7. Operacinės sistemos

Nežiūrint į tai, kad kompiuterių architektūros dalyko esmė sufokusuota ties aparatine kompiuterių dalimi, yra viena programinės įrangos rūšis kuri taip pat turi būti čia išnagrinėta – kompiuterių operacinės sistemos. Operacinė sistema yra programa, kuri valdo kompiuterio resursus, suteikia tam tikrą servisą programuotojui/vartotojui ir reguliuoja taikomųjų programų vykdymą. Minimalų supratimą apie operacines sistemas būtina žinoti jau vien tam, kad suvokti kaip CPĮ valdo visą kompiuterizuotą sistemą. Tai ypač palengvins pertraukčių efekto ir atminties hierarchijos valdymo nagrinėjimą.

Pradžioje apžvelgsime operacinių sistemų trumpą istoriją ir programuotojui/vartotojui teikiamo serviso galimybes. Pagrindinis dėmesys bus skirtas dviem operacinių sistemų funkcijoms, kurios yra labiausiai susiję su kompiuterių architektūros studijomis: programų vykdymas ir atminties valdymas.

7.1. Operacinių sistemų apžvalga

7.1.1. Operacinių sistemų tikslai ir funkcijos

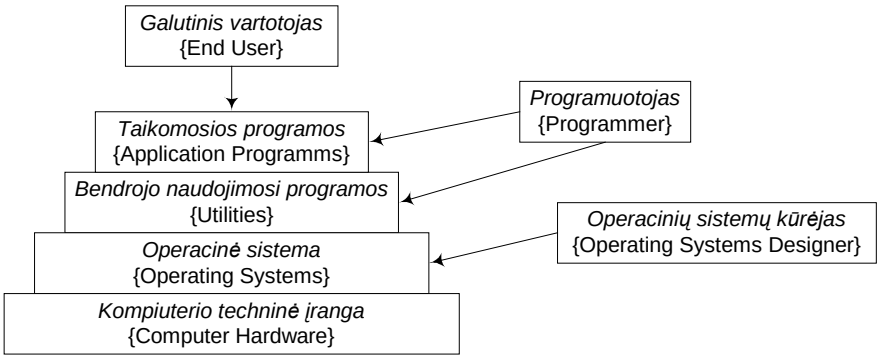
Operacinė sistema yra programa, valdanti taikomųjų programų vykdymą, o taip pat atliekanti interfeiso tarp kompiuterio vartotojo ir kompiuterio techninės įrangos funkciją. Galima teigti, kad operacinė sistema turi du tikslus arba vykdo dvi funkcijas:

- *Patogumas*: operacinė sistema kompiuterį daro patogesniu vartojimui.
- *Efektyvumas*: operacinė sistema užtikrina kompiuterio resursų efektyviausią naudojimą.

Išnagrinėkime detaliau šiuos du operacinės sistemos aspektus.

Operacinė sistema – interfeisas tarp kompiuterio ir vartotojo

Kompiuterizuotos sistemos techninės įrangos prigimtis yra hierarchinė. Lygiai tą patį galima pasakyti ir apie programinę įrangą (7.1 pav.).



7.1 pav. Kompiuterio sistemos lygmenys

Daugeliu atvejų pagrindinė kompiuterizuotos sistemos užduotis yra užtikrinti vienos arba kelių taikomųjų programų – aplikacijų vykdymą. Šių aplikacijų vartotojas vadinamas galutiniu vartotoju {end user} arba tiesiog *vartotoju,* tačiau jis nėra siejamas su kompiuterio architektūra. Tokiu būdu, kompiuterio sistemą vartotojas mato tiks iš savo aplikacijų pusės. Tokios aplikacijos, pvz., programavimo kalbos, yra kuriamos *aplikacijų programuotojo* (programinių produktų kūrėjo {application programmer}). Natūralu, jeigu aplikaciją kurti kaip mašininių instrukcijų, visiškai užtikrinančių kompiuterio techninės įrangos valdymą, rinkinį, daugeliu atvejų tai bus labai komplikuotas uždavinys. Šiam uždaviniui palengvinti taikomas sisteminių programų rinkinys. Šį rinkinį sudarančios programos vadinamos *utilitomis* {utilities} arba sisteminėmis programomis. Jos vykdo dažnai naudojamas funkcijas, kurios padeda, pvz., atidaryti programą, dirbti su failais ir valdyti Į/I įrenginius. Programuotojas, kuriant aplikacijas turi mokėti naudoti išvardytas galimybes, o veikianti aplikacija, savo ruožtu, vykdant tam tikras funkcijas, į savo darbą turi

įtraukti šias utilitas. Pati svarbiausioji sisteminė programa yra *operacinė sistema*. Operacinė sistema „paslepia“ nuo programuotojo techninės įrangos detales ir, be to, suteikia jam patogų interfeisą kompiuterizuotai sistemai valdyti. Operacinė sistema iš esmės veikia kaip tam tikras tarpininkas lengvinantis ir programuotojui, ir aplikacijoms naudoti jos (operacinės sistemos) galimybes ir servisą.

Trumpai tariant, operacinė sistema paprastai teikia servisą tokiose srityse:

- *Programos kūrimas* {program creation}. Operacinė sistema programuotojui suteikia įvairias galimybes ir servisą kuriant programas. Visa tai apibūdinama *utilitos* terminu.
- *Programos vykdymas*. Kad programa būtų įvykdyta reikia atlikti daug užduočių. Programos instrukcijos ir duomenys turi būti įkelti į pagrindinę atmintį, turi būti inicijuoti tam tikri failai ir Į/I įrenginiai, be to, turi būti parengti ir kiti kompiuterio resursai. Visą tai vartotojo vardan tvarko operacinė sistema.
- *Susisiekimas su Į/I įrenginiais*. Kiekvienas Į/I įrenginys savo veikimui reikalauja specifiško instrukcijų arba valdymo signalų rinkinio. Visus rūpesčius, kad programuotojas galėtų operuoti tik paprastais terminais ir sąvokomis, pvz., rašyti arba skaityti {read, write} šiuo atveju operacinė sistema užsikrauna sau.
- *Darbo su failais valdymas*. Failų atveju būtina ne tik nustatyti Į/I įrenginių prigimtį (diskiniai ar juostiniai įrenginiai), tačiau taip pat būtina taikyti tam tikrus failų saugojimo formatus konkrečiuose įrenginiuose. Ir vėl čia operacinė sistema turi rūpintis dėl visų smulkmenų. Dar labiau, sistemos su keliais vienu metu dirbančiais vartotojais atveju, operacinė sistema turi užtikrinti apsaugos mechanizmą, kuris valdys bendrų {shared} resursų (pvz., failų) naudojimą.
- *Sistemos vartojimas*: Kolektyvinės {public} arba bendro naudojimo kompiuterizuotos sistemos atveju, operacinė sistema kontroliuoja prisijungimą prie pačios kompiuterizuotos sistemos ir tam tikrų specifinių resursų naudojimą.

Daugiau operacinės sistemos serviso čia nenagrinėsime. Sukoncentruosime savo dėmesį į operacinės sistemos resursų valdymo funkcijas ir jų (funkcijų) pritaikymą kompiuterių architektūroje.

Operacinė sistema – resursų menedžeris

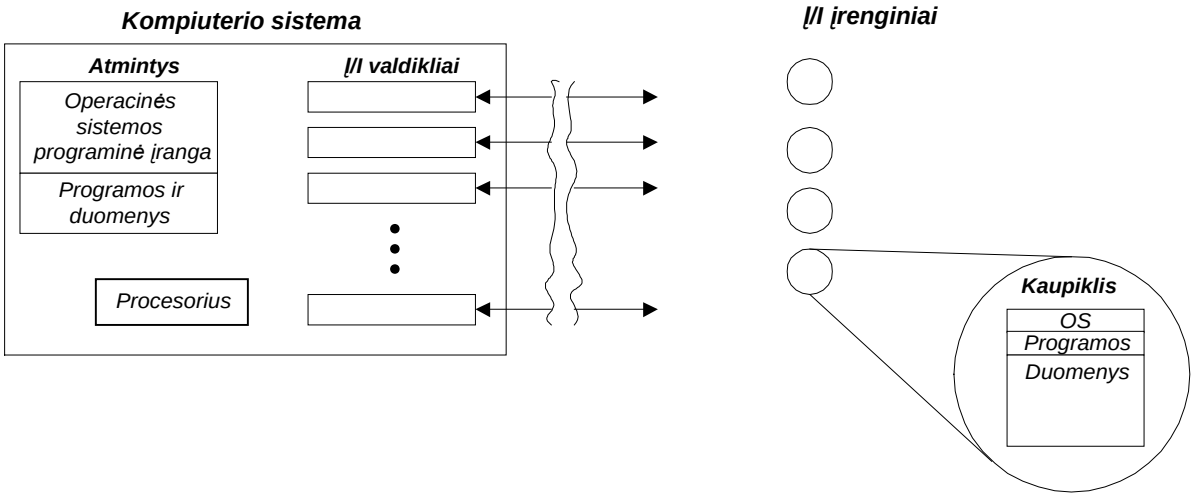
Kompiuteris iš esmės yra resursų duomenims siųsti, įsiminti ir apdoroti bei visoms šioms funkcijoms valdyti, rinkinys. Operacinė sistema užtikrina šių resursų valdymą.

Ar galime iš karto atsakyti į klausimą – kas tai per sistema (operacinė), kuri kontroliuoja keitimąsi duomenimis, duomenų įsiminimą ir apdorojimą? Iš vienos pusės peršasi labai paprastas atsakymas – taip. Bevaldant kompiuterio resursus, operacinė sistema taip pat kontroliuoja ir kompiuterio pagrindines funkcijas. Tačiau ši kontrolė vykdoma labai įdomiu būdu. Paprastai, kalbant apie tam tikrą kontrolės mechanizmą, mes jį įsivaizduojame kaip kažką išorinį atžvilgiu to, kas yra kontroliuojama. (Pavyzdžiui, šildymo sistema yra valdoma termostatu, kuris yra atskiras įrenginys ir nuo kaitinimo ir nuo šilumą skleidžiančio įrenginio). Tačiau operacinės sistemos atveju, taip nėra. Jos kontrolės mechanizmas yra neįprastas dviem aspektais:

- Operacinė sistema veikia tokiu pat būdu, kaip ir bet kuri kita kompiuterio programinė įranga – t. y. programa, kurią atlieka CPĮ.
- Operacinė sistema dažnai perleidinėja valdymą ir pati priklauso nuo CPĮ, kad vėl atgauti kontrolę.

Verta dar kartą pakartoti, kad operacinė sistema faktiškai yra ne kas daugiau, kaip kompiuterio programa. Skiriasi tik šios programos tikslai. Operacinė sistema procesoriui nurodinėja, kaip naudoti kitus kompiuterizuotos sistemos resursus ir kontroliuoja kitų (vartotojo) programų vykdymą. Tačiau tam, kad daryti abu šiuos dalykus, CPĮ turi nustoti vykdyti operacinės sistemos programą ir atlikinėti kitas – taikomąsias programas. Tokiu būdu, operacinė sistema perduoda procesoriui valdymą, kad jis darytų tam tikrą „naudingą“ darbą, o vėliau vėl sugrąžina sau kontrolę, siekiant procesorių paruošti kitai darbo daliai. Mechanizmas, užtikrinantis tokį funkcionavimą bus paaiškintas vėliau nagrinėjant operacines sistemas.

7.2 pav. pateikti pagrindiniai operacinės sistemos valdomi resursai. Operacinės sistemos tam tikra dalis saugoma kompiuterio pagrindinėje atmintyje. Ten yra operacinės sistemos branduolys {nucleus, core}, kurį sudaro operacinės sistemos dažniausiai naudojamos funkcijos ir kitos, duotu laiko momentu darbui būtinos,



7. 2 pav. Operacinė sistema – resursų menedžeris

operacinės sistemos dalys. Likusioje pagrindinės atminties dalyje saugomos kitos (vartotojo) programos ir duomenys. Šiek tiek vėliau pamatysime, kad šitų resursų išdėstymą pagrindinėje atmintyje kartu kontroliuoja ir operacinė sistema, ir procesoriuje esantis atminties schemotechninis menedžeris. Operacinė sistema taip pat sprendžia kuriuo laiko momentu vykdoma programa naudos tam tikrą Į/I įrenginį ir kontroliuoja jos (programos) darbą su failais.

7.1.2. Operacinių sistemų tipai

Operacinėms sistemoms sugrupuoti pagal tipus naudoja tam tikras charakteristikas. Šios charakteristikos dalinamos į dvi nepriklausomas kategorijas. Pirmoji kategorija apibūdina ar operacinė sistema yra paketinė {batch} ar interaktyvioji. *Interaktyviosios* sistemos atveju, vartotojas/programuotojas su kompiuteriu sąveikauja tiesiogiai, paprastai per klaviatūra/displėjus terminalą kontroliuojant tam tikrą darbą arba vykdant transakciją. Dar labiau, interaktyviojoje sistemoje priklausomai nuo aplikacijos pobūdžio, vartotojas gali susisiekti su kompiuteriu ir užduoties vykdymo metu. *Paketinė* operacinė sistema yra interaktyviosios sistemos priešinybė. Paketinės operacinės sistemos atveju, tam tikro vartotojo programos sudedamos į paketą su kitų vartotojų programomis ir atiduodami kompiuterio operatoriui. Po to, kai programa bus įvykdyta, jos darbo rezultatai išspausdinami ir grąžinami vartotojui. Dabar grynos paketinės sistemos yra retenybė. Tačiau, nagrinėjant modernias operacines sistemas, labai naudinga trumpai apsistoti ir ties paketinėmis sistemomis.

Minėtos nepriklausomos charakteristikų kategorijos rodo, ar užtikrina duotoji sistema *multiprograminį* darbo režimą {multiprogramming} ar ne. Multiprograminiam režimui esant, mėginama kiek įmanoma daugiau apkrauti procesorių, verčiant jį vienu metu dirbti daugiau negu su viena programa. Šiuo atveju į pagrindinę atmintį patalpinamos kelios programos ir procesorius tarp jų labai greitai persijunginėja. Šiam režimui alternatyva yra vienprogramė sistema, kai vienu metu gali būti vykdoma tik viena programa.

