įvertinti kreipimosi laiką. Esmė, kad kiekvieną kartą, prieš keičiant puslapių skaitiklių reikšmes, skaitiklių reikšmės perstumiamos vienu bitu į dešinę, o paskui prie pirmojo iš kairės pusės esančio bito pridedamos naujos bitų R reikšmės. Nuo LRU skiriasi, jog sendinimo algoritmas gali atsekti tik vėliausių 16 kreipinių eigą.

**Puslapių kilnojimas:**

Rezidentinė proceso puslapių grupė – tai toks į PA įkeltų puslapių skaičius, kuriam esant procesas gali būti efektyviai vykdomas. Jei PA įkelta per mažai puslapių, tai vykdomas procesas dažnai pertraukiamas dėl puslapio trūkumo. Pertraukimo metu OS yra priversta įkelti trūkstamą proceso puslapį ir juo pakeisti vieną iš atmintinėje esančių puslapių. Jeigu tokios pertrauktys dažnos, tai OS pradeda daug laiko skirti vien puslapių mainams – puslapiams kilnoti tarp disko ir PA. Toks dažnas kilnojimas vadinamas šiukšlinimu (trashing), nes OS didelę laiko dalį nevykdo procesų. Vienas iš sprendimo būdų būtų procesams reikalingo rezidentinio puslapių skaičiaus nustatymas. Jeigu jį būtų galima nustatyti, tai atliekant ilgalaikį planavimą, galima būtų paleisti tik tuos procesus, kurių atmintinės srities poreikiai galėtų būti tenkinami. O jeigu dideli poreikiai? Siūloma taikyti rezidentinės aibės modelį (working set model), pagal kurį procesas gali būti laikomas PA tik tokiu atveju, jeigu visi puslapiai, kuriais procesas tuo metu naudojasi, gali būti PA. Jeigu puslapių poreikiai auga, o PA nėra vietos, procesas iškeliamas iš PA. Išmetus kiti užbaigiami greičiau. Dažniausia kliūtis – reikalavimas sekti rezidentinius puslapius, procesų kreipinius į puslapius, keisti šių puslapių aibę, įkeliant naujus ir išmetant senus puslapius.

# I/O įtaisai

## I/O įtaisai (bendrai)

OS valdo daug įvairių įvesties ir išvesties įtaisų. Jų valdymui skiriama nemaža dėmesio, nes jie yra labai skirtingi, jie skiriasi vienas nuo kito įvairiais aspektais. Kai kurie iš jų yra skirti bendrauti su vartotoju, tokių įtaisų pavyzdžiai gali būti: spausdintuvai, vaizduokliai, klaviatūra, pelė. Kiti įtaisai, kaip diskai, magnetinės juostos, sensoriai ir kt., skirti tik komunikuoti su elektroniniais įrenginiais. Įtaisai skiriasi jų kontrolės sudėtingumu, duomenų kodavimu. Skiriasi ir duomenų perdavimo vienetais. Šiuo požiūriu įtaisai gali būti skirstomi į blokinius ir simbolinius. Blokiniuose įtaisuose duomenys yra saugomi ir į juos yra perduodami tarpusavyje nepriklausomais fiksuoti ilgio blokais. Simboliniai pasižymi tuo, kad duomenys į juos/iš jų perduodami kaip baitų srautas.

## I/O valdymo tikslai

Pagrindiniai 2 tikslai:

* Siekiama užtikrinti efektyvų sistemos funkcionavimą;
* Siekiama pateikti vartotojui bendrą, nepriklausantį nuo įrenginio, sąveikos su įrenginiais mechanizmą.

