Operacinių sistemų konspektas

# Mirties taškas

## Mirties taškas. Jo susidarymo sąlygos. Resursų priskyrimo grafai ir mirties taško situacija

Mirties taškas sistemoje - tai tokia situacija, kai ilgam laikui yra užblokuojama grupė procesų, kurie varžosi dėl tų pačių sistemos išteklių arba komunikuoja su kitais, užblokuotais procesais.

Mirties taškas kyla kai du ar daugiau procesų turi konfliktuojančius poreikius.

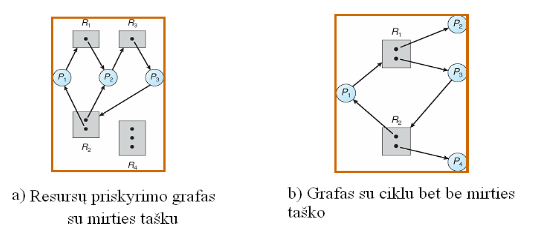
Mirties taškas sistemoje tampa galimu, jei sistemoje susidaro šios sąlygos:

Tarpusavio išskirtinumo

Turėjimo ir laukimo

Neperėmimo

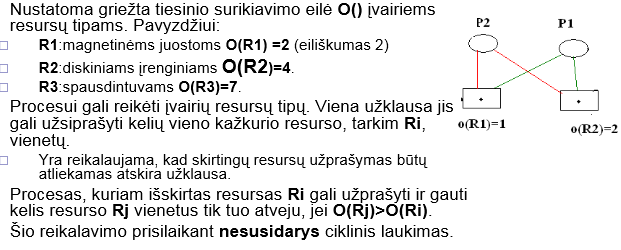
Ciklinio laukimo sąlyga



Dažniausiai nėra bendro sprendinio, kuris nusakytų kaip elgtis mirties taško atveju. Kai kurių operacinių sistemų kūrėjai skaito, kad jų sistemose mirties taškas negali niekad susidaryti.

Pavyzdžiu gali būti operacinė sistema - Unix SVR4

## Mirties taško prevencija: a) tiesioginiai būdai (jų galimybės, taikomi metodai)



b) netiesioginiai būdai

Netiesioginiai metodai yra orientuoti į pirmas tris sąlygas

1.Tarpusavio išskirtinumo sąlyga

♦Šios sąlygos egzistavimo dažnai negalima suardyti.

2.Turėjimo ir laukimo sąlyga

♦Šią sąlygą galima suardyti, reikalaujant kad procesas visų jam reikalingų resursų užprašytų iš karto.

♦Tokio metodo minusai:

**ν** Gali gautis badavimo situacija.

**ν** žemas resursų panaudojimas.

**ν** procesas turi žinoti, kokių resursų ir kiek jų jam reikės.

3. Neatėmimo sąlyga

♦ Ši sąlyga gali būti sutrukdoma keliais būdais. Metodo esmė yra ta, kad iš proceso yra atimami jam anksčiau išskirti resursai.

**ν** Jei procesas turi kažkurį resursą ir jam negali būti tuojau pat paskirti kiti jam tuo momentu reikalingi resursai, tai jo turėti resursai iš jo yra atimami.

**ν** Bet, jei procesas turi atlaisvinti resursą, kurį jis naudoja, šio resurso būvis turi būti išsaugomas, kad vėliau būtų galima pratęsti (atstatyti) proceso vykdymą.

## Mirties taško vengimas.

Leidžiama įvykti pirmoms trims sąlygoms, tačiau vykdomas protingas resursų išskyrimas, kuriam esant mirties taškas niekad nebus pasiektas.

Taikant šį metodą yra leidžiama didesnė konkurencija dėl resursų nei prevencijos atveju.

**ν** Taikomi du būdai:

♦Procesas nepradedamas, jei jo resursų užklausa gali vesti į mirties taško susidarymą.

♦Resursų padidinimo užklausa nevykdoma, jei toks išskyrimas ves prie mirties taško susidarymo.

**ν** Abiem šiais atvejais iš anksto turi būti nusakomi maksimalūs proceso poreikiai resursams.

## Bankininko algoritmas, jo taikymo pavyzdys.

♦Bet kuriuo momentu sistemos būvis gali būti apibrėžiamas R(i), C(j,i) reikšmėmis, kur i žymi R resurso tipą, o j procesą.

♦Matricos A elementas A(j,i) parodo, koks i–tojo resurso kiekis jau yra skirtas j-tajam procesui. (visoms j,i reikšmėms).