Remiantis šiomis dvejomis nepriklausomomis kategorijomis galima išvardyti keturis operacinių sistemų tipus (7.1 lentelė). Šiuos skirtingus tipus aprašyti geriausias būdas – padaryti trumpą operacinių sistemų istorijos apžvalgą.

7.**1 lentelė.** Operacinių sistemų kategorijos

	Paketinės {batch}	Interaktyviosios
Viena užduotis duotu laiko momentu	elementarios paketinės operacinės sistemos	Dedikuotos operacinės sistemos
Multiprograminis darbo režimas {multiprogramming}	Išvystytos paketinės operacinės sistemos	Laiką skirstančios operacinės sistemos

Ankstyvosios sistemos

Pirmuosiuose kompiuteriuose programuotojai betarpiškai sąveikaudavo su kompiuterio technine įranga. Šios mašinos buvo valdomos iš pulto/konsolės, kurioje buvo displėjus iš lempučių, tumblerių rinkinys – tam tikra įvesties įrenginio rūšis ir spausdintuvas. Mašininiu kodu parašytos programos buvo įkeliamos įvesties įrenginio (pvz., perfokortų skaitytuvo). Jeigu dėl klaidos programos vykdymas buvo nutraukiamas, klaidos sąlygos buvo indikuojamos tam tikromis lemputėmis.

Laikui bėgant buvo kuriama nauja techninė ir programinė įranga. Techninėje įrangoje jau buvo juostiniai kaupikliai ir labai spartūs spausdintuvai. Programinėje įrangoje atsirado kompiliatoriai, asembleriai ir bendrojo naudojimo funkcijų bibliotekos. Bendrojo naudojimo funkcijas nuo šiol galima buvo įterpti į taikomąją programą nerašant jų teksto.

Tose ankstyvosiose sistemose reikėjo pastoviai spręsti dvi pagrindines problemas:

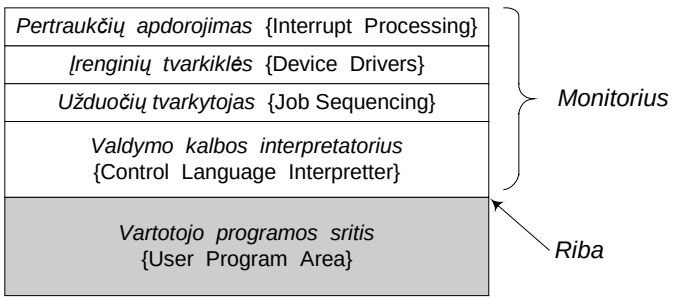
- *Darbo planavimas* {scheduling}. Vykdomų programų daugumoje buvo naudojamas iš anksto sudarytas grafikas laiko rezervui užtikrinti. Paprastai iš pusvalandžio trukmės laiko atkarpų (arba panašiai) vartotojas sudarydavo savo aplikacijos vykdymo laiko bloką. Jeigu vartotojas numatydavo darbui pvz., valandą, o programa buvo įvykdoma per 45 minutes – likusįjį laiką kompiuteris stovėdavo be darbo. Ir atvirkščiai, vartotojo programa galėjo būti sustabdyta per anksti, kai dar nebuvo gautas sprendinys.
- *Parengiamasis laikas*. Paprastosiose taikomose programose, vadinamose *uždaviniais* {job} galėjo būti numatyta kompiliatoriaus ir aukšto lygio programavimo kalbos įkėlimas į atmintį, sukompiliuotos programos (objektinės programos) įsiminimas ir bendras – objektinės programos ir bendrojo naudojimo funkcijų apjungimas {linking}. Kiekviename iš išvardytų žingsnių galėjo prireikti fiziškai montuoti arba išmontuoti juostinį kaupiklį arba įstatyti perfokortų bloką. Tokiu būdu, programą rengiant buvo eikvojama daug laiko.

Paprastos paketinės operacinės sistemos

Pirmosios skaičiavimo mašinos buvo labai brangios, todėl buvo labai aktualu maksimaliai išnaudoti jų resursus ir galimybes. Prastovų laikas dėl neracionalaus planavimo ir pasirengimo buvo nepageidautinas.

Siekiant pagerinti kompiuterių eksploataciją buvo sukurtos vadinamosios paprastos paketinės operacinės sistemos. Tokiose sistemose, vadinamose *monitoriais* {monitor} vartotojui daugiau nereikėjo tiesiogiai sąveikauti su kompiuteriu. Vartotojas magnetinėje juostoje arba perfokortose parengtą užduotį perduodavo kompiuterio operatoriui. Pastarasis kelis uždavinius sudėdavo į paketą, įkeldavo į įvesties įrenginį ir tolimesnį apdorojimą valdė monitorius.

Norint suprasti, kaip tokia schema veikia, reikia į ją pažiūrėti iš dviejų pozicijų: iš paties monitoriaus ir CPĮ. Iš monitoriaus pažiūros atrodo, kad būtent monitorius kontroliuoja visą įvykių seką. Kad taip būtų, monitorius pastoviai saugomas pagrindinėje atmintyje (todėl jį dažnai vadina *rezidentiniu monitoriumi*) ir gali būti vykdomas (7.3 pav.). Atliekant tam tikrą uždavinį monitorius atmintyje skaitomas tik vieną kartą. Kai jis skaitomas (pagal jo instrukcijas), esamas uždavinys patalpinamas į pagrindinės atminties specialią vartotojo programų sritį (7.3 pav.) ir visas kompiuterio valdymas perduodamas šiam uždaviniui. Kai uždavinys bus atliktas (išspręstas) iki galo, generuojama pertrauktis (vidinė CPĮ atžvilgiu), kuri kontrolę (iš uždavinio) vėl sugrąžina monitoriui. Monitorius gi, savo ruožtu, iš karto skaito kitą uždavinį. Išspręsto uždavinio rezultatai spausdinami ir atiduodami vartotojui.

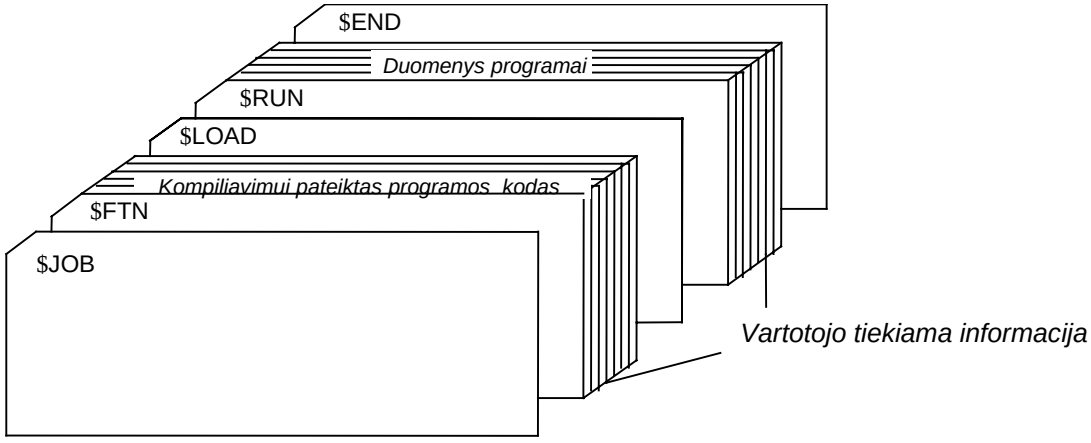


7. 3 pav. Pagrindinės (darbinės) atminties paskirstymas rezidentiniam monitoriui esant

Dabar apžvelgsime kaip tai viskas atrodo iš CPĮ pažiūros. Tam tikru laiko momentu CPĮ skaito instrukcijas iš tos pagrindinės atminties srities, kur saugomas monitorius. Šios instrukcijos procesoriui nurodo įkelti kitą uždavinį į atitinkamą pagrindinės atminties sritį. Kai uždavinys yra įkeltas, CPĮ monitoriuje aptinka šakojimosi instrukciją, kuri procesoriui nurodo tęsti vykdymą kitoje atminties srityje (t. y. pradėti vykdyti vartotojo uždavinį). Tuomet CPĮ pradeda vykdyti instrukcijas vartotojo programoje kol nenueis iki programos pabaigos arba neatsiras klaida. Bet kuriuo iš šių dviejų atveju CPĮ kitą instrukciją jau išrenka iš monitoriaus programos. Tokiu būdu, frazė „kontrolė perduota uždaviniui“ tiesiog reiškia, kad CPĮ instrukcijas dabar rinks ir vykdys iš vartotojo programos, o frazė „kontrolė sugrąžinama monitoriui“ – reiškia, kad CPĮ instrukcijas dabar rinks ir vykdys monitoriaus programoje.

Čia turi būti aišku, kad monitoriaus taikymas išsprendžia planavimo {scheduling} problemą. Uždaviniai pakete sustatomi į eilę taip, kad jie yra vykdomi, maksimaliu įmanomu greičiu be prastovų.

Dabar pažiūrėkime į uždavinio parengimo problemą. Monitorius geba ir ją puikiai sutvarkyti. Šiuo atveju kiekviename uždavinyje kartu su instrukcijomis yra vadinamoji *uždavinio valdymo kalba* {job control language}. Iš esmės tai yra speciali programavimo kalba užtikrinanti uždavinių instrukcijų sąveiką su monitoriumi. Paveiksle 7.4 pateiktas elementarus uždavinio įvesties perfokortomis pavyzdys. Šiame pavyzdyje vartotojas pateikia FORTRAN kalba parašytą programą ir tam tikrus šiai programai reikalingus duomenis. Be FORTRAN-o ir duomenų į uždavinio paketą įeina valdymo instrukcijos, kurios 7.4 pav. prasideda $ ženklu.



7.4 pav. Perfokortų blokas elementarioje paketinėje sistemoje

Iš aptarymo matyti, kad monitorius arba paketinė operacinė sistema yra paprasta kompiuterinė programa. Jos veikimas pagrįstas CPĮ galimybe instrukcijų rinktis iš įvairių pagrindinės atminties sričių, siekiant kad būtų galima kaitalioti valdymo perdavimą ir sugrąžinimą. Šiuo atveju yra tam tikri reikalavimai techninei įrangai:

- *Atminties apsauga* {memory protection}. Kol vykdoma vartotojo programa, atminties srities, kurioje saugomas monitorius, turinys neturi keistis. Jeigu toks mėginimas yra, CPĮ detektuoja klaidą ir kontrolę perduoda monitoriui. Monitorius esamą uždavinį sustabdo, siunčia pranešimą apie klaidą ir įkelia kitą iš eilės uždavinį.
- *Taktavimas* {timing}. Taimeris naudojamas tam, kad kompiuterizuotos sistemos negalėtų monopolizuoti kažkoks vienas uždavinys. Taimeris nustatomas kiekvieno uždavinio pradžioje (t. y. uždaviniui išskiriamas mašininio laiko limitas). Jeigu taimeryje nustatytas laiko intervalas pasibaigia, generuojama pertrauktis ir kontrolė sugrąžinama monitoriui.
- *Privilegijotosios instrukcijos*. Tam tikros instrukcijos pažymimos kaip privilegijuotosios ir jas vykdyti gali tik tai monitorius. Tokios yra, pavyzdžiui, Į/I instrukcijos, todėl visus Į/I įrenginius gali valdyti tik monitorius. Tai leidžia, pavyzdžiui, apsaugoti vartotojo programą nuo netyčinio valdymo instrukcijų iš kito uždavinio instrukcijų. Jeigu vartotojo programai reikia atlikti tam tikrą Į/I operaciją, ji turi „prašyti“ monitorių, kad pastarasis tai padarytų už ją. Jeigu CPĮ aptinka privelegiruotąją instrukciją vartotojo programos vykdymo metu, procesoriaus tam tikri techniniai įrenginiai tai supranta kaip klaidą ir valdymas sugrąžinamas monitoriui.

Tokiu būdu, mašininis laikas yra dalinamas tarp vartotojų programų ir monitoriaus veiksmų. Tam tenka aukoti du mašininius resursus: tam tikra atminties dalis skiriama monitoriui ir tam tikra mašininio laiko dalį taip pat eikvoja monitorius. Abi šios aukos yra kaip papildomos išlaidos. Tačiau, net šioms papildomoms išlaidoms esant elementariosios paketinės sistemos gerokai pagerina kompiuterio eksploataciją.

Patobulintos paketinės sistemos

Net automatiniam nuosekliam uždavinių vykdymui esant, kurį užtikrina elementarios paketinės sistemos, procesorius dažnai stovi be darbo. Problema yra tame, kad Į/I įrenginiai yra labai lėti lyginant su procesoriumi. Paveiksle 7.5 pateikti demonstraciniai apskaičiavimai. Duotajame pavyzdyje apskaičiuojama programa, kuri apdoroja įrašų bloką ir joje, vienam įrašui nuskaityti ir įrašyti vidutiniškai eikvojama 100 mašininių instrukcijų. Šiame pavyzdyje kompiuteris (tiksliau CPĮ) priverstas 96% savo laiko laukti, kol Į/I įrenginys užbaigs duomenų siuntimą! Paveiksle 7.6, a parodyta šios situacijos laiko diagrama. Be to, procesorius papildomai eikvoja šiek tiek vykdomojo laiko kol atitinkama Į/I instrukcija bus surasta ir parengta vykdymui.

