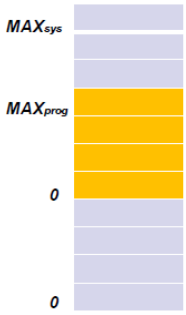
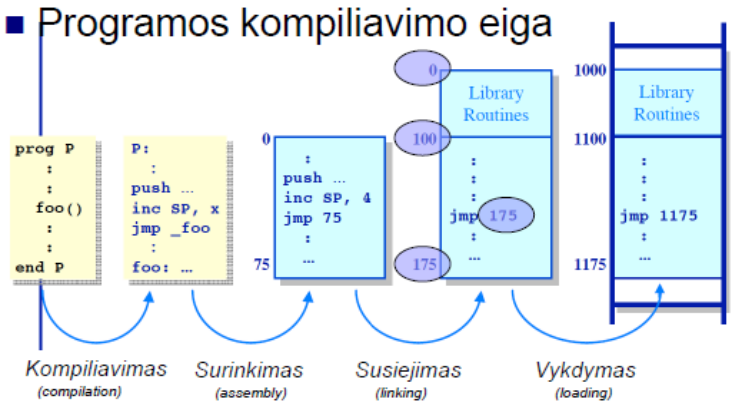
1. Atmintinės Valdymas - Įvadas (08T, 10T)

Pagrindinės atmintinės valdymo sąvokos

- Fizinė adresų sritis – techninės įrangos palaikoma adresų sritis (nuo 0 iki MAX_{sys})
- Loginė/virtuali adresų sritis – procesui matoma atminties sritis (nuo 0 iki MAX_{prog})
- Fizinis (absoliutusis) adresas, loginis (bazinis, realiatyvus) adresas



Loginių adresų formavimas:



Reliatyvus adresas – išreiškiamas bazinio adreso atžvilgiu (susiejimo etape tas 175)

Fiziniai adresai – priskiriami programos vykdymo metu (vykdymo etape tas 1175)

Antrinė atmintinė yra skirta saugoti ilgalaikiams duomenims, programoms (diskai, magnetinės juostos...), o pagrindinė atmintinė yra skirta saugoti programos, kuri vykdoma šiuo metu, duomenims.

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

Atmintinės valdymo schemos sprendžia problemas, kurių atsiranda keliant procesus iš antrinės atmintinės į pagrindinę:

- Kurią proceso dalį įkelti ir kada?
- Kur patalpinti įkeliamą procesą ar jo dalį?
- Kuriuos duomenis iškelti, kad būtų daugiau vietos kitiems, įkeliamiems procesams?

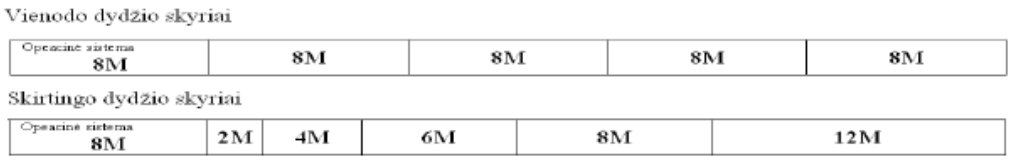
Būna:

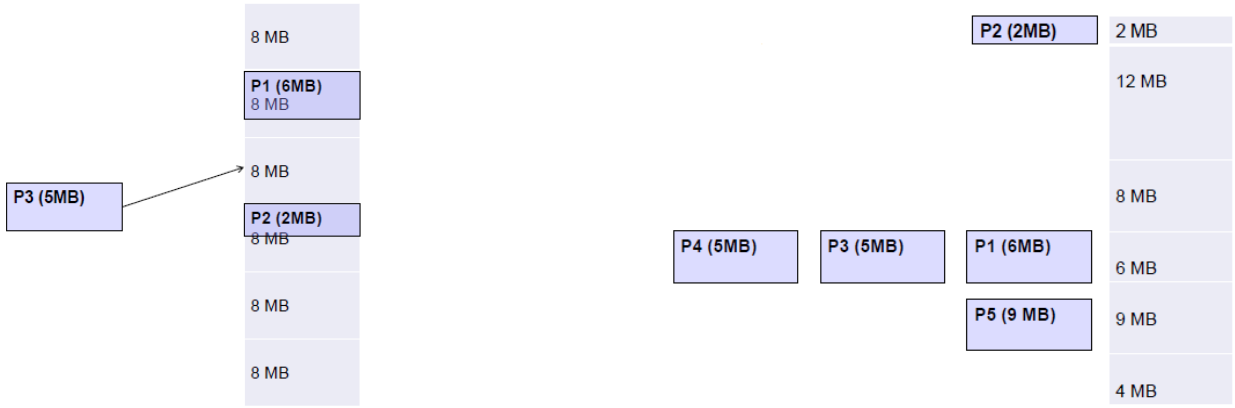
- Nuoseklių (ištisinių adresų) zonos skyrimas
    - Procesas egzistuotų kaip vientisas, nuoseklių adresų erdvėje esantis blokas
    - Tai labai paprastas atmintinės dalinimo būdas
    - Atsiranda problemos:
        - Tinkamo dydžio laisvo bloko atradimas
        - Netinkamas atmintinės panaudojimas
-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

Paprastas atmintinės valdymas (fiksuoti, dinamiškai formuojami skyriai. Procesų įkėlimo į šiuos skyrius algoritmai. Bičiulių sistema (angl. Buddy System))

Fiksuoti skyriai:

- Pagrindinė atmintis yra sudaloma į eilę nepersidengiančių skyrių (dalių). Šios dalys gali būti tiek vienodo, tiek skirtingo dydžio
- Procesas, kurio dydis yra mažesnis arba lygus skyriaus dydžiui, gali būti patalpinamas į šį skyrių
- Procesorius gali greitai persijungti tarp procesų
- Naudojami keli ribiniai registrai apsaugai nuo to, kad procesai negadintų vienas kito duomenų ar programos, kreipdamiesi į ne jam skirtą atmintinės bloką – tokie kreipiniai neleidžiami
- Jei visi skyriai yra užimti, operacinė sistema gali iškelti (swap) procesą iš jo užimamo skyriaus



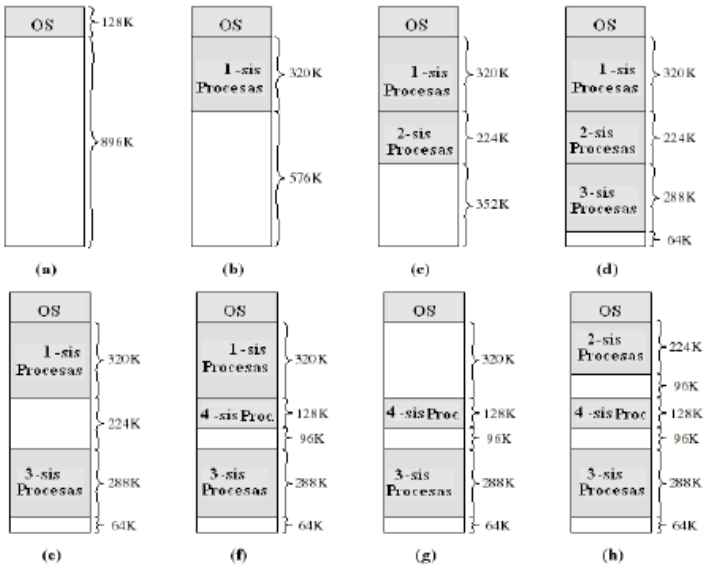


Algoritmas:

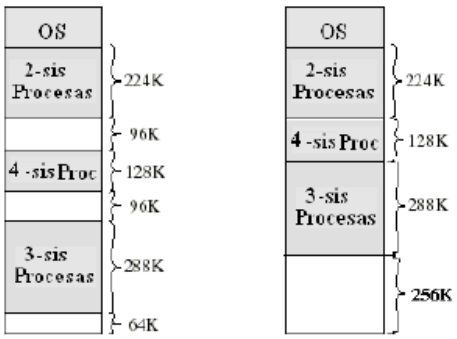
- Vienodo dydžio skyriuose procesai tiesiog dedami į tą skyrių, kur yra laisvos vietos. Jei keliant procesą į skyrių, visa eilė yra užimta užblokuotų procesų, užblokuotą procesą galima iškelti į antrinę atmintinę (pirma nuotrauka).
- Nevienodo dydžio skyriuose gali būti sudaroma daug eilių, kurios laukia tam tikro skyriaus atsilaisvinimo. Procesai pagal savo dydį laukia prie būtent to skyriaus. Taip bandoma sumažinti vidinės fragmentacijos problemą – procesą įtraukiant į eilę, atrenkamas mažiausias įmanomas jam sutalpinti skyrius (antra nuotrauka).

Dinaminiai skyriai:

- Taikant šį principą skyrių **kiekis**, jų **dydis** yra kintami
- Kiekvienam procesui jį talpinant pagrindinėje atmintinėje yra išskiriamas **tokio dydžio skyrius, kokio jis prašo**



Suspaudimas:

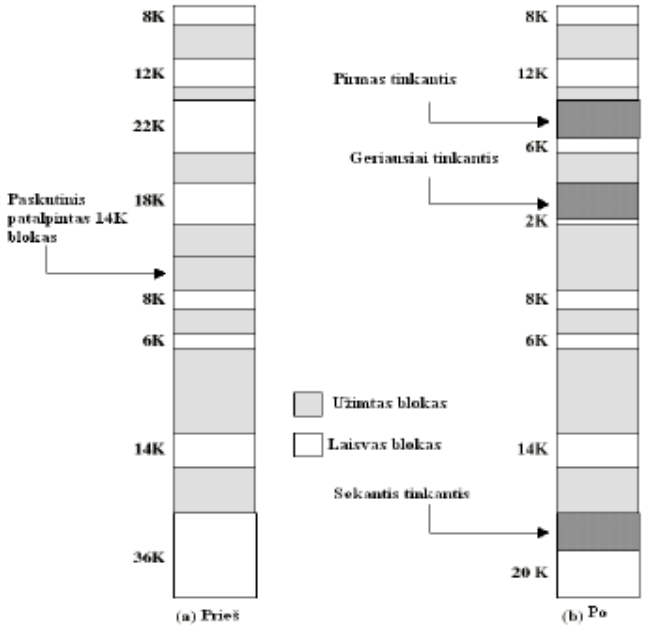


Talpinimo algoritmai dinaminių skyrių atveju:

- Paskirtis – nuspręsti, kurį laisvą atmintinės skyrių („skylę“) paskirti procesui, jį talpinant pagrindinėje atmintinėje

- Galimi ir naudojami šie algoritmai:
    - Geriausiai tinkančio - paieška yra susijusi su mažiausio neužimto bloko suradimu: likęs laisvas fragmentas bus gaunamas mažiausias
    - Pirmo tinkamo - taikant „pirmo tinkamo“ paiešką, skylės radimas sukasi apie atmintinės pradžią, jį taikant gaunama mažesnė fragmentacija nei „sekančio tinkamo“ atveju
    - Sekančio tinkančio - taikant „sekantis tinkamas“ algoritmą dažnai naujai talpinamam procesui yra priskiriamas didžiausias blokas, esantis pagrindinės atmintinės pabaigoje
    - Blogiausiai tinkančio – kartais yra naudojamas algoritmas, kuris proceso patalpinimui ieško blogiausiai tinkančios savo dydžiu „skylės“. Randamas didžiausias savo apimtimi blokas, į kurį patalpinus procesą jame lieka didžiausia neišnaudota erdvė. Yra tikimasi, kad į šią neišnaudotą erdvę vėliau bus galima patalpinti kitą procesą

Talpinimo algoritmų pavyzdys, talpinant 16K procesą atmintinėje:



Bičiuliška sistema (Buddy System):

- Bičiuliška sistema, tai algoritmas, kuriuo bandoma apeiti tiek fiksuotų, tiek dinaminių skyrių problemas
- Modifikuota šio algoritmo versija yra naudojama UNIX SVR4
- Atmintinės blokai, kurie yra išskiriami procesams yra 2^k dydžio

Algoritmo žingsniai:

- Pradžioje visa atmintinė yra laisva, taigi pradedama turint 2^U dydžio bloką. Tarkime, atsiranda pareikalavimas patalpinti S dydžio procesą
    - Jei $2^{U-1} < S <= 2^U$, tai procesui išskiriamas visas blokas 2^U
    - Priešingu atveju blokas sudalomas į dvi vienodo dydžio 2^{U-1} dalis (bičiulius)
        - Jei $2^{U-2} < S <= 2^{U-1}$, tai procesui išskiriama viena iš dalių (vienas bičiulis), o jei ne, tai viena iš dalių vėl yra daloma į dvi dalis

- Šis procesas yra kartojamas tol, kol gaunamas mažiausias blokas, kuris yra lygus arba didesnis nei S
- Pavyzdys: Turim 1MB = 2^{10}KB = 1024KB. Procesas prašo 100KB
- $2^9 = 512 < 100 < 2^{10} = 1024$ – netinka
- $2^8 = 256 < 100 < 2^9 = 512$ – netinka
- $2^7 = 128 < 100 < 2^8 = 256$ – netinka
- $2^6 = 64 < 100 < 2^7 = 128$ – tinka, skiriam 128KB

1 MB blokas	1 M			
100KB A procesas	A = 128 K	128 K	256 K	512 K

Bičiuliška sistema:

- Jei du bičiuliai tampa laisvais, bičiuliai yra apjungiami
- Operacinė sistema palaiko keletą sąrašų apie esančias skyles
  - I-tas sąrašas apima skyles, kurių dydis yra 2^i
  - Kai tik pora bičiulių atsiranda i-tame sąraše, jie yra išmetami iš šio sąrašo ir apjungiami į vieną bendrą skylę (I + 1) sąraše
- Atsiradus naujai užklausai, t.y. norint patalpinti k dydžio procesą, tokį:
  - Kuriam galioja: $2^{i-1} < k <= 2^i$
  - Yra patikrinamas i-tas sąrašas. Jei šis sąrašas yra tuščias, tikrinamas (i + 1) sąrašas
  - Jame radus skylę ji bus sudaloma į du bičiulius, viena iš dalių bus priskirta procesui, o kita įtraukta į i-tą sąrašą

Pavyzdys:

1 MB blokas	1 M				
100KB A procesas	A = 128 K	128 K	256 K	512 K	
240KB B procesas	A = 128 K	128 K	B = 256 K	512 K	
64KB C procesas	A = 128 K	C = 64K	64 K	B = 256 K	512 K
256KB D procesas	A = 128 K	C = 64K	64 K	B = 256 K	D = 256 K	256 K
Baigiasi B procesas	A = 128 K	C = 64K	64 K	256 K	D = 256 K	256 K
Baigiasi A procesas	128 K	C = 64K	64 K	256 K	D = 256 K	256 K
75KB E procesas	E = 128 K	C = 64K	64 K	256 K	D = 256 K	256 K
Baigiasi C procesas	E = 128 K	128 K	256 K	D = 256 K	256 K	
Baigiasi E procesas	512 K			D = 256 K	256 K	
Baigiasi D procesas	1 M				

Fragmentacijos sąvoka. Išorinė, vidinė fragmentacija.

Fragmentacija – tai toks reiškinys, kai saugojimo vieta yra naudojama neefektyviai, ko pasekoje mažėja talpa arba našumas, o dažnai ir abu.

Vidinė fragmentacija – priskirta (angl. allocated) daugiau atminties vietos, nei reikia, todėl lieka palikti nemaži gabalai nenaudojamos atminties vietos. Pavyzdžiui, atmintis gali būti paskirstyta į gabalus, kurie

dalinasi iš 4, 8 arba 16. Taigi, jei programai reikia 29 baitų, ji užima 32 baitų gabalą, kur likę 3 baitai lieka nenaudojami.

Išorinė fragmentacija – ji iškyla, kai laisva atmintis lieka išskaidyta į mažo dydžio blokus. Nesvarbu, kad turime laisvos vietos, bet ji yra padalinta į per mažas dalis, kad būtų panaudota tam tikrų programų. Žodis „išorinė“ reiškia, kad nenaudojama vieta yra už priskirtų (angl. allocated) vietų. Pavyzdžiui, įsivaizduokime, kad programai yra priskiriami trys iš eilės einantys atminties blokai, o kai galiausiai vidurinis blokas bus išlaisvintas, jame liks laisvos vietos, tačiau kai kurioms kitoms programoms gali prireikti daugiau vietos, nei yra šiame bloke.

0x0000	0x1000	0x2000	0x3000	0x4000	0x5000
A	B	C			
A		C			
A	C				

Fragmentacija naudojant fiksuotus skyrius:

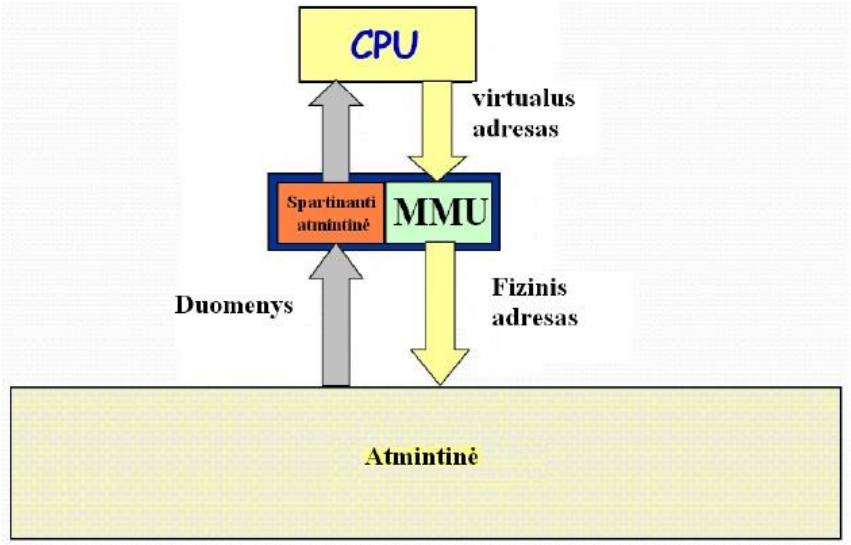
- Atsiranda **vidinės fragmentacijos** problema:
  - Nes nežiūrint kokia maža programa būtų – jai skiriamas visas skyrius ir jame gali būti daug nenaudojamos vietos
  - Skirtingo fiksuoto ilgio skyriai kiek sumažina šią problemą, tačiau problema išlieka

Fragmentacija naudojant dinaminius skyrius:

- Pagrindinėje atmintinėje labai greitai gaunamos skylės, kurios yra per mažos bet kokio proceso patalpinimui, ir reikia atlikti suspaudimus (**išorinė fragmentacija**)

2. Atmintinės Valdymas – Virtualioji atmintinė (08T, 10T)

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



- **Fizinis adresas** (absoliutinis adresas) nurodo objekto fizinę vietą pagrindinėje atmintinėje.

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

➢ Taikant virtualiosios atmintinės idėjas, procesai arba atskiros jo dalys gali būti ne tik įkeliamos bet ir iškeltos iš pagrindinės atminties, bei po to pakartotinai įkeltos.
➢ **Santykinis adresas** yra loginio adreso pavyzdys, kai adresas yra išreiškiamas kurio nors žinomo programos taško atžvilgiu.
➢ Santykiniai adresai yra dažniausia loginio adreso forma, naudojama modulinėse programose.
➢ Kai procesas yra perjungiamas į vykdymo būseną, į procesoriaus bazinį registrą yra įrašomas proceso pradžios fizinis 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.

- Reikalingas tam tikras dinaminis adreso transliavimo mechanizmas, kuris pakeistų virtualųjį adresą fiziniu adresu proceso vykdymo metu. Šiuo adreso transliavimu rūpinasi atmintinės valdymo įrenginys (MMU).
- **MMU** – memory management unit, tai atminties valdymo įrenginys. **Visi OS atminties adresai keliauja per jį ir jis virtualius adresus verčia fiziniais.**

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

Puslapiavimo idėja:

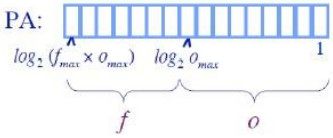
- Kiekvienas procesas yra suskaidomas į vienodo dydžio blokus, vadinamus puslapiais.
- Pagrindinė atmintinė taip pat yra sudalijama į fiksuoto puslapio dydžio blokus, vadinamus rėmais.

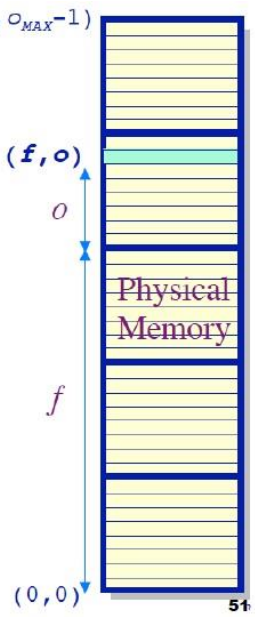
➢ Vieno ir to paties proceso puslapiai gali būti išbarstyti pagrindinėje atmintinėje. Programoje į duomenis ar komandas kreipiamasi pagal virtualųjį adresą, kuris susideda iš puslapio numerio bei poslinkio puslapio pradžios atžvilgiu.
➢ Skaidymo puslapiais sistema užtikrina transformavimą virtualaus puslapio numeriais, kurį naudoja programa į realų, fizinį puslapio adresą, kuris naudojamas pagrindinėje atmintinėje.
➢ Taikant skaidymo puslapiais mechanizmą, OS priversta pagrindinėje atmintinėje laikyti kiekvieno aktyvaus proceso puslapių lentelę.
➢ **Kiekvienas puslapių lentelės įrašas – tai nuoroda į numerį rėmo, į kurį iš tikrųjų yra įkeltas atitinkamas puslapis.**
➢ Kiekvienoje programoje, kuriai yra taikomas skaidymo puslapiais mechanizmas (puslapiavimas), kiekvienas loginis adresas turi būti išreikštas puslapio numeriu p ir poslinkiu tame puslapyje d.
➢ Kai programoje yra aptinkamas loginis adresas, išreikštas puslapio numeriu 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.
➢ Aukščiausieji adreso bitai naudojami kaip puslapių lentelės indeksas, jie nurodo, į kurį puslapį yra kreipiamasi. Žemiausieji – rodo poslinkį šiame puslapyje

Puslapiavimas (1)

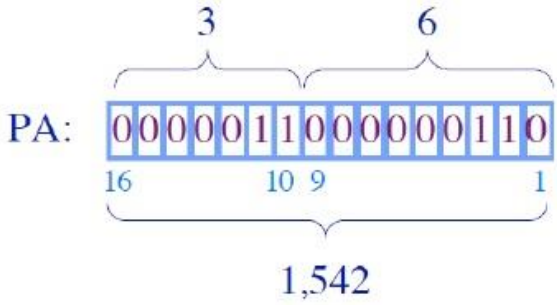
- Fizinė atmintis yra sudaloma į fiksuoto, puslapio dydžio blokus, vadinamus rėmais (frames).