♦Sumarinis esamas laisvas i–tojo resurso kiekis yra nusakomas vektoriaus V(i)-tąja komponente:

♦Matricos N elementai N(j,i) nusako, kiek i-to resurso reikia tam, kad j-tasis procesas galėtų baigti įvykdyti savo užduotį.

**ν** Sprendžiant, ar gali būti patenkinama proceso užklausa resursui, bankininko algoritmas testuoja, ar užklausos patenkinimo rezultate sistemos būvis gausis saugus.

♦Jei skyrus resursą procesui rezultate gausis saugus būvis, tai užklausa yra tenkinama,

♦priešingu atveju ji nėra tenkinama.

**ν** Būvis yra laikomas saugiu, jei galima sudaryti tokią procesų seką {P1...Pn}, kuriai esant bus galima kiekvienam iš sekoje esančių procesų priskirti visus to proceso įvykdymui reikalingus resursus. Esant saugiam būviui visada visi procesai bus pilnai įvykdyti.

**ν** Lai Q(j,i)tai i–tojoresurso kiekis, kurio tam tikru momentu užsiprašo j-tasisprocesas.

### Algoritmo žingsniai:

♦Jei Q(j,i)<=N(j,i) visiems i, tai einama prie sekančio žingsnio, priešingu atveju gaunama klaidos sąlyga (j-tojo proceso resursų pareikalavimas viršytas).

♦Jei Q(j,i)<=V(i) visiems i tai eiti prie sekančio žingsnio, priešingu atveju laukti, nes esamų laisvų resursų kol kas nepakanka.

♦ Pretenduojama į tai, kad resurso poreikiai bus tenkinami ir nustatomas naujas resurso išskyrimo būvis:

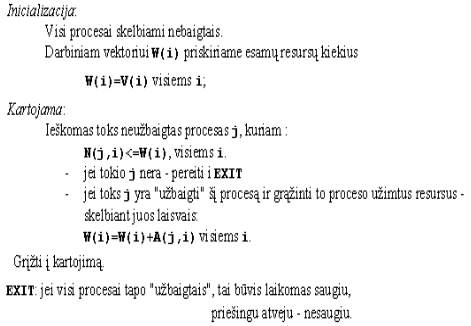
**ν** V(i)=V(i) -Q(j,i)visiems i

**ν** A(j,i)=A(j,i)+Q(j,i)visiems i

**ν** N(j,i)=N(j,i)-Q(j,i)visiems i

♦ Jei šio priskyrimo rezultate būvis gaunasi saugus, tai iš tikrųjų yra įvykdomas resurso Q(j,i) išskyrimas procesui j. Priešingu atveju j procesas turės laukti resurso Q(j,i) išskyrimo ir išsaugomas senas būvis.

## Saugaus būvio nustatymo algoritmas, jo taikymo pavyzdys



## Mirties taško nustatymo algoritmas, jo taikymo pavyzdys

**ν** Šiuo atveju procesų užklausos resursams visad yra tenkinamos. Operacinei sistemai tokiu atveju reikia:

♦algoritmo patikrinimui, ar nėra susidaręs mirties taškas.

♦algoritmo, kuris nusakytų, kaip išeiti iš šio mirties taško.

**ν** Tikrinimas, ar yra susidaręs mirties taškas gali būti atliekamas kartu su kiekviena užklausa resursams. Aišku, toks dažnas tikrinimas vartos daug CPU laiko.

**ν** Naudosime tas pačias matricas bei vektorius kaip ir resursų priskyrimo algoritme.

**ν** Bus atžymimi visi procesai, kurie nėra įėję į mirties taško situaciją. Pradžioje visi procesai yra nepažymėti.

**ν** Atliekama:

**ν** Atžymimas kiekvienas j-tas procesas, kuriam išskirtas resursų kiekis A(j,i)=0, kiekvienam i-tajam resurso tipui (kadangi šie procesai nėra mirties taške).