Į/I įrenginys skaito vieną įrašą	1,5 *ms*
CPĮ vykdo 100 instrukcijų	0,1 *ms*
Į/I įrenginys rašo vieną įrašą	1,5 *ms*
Viso:	3,1 *ms*

$$\textit{CPĮ išnaudojimo procentas} = \frac{0{,}1}{3{,}1} \cdot 100 \approx 3{,}2\ \%$$

7.5 pav. Kompiuterio išnaudojimo pavyzdys

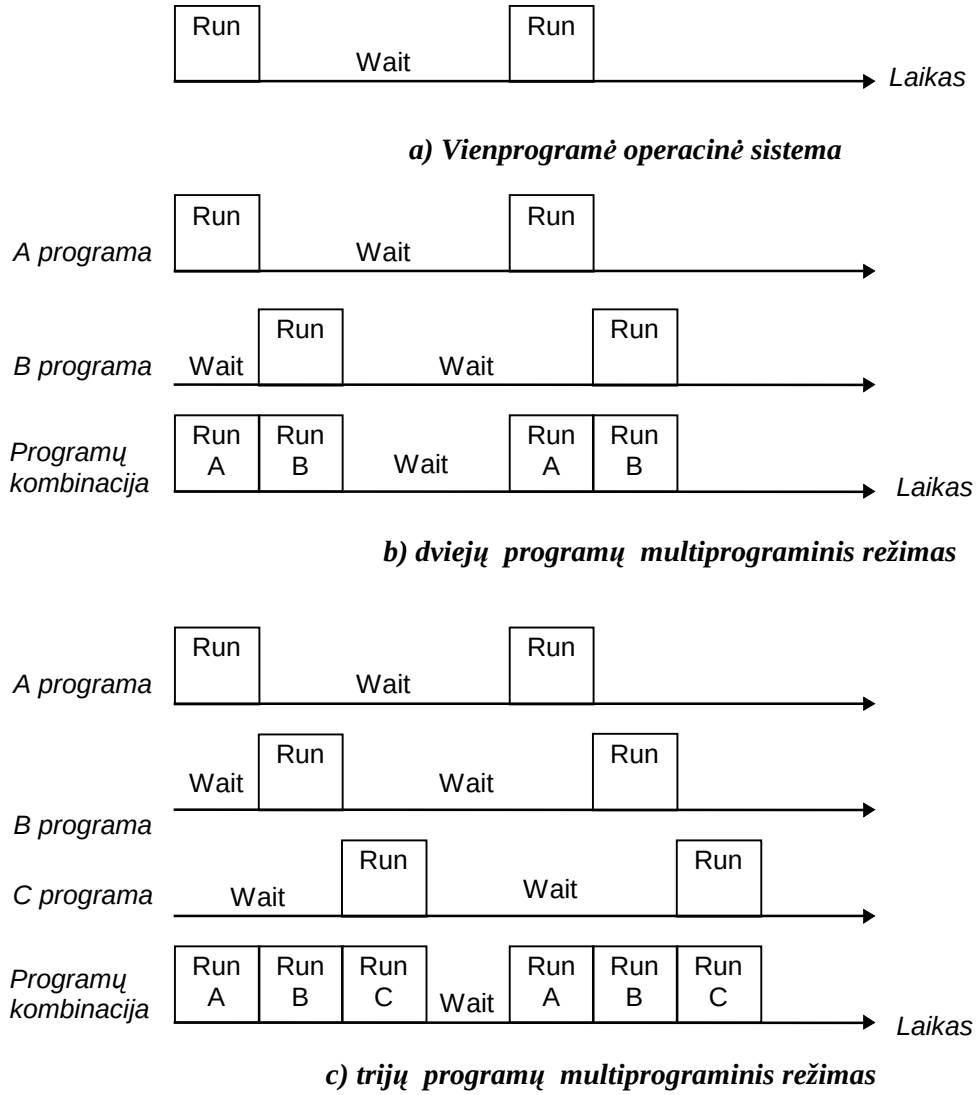
Tačiau toks neefektyvumas nėra neišvengiamas. Mes žinome, kad kompiuteryje turi būti pakankamai operatyviosios atminties operacinei sistemai (rezidentiniam monitoriui) ir nors vienai vartotojo programai patalpinti. Įsivaizduokime, kad atminties yra tiek daug, kad joje galima saugoti ir operacinę sistemą, ir ne vieną, o dvi vartotojo programos. Dabar, kai vienas uždavinys priverstas laukti kol bus įvykdyta Į/I operacija CPĮ gali persijungti į kitą uždavinį, kuriam šiuo momentu Į/I operacijos dar nereikia (7.6 pav. b). Dar labiau, atmintį galima taip išplėsti, kad joje vienu metu galima būtų saugoti tris, keturias ar daugiau programų ir persijunginėti tarp jų (7.6 pav. c). Šis procesas vadinamas *multiprogramininiu darbo režimu* {multiprogramming}. Tai yra modernių operacinių sistemų pagrindinis skiriamasis bruožas.

Multiprogramininio darbo režimo našumui iliustruoti išnagrinėkime vieną tokio režimo pavyzdį [10]. Tarkime, turime hipotetinę kompiuterio sistemą, kurioje yra 256 K žodžių atminties (nenaudojamos operacinės sistemos), diskinis įrenginys, terminalas ir spausdintuvas. Trys programos – JOB1, JOB2 ir JOB3 su 7.2 lentelėje išdėstytais atributais – pateiktos vykdymui vienu metu.

7.2 lentelė. Vykdomųjų programų atributų pavyzdžiai

	JOB1	JOB2	JOB3
Uždavinio pobūdis	intensyvūs skaičiavimai	intensyvios Į/I operacijos	intensyvios Į/I operacijos
Trukmė	5 min	15 min	10 min
Reikalaujama atmintis	50 K	100 K	80 K
Ar reikalingas diskinis įrenginys.?	Ne	Ne	Taip
Ar reikalingas terminalas?	Ne	Taip	Ne
Ar reikalingas spausdintuvas?	Ne	Ne	Taip

Tarkime, kad JOB2 ir JOB3 programoms procesorius reikalingas labai mažai, tačiau JOB3 programa labai intensyviai naudos diskinį įrenginį ir spausdintuvą. Vienprogramės {uniprogramming} operacinės sistemos atveju, visos šios programos bus vykdomos nuosekliai iš eilės. JOB1 uždavinys bus atliktas per 5 minutes. JOB2 – laukia šias 5 minutes ir toliau vykdoma per 15 minučių. JOB3 programa laukia 20 (5 + 15) minučių ir užsibaigia ties 30-ąja minute nuo to momento, kai ji buvo parengta vykdymui. Vidutinis

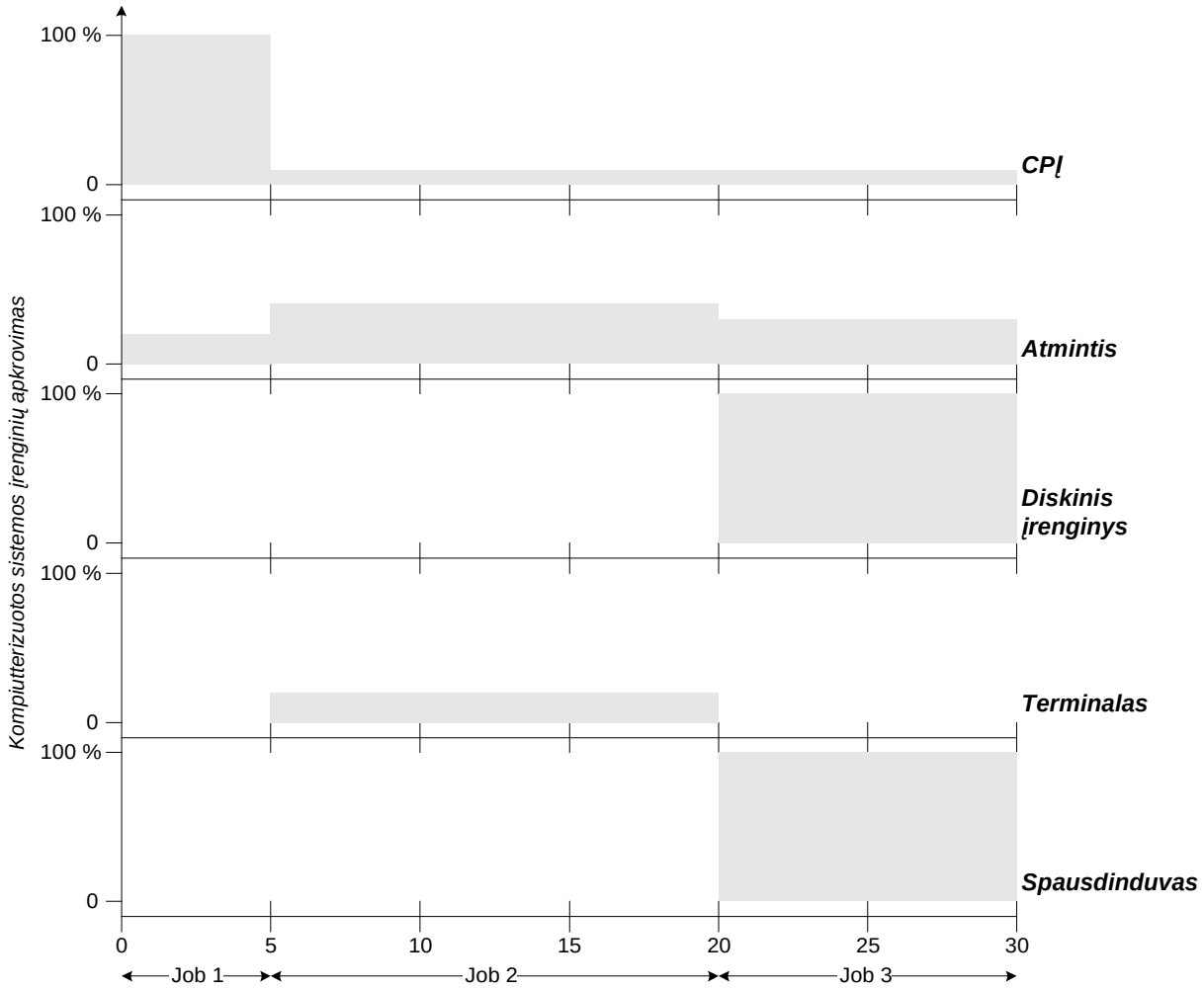


7.6 pav. Multiprograminio darbo režimo pavyzdžiai. Čia Run – procesoriaus veikimo laikas; Wait – procesoriaus prastovos laikas

resursų panaudojimas, kompiuterizuotos sistemos našumas ir reakcijos laikas, vienprogramės operacinės sistemos atveju, pateikti 7.3 lentelės atitinkamame stulpelyje. Nuoseklioji įrenginių {device-by-device} eksploatacija atvaizduota 7.7 pav. Akivaizdu, kad yra didelis neišnaudotas įrenginių eksploatacijos rezervas per visą 30 min intervalą.

Dabar įsivaizduokime, kad visi uždaviniai paleidžiami į darbą kartu taikant multiprograminę operacinę sistemą. Kadangi didelės resursų konkurencijos tarp uždavinių nėra, visi trys uždaviniai gali būti paleisti į darbą su minimaliu laiko skirtumu ir tam tikrą laiką būti kompiuteryje (tarkime savo Į/I operacijoms atlikti JOB2 ir JOB3 uždaviniai naudoja pakankamai procesorinio laiko). JOB1 uždaviniui atlikti pakanka 5 minučių, tačiau per šį laiką JOB3 bus tik įpusėjęs, o JOB2 uždavinio padarytas tik trečdalis. Visi trys uždaviniai bus padaryti 15 minučių bėgyje. Našumo didėjimas bus akivaizdus, kai išnagrinėsime atitinkamą (paskutinįjį) 7.3 lentelės stulpelį, kuris yra gautas iš 7.8 pav. pateiktos histogramos.

Kaip ir elementarios paketinės sistemos, multiprograminė paketinė sistema iš esmės taip pat yra programa, kurios veikimą turi užtikrinti tam tikri kompiuterio techninės įrangos ypatumai. Svarbiausios papildomos galimybės, kurios yra labai naudingos multiprograminiam darbo režimui, yra ta techninė įranga, kurioje numatyti Į/I pertraukties ir DMA mechanizmai. Pertraukčių valdomų Į/I arba DMA mechanizmo atveju CPĮ tam tikru laiko momentu bus nutrauktas, o valdymas perduotas operacinės sistemos pertrauktis apdorojančiai programai. Operacinė sistema tuomet galės kontrolę perduoti kitam uždaviniui.



7.7 pav. Resursų eksploatacijos histograma vienprogramės operacinės sistemos atveju

7.3 lentelė. Multiprogramavimo taikymo efektas kompiuterio resursų naudojimui

	Vienprogramė operacinė sistema	Multiprograminė operacinė sistema
Procesoriaus naudojimas	17 % (5/30 ≅ 17 %)	33 % (5/15)
Atminties naudojimas	30 % (230/256/3 ≅ 30 %)	67 % ((50/3+100+80·2/3)/256 ≅ 67 %)
Diskinio įrenginio naudojimas	33 % (10/30 ≅ 33 %)	67 % (10/15 ≅ 67 %)
Spausdintuvo naudojimas	33 % (10/30 ≅ 33 %)	67 % (10/15 ≅ 67 %)
Bendras laikas	30 min	15 min
Našumas	6 uždaviniai/val.	12 uždaviniai/val.
Vidutinis atsako laikas	18,3 min. ((5 + 20 + 30)/3)	10 min ((5 + 10 + 15)/3)

Multiprograminės operacinės sistemos lyginant su vienprogramėmis yra labai sudėtingos. Tam, kad visą laiką turėti kelis darbui parengtus uždavinius, jie turi būti saugomi atmintyje, kuriai savo ruožtu, reikia tam tikro *atminties tvarkymo* {memory management} mechanizmo. Be to, kai keli uždaviniai yra parengti vykdymui, procesorius turi pasirinkti, kurį iš jų vykdyti – tai reikalauja tam tikro algoritmo *darbo laiko planavimui* {scheduling}. Abi šias koncepcijas aptarsime vėliau.