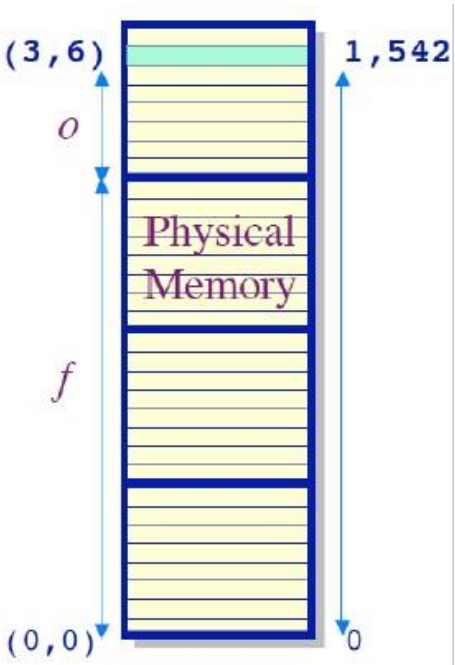
- Atminties adresas yra rinkinys (*f, o*):
  - *f* – rėmo numeris (f_{max} rėmų)
  - *o* –poslinkis nuo rėmo pradžios (o_{max} baitai/rėmai)
  - Fizinis adresas= $o_{\max} \times f + o$





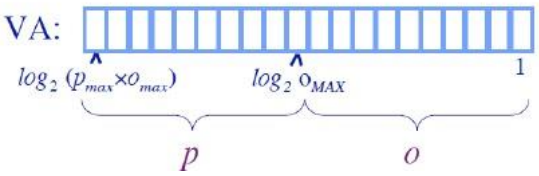
- Pavyzdys: 16 bitų adresų erdvė padalinta į 512 B rėmus.
  - Adresuojama vieta (3,6) = 1542

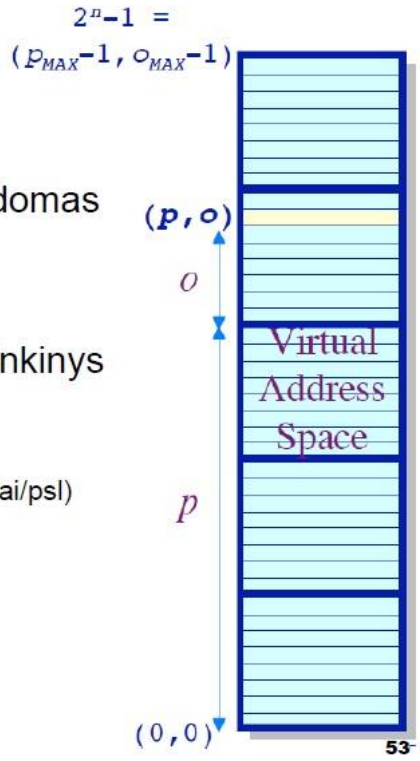




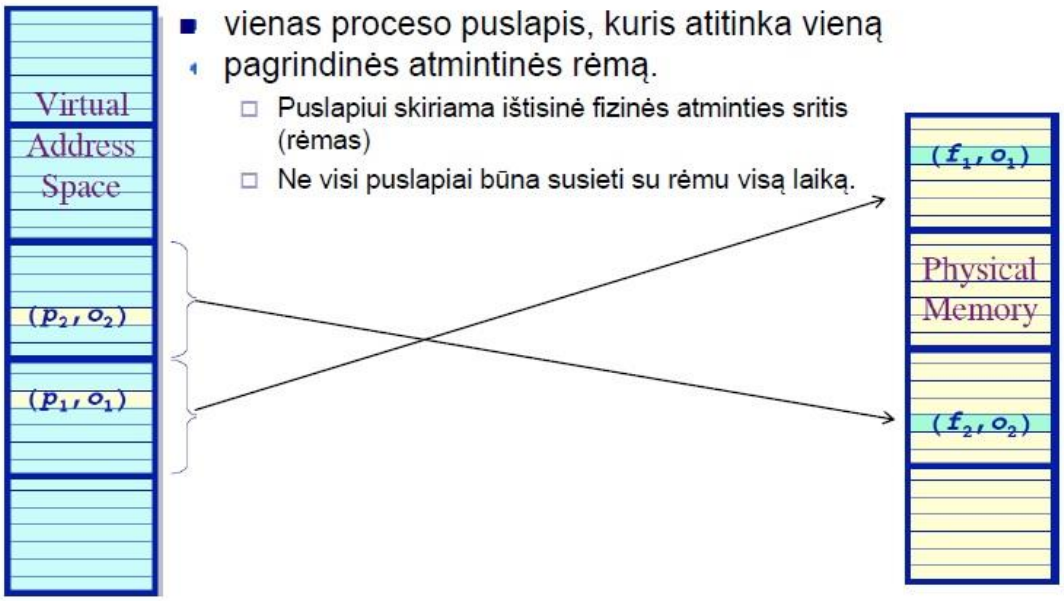
Puslapiavimas (2)

- Kiekvienas procesas yra suskaidomas į vienodo dydžio puslapius.
  - |puslapis|=|puslapio rėmas|
- Virtualus puslapio adresas yra rinkinys (*p*, *o*):
  - *p* – psl numeris (p_{max} psl)
  - *o* – poslinkis nuo psl pradžios (o_{max} baitai/psl)
  - Virtualus psl adresas= $o_{\max} \times p + o$





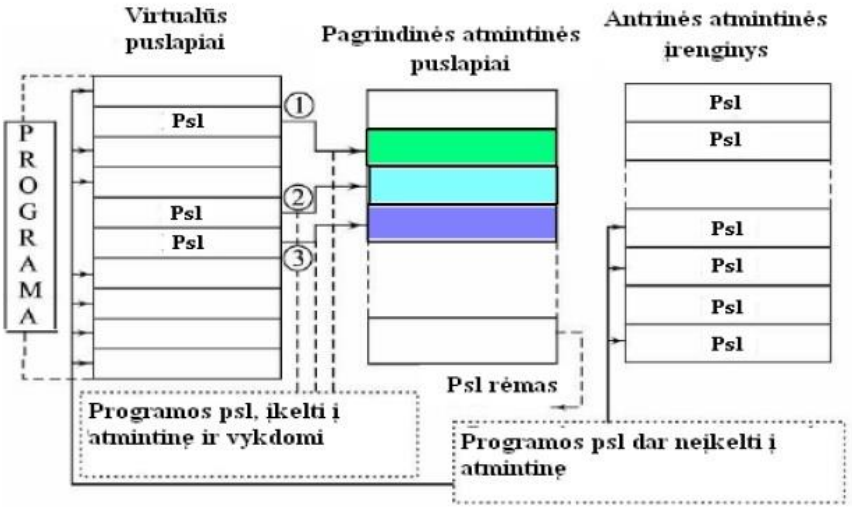
Puslapiavimas (3)



- vienas proceso puslapis, kuris atitinka vieną pagrindinės atmintinės rėmą.
  - Puslapiui skiriama ištisinė fizinės atminties sritis (rėmas)
  - Ne visi puslapiai būna susieti su rėmu visą laiką.

Proceso įkėlimas

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



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

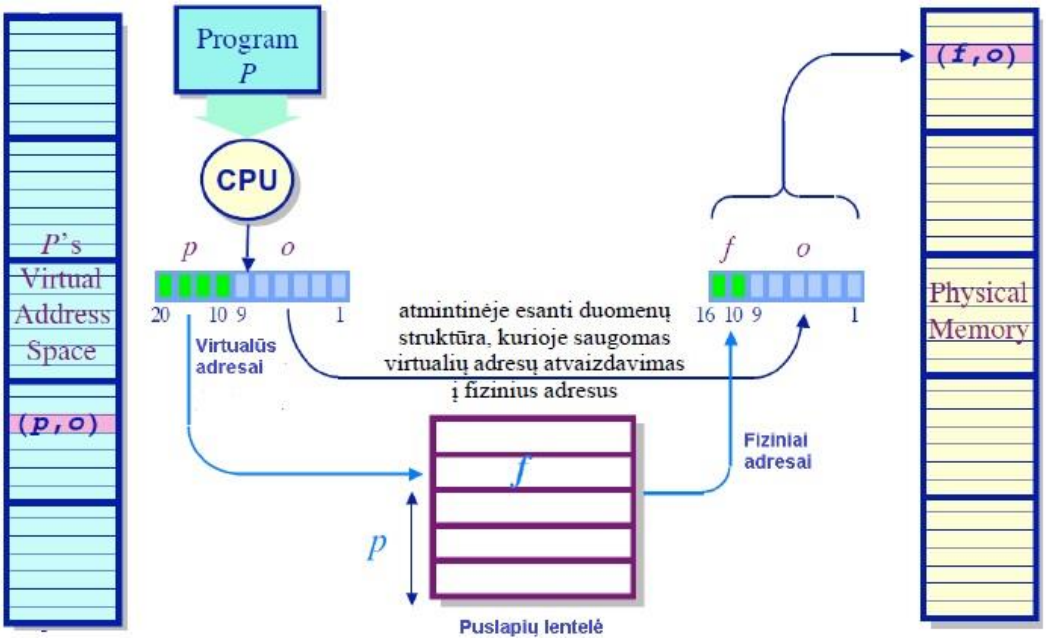
- Nustatomas proceso rezidentinių puslapių skaičius;
- Surandama pakankamai laisvų rėmų PA;
- Į juos įkeliami proceso rezidentiniai puslapiai.

- Darbo su virtualiąja atmintine principai buvo sukurti palaikyti 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.
- 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.

Fragmentacija naudojant puslapiavimą

- Nebelieka išorinės fragmentacijos :
  - Proceso puslapiai proceso talpinimo į pagrindinę atmintinę metu gali užimti laisvus rėmus (page frames).
  - Nauda:
    - procesui nebūtina užimti ištisinės adresų erdvės pagrindinėje atmintinėje
    - proceso puslapiai gali būti išbarstomi į egzistuojančius laisvus rėmus.
  - Procesui pasibaigus, tiesiog padaugėja laisvų rėmų.
- Kadangi puslapio (rėmo) dydis yra pakankamai nedidelis, tai sumažėja ir vidinė fragmentacija.

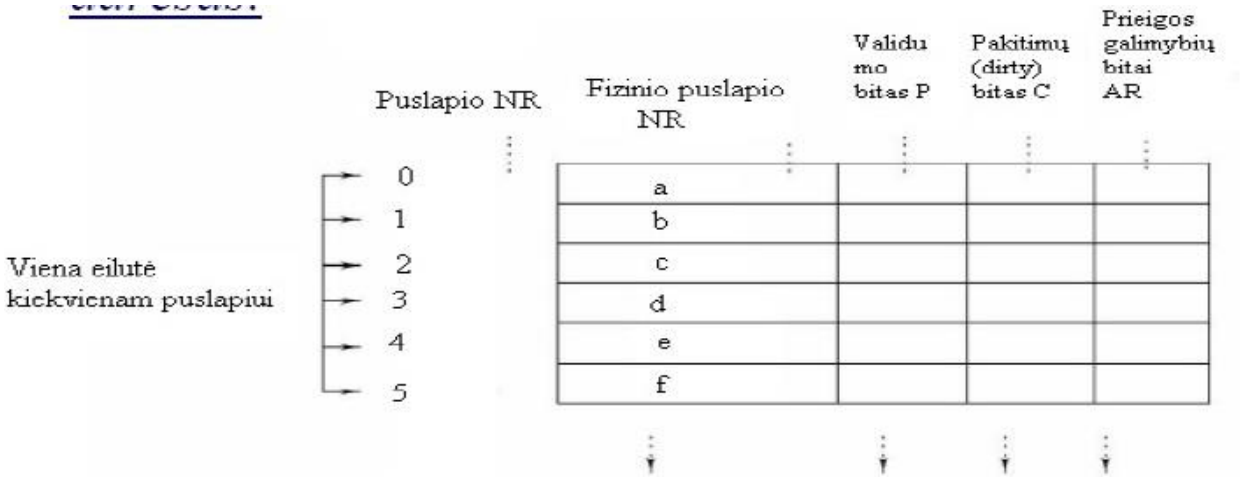
Virtualaus adreso transliavimas į fizinį



- 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 numeriu 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 2^N, 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 lentelės struktūra

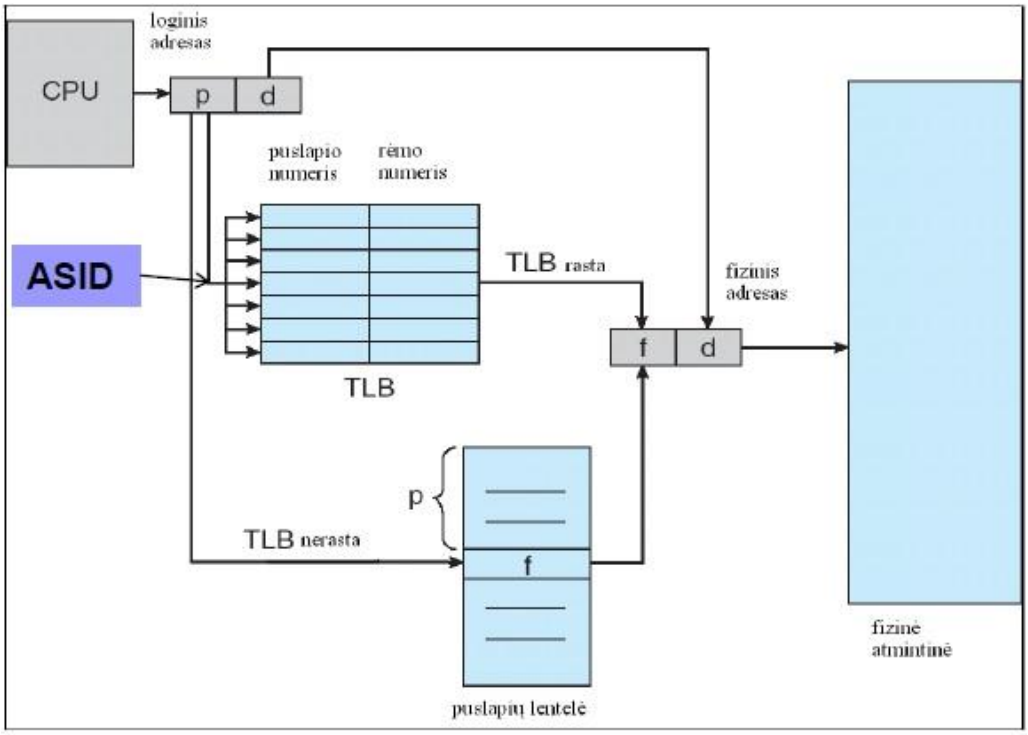
- Viename iš CPU registrų visada yra tuo metu vykdomo proceso puslapių lentelės **pradžios fizinis adresas.**
- **Puslapių lentelę sudaro** puslapio adresas, validumo bitas, pakitimų bitas ir prieigos galimybių bitai.
- Puslapių lentelė naudojama loginiam adresui į fizinį transliuoti, todėl ją reikia laikyti pagrindinėje atmintinėje.