**ν** Nustatomas darbinis vektorius W(i)=V(i), visoms i reikšmėms.

**ν** Kartojama:

**ν** Randamas nepažymėtas procesas j, kuriam Q(j,i)<=W(i) visoms reikšmėms. Sustojama, jei tokio j proceso nėra.

**ν** Jei toks j procesas yra: j-tasis procesas pažymimas ir nustatoma W(i)=W(i)+A(j,i), visoms i.

**ν** Grįžtama į kartojimą.

**ν** Gale: kiekvienas nepažymėtas procesas yra mirties taške.

## Išėjimas iš mirties taško

**ν** Aptikus susidariusį mirties tašką žudomas procesas, esantis mirties taško situacijoje.

**ν** Panaudojamas procesų suspendavimo/atnaujinimo mechanizmas

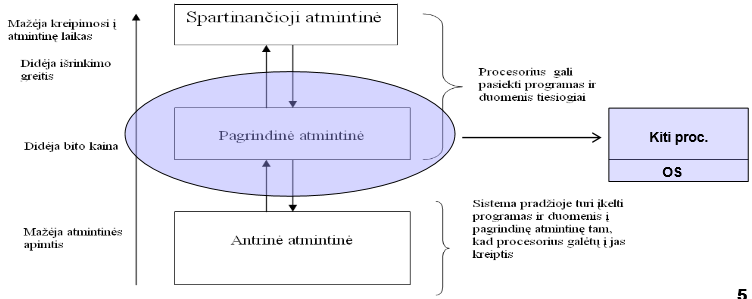
♦ Pavyzdžiui, procesai iškeliami į swap sritį

**ν** Procesų vykdymas grąžinamas atgal

♦ Reikia išsaugoti tam tikrų kontrolinių taškų kontekstą

♦ Neprarandamas su procesu atliktas darbas

# Atminties hierarchija ir charakteristikos

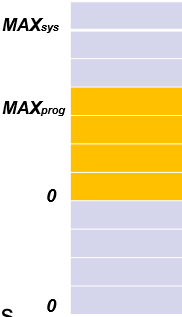


Reikalavimai pagrindinės atmintinės valdymui. Pagrindinės AV sąvokos

### Reikalavimai:

**ν** Patalpinimo vietos pakeitimo galimybė

**ν** Apsauga

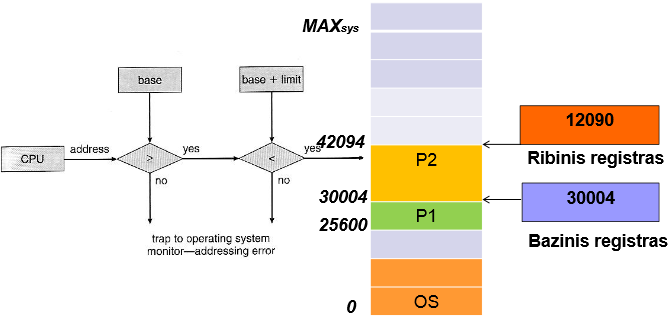
**ν** Dalinimasis

**ν** Loginė organizacija

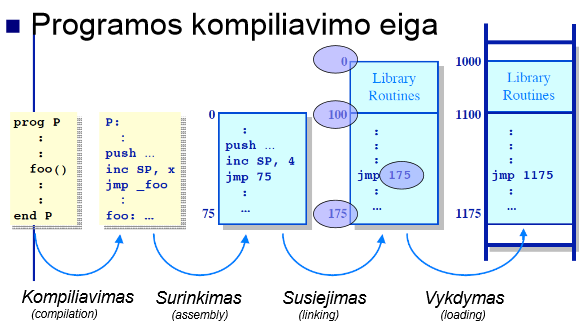
**ν** Fizinė organizacija

### AV sąvokos:

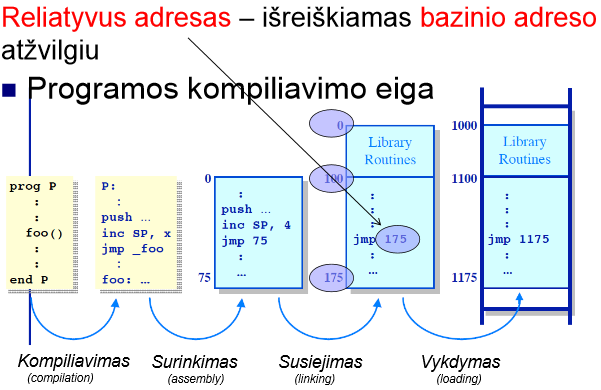
**ν** Adresų sritys. Fizinė sritis – techninės įrangos palaikoma adresų sritis. Nuo 0 iki MaxSys. Loginė virtuali adresų sritis – procesui matoma atmintinės sritis nuo 0 iki MaxProg.