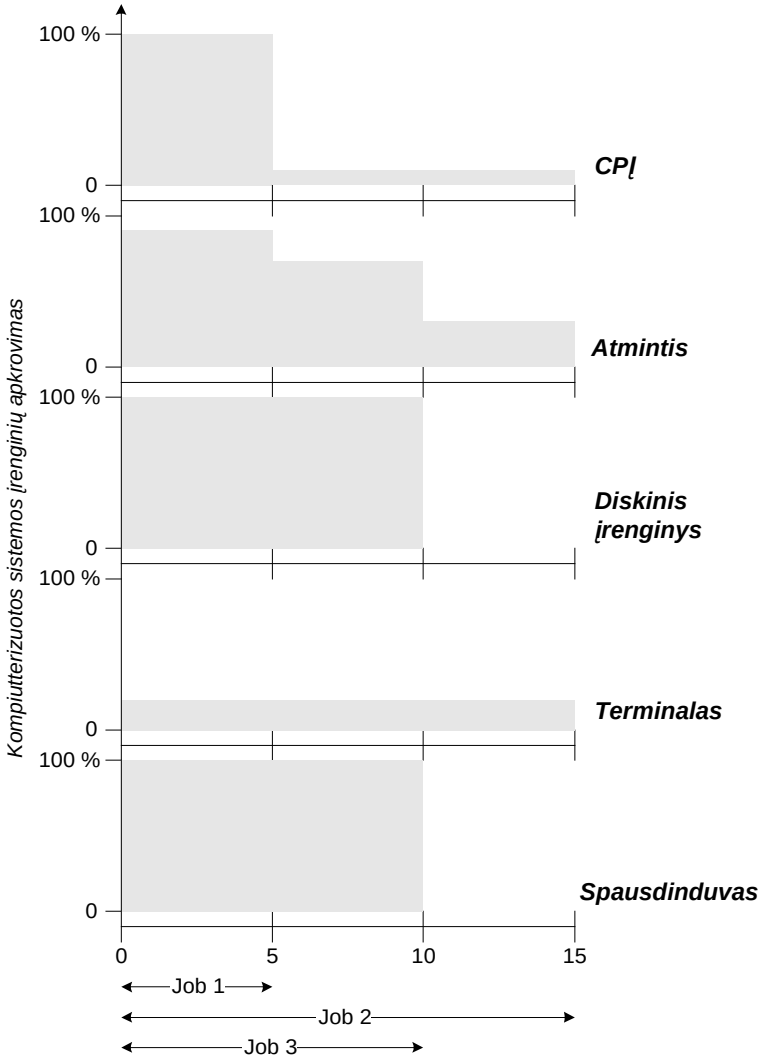
Laiko skirstymas – Time Sharing

Taikant multiprograminį darbo režimą paketinis procesas gali būti labai efektyvus. Tačiau, kai yra daug uždavinių labai pageidautina užtikrinti dar ir tokį režimą, kad vartotojas galėtų betarpiškai sąveikauti su

kompiuteriu. Iš tikrųjų, kai kuriuose uždaviniuose, tokiuose kaip, pavyzdžiui, transakcijos {transaction} procese, interaktyvus režimas tiesiog būtinas.

Nūnai gali būti iškelti (ir, iš tikro, gana dažnai keliami) reikalavimai interaktyviems skaičiavimams. Šitie reikalavimai tenkinami vadinamuosiuose *dedikuotuosiuose mikrokompiuteriuose* {dedicated microcomputers}. Interaktyviųjų skaičiavimų galimybės 1960-taisiais metais, kai dauguma kompiuterių buvo dideli ir brangūs, dar nebuvo. Vietoj šito, buvo sukurta *darbo laiko skirstymo* {time sharing} technologija.

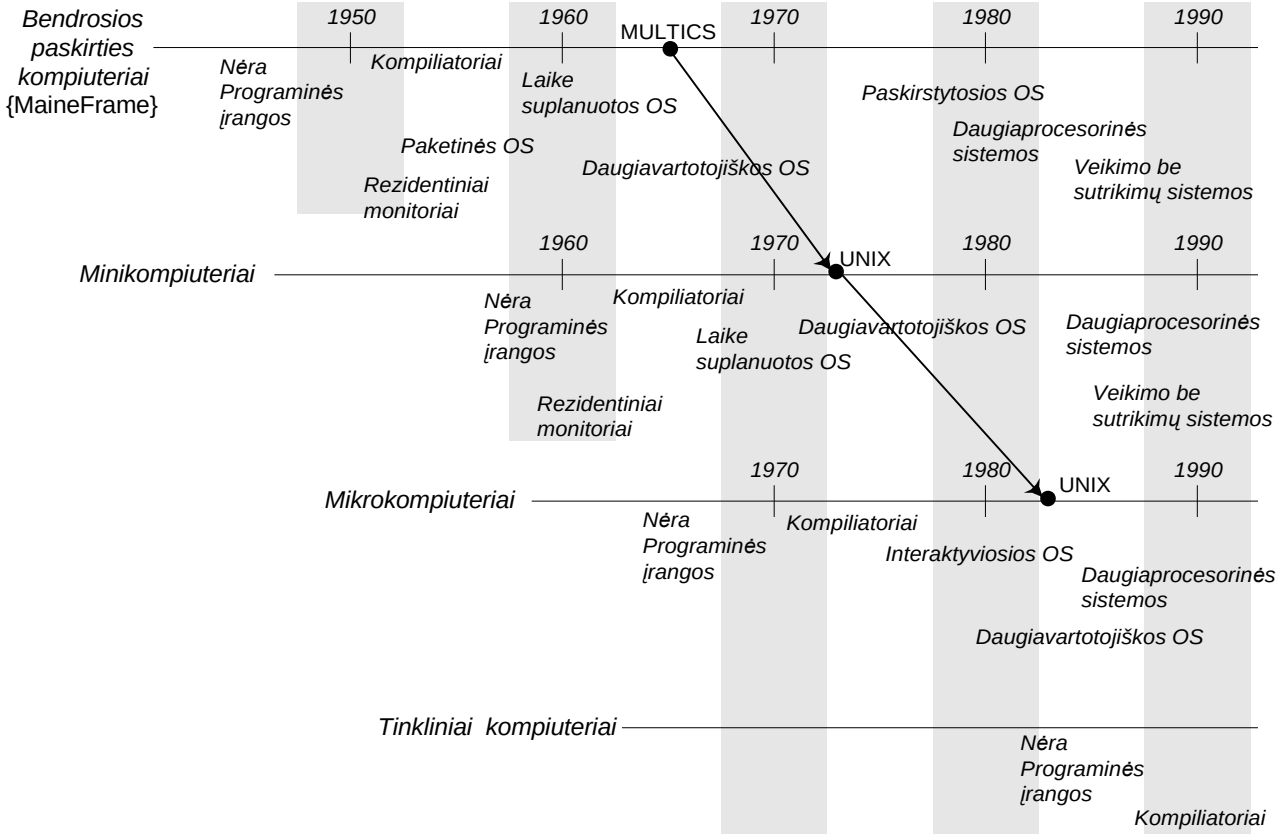
Kaip ir multiprograminis darbo režimas, kuris procesoriui leidžia vienu metu valdyti daug paketinių uždavinių, ši technologija taip pat gali būti taikoma aibei interaktyvių uždavinių valdyti. Šiuo atveju, terminas *darbo laiko skirstymas* atspindi tą faktą, kad procesoriaus laikas yra skirstomas tarp aibės vartotojų. Abi technologijos ir multiprograminis darbo režimas, ir laiko skirstymas iš esmės naudoja multiprograminį darbo režimą. Pagrindiniai šių technologijų skirtumai parodyti 7.4 lentelėje.



7.8 pav. Resursų eksploatacijos histograma multiprograminės operacinės sistemos atveju

7. 4 lentelė. Paketinio programavimo ir laiko skirstymo technologijų skirtumai

	Paketinis multiprograminis darbo režimas {Batch Multiprogramming}	Laiko skirstymas {Time Sharing}
Principinis tikslas	maksimizuoti procesoriaus naudojimą	minimizuoti atsako laiko
Operacinės sistemos instrukcijų šaltinis	uždavinio valdymo programavimo kalbos instrukcijos (tiekiamos kartu su uždaviniu)	iš terminalo įvedamos komandos



7. 9 pav. Operacinių sistemų koncepcijos ir jų galimybių migracija

Kompiuterių klasės

Operacinės sistemos pagrindinai buvo kuriamos ir tiriamos dideliems {mainframe} kompiuteriams. Atsiradus minikompiuteriams, o po to ir mikrokompiuteriams, jų techninei įrangai besivystant, visos didelių kompiuterių technologijos buvo perorientuotos į šiuos, skirtingų klasių kompiuterius.

7.9 pav. šis fenomenas yra parodytas. Operacinių sistemų koncepcijų ir galimybių migracijos geru pavyzdžiu gali būti UNIX sistemos evoliucija iš Multics sistemos. Multics – tai operacinė sistema sukurta kartu M.I.T., Bell Laboratories ir Division of General Electric kompanijomis ir vėliau tapusi Honeywell sistemos dalimi. Multics operacinė sistema iki šiol pardavinėjama Honeywell kompanija ir iki šiol labai teigiamai vertinama išvystytose sistemose, ypač saugumo {security} srityje. Labai daug idėjų, kurios iš pradžių buvo skirtos Multics sistemai, vėliau Bell Labs firmos buvo pritaikytos kuriant UNIX operacinę sistemą. UNIX operacinė sistema tapo viena populiariausių sistemų minikompiuterių klasėje ir dabar plačiai juose naudojama.

7.2. Mašininio laiko planavimas – Scheduling

Nūdienių operacinių sistemų pagrindinis uždavinys – užtikrinti multiprograminį darbo režimą (neskaitant tų mikrokompiuterių su kuriais dirbo tik vienas vartotojas). Multiprograminio režimo atveju, operatyviojoje atmintyje vienu metu saugoma daug uždavinių {jobs} arba vartotojų programų. Kiekvienas uždavinys duotu laiko momentu yra arba apdorojamas procesoriaus, arba laukia kol bus pastebėta jo išstatyta pertrauktis. Kai procesorius vykdo vieną uždavinį, kitam jis yra neprieinamas ir pastarasis privalo laukti.

Multiprograminio darbo režimo esmę sudaro *mašininio laiko planavimas* {scheduling}. Faktiškai yra žinomi trys mašininio laiko planavimo būdai (7.5 lentelė). Visus juos išnagrinėsime, tačiau iš pradžių aptarsime *proceso* koncepciją. Pirmą kartą šis terminas buvo panaudotas 1960-aisiais metais, kuriant Multics operacinę sistemą. Tai yra šiek tiek bendresnis negu *uždavinys* terminas. *Proceso* terminui buvo sukurta nemažai apibūdinimų, pvz.,

- programa, kuri yra vykdymo stadijoje;
- programinis objektas, kuriam yra skiriamas CPĮ;
- programos „atgaivintoji dvasia“ {the “animated spirit” of program}.

Proceso koncepcija taps aiškesnė, kai išsamiau nagrinėsime mašininio laiko planavimą.

7. 5 lentelė. Mašininio laiko planavimas multiprograminio darbo režimo atveju

Aukšto lygio planavimas {High-Level Scheduling}	Sprendžiama, kaip optimaliai programas sudėlioti į eilę
Trumpo (artimesniojo) laiko planavimas	Sprendžiama, kuris iš esamų procesų gali būti vykdomas procesoriaus
Į/I planavimas	Sprendžiama, kuris iš esamo proceso neužbaigtų Į/I užklausų gali būti apdorojamas tam tikro (šiuo metu pasirengusio) Į/I įrenginio

7.2.1. Aukšto lygio mašininio laiko planavimas

Kompiuteryje esantis aukšto lygio mašininio laiko planuotojas sprendžia, kuri iš programų yra talpinama į kompiuterizuotą sistemą vykdymui. Tokiu būdu, aukšto lygio planuotojas kontroliuoja *multiprograminio darbo režimo laipsnį* {degree of multiprogramming} – t. y. procesų skaičių atmintyje. Tas, kas pagal planuotojo sprendimą bus patalpinta į kompiuterizuotą sistemą – uždavinys arba programa, tampa procesu ir statomas į trumpo (artimesnio) laiko planuotojo eilę. Akivaizdu, kad kuo daugiau procesų bus pradėta vykdyti, tuo mažesnė vykdymo laiko dalis atitiks kiekvienam procesui. Ilgesniojo laiko planuotojas turi taip apriboti multiprograminio režimo laipsnį, siekiant užtikrinti tinkamą esamų procesų apdorojimą.

Paketinėse sistemose planuotojas naujai atsiradusį uždavinį talpina į diskinį kaupiklį ir laiko jį eilėje arba vadinamojoje laukimo linijoje {waiting line}. Aukšto lygio planuotojas iš šios eilės uždavinį pasiima tik tada, kai jis gali tai padaryti. Šiuo atveju jam reikia priimti du sprendimus. Pirmą, planuotojas turi nuspręsti kiek procesų – vieną ar daugiau, jis gali priimti. Tokį sprendimą reikia daryti visada, kai baigiama vykdyti esamą procesą. Antrą, planuotojas turi nuspręsti, kuris uždavinys (ar uždaviniai) gali būti įtraukti į procesus. Šiuose kriterijuose gali būti numatyti prioritetai, spėjamojo vykdymo laiko ir Į/I reikalavimai.

Sistemoje su skirstamuoju mašininiu laiku {time sharing}, užklausos procesui generuojamos kai prie kompiuterizuotos sistemos mėgina prisijungti vartotojas. Laike paskirstytieji (suplanuotieji) vartotojai į eilę paprastai nestatomi ir nelaukia kol sistema juos galės aptarnauti.Šiuo atveju būtent operacinė sistema priima visus autorizuotus vartotojus kol kompiuterizuota sistema neprisisotins. Tuomet vartotojo mėginimas įeiti į kompiuterizuotą sistemą susilauks pranešimo apie tai, kad sistema yra užpildyta {system is full}, o pats vartotojas sistemoje yra laukiamas vėliau.