- 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 bitus, 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.

Adreso transliavimas naudojant TLB įrašus

- Kadangi transliuojant adresą, pradžioje reikia kreiptis į puslapio lenteles, esančias atmintinėje, ir iš ten gauti numerį (Ar adresą) rėmo, į kurį yra įkeltas šis puslapis, tai procesui paspartinti naudojamos TLB lentelės, kurios saugo dalį puslapių lentelės informacijos.
- TLB lentelės yra laikomos procesoriaus spartinančiojoje atmintinėje ir jas naudoja atmintinės valdymo įranga virtualiojo adreso transliacijai pagreitinti.
- **TLB lentelės saugo virtualiųjų puslapių numerių bei jų fizinių adresų tarpusavio atitikimus**. Jei reikalaujamas adresas randamas TLB lentelėje jis transliuojamas greitai.
- Kaip alternatyva hierarchinėms puslapių lentelėms naudojamos invertuotos puslapių lentelės. Taikant šias 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.



Su puslapiavimu susijusios problemos ir sprendimai

- Du kreipiniai į atmintį. Sprendimas: TLB lentelės
- Puslapio dydis. Sprendimas: 4KB, 8KB arba 16 KB
- Rezidentinis proceso dydis. Sprendimas: vienodi rėmų kiekiai; rėmų kiekis proporcingas procesų dydžiui

Skirstymo puslapiais pranašumai:

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

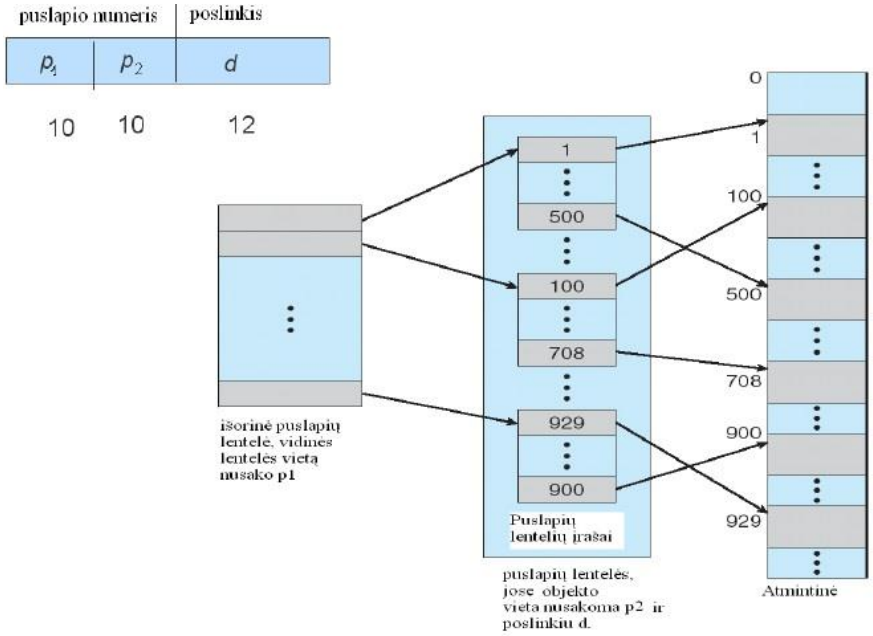
Skirstymo puslapiais problemos:

- Puslapių lentelės yra didelės apimties
- Sunku iš anksto skirti sritį

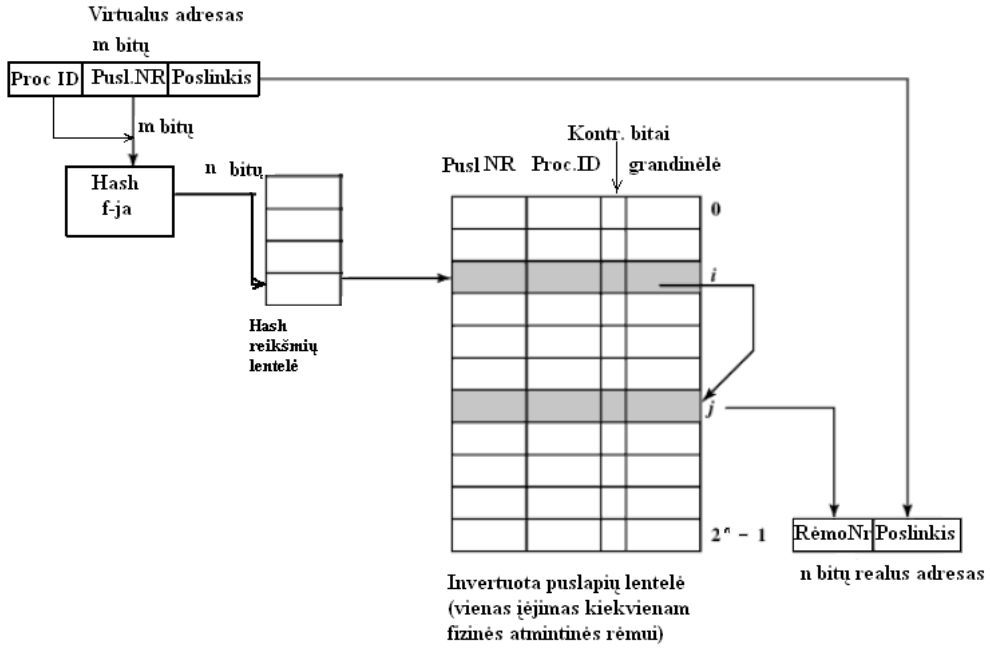
Puslapių lentelių tipai:

- vieno lygio
- hierarchinės
- invertuotos santraukos (hash) tipo

Dviejų lygių hierarchinė puslapių lentelė



Invertuoto tipo puslapių lentelė



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

- 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:

Bendrai naudojantis segmentais, kiekvieno proceso segmentų lentelėse yra įrašai, kurie rodo į tas pačias pagrindinės atmintinės vietas. Bendrai naudojamų segmentų dažniausiai neleidžiama modifikuoti.

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 (puslapių keitimo strategijos, puslapių kilnojimas)

Puslapių mainai

Tai gali būti atliekama atsiradus kreipiniui į tam tikrą, pagrindinėje atmintinėje (PA) dar nesantį, proceso puslapį, todėl iškyla poreikis įkelti puslapį 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 pagrindinėje atmintinėje).
- 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 galiojimo 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

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. Po šio keitimo procesoriaus rezidentinius skaičius gali ir padidėti ir sumažėti.

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 kreipinams 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.
- **Taikomas laikrodžio algoritmas;**
  - 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.
  - Naudojamas žiedinis įkeltųjų puslapių sąrašas, kuriame laikrodžio „rodyklė“ rodo į seniausią puslapį.
- **Keičiamas mažiausiai pastaruoju metu naudotas puslapis (LRU – Least Recently Used);**
  - Mažiausiai naudoto puslapio keitimo algoritmas (LRU) yra 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.

- 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.

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

Puslapių kilnojimas

Rezidentinė proceso puslapių grupė – tai toks į pagrindinę atmintinę 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**, nes OS didelę laiko dalį nevykdo procesų.

Kaip sprendžiama problema, susijusi su šiukšlinimo situacija?
Venas iš sprendimo būdų būtų procesams reikalingo rezidentinio puslapių skaičiaus nustatymas.

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.

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.---

3. Įvesties ir Išvesties (I/O) Sistema – I/O Įtaisai (12T)

I/O Valdymo tikslai

I/O Įrenginių tipai:

- Orientuoti į žmogų
  - Naudojami žmogaus komunikacijose: spausdintuvai, terminalai, ekranai, klaviatūra, pelė
- Orientuoti į sistemą

    - Naudojami komunikacijose su elektroniniais įrenginiais: Diskai, magnetinės juostos, kontroleriai, davikliai
- Orientuoti į tinklines komunikacijas
    - Modemai, faksai, ...

OS ir I/O valdymo problemos

- Multiprogramavimas leidžia procesams laukti I/O veiksmų pabaigos, o tuo tarpu kiti procesai gali būti vykdomi.
- OS turi kontroliuoti visus I/O įrenginius
- OS turi suderinti greičių skirtumus
    - Dauguma I/O įrenginių yra lėti lyginant su pagrindine atmintine
- Siekiama traktuoti visus I/O įrenginius vienodai
    - Slėpti jų I/O detales, žemo lygio funkcijose, taip kad aukštesniame lygyje ir procesuose būtų galima į juos kreiptis paprastais veiksmais: read, write, open, close, lock, unlock...

I/O valdymo uždaviniai

- Generuoti komandas I/O įrenginiams.
- Perimti ir apdoroti šių įrenginių generuojamus pertraukčių signalus.
- Apdoroti klaidas.
- Užtikrinti nuo įrenginių nepriklausomą sąsają tarp šių įrenginių ir likusios sistemos dalies.

<u>I/O valdymo tikslai</u>

- Siekiama užtikrinti **efektyvų** sistemos funkcionavimą.
    - Aptarnauti įvairių procesų užklausas.
    - Pagreitinti duomenų perdavimą.
- Siekiama pateikti vartotojui bendrą, **nepriklausančią** nuo įrenginio, sąveikos su įrenginiais mechanizmą
    - Vartotojo lygmenyje sąveika su I/O įrenginiais yra daroma nepriklausoma nuo fizinių įrenginio charakteristikų
    - Programa neturi būti keičiama, pritaikant ją kiekvienam konkrečiam įrenginiui
    - Universalus vardų panaudojimas: loginiai įrenginio vardai: /dev/cdrom; /dev/hd0

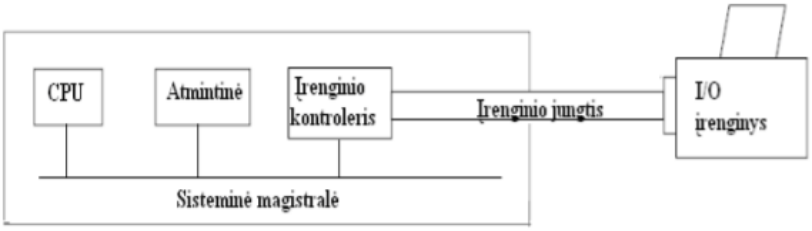
I/O įrenginių skirtumai

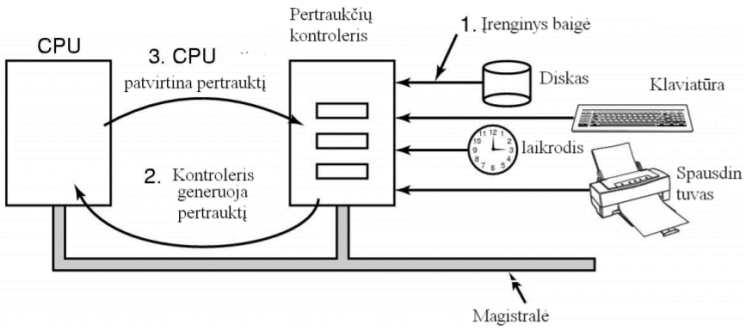
- Duomenų perdavimo vienetas
- Duomenys gali būti perduodami kaip baitų srautas (stream of bytes) , arba blokais
    - Skiriami blokinio ir srautinio tipo įrenginiai.
- Duomenų atvaizdavimo forma
    - Naudojamos tam tikros kodavimo sistemos
- Kontrolės sudėtingumas
- Reakcija į klaidos situacijas