Dauguma išorinių įrenginių yra gerokai lėtesni nei procesorius, I/O operacijų greitis nesuderinamas su procesoriaus atliekamų veiksmų greičiu. Efektyvų funkcionavimą siekiama užtikrinti, taikant multiprogramavimo principus – leidžiama procesus, kurie laukia I/O, užblokuoti, o kiti yra vykdomi. Jeigu visi yra užblokuojami, taikomas pristabdymo mechanizmas, iškeliant dalį į išorinius įrenginius ir į jų vietą įkeliant naujus procesus. Jeigu programa skaito failą, tai vartotojui, rašančiam programą, turėtų būti nesvarbu, ar tas failas disketėje, ar standžiajame diske, programa neturi būti keičiama, pritaikant ją kiekvienam konkrečiam įrenginiui. Kitas svarbus momentas yra įrenginio generuojamų klaidų apdorojimas. Paprastai stengiamasi šio tipo klaidas apdoroti kaip galima arčiau pačių įrenginių. Kontroleris bando pataisyti ją pats, jei nepavyksta, klaidą apdoroti perduodama įrenginio tvarkyklei. Kadangi norima užtikrinti, kad vartotojas galėtų vienodai traktuoti visus I/O įrenginius, stengiamasi nuo jo paslėpti įvairias I/O įrenginių detales, žemo lygio funkcijas.

## Įrenginių kontrolieriai

Tai sąsajos korta arba modulis, kuris „kontroliuoja“ išorinius įrenginius, pavyzdžiui, standžiojo disko kontroleris. Dažniausiai jie turi nedidelio dydžio (kelių kilobaitų) atmintinę ir primityvius bendros paskirties procesorius, bet būna ir išimčių (kaip spausdintuvų). Kiekvienas turi keletą registrų, kurie naudojami kaip komunikacija su procesoriumi. OS gali perduoti tam tikras komandas kontrolerio valdomam įrenginiui. Skaitydama iš šių registrų, OS gali sužinoti apie įrenginio būseną. Naudojami I/O prievado numeriai. 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 2 sudedamąsias dalis:

* Kontrolerį ir įrenginio sąsają;
* Kontrolerį ir kompiuterių sistemos sąsają.

Kontroleris su įrenginiu jungiamas per įrenginio jungtį. Pertrauktys iš kontrolerio siunčiami TĮ IRQ (Interrupt ReQuest line) – pertraukčių siuntimo linija. OS aptikus pertrauktį, programa nutraukiama.

[Neužbaigta, per daug teksto...]

## I/O veiksmų valdymas

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

* Programuojamosios I/O;

OS perduoda proceso I/O komandą įrenginio kontroleriui, palikdama procesą, iškvietusį I/O veiksmus, aktyviojo laukimo būsenos. Šio pagrindinė problema, kad procesorius yra priverstas ilgai laukti, kol I/O modulis pasiruošia priimti ar perduoti duomenis.

* Pertrauktimis grindžiamos I/O;

Procesoriui sudaroma galimybė vykdyti kitus veiksmus, kol vykdomi įvesties/išvesties veiksmai. Šis metodas reikalauja sudėtingesnės TĮ ir PĮ, bet leidžia efektyviau išnaudoti procesoriaus laiką. Procesui iškvietus sisteminį kvietinį, susijusį su I/O veiksmais, procesas stabdomas, spausdinamieji dydžiai perkeliami į branduolio buferį, OS pateikia I/O komandą kontroleriui ir pirmasis simbolis yra perkeliamas į kontrolerio duomenų registrą, kai tik šis gali jį priimti. Kontroleris praneša apie simbolio spausdinimo pabaigą ir pasiruošimą priimti kitą simbolį, siųsdamas pertraukties signalą. OS apdoroja: procesas stabdomas, apdorojanti programa patikrinta, ar liko neatspausdintų simbolių, ir, jeigu taip, perkelia kitą simbolį, jei ne – aktyvinama užblokuota programa. Pagrindinis trūkumas: pertrauktys generuojamos ties kiekvienu simboliu.

* Tiesioginės prieigos atmintinės (DMA – Direct memory Access).

OS pateikia I/O komandą DMA moduliui ir normaliai funkcionuoja toliau. DMA kontroleris inicijuoja skaitymo arba rašymo komandą. DMA kontroleris perduoda priimamus baitus tiesiai į PA be procesoriaus įsikišimo. Procesoriaus darbas nutraukiamas tik tada, kai visas informacijos blokas yra perduotas į PA.