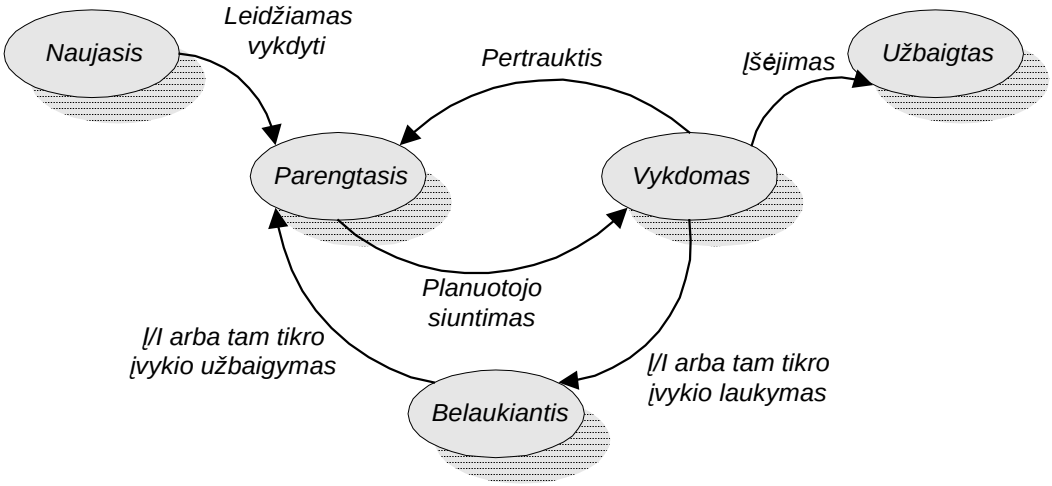
7.2.2. Trumpalaikis planavimas

Operacinės sistemos aukšto lygio planuotojas į darbą įtraukiamas palyginus retai ir jis daro bendresnius sprendimus: ar pradėti naują procesą ir kurį būtent. Trumpalaikis planuotojas, vadinamas dar *dispečeriu*, veikia dažnai ir priima konkretesnius sprendimus – kuris būtent uždavinys bus vykdomas toliau.

Proceso etapai

Norint suprasti kaip veikia trumpalaikis planuotojas, turime išnagrinėti proceso etapo (stadijos) koncepciją. Per visą proceso vykdymo laiką jo statusas (būvis) keičiasi daug kartų. Kiekvienu duotu laiko momentu jo statusas apibūdinamas kaip tam tikras *etapas (stadija). Stadijos* terminas čia taikomas todėl, kad

jis reiškia, kad esti tam tikros esamo momento statusą apibūdinančios informacijos. Procesą paprastai apibūdina penkios stadijos (7.10 pav.):

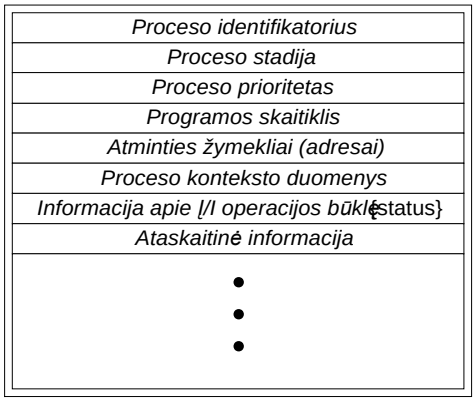


7.10 pav. Proceso stadijos

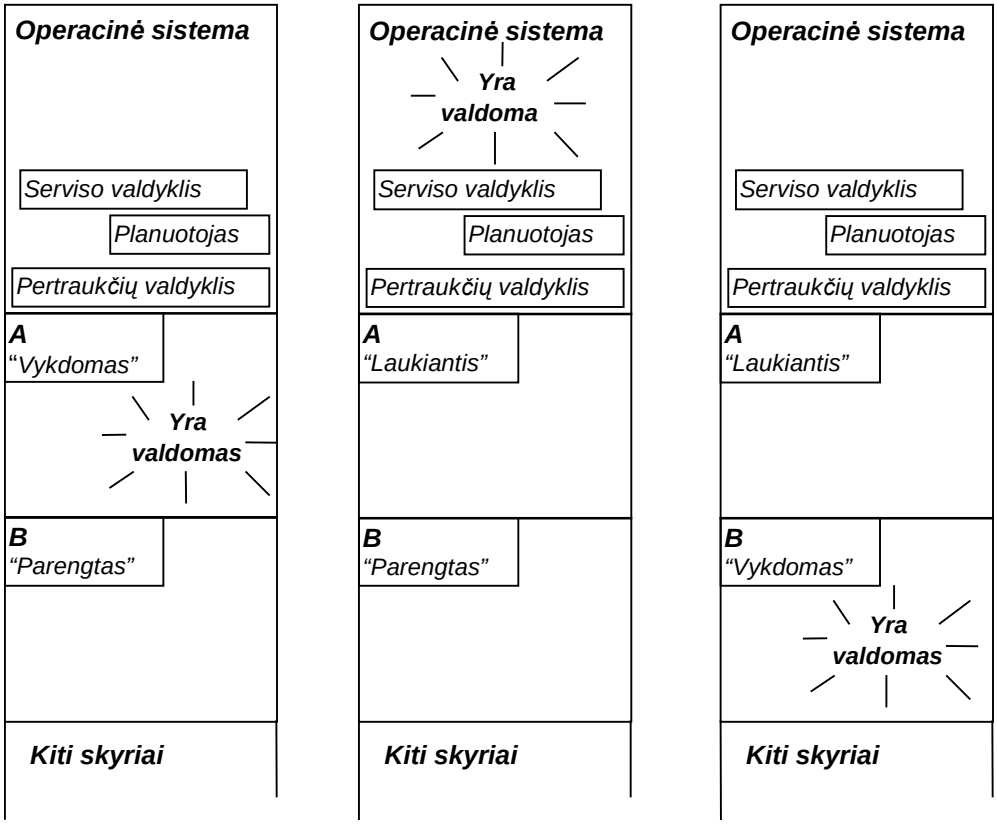
- *Naujasis* {New} – tam tikra programa yra pasirenkama aukšto lygio planuotojo, tačiau dar neparengta vykdymui. Operacinė sistema, procesą perkeliant į parengimo stadiją jį inicijuoja.
- *Parengtasis* {Ready} – procesas yra parengtas vykdymui ir tik laukia procesoriaus kvietimo.
- *Vykdomasis* {Running} – procesorius vykdo esamą procesą.
- *Laukiantysis* {Waiting} – procesas laikinai sustabdomas ir laukiama tam tikrų sistemos resursų, pvz., Į/I suteikimo.
- *Nutrauktas* {Halted} – operacinė sistema procesą nutraukia ir sunaikina.

Apie kiekvieną vykdomą procesą operacinė sistema turi turėti jo stadijos ir kitą procesui vykdyti būtiną informaciją. Todėl, operacinės sistemos požiūriu, kiekvienas procesas atrodo kaip *proceso valdymo blokas* (7.11 pav.), kuriame paprastai būna:

- *Identifikatorius*. Kiekvienas esamas procesas turi unikalų identifikatorių (numerį).
- *Stadija*. Esamo proceso stadija (naujas, parengtas ir pan.).
- *Prioritetas*. Santykinis proceso prioriteto numeris.
- *Programos skaitiklis*. Esamoje programoje kitos (n + 1) instrukcijos adresas.
- *Atminties žymekliai* {Memory pointers}. Proceso išdėstymo atmintyje pradžia ir pabaiga (adresai).
- *Konteksto duomenys* {Context Data}. Faktiškai tai yra procesoriaus registrų turinys, proceso vykdymo metu. Konteksto duomenys ir programos skaitiklio reikšmė yra išsaugomi, kai procesas išeina iš vykdymo {running} stadijos. Procesorius šiuos duomenis analizuoja kai vėl atnaujinamas proceso vykdymas.
- *Į/I operacijos statuso informacija*. Joje yra neįvykdytų Į/I užklausų sąrašas, išvardyti procesą atitinkantys Į/I įrenginiai ir failai ir t. t.
- *Ataskaitinė informacija*. Joje gali būti duomenys apie procesoriaus panaudojimą, laiko limitus irpan.



7. 11 pav. Proceso valdymo blokas {Process Control Block}



7.12 pav. Mašininio laiko planavimo pavyzdys

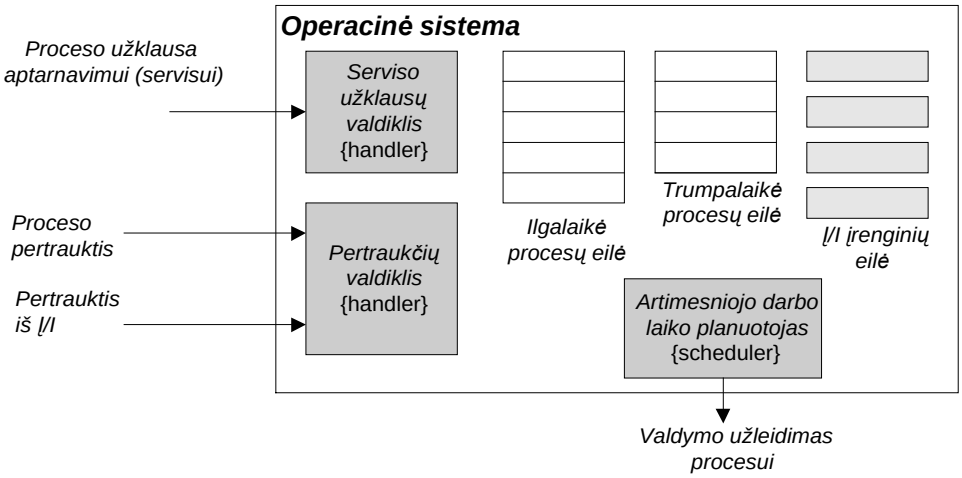
Tuo metu kai procesorius pradeda vykdyti naują uždavinį arba aptarnauti vartotojo užklausą, sukuriamas tuščias proceso valdymo blokas ir atitinkamas procesas pervedamas į naujo {new} proceso stadiją. Po to, kai operacinė sistema tinkamai užpildys proceso valdymo bloką, procesas perkeliamas į parengtojo proceso stadiją.

Mašininio laiko planavimo metodai

Norint suprasti kaip operacinė sistema valdo įvairių atmintyje esančių uždavinių planavimą išnagrinėkime 7.12 pav. pateiktą elementarų pavyzdį. Šiame paveiksle parodyta, kad duotu laiko momentu atmintis yra suskirstyta į tam tikrus skyrius. Natūralu, kad operacinės sistemos branduolys visą laiką yra atmintyje (rezidentinis). Be to, yra tam tikras skaičius suaktyvintų procesų, tarp jų *A* ir *B*, kurie atmintyje irgi užima tam tikrus skyrius.

Analizę pradėsime tuo momentu kai *A* procesas yra vykdomas (7.12 pav. a). Šiuo atveju, procesorius vykdo *A* atminties skyriuje esančios programos instrukcijas. Po tam tikro laiko (7. 12 pav. b) procesorius nustoja vykdyti *A* proceso instrukcijas ir pradeda vykdyti operacinės sistemos atminties srityje esančias instrukcijas. Tai gali atsitikti dėl vienos iš trijų priežasčių:

1. Procesas *A* operacinei sistemai siunčia aptarnavimo užklausą (pvz., Į/I užklausą). Proceso *A* vykdymas bus sustabdytas kol operacinė sistema nepatenkins šios užklausos.
2. Procesas *A* nutraukiamas *pertraukties*. Kaip žinia, pertrauktis yra signalas, kurį techninė įranga generuoja procesoriui. Kai aptinkamas šis signalas, procesorius nutraukia *A* proceso vykdymą ir susisiekia su pertraukties aptarnaujančiu mechanizmu operacinėje sistemoje. Pertrauktis gali atsirasti dėl daugybės priežasčių susijusių su *A* procesu. Pavyzdžiui, tokia klaida {error}, kaip bandymas įvykdyti privilegijuotą instrukciją. Kitas pavyzdys, laiko limito baigimas {timeout}. Tam, kad koks nors vienas procesas nemonopolizuotų procesoriaus, kiekvieną procesą procesorius gali vykdyti tik tam tikrą iš anksto nurodytą laiką (išskiriamas laiko limitas procesui).



7.13 pav. Pagrindiniai multiprogramaminį režimą užtikrinantys operacinės sistemos elementai

3. Koks nors su *A* procesu nesusijęs įvykis, kuris iššaukia pertrauktį. Pavyzdžiui, Į/I operacijos vykdymas.

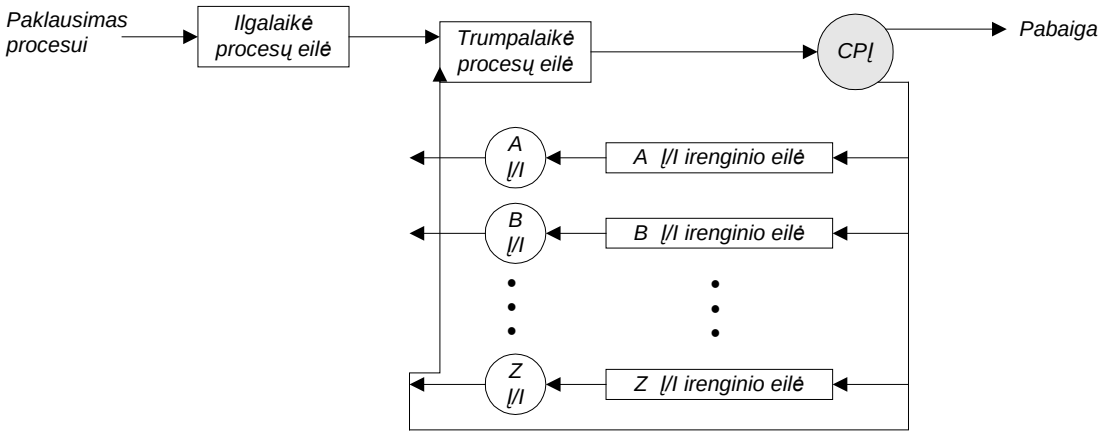
Nepriklausomai nuo šių trijų priežasčių, pasekmės bus tokios. Procesorius *A* proceso valdymo bloke išsaugo *A* proceso esamo konteksto duomenis ir *A* programos skaitiklio reikšmę, o po to pradeda vykdyti operacinės sistemos instrukcijas. Operacinė sistema turi atlikti tam tikrą darbą, tarkime inicijuoti Į/I operaciją. Tada, trumpalaikis {short-term} operacinės sistemos planuotojas nustato, kuris iš procesų bus vykdomas toliau. Mūsų pavyzdyje, tai *B* procesas. Operacinė sistema šiuo atveju atkuria *B* proceso konteksto duomenis ir šį procesą pradeda vykdyti nuo tos vietos, kur jis buvo anksčiau nutrauktas (7.12 pav. c).