Įrenginių kontroleriai. I/O Veiksmų vykdymas

Įrenginių kontroleriai

- Tai sąsajos korta arba integruotas modulis, kuris „kontroliuoja“ išorinius įrenginius, pvz, kieto disko kontroleris.
- Dažniausiai kontroleriai turi gana primityvius bendros paskirties procesorius bei nedidelį kiekį atminties (kelis kB), bet būna ir išimčių: kai kurių spausdintuvų kontroleriai yra pakankamai galingi, kad interpretuotų aukšto lygio grafinio programavimo kalbas.
- Nuo įrenginio kontrolerio duomenys perduodami sistemine magistrale. I/O adresas – tai pavienis adresas ar adresų grupė duomenų siuntimui/priėmimui iš/į įrenginio kontrolerio

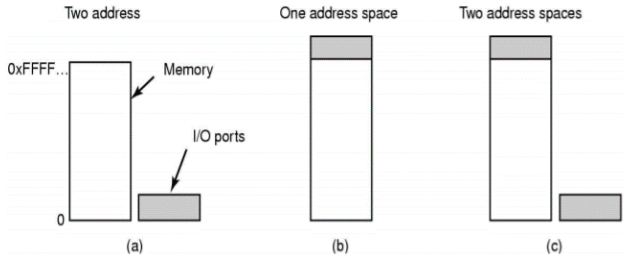




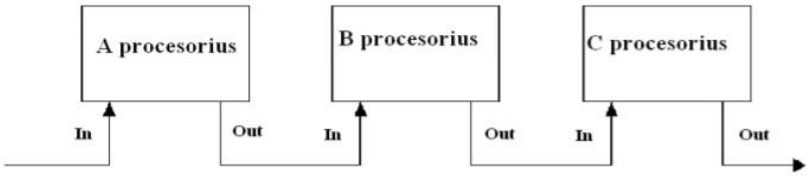
Kontrolerio registrai

- Naudojami komunikacijose su procesoriumi:
  - Perduoti tam tikras komandas įrenginiui
  - Sužinoti įrenginio būklę
- Registrų adresavimui gali būti naudojami:
  - I/O porto (prievado) numeriai,
  - Registrams yra skiriama vieta atmintyje, jauniausiuose jos adresuose
  - Hibridinis būdas, kai duomenų buferiai atmintinėje ir portai adresavimui kontrolerio registrų (wtf... jeigu mes tokį parašytume sakinį tai nukirstu kojas čj)

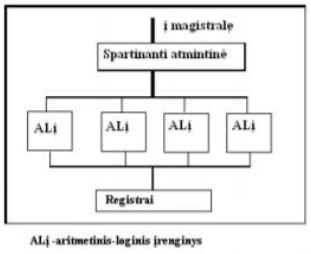
CPU ir kontrolerio registrai:



Pertrauktis ir procesoriai



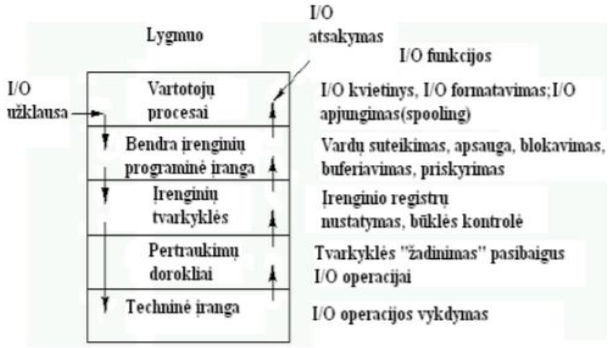
Procesorių sujungimas vamzdžio principu



superskaliarinio procesoriaus architektūra

I/O valdymo lygiai

- Vartotojo lygyje jam yra pateikiama įvairių funkcijų biblioteka (read(), open(), printf() ir pnš.)
- Vartotojo veiksmams atlikti skiriamos įvairios programos, tokios kaip pvz cat, tar ir t.t., įvairios prieigos prie dedikuotų įranginių (spausdintuvų, tinklo jungčių priemonės)



Bendra įrenginių programinė įranga

- Teikia vartotojui vienodą sąsają su skirtingais įrenginiais
  - Siekiama užtikrinti, kad vartotojas galėtų vienodai traktuoti visus I/O įrenginius
  - Stengiamasi nuo jo paslėpti I/O detales, žemo-lygio funkcijas.
  - Vartotojui leidžiama operuoti bendromis sąvokomis (read, write, open, close ir t.t.)
- Šiame lygyje yra sprendžiami ir įrenginių apsaugos klausimai
- Rūpinamasi įrenginių įvardinimu, I/O buferiais, įrenginių priskyrimu, klaidų, susijusių su įrenginiais, apdorojimu.

Įrenginių tvarkyklės

- Kiekvienas prie kompiuterio prijungtas įrenginys reikalauja tam tikros tą įrenginį valdančios programos, kuri yra vadinama **įrenginių tvarkykle**
- Kiekvieno tipo įrenginiui reikalinga atskira, jam skirta tvarkyklė
- Įrenginių tvarkyklės konvertuoja abstrakčią užklausą, pvz., **perskaityti n reikšmę**, į specifinę, įrenginiui orientuotą užklausą bei atitinkamas komandas, skirtas šio įrenginio kontroleriui.
- Jos atsakingos už keletą funkcijų
  - Jos turi priimti skaitymo arba rašymo užklausas, atėjusias iš aukštesnio, nuo įrenginio nepriklausomo lygmens ir jas vykdyti, patikrinusios pateiktus įvedimo parametrus.
  - Tvarkyklės turi sugebėti **patikrinti** įrenginį, sugebėti jį inicializuoti:
    - Kontrolė, per kurią turi atlikti to įrenginio tvarkyklė, susiveda į eilės komandų perdavimą šio įrenginio kontroleriui
    - Tvarkyklė gali patikrinti ar kontroleris ją priėmė ir ar yra pasiruošęs priimti sekančią komandą.
- Atlikusi visą komandų seką tvarkyklė daro **klaidų tikrinimą** ir jei klaidų neranda, tvarkyklė gali atlikti duomenų perdavimą aukštesniam, nuo įrenginio nepriklausančiam lygmeniui.

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

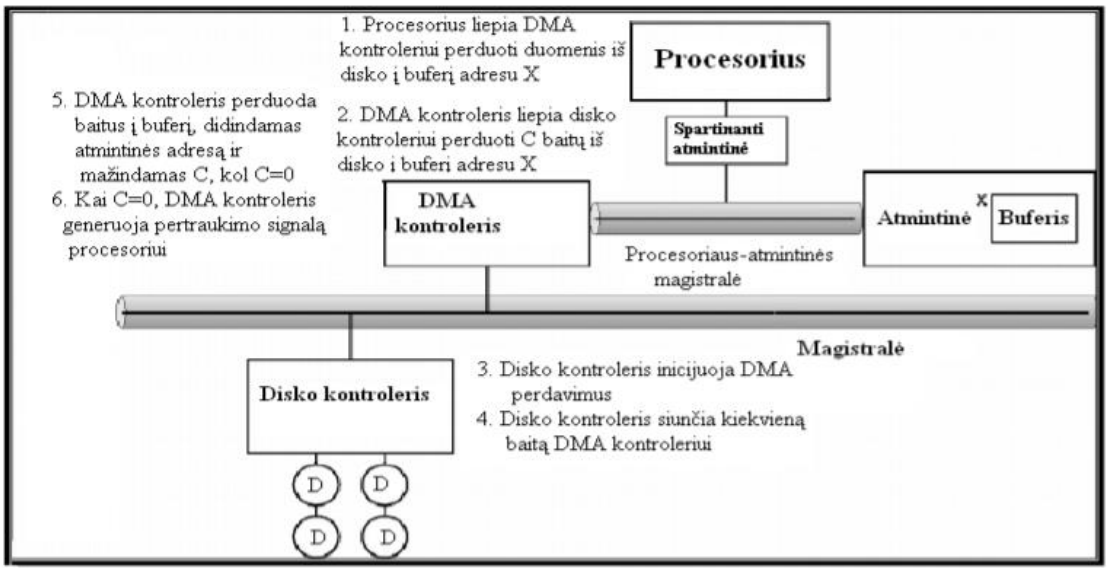
Programuojamas I/O

- Operacinė sistema pateikia I/O komandą įrenginio kontroleriui
- Procesas yra „Užimto laukimo“ būvyje, periodiškai apklausia kontrolerį, norėdamas nustatyti, ar I/O operacija yra baigta
- Problema tame, kad procesorius yra priverstas ilgai laukti, kol I/O modulis tampa pasiruošęs priimti ar perduoti duomenis.

Pertrauktimis grindžiamas I/O

- I/O komanda yra išduodama kontroleriui
- Procesorius tęsia komandų vykdymą
- I/O modulis atsiunčia pertrauktį, kai baigia veiksmą.
  - Trūkumas tame, kad pertrauktys yra generuojamos ties kiekvienu simboliu.
  - Pats pertraukties apdorojimas užima kažkiek laiko, pagal šią schemą neefektyviai panaudojamas procesorius

Tiesioginė prieiga prie atminties:



- **DMA modulis** kontroliuoja duomenų apsikeitimą tarp atminties ir I/O įrenginio
- **Procesorius** pertraukiamas tik po to, kai visas blokas yra perduodamas.
- **DMA privalumas** yra tame, kad naudojant šį metodą yra atlaisvinamas procesorius nuo skaitymo/rašymo vykdymo iš kontrolerio į pagrindinę atmintinę, sumažėja pertraukčių kiekis.

4. Įvesties ir Išvesties (I/O) Sistema – Diskai (12T)

Fizinė disko struktūra

Šiuolaikinio disko komponentai:

1. **Plokštelė**. Apvalus kietas paviršius ant kurio nuolat saugomi duomenys sukeliant magnetinius pokyčius.
    - Diskas gali turėti kelias plokšteles
    - Kiekviena plokštelė turi dvi puses, kurios vadinamos **paviršiumi**.
    - Plokštelės paprastai pagamintos iš kietos medžiagos (pvz. aliuminis) ir tuomet padengiamos plonu magnetiniu sluoksniu, kuris leidžia kietajam diskui nepertraukiamai saugoti bitus, net kai jis yra išjungtas.
2. **Velenas**. Plokštelės yra pritvirtintos prie velenėlio sujungto su motoriuku, kuris būdamas įjungtas suka plokšteles pastoviu nustatytu greičiu.
    - Greitis matuojamas **apsisukimais per minutę (RPM)**.
    - Dažniausiai tarp 7200 – 15 000 RPM.
3. **Takelis**. Duomenys užkoduoti ant kiekvieno paviršiaus sektorių koncentriniuose ratuose. Vienas koncentrinis ratas yra takelis.
    - Vienas paviršius susideda iš tūkstančiai tūkstančių takelių.
4. **Galvutė**. Mechanizmas, kuris leidžia skaityti ir rašyti kiekviename disko paviršiuje.
    - Skaito magnetinius raštus išsaugotus disko paviršiuje arba sužadina pasikeitimus jame.
    - Viena galvutė vienam paviršiui.

5. **Ranka**. Disko galvutės yra pritaisytos prie disko rankos galinčios judėti skersai paviršiaus prie to takelio pozicijos, kurį turi nuskaityti galvutė.

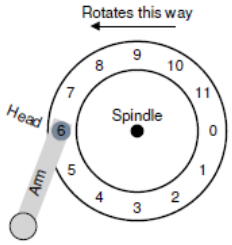


Figure 37.2: **A Single Track Plus A Head**

Disko užklausų vykdymas

Gavusi sąrašą I/O užklausų disko planuoklė (scheduler) išanalizuoja užklausas ir nusprendžia, kurią iš jų atlikti pirmiau.