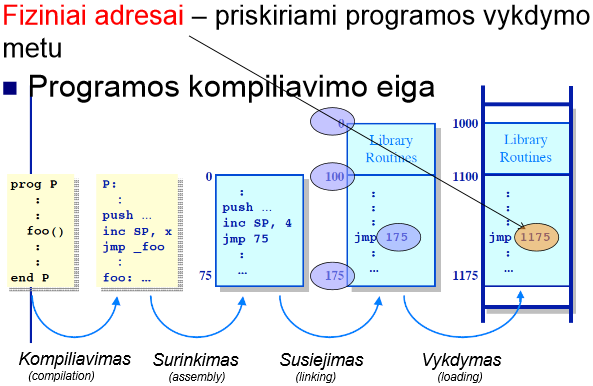
**ν L**oginių adresų sritį apibrėžiantys registrai

**ν** Loginių adresų formavimas



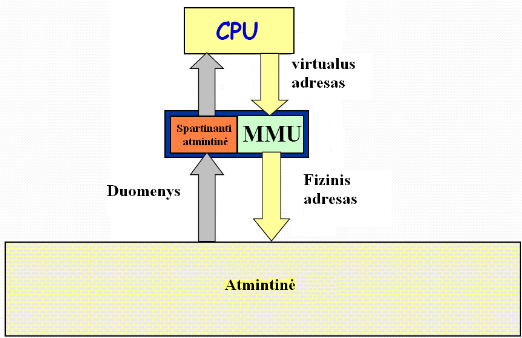
Programos kompiliavimo etapai, jų paskirtis, sprendžiami uždaviniai



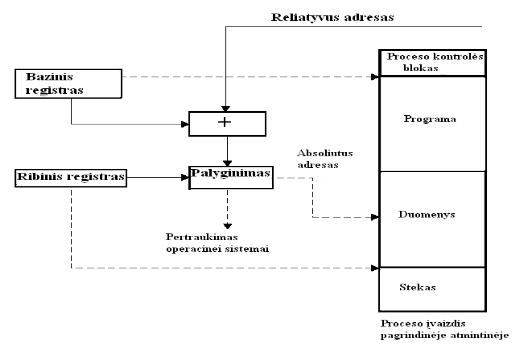


Adresų transliacijos schemos

### Loginio adreso transliavimas į fizinį



### Reliatyvaus adreso transliacija į fizinį adresą



Rezidentinės proceso dalies sąvoka

Į pagrindinę atmintinę įkelta proceso dalis dar yra vadinama rezidentine arba darbine dalimi.

**ν** Kodėl gi proceso vykdymui pakanka mažesnės srities nei viso proceso dydis?

♦Tai surišta su lokališkumo principu, kuris pasireiškia tuo:

**ν** dažniausiai didelę proceso vykdymo laiko dalį procesas vykdo nedidelę komandų aibę (vyksta ciklas)

**ν** naudoja greta esančius duomenis,

**ν** dauguma skaičiavimų yra vykdomi nuosekliai.

♦ Todėl bet kuriuo momentu procesui pakanka nedidelės rezidentinės srities pagrindinėje atmintinėje.

Atmintinės valdymo sprendžiami uždaviniai

**ν** 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?

Proceso patalpinimo pagr. atmintinėje būdai

### **ν** Nuoseklių (ištisinių adresų) zonos skyrimas

♦ procesas egzistuotų kaip vientisas, nuoseklių adresų erdvėje esantis blokas.

♦tai labai paprastas atmintinės dalinimo būdas,

♦atsiranda problemos:

**ν** tinkamo dydžio laisvo bloko atradimas

**ν** netinkamas atmintinės panaudojimas.

### **ν** Neištisinės srities skyrimas

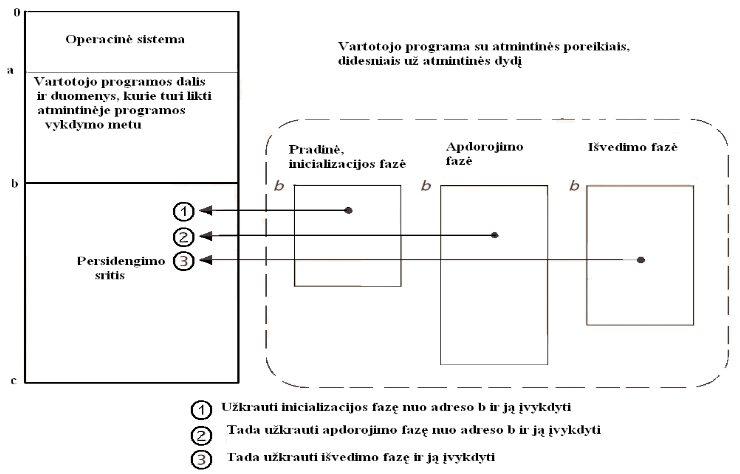
♦procesas, jo duomenys skaldomi tam tikro dydžio, atskirose atmintinės vietose talpinamais gabalais (puslapiais, segmentais).

♦lengviau atrasti tinkamas proceso patalpinimui vietas atmintinėje,

♦leidžia padidinti procesų, vienu metu esančių pagrindinėje atmintinėje kiekį,

realizacija yra sudėtingesnė.