## Įrenginių tvarkyklės

Kiekvieno įrenginio kontroleris turi registrus, į kuriuos įrašomos komandos, kurias turi atlikti įrenginys. Šių registrų skaičius, komandos, kuriomis operuoja atskiras įrenginys, gali skirtis, imant vieną ar kitą įrenginį. Dėl to kiekvienas prie kompiuterio prijungtas įrenginys reikalauja tam tikros to įrenginio valdymo programos, kuri yra vadinama įrenginių tvarkykle. OS paprastai skirsto visas įrenginių tvarkykles į 2 kategorijas. Tai tvarkyklės, skirtos blokiniams įrenginiams (pvz., diskams) ir simboliniams įrenginiams (pelei, klaviatūrai). Dauguma OS apibrėžia 2 sąsajas, kurias turi tenkinti blokinių ir simbolinių įrenginių tvarkyklės. Šias sąsajas sudaro rinkinys procedūrų, kurias gali naudoti likusi OS dalies, kad priverstų įrenginių tvarkyklę atlikti tam tikrus veiksmus. Jos atsakingos už keletą funkcijų: jos turi priimti skaitymo/rašymo užklausas; turi sugebėti patikrinti įrenginį, sugebėti jį inicijuoti.

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

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

* Bendros sąsajos su įvairiomis įrenginių tvarkyklėmis sudarymą;

Pagrindinė paskirtis – atlikti tas I/O funkcijas, kurios yra bendros visiems įrenginiams, bei užtikrinti vienodą sąsają vartotojo lygio programoms. Užtikrinus tai, daug paprasčiau įdiegti į sistemą naują įrenginį, nes aiškiai žinoma, kokias funkcijas ji turi atlikti.

* Buferių naudojimą I/O veiksmams;

Kaip minėta, mainai vyksta lėtai. Nenaudojant buferių, procesai yra blokuojami ir turi laukti, kai tik jie paprašo įvesties ar išvesties. Nuskaičius simbolį, generuojama pertrauktis, šią pertrauktį apdorojanti procedūra perduoda nuskaitytą simbolį į vartotojo sritį. Įrenginys taip pat turi sustoti ir laukti, kol procesas duos kitą skaitymo ar rašymo komandą. OS I/O užklausoms gali skirti buferį branduolio srityje ir liepti pertrauktis apdorojančiai programai dėti informaciją čia. [Reiktų paveiksliukų ir dar šiek tiek teksto, bet jo ten per daug]

* Įrenginių priskyrimą ir priskyrimo atšaukimą;

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 įgyvendinamos kaip sisteminis kreipinys, prašantis atverti įrenginį atitinkantį specialųjį failą. Įrenginys atlaisvinamas, užveriant šį failą.

* Nepriklausomo nuo įrenginio bloko dydžio palaikymą.

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.

**Vartotojo lygmens įranga:**

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

# Diskai

## Fizinė disko struktūra

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

## Disko užklausų vykdymas

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

* Paieškos;

Laikas, per kurį disko svirtelės nustatomos ties reikiamu cilindru. Jis priklauso nuo to, kaip greitai gali judėti svirtelės.

* Sukimosi;

Laikas, per kurį reikiamas sektorius atsiranda ties nuskaitymo galvute.

* Perdavimo laiko.

Laikas, per kurį duomenys nuo disko paviršiaus perduodami į disko kontrolerį, kuris persiunčia juos į PA.

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

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

## RAID architektūra

Diskų funkcionalumui padidinti ir duomenų praradimams minimizuoti naudojama RAID (Redundant Array of Independent Disks) architektūra. RAID technologija apima RAID 0 – RAID 7. Naudojant architektūrą, siekiama sutrumpinti informacijos mainų laikus bei tam tikru perteklumu padidinti saugomos informacijos patikimumą. Naudojami tam tikri principai:

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

RAID architektūros (šiek tiek sutrumpintai):

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