SJF: **Shortest Job First.** Kitaip nei užduočių planavime, kur kiekvienos užduoties ilgis paprastai nėra žinomas, tai disko planavime galima ganėtinai tiksliai numatyti kiek laiko įvykdyti užklausą užtruks. Apskaičiuodama užklausos takelio pasiekimo (seek) ir galimą sukimosi vėlavimą (delay) disko planuoklė gali žinoti kaip ilgai kiekviena užklausa užtruks ir taip parinkti tą, kuri užims mažiausiai laiko, aptarnauti pirma. Taigi planuoklė bando taikyti **SJF (shortest job first) principą**.

SSTF: Shortest Seek Time First (trumpiausias pasiekimo laikas pirma). Vienas pirmųjų disko planuoklės metodų **SSTF** (arba SSF shortest seek first) surikiuoja I/O užklausų eilę pagal takelį, paimti užklausą, kurios takelis yra arčiausiai galvutės. Pvz.: galvutė yra ties plokštumos centre esančiu takeliu, eilėje yra dvi užklausos į 21 (takelis per vidurį) ir 2 (išorinis takelis) sektorius. Pirmiausia bus aptarnaujama užklausa į 21, o po jos į 2 sektorių.

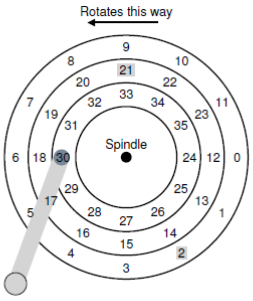


Figure 37.7: **SSTF: Scheduling Requests 21 And 2**

Minusai:

- OS nežino disko geometrijos, o tiesiog mato blokų masyvą, todėl vietoje SSTF OS paprasčiau yra implementuoti NBF (nearests-block-first, artimiausias blokas pirmiau), kuri pirmiau aptarnauja artimesnį bloką.
- Fundamentali problema – badavimas. Jeigu didžioji dauguma esančių eilėje užklausų bus į centrini takelį, užklausos į visus kitus takelius bus tiesiog ignoruojamos.

Elevator (a.k.a. SCAN or C-SCAN). Pagal šį algoritmą galvutė tiesiog periodiškai pereina skersai disko plokštumos aptarnaudama užklausas to takelio, virš kurio tuo metu yra. Taigi jeigu užklausa atėjo į bloką takelio, kuris jau buvo aptarnautas konkretaus perėjimo metu, ji nebus vykdoma iš karto, bet įdėta į eilę ir aptarnauta kito perėjimo metu.

Atmainos:

- **F-SCAN** – užšaldo užklausų eilę kol pasibaigia konkretus perėjimas, o įdeda į kitą eilę, kuri bus vykdoma kito perėjimo metu. Tai leidžia išvengti badavimo, jeigu perėjimo metu būtų siunčiama daug užklausų į dar neaptarnautus takelius.
- **C-SCAN (circular SCAN)** – vietoj pereinant viena kryptimi, galvutė vaikšto abiejomis kryptimis (iš išorės į vidų ir atgal). Šis algoritmas primena liftą, kuris kyla aukštyn arba žemyn, bet neaptarnauja tiesiog artimiausiai esančio aukšto (leidžiantis iš 10 aukšto į pirmą pasiekus 3 aukštą ir gavus iškvietimą į 4 liftas nusileis iki 1 ir tik tuomet kils į 4 aukštą).

SPTF: Shortest Positioning Time First (SATF shortest access time first). SSTF ir Elevator metodų minusas, kad jie neįvertina disko sukimosi greičio. Jeigu galvutės paslinkimo laikas yra trumpesnis negu disko apsisukimo, tuomet yra logiškiau aptarnauti užklausas tų sektorių prie kurių galvutė atsidurs greičiau. Pvz.: iš 30 bloko pagal SSTF peršokus į vidurinį takelį reiktų palaukti kol diskas apsisuks ir priartės 16 bloko užklausa, todėl logiškiau jeigu galvutė pirmiau pereitų į išorinį takelį ir aptarnautų 8 bloką pirma, nes jį pasieks jau dabartinio apsisukimo metu. Bet šio algoritmo taip pat beveik neįmanoma įgyvendinti OS, kadangi ji nežino takelių ribų ir kurioje pozicijoje galvutė yra disko sukimosi atžvilgiu.

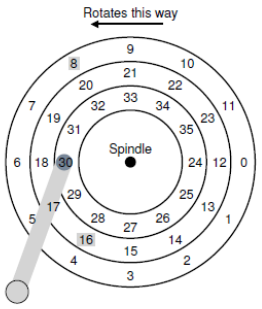


Figure 37.8: SSTF: Sometimes Not Good Enough

RAID architektūra

Raid – Redundant Array of Inexpensive Disks (perteklinis nebrangių diskų masyvas). Tai technika naudojanti kelis diskus kartu tam, kad sukurtų greitesnę, talpesnę ir patikimesnė diskinę sistemą. Terminas pristatytas 1980 m.

Išoriškai RAID atrodo kaip paprastas diskas: grupė blokų iš kurių galima skaityti arba į kuriuos galima rašyti. Viduje RAID – sudėtingas žvėris, susidedantis iš daugiau nei vieno diskų, atminties (pastovios ir ne) ir vieno ar kelių procesorių tvarkančių sistemą. Techninės įrangos prasme tai labai panašu į kompiuterinę sistemą specializuotą tvarkyti diskų grupės užduotis.

RAID turi daugybę pranašumų prieš vieną diską:

- Darbo sparta (performance) – naudojant keletą diskų lygiagrečiai gerokai pagreitina I/O kartus.
- Talpa (capacity) – daugiau diskų, daugiau telpa duomenų.
- Patikimumas (reliability) – paskirstyti duomenis tarp kelių diskų be RAID padaro duomenis pažeidžiamus, jei bus prarastas bent vienas diskas, bet su dubliavimu RAID gali toleruoti disko praradimą ir veikti lyg nieko nebūtų įvykę.

RAID taip pat pasižymi skaidrumu (transparency – savybė, kai naujo funkcionalumo pridėjimas nereikalauja jokių pokyčių sistemoje) jį galima paprasčiausiai instaliuoti vietoje paprasto disko ir sistemoje (kompiuteriuose, OS'ose) nieko keisti nereikia. Išsprendus šią diegimo problemą RAID buvo sėkminga nuo pat pradžių.

P.S. šiaip medžiagos apie RAID duota 19 puslapių anglų kalba. Čia yra pagrindas, bet jei norit galit pasiskaityti viską dėl geresnio supratimo.

5. Failų Sistema – Failų sistemų pagrindai (14T)

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

Pagrindinis kompiuterių tikslas yra kurti duomenis, jais manipuliuoti, juos saugoti ar esant reikalui pateikti. Failų sistema pateikia būdą kaip saugoti, pateikti ir valdyti informaciją nuolatinio saugojimo įrenginyje, kokiu yra diskas.

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

Failų sistema teikia metodą, skirtą failų saugojimui ir organizavimui:

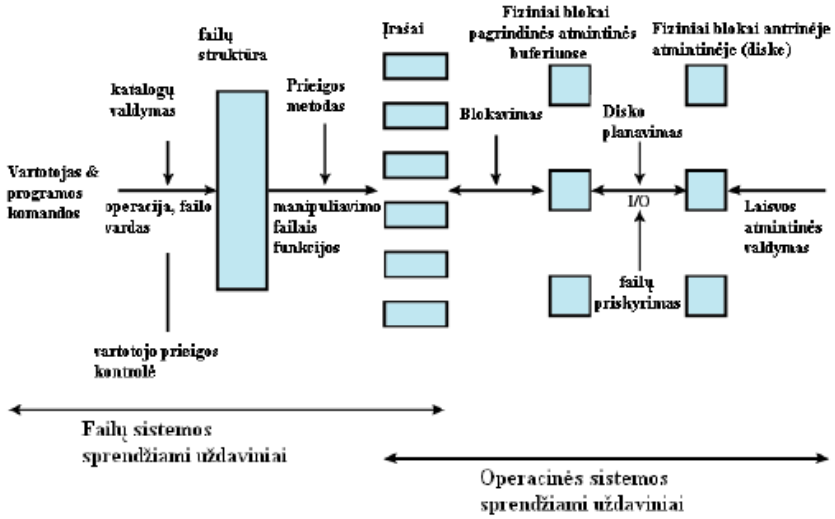
- Teikti loginį (abstraktų) požiūrį į failus ir katalogus
- Užtikrina loginę failų organizaciją
- Padėti efektyviai naudoti atmintinės įrenginius
- Palaikyti bendrą naudojimąsi duomenimis

Diskai yra atmintinės įrenginiai, skirti failų sistemos saugojimui. Formatuodami diską, mes „įkurdiname“ jame atitinkamą failų sistemą (FAT, NTFS, UNIX, ext2, ext3, ext4...)

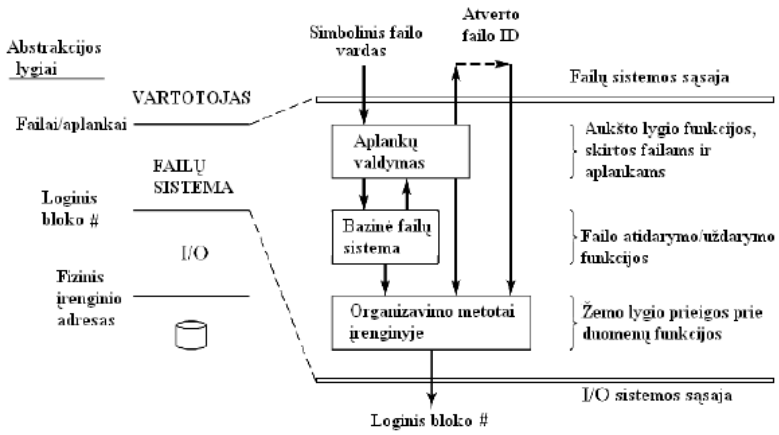
Failų sistema apima:

- Diske saugomų failų visumą
- Duomenų struktūras, reikalingas failams tvarkyti (aplankai, failų deskriptoriai, informacija apie vietą diske)
- Failus tvarkančias programas (skirtas failų kūrimui, naikinimui, kopijavimui, perkėlimui, įrašymui ir t.t.)

Failų sistemos sprendžiami uždaviniai:



Abstrakcijos lygiai:



Hierarchijos modelis (nežinau, ar reikia):

- Abstrakti vartotojo sąsaja
    - Vartotojui yra pateikiama vartotojo lygio sąsaja, kuria naudodamasis jis gali naudotis failų bei aplankų sąvokomis operuodamas jomis tiek savo programose, tiek veiksmuose su duomenimis
- Aplankų (katalogų) valdymas
    - Aplankų valdymo pagrindinė paskirtis yra pagal loginį failo vardą pateikti šį failą atitinkantį unikalų failo identifikatorių, atrasti failo aprašą (descriptor)
- Bazinė failų sistema
    - Atverti, užverti (open/close) failus
- Fiziniai organizavimo metodai
    - Atvaizduoti failo duomenis į disko blokus

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

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

Failo charakteristikos:

- Failo tipas – reikalingas tik sistemoms, kurios palaiko skirtingus failų tipus
- Failo savininkas – šie duomenys naudojami apskaitai, prieigos prie failų nustatymui
- Leidimai prieigai prie failo – nusako kas ir kokius veiksmus gali atlikti su failu
- Data, laikas – failo sukūrimo, modifikavimo, prieigos data
- Failo vieta diske
- Failo dydis ir kitos

Veiksmai su failais: sukurti, išmesti, ieškoti, atverti, uždaryti, nustatyti jo charakteristikas, modifikuoti turinį.