Perdengimo mechanizmas

  
Fiksuoto dydžio skyriai, jų pliusai, minusai

**ν** Pagrindinė atmintis yra sudaloma į eilę nepersidengiančių skyrių (dalių). Šios dalys gali būti tiek vienodo tiek skirtingo dydžio.

**ν** Procesas, kurio dydis yra mažesnis arba lygus skyriaus dydžiui, gali būti patalpinamas į šį skyrių.

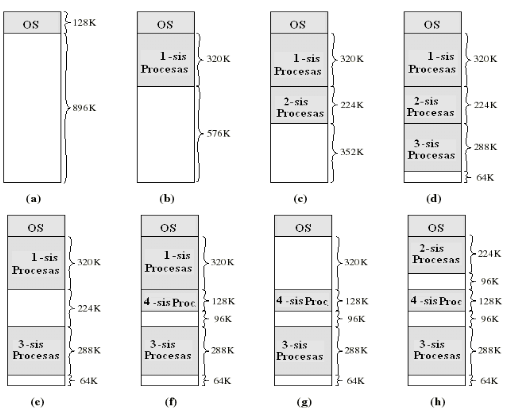
**ν** Procesorius gali greitai persijungti tarp procesų.

**ν** 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.

**ν** Jei visi skyriai yra užimti, operacinė sistema gali iškelti (swap) procesą iš jo užimamo skyriaus.   
Dinaminis skyrių formavimas: pliusai, minusai. Talpinimo algoritmai dinaminių skyrių atveju (taikymo pavyzdžiai), algoritmų pliusai, minusai

**ν** Taikant šį principą skyrių kiekis, jų dydis yra kintami.

**ν** Kiekvienam procesui jį talpinant pagrindinėje atmintinėje yra išskiriamas tokio dydžio skyrius, kokio jis prašo.



### Talpinimo algoritmai dinaminių skyrių atveju:

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

**ν** Galimi ir naudojami šie algoritmai:

♦Geriausiai tinkančio

♦Pirmo tinkamo

♦Sekančio tinkančio.

♦Blogiausiai tinkančio

Talpinimo Algoritmų Savybės:

**ν** Taikant „sekantis tinkamas“ algoritmą dažnai naujai talpinamam procesui yra priskiriamas didžiausias blokas, esantis pagrindinės atmintinės pabaigoje.

**ν** Taikant pirmo tinkamo paiešką, skylės radimas sukasi apie atmintinės pradžią, jį taikant gaunama mažesnė fragmentacija nei „sekančio tinkamo“ atveju.

**ν**„Geriausiai tinkančio“ paieška yra susijusi su mažiausio neužimto bloko

suradimu: likęs laisvas fragmentas bus gaunamas mažiausias.

**ν**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.

Informacijos apie užimtas ir laisvas sritis saugojimas

Atminties dalinimas naudojant „bičiuliškos sistemos“ algoritmą (taikymo pavyzdys)

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

**ν**Modifikuota šio algoritmo versija yra naudojama UNIX SVR4 .

**ν**Atmintinės blokai, kurie yra išskiriami procesams yra 2^{K} dydžio

**ν** 27=128<100<28=256 –Netinka

**ν** 26= 64<100<27=128-tinka, skiriam 128 KB

**ν**Jei du bičiuliai tampa laisvais , bičiuliai yra apjungiami.

**ν**Operacinė sistema palaiko keletą sąrašų apie esančias skyles.

♦i-tas sąrašas apima skyles, kurių dydis yra 2^{i}.

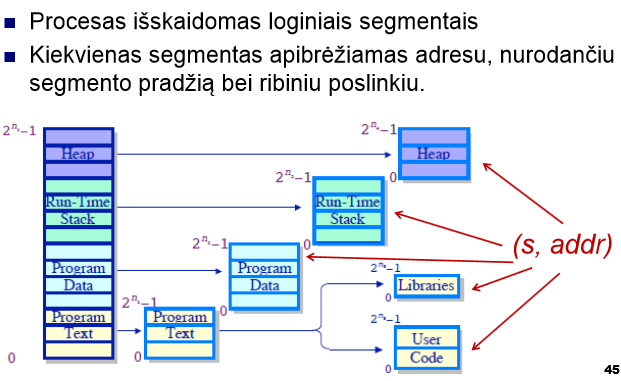
♦Kai tik pora bičiulių atsiranda i-tame sąraše, jie yra išmetami iš šio sąrašo ir apjungiami į vieną bendrą skylę (i+1) sąraše.