Išnagrinėtas (7.12 pav.) pavyzdys rodo pagrindinius artimesnio laiko {short-term} planuotojo veiksmus. 7.13 pav. pateikti operacinės sistemos elementai, kurie yra įtraukti į multiprograminį darbo režimą ir procesų planavimą. Taigi jei yra pertrauktys, operacinė sistema iš procesoriaus perima pertraukčių valdymo mechanizmą, o jeigu atsiranda užklausos aptarnavimui {service-call} – aptarnavimo mechanizmo valdymą. Kai pertraukčių arba aptarnavimo mechanizmai yra perimti, trumpalaikis (artimesniojo darbo) planuotojas turi pasirinkti kitą procesą vykdymui.

Savo funkcijoms vykdyti, operacinė sistema sudaro ir tvarko kelias eiles (7.13 pav.). Kiekviena eilė iš esmės yra tam tikrų kompiuterio resursų laukiančių procesų sąrašas. *Ilgalaikė eilė* yra uždavinių, laukiančių savo eilės būti vykdomiems, sąrašas. Tam tikroms sąlygoms atsiradus, aukšto lygio planuotojas sutvarko atmintį ir pradeda naują procesą vienam iš belaukiančių uždavinių spręsti. *Trumpalaikė eilė* sudaryta vien iš darbui parengtų procesų. Kiekvienas iš šių procesų gali būti tuoj pat pradėtas vykdyti. Trumpalaikio planuotojo tikslas surasti iš jų tinkamą projektą. Paprastai tai daroma taikant ciklišką algoritmą {round-robin}, kuris užtikrina kiekvieno eilėje esančių procesų vykdymą. Čia taip pat gali būti taikomi prioritetiniai lygiai. Galiausiai kiekvienam Į/I įrenginiui yra nuosavos Į/I eilės. Kiekvienas Į/I įrenginys gali būti reikalingas daugiau kaip vienam procesui. Visi procesai, šiuo atveju, surikiuojami į tam tikro *Į/I įrenginio eilę*.

7.14 pav. parodyta, kaip operacinė sistema valdo proceso vykdymą kompiuteryje. Kiekvienas numatomas procesas (paketinis darbas arba interaktyvusis darbas su vartotoju) statomas į ilgalaikę eilę. Kai tik atsilaisvina resursai, numatomas procesas {process request} tampa tikru procesu, pervedamas į parengto proceso stadiją ir statomas į trumpalaikę eilę. Procesorius persijunginėja tarp operacinės sistemos instrukcijų vykdymo ir vartotojo proceso vykdymo. Tuo laiku, kai valdymas priklauso operacinei sistemai (o ne procesoriui), nusprendžiama, kuris procesas iš trumpalaikės eilės bus vykdomas sekančiu. Tuomet operacinė sistema užbaigia savo tiesioginius uždavinius ir procesoriui perduoda pasirinktą procesą.

Anksčiau jau buvo minėta, kad procesas gali būti pristabdytas dėl įvairių priežasčių. Jeigu jis sustabdomas dėl to, kad reikia įvykdyti Į/I operaciją, tada jis statomas į Į/I operacijų eilę. Jeigu jis sustabdomas dėl to, kad baigėsi jo laiko limitas arba operacinė sistema turi atlikti kažkokius neatidėliojamus veiksmus, tada jis pervedamas į parengto proceso stadiją ir vėl statomas į trumpalaikę eilę.



7.14 pav. Procesoriaus darbo planavimo eilių diagrama

Galiausiai, buvo minėta, kad operacinė sistema taip pat valdo Į/I operacijų eiles. Kai tam tikra Į/I operacija įvykdoma, operacinė sistema aptarnautą procesą „išbraukia“ iš Į/I operacijų eilės ir stato jį į trumpalaikę eilę. Toliau, operacinė sistema renkasi kito Į/I operacijos belaukiančio proceso (jeigu, suprantama, toks yra) ir signalizuoja atitinkamam Į/I įrenginiui, kad reikia apdoroti procesą.

Buvo minėta, kad pagrindinis operacinės sistemos uždavinys yra valdyti kompiuterio resursus. 7.14 pav. parodyta, kaip operacinė sistema taikant eilių mechanizmą valdo procesoriaus laiką ir Į/I įrenginius. Liko dar vienas labai svarbus resursas – atmintis. Atminties valdybą nagrinėsime kitame skyriuje.

7.3. Atminties tvarkymas – Memory Management

Vienprogramių operacinių sistemų atveju, pagrindinė atmintis padalinta į dvi dalis: viena – operacinei sistemai (rezidentinis monitorius), o kita – taikomajai programai, kuri esamu momentu yra vykdoma. Multiprograminei sistemai esant, „vartotojiška“ atminties dalis turi būti dalinama toliau, siekiant užtikrinti aibės procesų veikimą. Šis tolimesnis atminties dalinimas vykdomas operacinės sistemos dinamiškai ir vadinamas *atminties tvarkymu* {memory management}.

Efektyvus atminties tvarkymas yra tiesiog gyvybiškas multiprograminėms sistemoms. Jeigu operatyviojoje atmintyje patalpinti tik keli iš esamos aibės procesai, tada likusieji didžiąją laiko dalį lauks savo eilės būti įkeltiems į atmintį ir procesorius bus priverstas šiek tiek laiko stovėti be darbo. Tokiu būdu atmintyje turi būti galimybė išdėstyti kuo daugiau procesų.

7.3.1. Procesų apykaita – Swapping

Grįžkime prie 7.14 pav., kuriame aptarėme tris eilių tipus: procesų ilgų eilių sudarymas naujiems (būsimiems) procesams, procesų, kurie yra pasiruošę naudoti procesorių trumpų eilių sudarymas ir įvairių Į/I procesų, kurie dar nepasirengę naudoti procesorių, eilių sudarymas. Prisiminkime, kad šių sudėtingų mechanizmų atsiradimo priežastis yra ta, kad Į/I aktyvumas yra žymiai lėtesnis negu skaičiavimo operacijų, todėl vienprogramėse sistemose procesorius priverstas didelę laiko dalį stovėti be darbo.

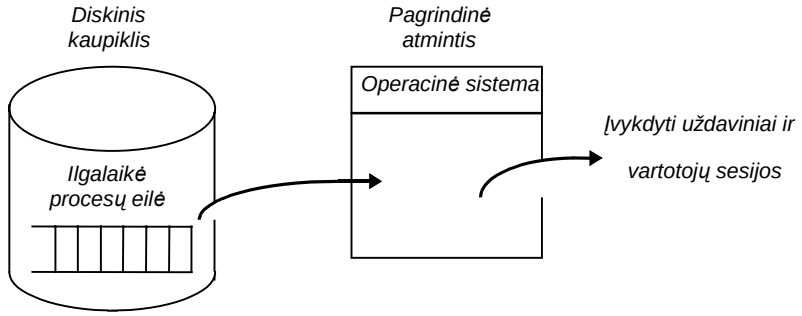
Tačiau 7.14 pav. pateiktas pavyzdys šios problemos iki galo neišsprendžia. Tai tiesa, kad šiuo atveju atmintyje saugoma aibė procesų ir, kad procesorius gali pereiti nuo vieno proceso prie kito, jeigu prieš tai buvęs procesas kažko laukia. Bet procesorius vis tiek yra daug spartesnis už Į/I mechanizmą, kuris yra *bendras visiems* atmintyje esantiems ir Į/I procedūrų laukiantiems procesams. Tokiu būdu, net multiprograminei sistemai esant procesorius priverstas stovėti be darbo gana daug laiko.

Ką gi daryti? Pagrindinę atmintį galima išplėsti ir, tokiu būdu, talpinti joje daugiau procesų. Tačiau, šiame priėjime yra du trūkumai. Pirmas, operatyvioji atmintis yra brangi net ir šiandien. Antras, pingant

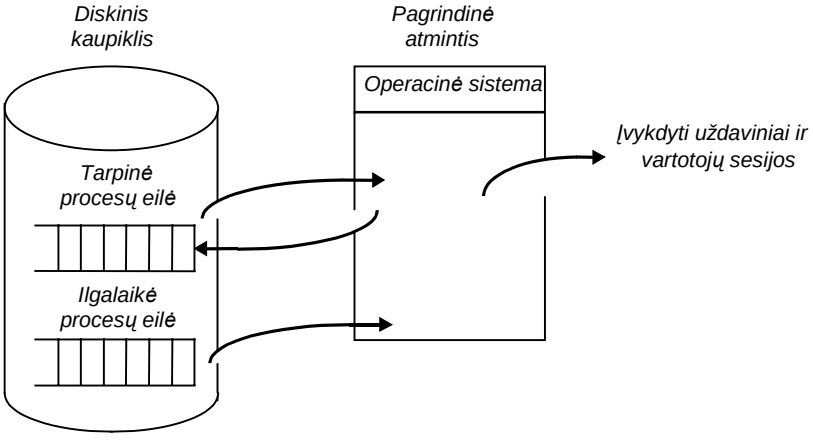
operatyviajai atminčiai programuotojų poreikiai proporcingai didėja. Tokiu būdu, didėjant atminties talpai, stambėja patys procesai ir vietos jiems vis tiek neužtenka.

Kitas problemos sprendimo būdas yra 7.15 pav. parodyta vadinamoji procesų *apykaita* {swapping}. Ilga procesų eilė paprastai saugoma diskiniame kaupiklyje. Jie iš ten išimami po vieną kai tik atsiranda laisvoji vieta operatyviojoje atmintyje. Kai procesai įvykdomi, jie iš atminties pašalinami. Gali susikloti tokia situacija, kai ne vienas atmintyje esantis procesas nebus parengtas vykdymui (pvz., dar neįvykdė laukiamos Į/I operacijos). Tada procesorius vieną iš šių procesų sugrąžina atgal į diską į *tarpinę* {intermediate} *eilę*. Tai yra esamų procesų, laikinai išimtų iš atminties, eilė. Operacinė sistema šiuo atveju iš tarpinės eilės į atmintį perkelia kitą procesą, arba ima naują procesą iš ilgos procesų eilės. Toliau jau yra vykdomas naujai atsiradęs procesas.

Tačiau procesų apykaita yra Į/I pobūdžio operacija ir, todėl, procesų vykdymo problema, tokiu būdu, gali tik pablogėti. Bet, kadangi diskinė Į/I, bendruoju atveju, yra sparčiausia visoje kompiuterizuotoje sistemoje, (t. y. lyginant su juostiniais kaupikliais arba spausdintuvais), procesų apykaitos taikymas paprastai gerina kompiuterio galimybes. Labiau patobulinta schema, kurioje, be to, yra ir virtualioji atmintis, kompiuterio našumą didina labiau negu paprasta procesų apykaita. Tai aptarsime šiek tiek vėliau. Pasiruošimui iš pradžių aptarkime skyrių sudarymą {partitioning} atmintyje ir *puslapiavimą* {paging} – atminties padalinimą į puslapius.



a) Elementarus uždavinio planavimas



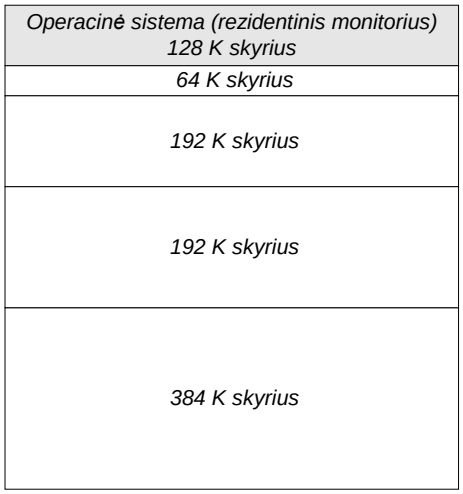
b) Proccesų apykaita

7.15 pav. Procesų apykaitos taikymas

7.3.2. Skyrių atmintyje sudarymas – Partitioning

Operacinė sistema pagrindinėje atmintyje užima fiksuoto dydžio sritį. Likusioji atmintis suskaidoma į skyrius, kuriuose galima išdėstyti procesus. Paprasčiausia skyrių sudarymo schema yra *fiksuoto dydžio skyrių* sudarymas taip, kaip parodyta 7.16 pav. Pažymėkime, kad, nors skyrių dydžiai yra fiksuoti, tačiau jie

nėra lygūs. Kai procesas yra perkeliamas į atmintį, jis išdėstomas mažiausiame iš tų, kuriuose minėtas procesas tik gali tilpti, skyriuje.



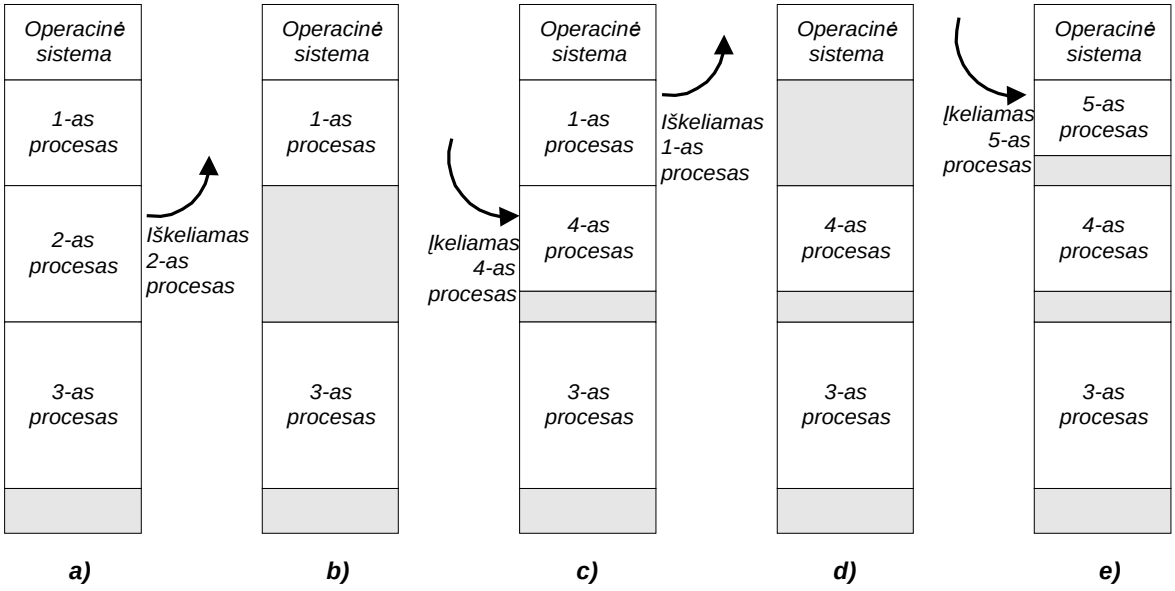
7.16 pav. Fiksuoto dydžio skyrių pavyzdys

Tačiau, net skirtingų fiksuoto dydžio skyrių taikymas negelbėja nuo atminties resursų praradimo. Daugumoje praktinių atvejų konkrečiam procesui nereikia tiek atminties, kiek jam suteikia skyrius. Pavyzdžiui, jeigu procesui reikia 128 K atminties, tai 7.16 pav. pateiktuose skyriuose jis užims 192 K skyrių, tokiu būdu iš kito galimo proceso eliminuodamas 64 K.