Prieigos prie failų metodai:

- Nuosekli prieiga
    - Visi baitai/įrašai skaitomi nuosekliai nuo pradžios

    - Negalima šokti atgal skaitymo metu (reikia persukti juostą)
    - Yra taikoma magnetinių juostų atveju
- Tiesioginė prieiga
- Atsitiktinė
    - Baitai arba įrašai skaitomi bet kuria tvarka
    - Tai svarbu duomenų bazių sistemose
- Asociatyvi
    - Dar vadinama adresuojama pagal turinį atmintis
        - Atmintinė adresuojama pagal turinį, ne pagal adresą
        - Naudojama labai greitos paieškos taikomosioms programoms
        - Asociatyvi atmintis yra brangi

Aplankai (katalogai):

- Pats aplankas taip pat yra failas, tik su specialia struktūra
    - Kiekvienas įėjimas aplanke – tai simbolinis failo vardas kartu su nuoroda į tai, kur failas yra patalpintas diske
    - Be to dar gali būti saugoma informacija apie failo dydį, tipą, prieigos galimybes, modifikavimo ir sukūrimo laikai
- Aplankai turi dvi paskirtis:
    - Vartotojams jie leidžia struktūrizuoti failų talpinimą failų sistemoje, grupuojant juos pagal tam tikras savybes, leidžia skirtinguose aplankuose saugomiems failams naudoti vienodus vardus
    - Failų sistemos požiūriu aplankai leidžia atskirti loginę failų struktūrą nuo fizinio failų patalpinimo diske
- Aplankai teikia informaciją, kurios reikia norint greitai surasti failo duomenų blokus diske
- Aplankų įrašuose saugoma informacija:
    - Galima saugoti tik simbolinius failų vardus ir metaduomenis
    - Galima saugoti tik simbolinius failų vardus ir nuorodą į aprašą

Bendras naudojimasis failais ir jų apsauga

Bazinė failų sistema:

- Atvertų failų lentelė (OFT)
- Open komanda:
    - Tikrina prieigos teises
    - Priskiria vietą atvertų failų lentelėje
    - Priskiria (read/write) buferius
    - Užpildo OFT lentelės įrašą
        - Inicializacija (pvz. Einamoji pozicija)
        - Informacija iš failo aprašo (pvz. Failo ilgis, vieta diske)
        - Nuorodos į priskirtus buferius
    - Grąžina OFT indeksą
- Close komanda:
    - Perkelia modifikuotus buferius į diską

    - Atlaisvina buferius
    - Atnaujina failo aprašą (failo ilgis, vieta diske, apskaitos informacija)
    - Atlaisvina OFT įėjimą

Failų sistemos šifravimas (apsauga):

- EFS (The Encrypting File System) teikia šifravimo technologiją, skirtą šifruotų failų saugojimui NTFS diskuose. EFS apsaugo failus nuo atakuotojų.
- Šifravimas yra skaidrus atžvilgiu vartotojo, kuris darė šifravimą:
    - Sistema atšifruoja failą tam vartotojui, kai jis į jį kreipiasi
    - Kai failas yra išsaugomas, jis vėl yra šifruojamas
- Vartotojai, kurie nėra autorizuoti prieigos prie šifruotų failų atžvilgiu gaus pranešimą „Access denied“ jei jie bandys atlikti open, copy, move ar rename šifruotam failui ar katalogui
- EFS privalumai:
    - Jei failą pažymėjom kaip šifruojamą, jis visad bus užšifruotas, vartotojui nereiks apie tai rūpintis, jam nereikės atsiminti slaptažodžio atšifravimui
    - Kieta sauga – raktai nėra grindžiami vartotojo įvesta pass-phrase, EFS generuoja raktus, kurie atsparūs žodyno atakoms
    - Visi šifravimai vykdomi branduolio būvyje
    - EFS teikia gerą duomenų atkūrimo mechanizmą – leidžia atstatyti duomenis net jei darbuotojas, kuris paliko organizaciją yra juos užšifravęs
- EFS naudoja simetrinio rakto šifravimą kartu su viešo rakto technologija failų apsaugai. Failų duomenys šifruojami naudojant DESX algoritmą.
- Šifravimui naudojamas raktas yra šifruojamas naudojant viešą/slaptą raktą ir yra saugomas kartu su failu. Dviejų algoritmų naudojimą skatina šifravimo greitis
- Užšifravus failą yra kuriamas specialus įrašas vartotojui „Data Decryption Field (DDF)“, kuriame šifravimo raktas išsaugomas užšifruotas vartotojo viešu raktu
- Taip pat yra sukuriamas duomenų atstatymo įrašas „Data Recovery Field (DRF)“, kuriame raktas patalpinamas jį užšifravus atstatymo agento viešu raktu
- Baigus šifravimą duomenys yra pakeičiami šifruotais

Dešifravimas:

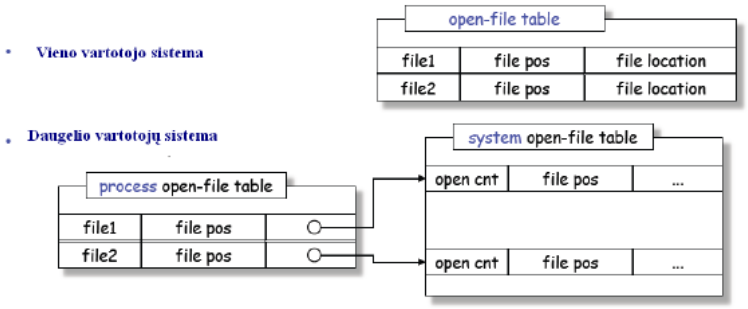
- Kai failas yra užšifruotas, tik tas vartotojas, kuris turi laukus DDF arba DRF gali turėti prieigą
- Įmanoma, kad keli vartotojai turėtų prieigą prie failo. Tai daroma sujungiant kelių vartotojų raktus į raktų grandinėlę ir užšifruojant raktą šių vartotojų viešais raktais
- Dešifravimas yra atliekamas kaip atvirkščias veiksmas šifravimui

Kopijų saugojimas (backup):

- Ką saugoti?
    - Operacinę sistemą – gal saugoti jos įvaizdį (image)
    - Duomenis
- Kur saugoti?
    - Atskiri diskai
    - USB flash tipo įrenginiai

    - USB išoriniai diskai
- Įvesti tam tikrą tvarką saugant duomenis, kuriuos norima atstatyti

Failų lentelė



Fiziniai failų sistemos organizavimo būdai

Fizinė failų organizacija aprašo failo išdėstymo išorinėje atmintyje (pvz. Diske), taisykles. Failas susideda iš fizinių įrašų – blokų. Blokas – tai mažiausias duomenų vienetas, kuriuo išorinis įrenginys apsikeičia su pagrindine atmintine. Priskiriant disko blokus failams yra siekiama efektyviai išnaudoti disko erdvę.

Fizinio organizavimo metodai:

- Problema – paskirti failams vietą diske: Failai – tai blokų eilė
    - Disko erdvė turi būti išnaudota efektyviai
    - Failus turi būti lengva išrinkti iš disko
- Naudojamas talpinimo strategijos:
    - Į nuosekliai einančius blokus
    - Į blokus, sujungiant juos į sąrašą
    - Į blokus, formuojant indeksų sąrašą

Nuoseklus talpinimas:

- Failas diske talpinamas į nuosekliai einančius disko blokus. Šis talpinimo metodas reikalauja failams priskirti tiek nuosekliai einančių blokų, kiek reikalauja to failo dydis
    - Paprastai įgyvendinamas
    - Greita nuosekli prieiga (minimalus galvutės judesys)
    - Sunku atlikt įterpimą/išmetimą
    - Sunku nuspręsti, kiek vietos priskirti pradžioje
    - Išorinė fragmentacija galima

Surištas sąrašas:

- Šiuo atveju aplanke yra nurodomas failo pradžios blokas, o jau tame bloke patalpinama nuoroda į sekantį failo bloką
    - Paprasta įterpti/išmesti, nėra išorinės fragmentacijos
    - Nuoseklus išrinkimas mažiau efektyvus (paieška, vėlinimas)
    - Nėra galimas tiesioginis kažkurio bloko išrinkimas
    - Mažas patikimumas (suirus grandinėlei)

Indeksinė organizacija:

- Suformuojama indeksų lentelė, rodanti, kuriame fiziniame bloke yra patalpintas atskiras failo blokas
- Paprasčiausiu atveju ši indeksų lentelė gali būti laikoma failo apraše
- Esant tokiam talpinimui paprasta atlikti įterpimo, išmetimo veiksmus, galimas tiek nuoseklus, tiek tiesioginis failo blokų išrinkimas
- Trūkumas pasireiškia tame, kad failo dydžius gali riboti indeksų lentelės dydžiai

Laisvos disko atmintinės valdymas

- Panašios problemos kaip ir valdant pagrindinę atmintinę
    - Saugoma informacija apie laisvus blokus surišto sąrašo metodu
        - Surišti individualius blokus mažai efektyvu, nėra taikomas blokų apjungimas, kad būtų mažinamas paieškos laikas
        - Surišamos nuoseklių blokų grupės. Grupė blokų yra priskiriama arba atlaisvinama vienu metu
    - Dvejetainio žemėlapio formavimas (bitmap)
        - Analogiškai kaip ir pagrindinės atmintinės atvejui
        - Trūkumas dvejetainio žemėlapio yra tame, kad ieškant laisvo bloko failų sistemai gali tekti peržiūrėti visą dvejetainį žemėlapį
        - Reliatyviai paprastas metodas – lengva rasti n laisvų gretimų blokų
        - Nėra efektyvu, jei visas žemėlapis nėra laikomas pagrindinėje atmintinėje
        - Sunku naudoti, jei diskai yra labai didelės talpos
- Nėra tikslinga surišti į sąrašą visus atskirus laisvus blokus po vieną
- Jei yra keli iš eilės einantys laisvi blokai, tai jie galės būt ir priskiriami esant poreikiui, todėl laisvų blokų sąrašas ir formuojamas surišant į sąrašą laisvų blokų grupes
- Laisvi blokai yra priskiriami failams imant juos iš sąrašo pradžios, o atsilaisvinę blokai yra įjungiami į šio sąrašo pabaigą

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

Windows failų sistema:

- Įrašus apie talpinimo vietą deda į failų talpinimo lentelę, vadinamą FAT (File Allocation Table)
    - FAT privalumai yra tame, kad informacija apie disko blokų užimtumą yra saugoma nedidelėje diske esančioje lentelėje
    - Lentelė indeksuojama blokų numeriais – vienas įėjimas-vienas blokas
    - Užimti blokai tarpusavy surišami sąrašu
    - Neužimti blokai indikuojami 0-nės reikšmės įėjimu
    - Patogus yra laisvų blokų suradimas ir priskyrimas failui – šiuo tikslu pakanka surasti neužimtus FAT lentelės įrašus
    - Kiek sudėtingesnis yra nuoseklus failo blokų skaitymas
    - Naudoja MS-DOS ir OS/2

MS DOS direktorijos įėjimas:

- Naudoja FAT12 arba FAT16 lenteles
- Naudoja 8+3 simbolių failų vardus

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

Windows 98 failų sistema:

- Naudoja FAT32 failų patalpinimo lenteles
- Naudoja ilgus 255 simbolių failų vardus

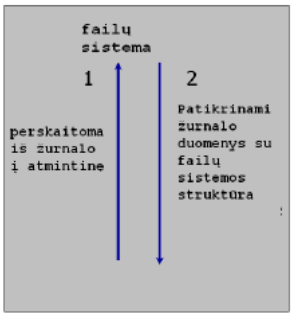
Unix/Linux failų sistemos:

- UFS, NFS
- Ex, ext2, ext3, ext4

Žurnalinė failų sistema:

- Operacijos, apie kurias rašomi įrašai:
    - Create, link, mkdir, truncate, allocating write, ...
- Kiekviena sistemos vykdoma operacija gali būti surišta su eilės metaduomenų keitimais (naujinimais)
- Realūs įrašymai į diską gali būti vykdomi:
    - Asinchroniškai
    - Prieš darant įrašą
- Failų sistemose, kurios naudoja žurnalus, visi vykdomi pakeitimai yra fiksuojami žurnaluose
    - Šie įrašai paprastai laikomi atskirame failų sistemos skyriuje
- Kokie duomenys gali būti rašomi į žurnalus?
    - Pilnas duomenų įrašas
        - Į žurnalą yra įrašomi visi duomenys
        - Tai duoda didelę failų sistemos neprieštaringumo garantiją
        - Reikalauja daug pastangų ir darbo sąnaudų
    - Metaduomenų įrašas
        - Yra greitesnis
        - Įrašomi tik metaduomenys (failo ar aplanko)
- Panaudojimas:
    - Sumontavus failų sistemą, failų sistemos tvarkyklė atlieka patikrinimą, nustatydama ar failų sistema yra stabili
    - Jei dėl kažkurių priežasčių reikia sutvarkyti metaduomenis, vietoj to, kad atlikti pilną metaduomenų skanavimą kaip fsck, yra peržiūrimas žurnalas
    - Kadangi žurnale yra chronologiški logai visų nesenų metaduomenų pakeitimų, tai ji paprasčiausiai patikrina tuos metaduomenų keitimo įrašus, kurie nesenai yra užregistruoti
    - Šis procesas yra žymiai greitesnis, nei pilnos failų sistemos struktūros analizė
    - Taigi sistema gali būti atstatoma per keletą sekundžių ir vėl tampa prieinama klientams
- Žurnalinės failų sistemos privalumai
    - Asinchroninis metaduomenų rašymas
    - Greitas atstatymas: jis priklauso tik nuo žurnalo dydžio, o ne nuo failų sistemos dydžio
- Trūkumai
    - Papildomi įrašymai į diską