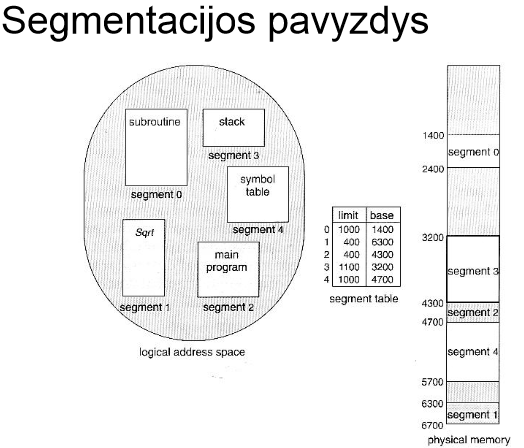
**ν**Atsiradus naujai užklausai, t.y. norint patalpinti K dydžio procesą, tokį:

♦kuriam galioja: 2^{i-1} < k <= 2^{i}

♦yra patikrinamas i-tas sąrašas. Jei šis sąrašas yra tuščias, tikrinamas (i+1) sąrašas.

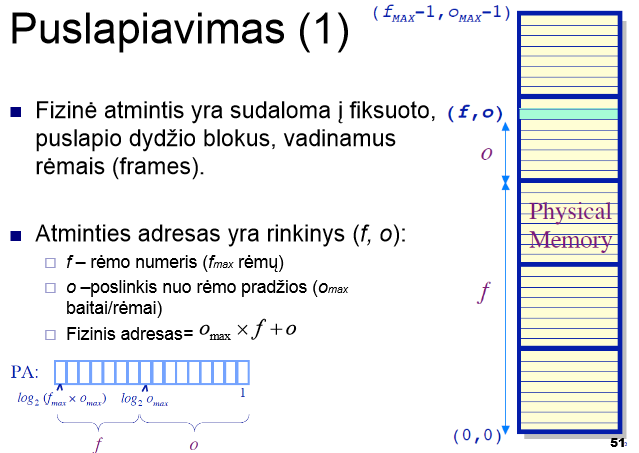
♦Jame radus skylę ji bus sudaloma į du bičiulius, viena iš dalių bus

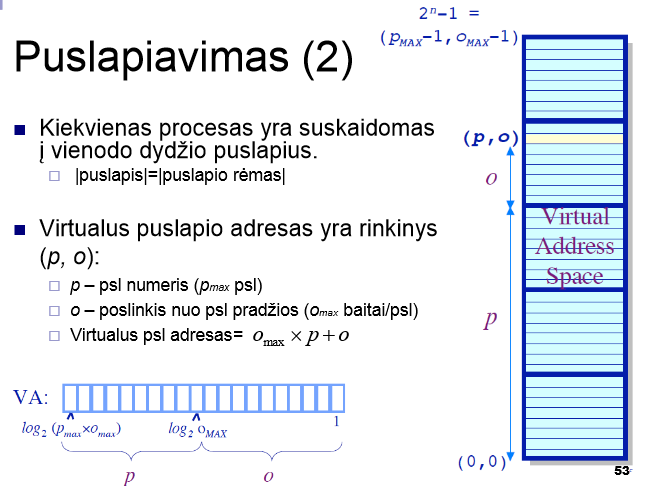
priskirta procesui, o kita įtraukta į i-tą sąrašą.  
Atminties skirstymas taikant segmentacijos principus (pliusai, minusai), adreso transliavimo schema. 

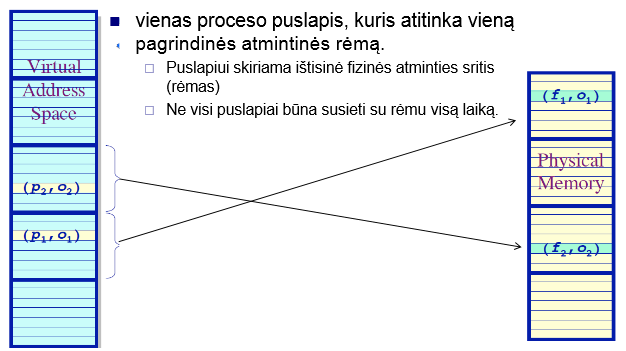


**ν**Išorinė fragmentacija egzistuoja.

**ν**Gali būti sprendžiama skaidant segmentą jo viduje (puslapiuoti segmentai)

Puslapiavimo principai (adreso sudedamosios dalys), adreso transliacijos schema. Su puslapiavimu susijusios problemos





Fragmentacija naudojant išorinį puslapiavimą:

**ν** Nebelieka išorinės fragmentacijos :

♦Proceso puslapiai proceso talpinimo į pagrindinę atmintinę metu gali užimti laisvus rėmus (page frames).