Labiau efektyvesnis yra *kintamojo dydžio skyrių* metodas. Kai procesas perkeliamas iš diskinio kaupiklio į atmintį, jis užima būtent tiek vietos, kiek jam reikia ir nedaugiau. Išnagrinėkime 7.17 pav. pateiktą pavyzdį. Iš pradžių pagrindinė atmintis būna tuščia, išskyrus srities, kur išdėstyta operacinė sistema. Pirmieji trys procesai įkeliami į atmintį pradedant nuo tos vietos kur baigiasi operacinė sistema (7.17 pav. a). Šiuo atveju atminties gale susidaro neužpildyta sritis, kuri yra permaža ketvirtajam procesui išdėstyti. Kai 2-as procesas eliminuojamas iš atminties {swapped out}, atsiranda pakankamai vietos 4-am procesui. Kadangi mūsų atveju 4-as procesas yra mažesnis už antrąjį, atsiranda dar viena truputi neužpildyta atminties sritis (7.17 pav. b ir c). Šiame pavyzdyje parodyta, kad metodas pradžioje veikia labai gerai, tačiau galiausiai situacija gali susiklostyti taip, kad atmintyje bus labai daug neužpildytų sričių. Laikui bėgant, atmintis tampa vis labiau *fragmentuota* (7.17 pav. e) ir jos naudojimo efektyvumas mažėja. Šią problemą padeda išspręsti sutankinimo {compaction} metodas: laikas nuo laiko operacinė sistema procesus atmintyje sustumia į vieną bloką. Tačiau, šio metodo realizacija reikalauja papildomų procesoriaus laiko išlaidų.

Prieš nagrinėjant kaip išspręsti skyrių sudarymo problemas, turime gerai suvokti vieną svarbų momentą. Jeigu šiek tiek gyliau panagrinėti 7.17 pav., taps akivaizdu, kad įkėlimas į pagrindinę atmintį visada vyksta į skirtingas vietas (su skirtingais adresais). Dar labiau, jeigu naudojamas sutankinimas, informacija bus sustumta visoje atmintyje. Atmintyje vykstantys procesai susidaro iš instrukcijų ir duomenų. Pačios instrukcijos savyje turi turėti tokius atminties pozicijų adresus:

- Duomenų adresus.
- Instrukcijų, taikomų šakojimuisi, adresus.



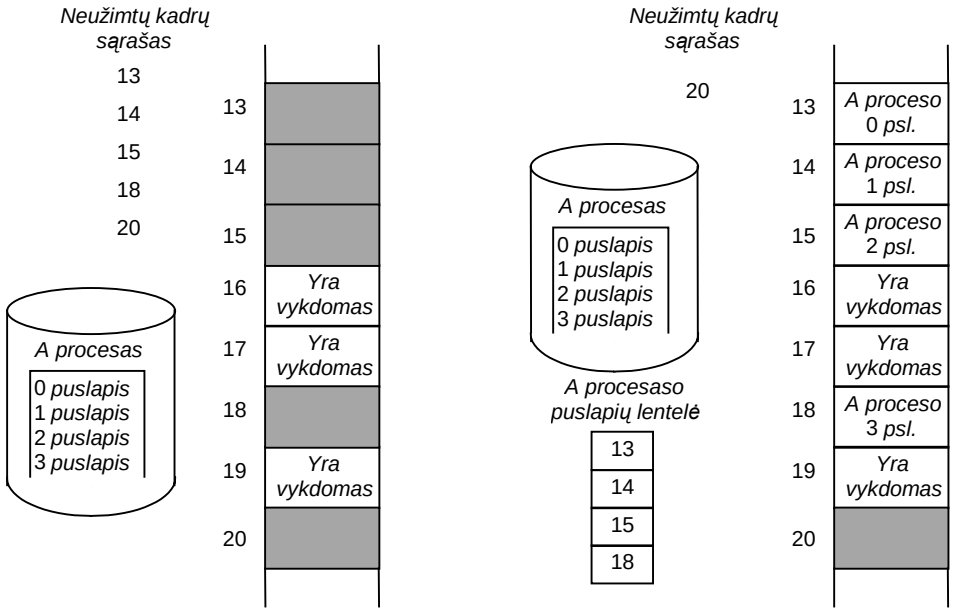
7.17 pav. Kintamojo dydžio skyrių sudarymo efektas

Tačiau matyti, kad šie adresai nėra fiksuoti! Jie pastoviai keisis, kai vyks procesų apykaita. Šiai problemai išspręsti adresai skiriami į loginius ir fizinius. *Loginiai adresai* išreiškiami tam tikru skaičiumi (pozicija programos pradžios atžvilgiu). *Fiziniai adresai*, suprantama, yra tikros ląstelių pozicijos atmintyje. Vykdant procesą procesorius loginius adresus automatiškai keičia į fizinius adresus, prie kiekvieno loginio pridedant proceso starto poziciją (*bazinį adresą*). Tai yra dar vienas iš pavyzdžių, kai CPĮ projektuojami atsižvelgus į operacinės sistemos poreikius. Konkreti šių įrenginių realizacija priklauso nuo taikomos atminties tvarkymo strategijos. Vėliau pamatysime šių strategijų pavyzdžius.

7.3.3. Padalinimas į puslapius – Paging

Ir fiksuoto, ir kintamo dydžio skyriai nėra efektyvūs atminties naudojimo atžvilgiu. Tačiau, įsivaizduokime, kad atmintis yra padalinta į vienodo, fiksuoto, palyginus mažo dydžio skyrelius, o kiekvienas procesas, taip pat padalintas į analogiško fiksuoto dydžio fragmentus. Šiuo atveju, programos fragmentai, kuriuos vadina *puslapiais* {pages}, gali būti priskirti atitinkamiems atminties skyreliams, kuriuos vadina *kadrais* {frames} arba puslapių kadrais {page-frames}. Tada gaunasi, kad neišnaudotas atminties dydis yra lygus paskutinio puslapio (arba kadro) likučiui.

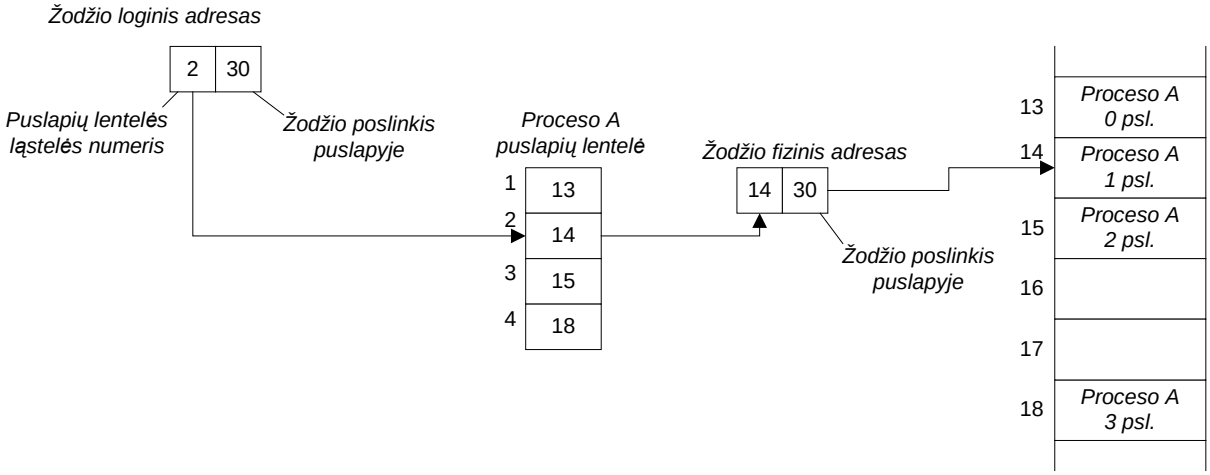
7.18 pav. pateikti puslapių ir kadrų naudojimo pavyzdžiai. Tam tikru laiko momentu atmintyje yra ir užimtų ir laisvų kadrų. Neužimtųjų kadrų sąrašą sudaro ir atnaujina operacinė sistema. Pavyzdžiui, *A* procesas saugomas diskiniame kaupiklyje ir susidaro iš keturių puslapių. Kai ateina momentas šį procesą paleisti į darbą, operacinė sistema atmintyje suranda keturis laisvus kadrus ir į juos patalpina minėtą procesą.



7.18 pav. Laisvųjų kadrų {frame} išdėstymas

Dabar įsivaizduokime, kad nebeliko laisvų gretimų kadrų, kuriuose galima būtų išdėstyti procesą. Ar tai yra neapeinama kliūtis operacinei sistemai išdėstant *A* procesą? Ne, taip nėra, kadangi galima pritaikyti loginių adresų koncepciją. Tada vien bazinio adreso nepakaks. Suprantama, kad operacinė sistema sudaro kiekvienam procesui *puslapių lentelės* {page table}. Puslapių lentelėje yra užrašyta, kur atmintyje yra kiekvienas kadras, kuriame saugomas tam tikras proceso puslapis. Kaip ir paprasto skyrių sudarymo atveju loginis adresas yra tam tikro žodžio padėtis programos pradžios atžvilgiu; CPĮ šią padėtį transliuoja į fizinį adresą. Sudarinėjant puslapius {paging}, loginių adresų transliavimą į fizinius užtikrina CPĮ schemotechniniai įrenginiai. Toks CPĮ turi „žinoti“, kaip reikia kreiptis į vykdomo proceso puslapių lentelę. Gaunant loginį adresą (puslapio numerį/santykinį adresą) CPĮ, taikant puslapių lentelę, generuoja fizinį adresą (kadro numeris/santykinis adresas). Toks pavyzdys pateiktas 7.19 pav.

Šis būdas įgalina išspręsti anksčiau iškilusias problemas. Pagrindinė atmintis padalinama į aibę mažo pastovaus dydžio kadrus. Kiekvienas procesas, savo ruožtu dalinamas į analogiško dydžio puslapius: mažam procesui pakaks kelių kadrų, dideliam jų prireiks daugiau. Kai procesas pasirenkamas vykdymui, jo puslapiai įkeliami į laisvus kadrus ir sudaroma atitinkama puslapių lentelė.



7.19 pav. Loginiai ir fiziniai adresai

7.3.4. Virtualioji atmintis

Puslapių „poreikis“

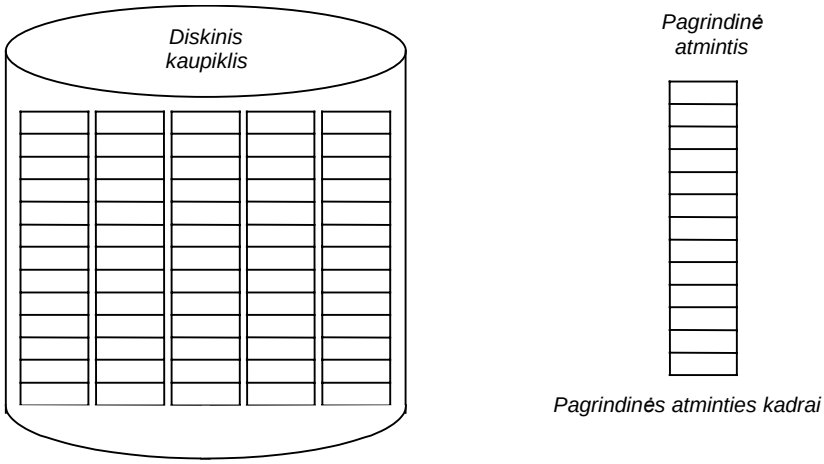
Puslapių sudarymo metodo taikymas įgalino kurti efektyvias multiprogramines operacines sistemas. Taip pat yra labai svarbu, kad paprasta proceso skaidymo į puslapius taktika įgalino sukurti kitą svarbią koncepciją: virtualiąją atmintį.

Norint suprasti kas yra virtualioji atmintis, reikia patikslinti ką tik išnagrinėtą puslapių sudarymo schemą. Tas patikslinimas yra *reikalaujamo puslapio (deficitinio puslapio)* {demand paging} sąvoka, kuri, paprastai reiškia, kad kiekvienas proceso puslapis įkeliamas į atmintį tada ir tik tada, kai jis reikalingas t. y., kai jaučiasi jo stoka.

Išnagrinėkime stambų procesą, kuriame be ilgos programos yra dar keli duomenų masyvai. Tam tikru trumpu laiko momentu, proceso vykdymas gali būti apribotas tik programos nedidele dalimi (pvz., tam tikra paprograme) ir, galbūt, tik vienas ar keli duomenų masyvai čia bus reikalingi. Tai yra vadinamasis *lokalizacijos* principas. Akivaizdu, kad yra neracionalu kelti į atmintį dešimtis proceso puslapių jeigu iki programos trumpalaikio pristabdymo {suspended} bus panaudoti tik keli jos puslapiai. Bus daug geriau, jeigu atmintis bus naudojama tik keliems puslapiams įkelti iš visų turimų. Tada, jeigu programoje atsiras šakojimasis į instrukciją, kurios puslapio šiuo metu atmintyje nėra, generuojamas page fault *(puslapio klaida)* signalas. Pagal šį signalą operacinė sistema į atmintį turi įkelti reikiamą puslapį.