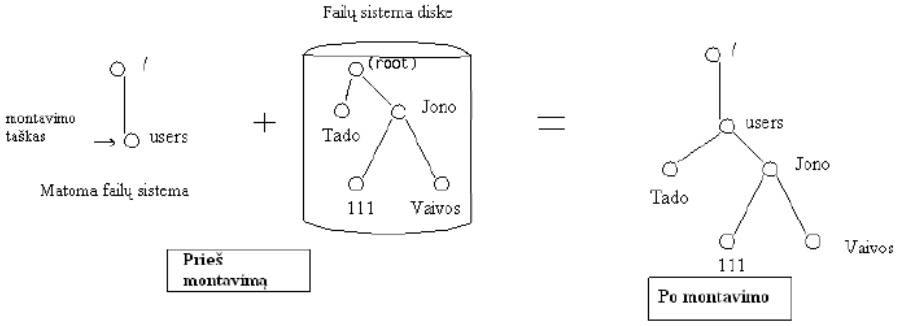
- Reikia vietos žurnalų saugojimui (nereikšminga)
- Dauguma modernių Linux failų sistemų naudoja žurnalines failų sistemas



Failų sistemos montavimas

Kiekviena diske esanti failų sistema turi savo šakninį katalogą. Galima sukurti bendrą failų sistemą ją sumontuojant iš atskirų failų sistemų (sukurti ryšį tarp katalogo vienoje failų sistemoje ir šakninio katalogo kitoje failų sistemoje). Tai leidžia sukurti bendrą medį iš kelių failų sistemų. Tai gali būti išplečiama ir per tinklą, sumontuojant kelių kompiuterių failų sistemas.

- Atskirame diske esančios failų sistemos įjungimas į egzistuojančią katalogų struktūrą yra žinomas kaip failų sistemos montavimas
- Primontuojama failų sistema gali rastis viename iš tiesiogiai prijungtų diskų (lokali failų sistema) arba būti nutolusios tinklinės failų sistemos dalimi
- Montavimas susieja primontuojamą failų sistemą su tam tikru egzistuojančios failų sistemos katalogu
- Iki montavimo failai, esantys primontuojamame diske nėra pasiekiami vartotojams
- Katalogas, kuriame vykdomas montavimas yra vadinamas montavimo tašku
- Motavimo tašku turi būti tuščias egzistuojančios sistemos katalogas



Išmontavus failų sistemą ji tampa laikinai neprieinama vartotojams, bet išmontavimas neišmeta failų sistemos iš disko. Sumontuota failų sistema yra automatiškai išmontuojama po kompiuterio stabdymo komandos „shutdown“.

6. Svečių Paskaitų Medžiaga (11T, 13T)

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

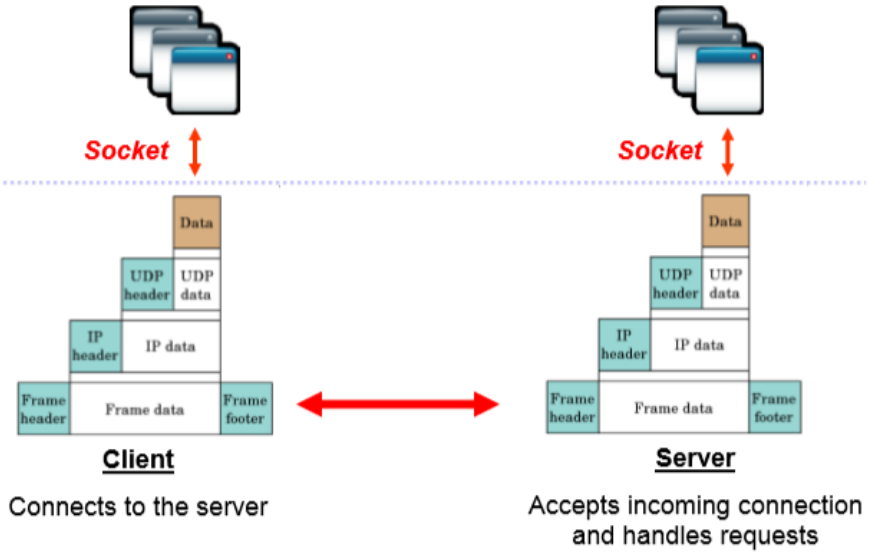
Tinklo soketas yra virtualus endpointas parūpinantis kelias funkcijas duomenų keitimuisi. Soketą apibrėžia IP adresas ir prievado (porto) numeris. Komunikavimas vyksta tarp dviejų procesorių, taigi yra nusakomas dviem galiniais taškais (endpointais)

Socket API yra aplikacijos programavimo sąsaja parūpinta operacinės sistemos, kuri leidžia aplikacijoms kontroliuoti ir naudotis tinkle soketais. Interneto soketų API paprastai yra paremti Berkeley soketų standartu. Berkeley soketai leido programuotojams panaudoti internet galimybes savo produktuose, šios API tikslas – leisti vartotojams kurti aplikacijas , kurios bendrauja tarpusavyje komunikuodamos tinkle.

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

- Prievado (porto) numerio surišimas su soketu, ryšio inicijavimas arba priėmimas sokete, soketo sukūrimas, suardymas
- Duomenų rašymas/skaitymas per soketą

Tinklo soketas inkapsuliuoja visus sluoksnius ir parūpina pagrindines rutinas ir funkcijas programuotojui (developer) ir aplikacijoms



Ką parūpina tinklo soketai ir API?

- Hostname konvertavimą į IP adresą ir atvirkščiai naudojant DNS sistemą
- Prisijungimą prie nutolusio hosto ir sesijos atidarymą
- Sesijos valdymą/tvarkymą
- Duomenų siuntimą ir priėmimą, kada įmanoma
- Sesijos uždarymą pabaigoje

Skirtumai tarp TCP ir UDP protokolų

UDP yra stateless protokolas

- Serveriui nereikia priimti ryšio
- Serveris tiesiog priima ir išsiunčia paketus BE sesijos valdymo/tvarkymo (without managing sessions)
- Greitos užklausos (requests) ir atsakymai
- Mažas duomenų kiekis

TCP yra stateful protokolas

- Klientas turi inicijuoti ryšį
- Serveris turi priimti ryšį
- Didelis duomenų kiekis

Mobiliosios OS: mobilieji įtaisai, jų savybės. Android OS program steko architektūra. Dalvik virtuali mašina. Apribojimai programinei įrangai

Mobilieji įtaisai: išmanieji telefoani, planšetės, išmanieji laikrodžiai etc.

Svarbiausios išmaniojo mobiliojo įtaiso savybės:

- Platforma (OS) su SDK, kurių dėka galima kurti ir valdyti programas
- Galima vienu metu dirbti keletu program (multitasking)
- Lanksčiai konfigūruojamas, didelis jungiamumas
- Mobiliosios OS, lyginant su įprastinėmis OS, yra paprastesnės, jose dažnai naudojami tik mobilieji ar bevieliai prisijungimo būdai, mobiliajai aplinkai adaptuoti daugialypės terpės formatai ir įvairūs įvesties metodai.

Populiariausios OS:

- Android
- iOS
- Symbian
- Windows
- Bada

Android yra atvirojo kodo platforma paremta Linux OS branduoliu. Tai pilnas programinės įrangos stekas, kurį be modifikuoto Linux OS branduolio sudaro tarpinė programinė įranga ir pagrindinės taikomosios programos.

Android OS program stack sudaro 4 sluoksniai:

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

Andoid Runtime aplinka veikia virš Linux OS branduolio ir užtikrina sąveiką su žemo lygio technine įranga, valdo procesų gyvavimo laiką ir taikomųjų programų atmintį.

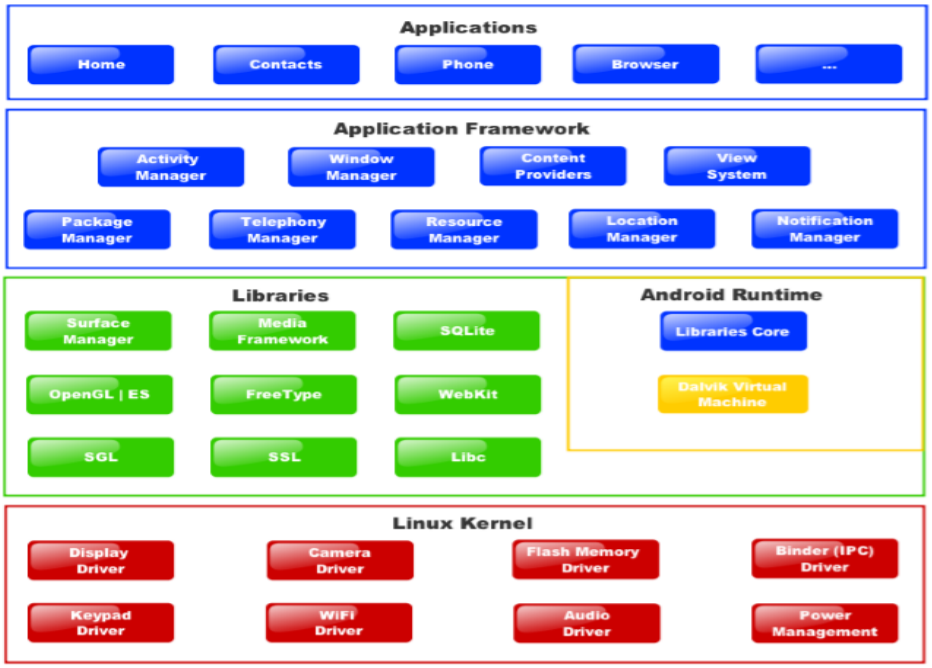
Android OS apima grupę specialiai operacinei sistemai sukurtų sisteminių C/C++ bibliotekų, kuriomis vartotojai – programų kūrėjai gali naudotis per programų karkasą. Kai kurios bibliotekos turi savo šaknis atviro kodo projektuose. Dėl licencijavimo konfliktų buvo nuspręsta Android OS įdiegti savą C biblioteką ("Bionic") ir sukuti specifinę Java programų vykdymo aplinką – **Dalvik virtualią mašiną.**

Dalvik Virtuali Mašina

Programos rašomos Java kalba, tačiau nėra JVM. Java klasės kompiliuojamos į Delvik (.dex) vykdomuosius failus ir vykdomi Dalvik VM, kuri yra sukurta specialiai Android. DVM specialiai optimizuota įrenginiams su ribotu energijos šaltiniu, atmintimi ir CPU.

Dalvik VM (DVM) – registrais paremta virtuali mašina, kuri buvo optimizuota siekiant užsitikrinti, kad mobilusis įranginys galėtų efektyviai vykdyti daugelį mašinos egzempliorių.

Lyginant su Java VM, dėl *dex failų ir Dalvik VM struktūros, programų vykdymui naudojama daug mažiau atminties. Android OS kiekviena Android taikomoji programa veikia kaip atskiras, vieną giją turintis procesas savo Dalvik virtualioje mašinoje, perduodama visa atminties ir procesų valdymo atsakomybę programų vykdymo aplinkai.



Apribojimai programinei įrangai:

- Programos neturi būti uždaromos, kai su jomis einamuoju momentu yra atlikti norimi veiksmai (pvz. Orų prognozės)
- Mobilieji įrenginiai neturi virtualiosios atminties, todėl susiduria su gana dideliais atminties naudojimo apribojimais
- Vartotojų patogumui persijungimas tarp taikomųjų program mobiliąjame įrenginyje turi būti labai greitas. Naujai programai paleisti leidžiama skirti mažiau nei 1 sekundę.
- Programos turi būti kuriamos naudojantis vienu API, kuriame yra pakankamai funkcijų visoms užduotims realizuoti ir kuris turi būti prieinamas trečiųjų šalių kūrėjams.