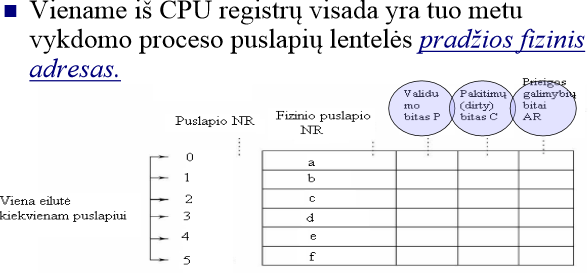
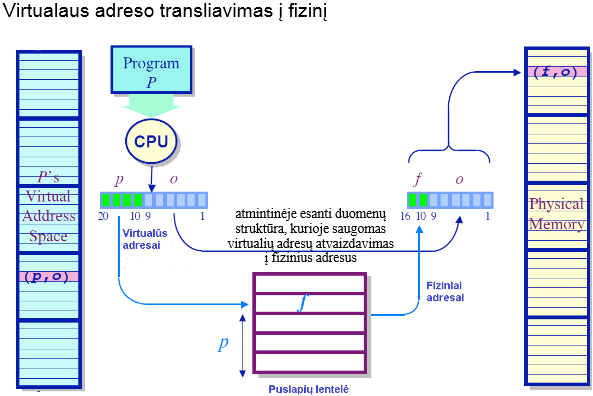
♦Nauda:

**ν** procesui nebūtina užimti ištisinės adresų erdvės pagrindinėje atmintinėje

**ν** proceso puslapiai gali būti išbarstomi į egzistuojančius laisvus rėmus.

♦Procesui pasibaigus, tiesiog padaugėja laisvų rėmų.

**ν** Kadangi puslapio (rėmo) dydis yra pakankamai nedidelis, tai sumažėja ir vidinė fragmentacija.



### 

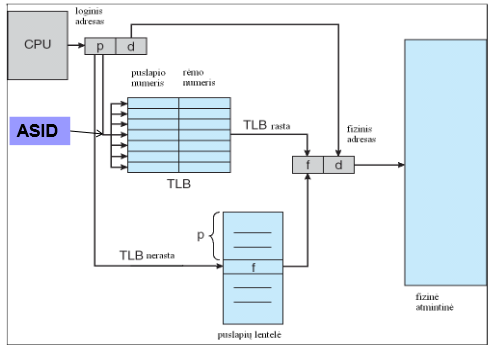
### Puslapiavimo problemos ir sprendimai:

**ν** Du kreipiniai į atmintį – TBL lentelės;

**ν** Puslapio dydis – 4KB, 8KB arba 16KB

**ν** Rezidentinis proceso dydis – vienodi rėmų kiekiai ir rėmų kiekis proporcingas procesų dydžiui.

Adreso transliavimo schema naudojant TLB įrašus

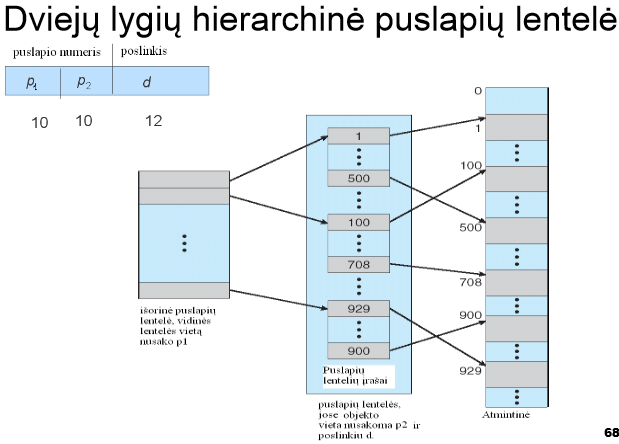


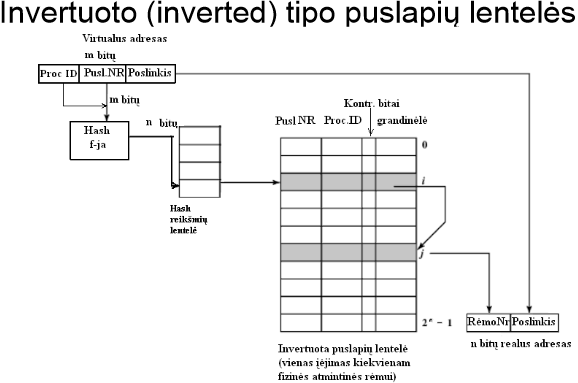
Puslapių lentelių tipai, jų naudojimo principai transliuojant adresą.

**ν** Vieno lygio,

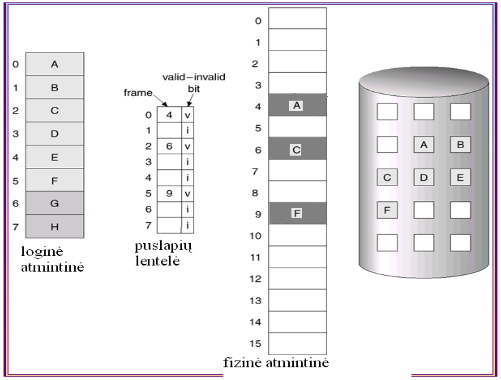
**ν** hierarchinės,

**ν** invertuotos santraukos (hash) tipo, puslapių lentelės





Puslapio trūkumo situacija ir jo apdorojimo schema



### Puslapio trūkumo apdorojimas

1.Techninė įranga praneša branduoliui

2.Išsaugomi pagrindiniai registrai

3.OS nusprendžia, koks virtualus puslapis yra reikalingas

4.OS patikrina ar adresai yra teisėti, ieško tinkamo pagr. atmintinės rėmo.

5.Jei parinktas rėmas yra modifikuotas (dirty), jis perrašomas į diską.

6.OS įkelia naują puslapį iš disko

7.Atnaujinamos puslapių lentelės

Grįžtama prie komandos sukėlusios puslapio trūkumą:

Planuojamas proceso tęsimas

Atstatomi registrai

Programa tęsia savo veiksmus  
Rezidentinis puslapio dydis, jį apibrėžiantys veiksniai, puslapių mainų įtaka jo dydžiui.

### Rezidentinių puslapių kiekis:

* Rezidentinio kiekio minimumo bei maksimumo reikšmės.

Maksimalų kiekį riboja aktyvių procesų kiekis

Maksimalų kiekį riboja laisvų rėmų kiekis.

* Priskyrus mažai puslapių gali smarkiai išaugti puslapių mainai.
* Minimalų kiekį nusako architektūros naudojamos komandos:

Turi valdyt tiek skirtingų rėmų, į kiek skirtingų rėmų gali kreiptis bet kuri komanda.

Komanda taip pat gali būti susieta su kelias rėmais

* Kai laisvos pagrindinės atmintinės kiekis nukrinta žemiau leistinos ribos, operacinė sistema iškelia tuos proceso puslapius, kurie viršija minimumo reikšmę.
* Jei procesas turi mažiau puslapių nei min –procesas suspenduojamas

Operacinė sistema, įkeldama puslapius į pagrindinę atmintinę gali elgtis įvairiai:

tai gali būti atliekama atsiradus tam tikro puslapio poreikiui (demand paging) - pradžioje gali būti įkeliamas tik pirmas reikalingas puslapis, o po to keliamas tas puslapis, į kurį vyksta kreipinys. Arba operacinė sistema gali bandyti nustatyti, kurių puslapių procesui reikės ir iš anksto įkelti keletą puslapių (prepaging).

Keičiant puslapius galima:.

Lokali politika–keičiamas tam pačiam procesui priklausantis puslapis nauju puslapiu.

Globali – keičiamas bet kuriam procesui priklausantis puslapis.

### Puslapių mainai

Mainus sukelia tai, kad atmintinės kiekis yra ribotas.

* Operacinė sistema, norėdama įkelti trūkstamą puslapį turi surasti jam vietą:

Ieškomas laisvas rėmas.

Jei laisvo rėmo nėra, OS turi išlaisvinti vieną iš užimtų rėmų.

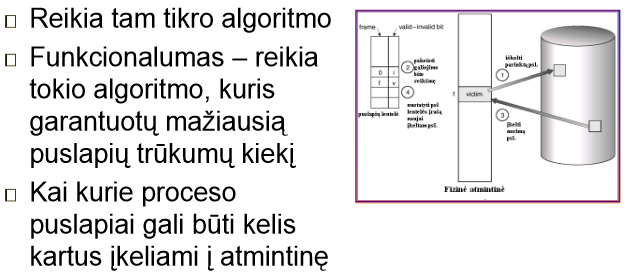
* Trūkstamas puslapis įkeliamas, suaktyvinus skaitymo iš disko užduotį.

Įkėlus puslapį daromi pakeitimai puslapių lentelėje,

įjungiamas įkelto puslapio galiojimo bitas

išjungiamas iškeliamo puslapio galiojimo bitas

Puslapių keitimo algoritmai, kiekvieno jų pliusai, minusai



Bendrai naudojamos atmintinės sritys