Tokiu būdu, tam tikru laiko momentu atmintyje yra tik keli puslapiai iš kiekvieno duoto proceso. Dar labiau, taupomas laikas, kadangi nenaudojami puslapiai nėra kaitaliojami {swapped} į atmintį ir iš jos. Tačiau operacinei sistemai turi būti aišku, kaip tokią sudėtingą schemą valdyti. Kai operacinė sistema talpina puslapį į atmintį, tam tikrą kitą puslapį reikia iš atminties išimti. Jeigu išimamas puslapis, kuris buvo ką tik naudojamas, labai dažnai būna, kad šį puslapį tenka naudoti vėl, praktiškai be jokios uždelsos. Dažnas tokių įvykių pasikartojimas iššaukia vadinamąjį *atminties perkrovos* efektą {trashing}: kai procesorius didžiąją savo laiko dalį naudoja ne instrukcijoms vykdyti, bet puslapių į atmintį ir iš atminties kaitaliojimui. Atminties perkrovos problemai spręsti 1970-taisiais metais buvo atlikta labai daug tyrimų ir sukurta nemažai sudėtingų, bet efektyvių algoritmų. Jų esmė yra tame, kad analizuojant proceso priešistoriją operacinė sistema, bando nuspėti, kuris puslapis mažiausiai tikėtina, bus naudojamas artimesnėje ateityje.

Taikant deficitinių puslapių metodą nėra reikalo krauti į pagrindinę atmintį visą procesą. Šis faktas turi labai svarbias pasekmes: *proceso apimtis gali būti didesnė už visą pagrindinę atmintį.* Tokiu būdu, vienas pagrindinių programavimo apribojimu yra pašalinamas. Be deficitinių puslapių metodo programuotojas turėjo tiksliai žinoti kokiais atminties resursais jis gali disponuoti. Jeigu programa gaudavosi per daug didelė, programuotojas privalėjo sugalvoti kaip programą restruktrūktrizuoti į dalis, kurios jau galėtų būti talpinamos į atmintį. Deficitinių puslapių metodui esant, šį darbą vykdo operacinė sistema ir kompiuterio techninė įranga. Dabar, jeigu programuotojas susirūpins dėl atminties, jis gali disponuoti didžiule atmintimi, kurios apimtis ribojama tik laisvąja diskinio kaupiklio talpa. Taikant deficitinių puslapių metodą operacinė sistema įkelia į pagrindinę atmintį tik dalį esamo proceso.



7.20 pav. Virtualioji atmintis

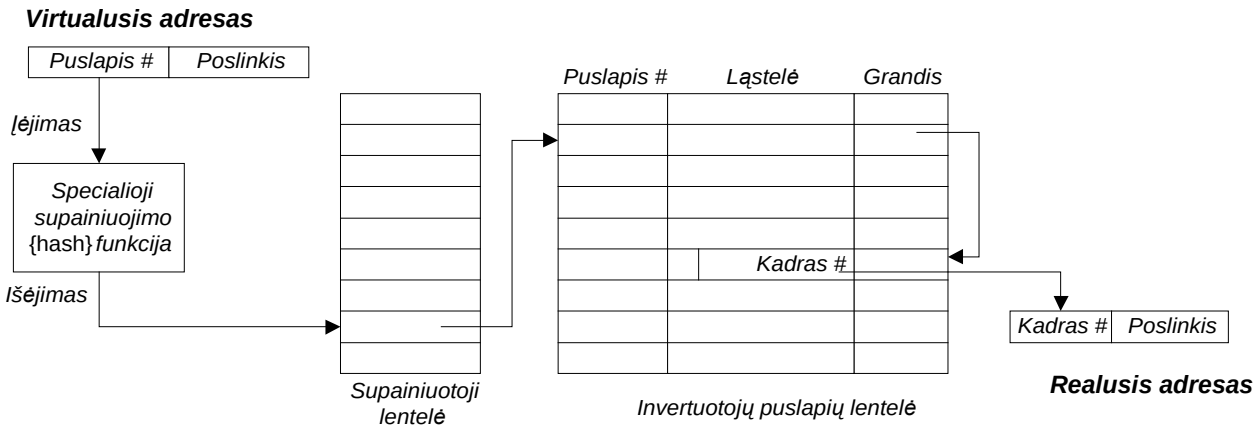
Kadangi vykdomi tik tie procesai, kurie yra šiuo metu pagrindinėje atmintyje, ši atmintis dažnai vadinama realiąja atmintimi {real memory}. Bet programuotojas arba vartotojas gali manyti, kad turi daug didesnę atmintį, kuri išdėstoma diskiniame kaupiklyje. Pastaroji atmintis vadinama *virtualiąja atmintimi* (*tikrąja atmintimi* – 7.20 pav.). Virtualioji atmintis užtikrina labai efektyvų multiprograminį darbo režimą ir išlaisvina vartotoją iš ankštos pagrindinės atminties.

7.3.5. Puslapių lentelės struktūra

Tokiu būdu, žodžio skaitymo atmintyje pagrindiniame mechanizme yra numatyta virtualiųjų (loginių) adresų, turinčių puslapio numerį ir postūmio {offset} reikšmę, transliacija į fizinius adresus, turinčius kadro numerį ir postūmio reikšmę, taikant puslapio lentelę. Puslapių lentelė yra kintamo, priklausančio nuo proceso dydžio, ilgio, todėl jos negalima saugoti CPĮ registruose. Reiškia, ji turi būti saugoma pagrindinėje atmintyje. 7.19 pav. pateikta šios schemos techninė realizacija. Kai prasideda tam tikro proceso vykdymas, CPĮ registruose saugomas šio proceso puslapių lentelės pradžios adresas. Loginio adreso puslapio numeris, šiuo atveju, naudojamas šiai lentelei indeksuoti ir nustatyti atitinkamą proceso kadro numerį. Norint gauti realųjį atitinkamą adresą (fizinį), šis numeris apjungiamas su virtualiojo adreso poslinkio dalimi.

Išnagrinėkime, kiek ląstelių gali būti puslapio lentelėje. Kompiuterizuotų sistemų daugumoje vienam procesui sudaroma viena puslapių lentelė. Tačiau virtualiojoje atmintyje kiekvienas procesas gali užimti labai daug vietos. Pavyzdžiui, VAX architektūros kompiuteriuose kiekvienam procesui numatyta iki $2^{31} = 2{,}14748 \cdot 10^9 \approx 2$ GB virtualiosios atminties. Taikant $2^9 = 512$ baitų dydžio (t.y. diskinio kaupiklio segmento dydžio) puslapius, tai reiškia, kad kiekvienam procesui puslapio lentelėje gali prireikti iki $2^{22} = 4194304 \approx 4$ M ląstelių ($2^9 \cdot 2^{22} = 2^{31}$). Akivaizdu, kad tokiu atveju, atminties dalis, skirta puslapių lentelėms, gali pasidaryti pernelyg didelė. Šiai problemai išspręsti, daugelis virtualiosios atminties schemų puslapių lenteles saugo ne realiojoje atmintyje, bet toje pačioje virtualiojoje. Tai reiškia, kad puslapių lentelė, lygiai kaip ir patys puslapiai yra puslapiavimo mechanizmo {paging} objektas. Kai prasideda proceso vykdymas, pagrindinėje atmintyje turi būti saugoma mažiausiai puslapių lentelės tam tikra dalis (kurioje yra tame tarpe ir dabar vykdomo puslapio ląstelė). Kai kurie procesoriai, didelio dydžio lentelėms organizuoti, gali naudoti dviejų lygmenų schemas. Tokioje schemoje taikoma vadinamoji puslapių direktorija (faktiškai tai yra lentelių direktorija arba lentelė, kurioje kiekviena ląstelė atitinka tam tikrą puslapių lentelę). Tokiu būdu, jeigu puslapio direktorijos dydis yra X, o maksimalus puslapių lentelės ilgis Y, tada procesas gali būti sudarytas iš $X \times Y$ puslapių. Dažniausiai, maksimalus puslapių lentelės ilgis ribojamas vieno puslapio dydžiu. Dviejų lygmenų lentelių organizacijos pavyzdį pamatysime vėliau, kai nagrinėsime Pentium® procesoriaus atminties tvarkymą.

Be vieno – arba dviejų lygmenų puslapių lentelių virtualiosios atminties adresavimo metodų naudojama ir vadinamoji *invertuotųjų puslapių lentelių* struktūra (7.21 pav.). Šis priėjimas taikomas IBM AS/400 ir visuose RISC sistemose, tame tarpe ir PowerPC.



7.21 pav. Invertuotųjų puslapių lentelės struktūra

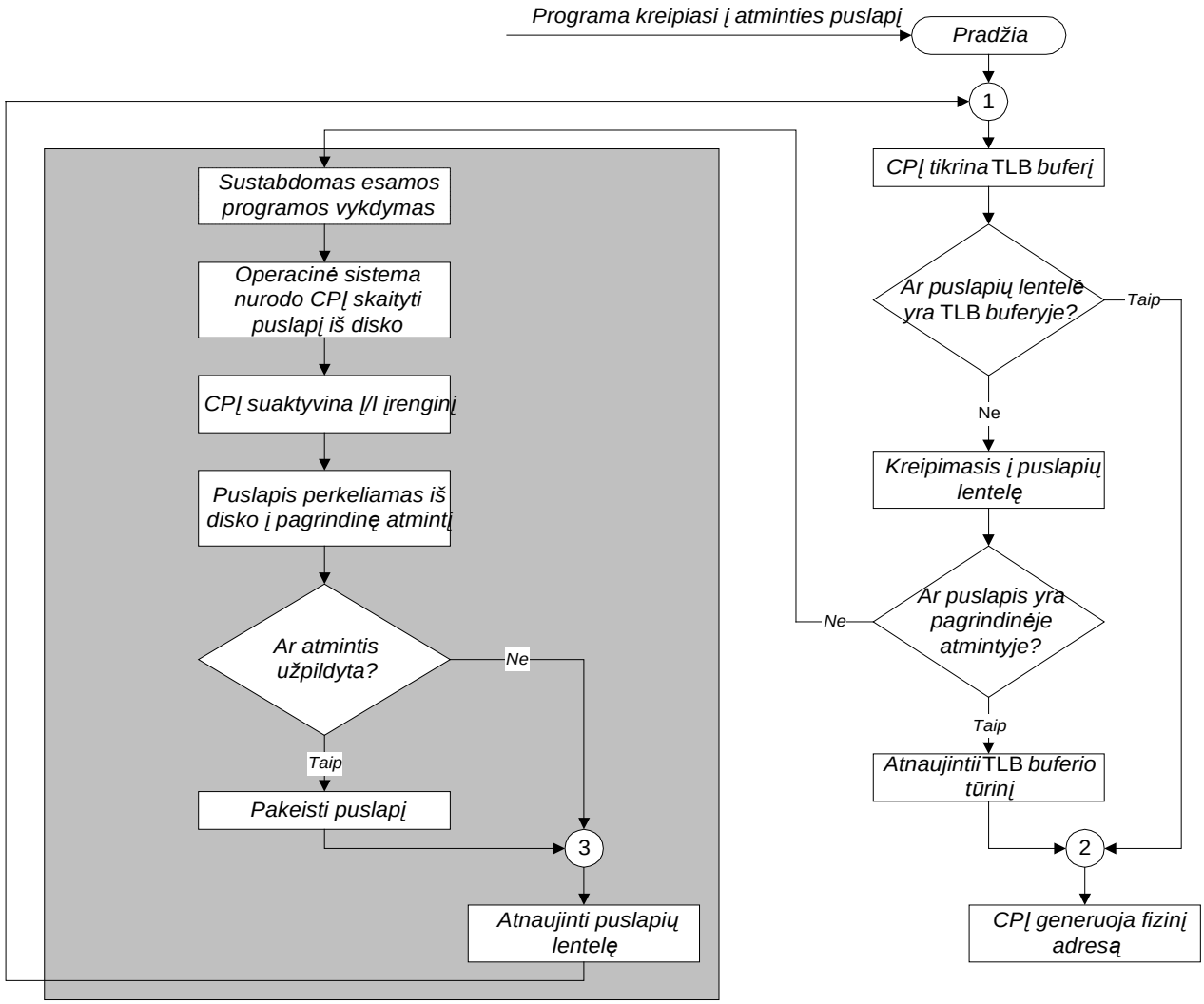
Šiam priėjimui esant, virtualiųjų adresų puslapių numerių porcija atvaizduojama vadinamojoje supainiotojoje {hash} lentelėje, taikant elementarią supainiojimo {hashing} funkciją[1]. Supainiotojoje lentelėje yra nuoroda į invertuotųjų puslapių lentelę, kurioje yra puslapių lentelės ląstelės. Tokioje struktūroje viena supainiotosios ir viena invertuotųjų puslapių lentelės ląstelės atitinka kiekvieną realiosios, o ne virtualiosios atminties puslapį. Tokiu būdu, nežiūrint į tai, koks gali būti procesų skaičius arba virtualiosios atminties dydis, puslapių lentelėms reikia fiksuoto dydžio realios atminties srities. Kadangi į kiekvieną supainiotosios lentelės ląstelę gali būti atvaizduotas daugiau negu vienas virtualusis adresas, persiliejimo {overflow} procesui valdyti taikomas sujungimo grandimi {chaining} metodas. Supainiojimo metodas atsirandančias grandis įtakuoja taip, kad jos gaunasi labai trumpos – kiekvienoje tik viena arba dvi ląstelės.

7.3.6. Peržiūros iš šalies transliacijos buferis – Translation Lookaside Buffer

Potencialiai kiekviena kreiptis į virtualiąją atmintį gali iššaukti dvi kreiptis į fizinę atmintį: vieną, kad pasirinkti atitinkamą puslapių lentelės ląstelę ir vieną, kad pasirinkti reikiamų duomenų. Tokiu būdu, tiesioginė virtualiosios atminties schema gali iššaukti kreipties į atmintį trukmės dvigubėjimo efektą. Šiai problemai išspręsti, virtualiosios atminties schemų dauguma puslapių lentelės ląstelėms gali naudoti specialią spartinančiąją atmintį („kešą“), paprastai vadinamą peržiūros iš šalies transliacijos buferiu {Translation Lookaside Buffer – TLB}. Šis „kešas“ veikia panašiu principu kaip ir pagrindinės atminties „kešas“ ir jame saugomos tos puslapių lentelės ląstelės, kurios yra dažniausiai naudojamos. 7.22 pav. pateiktas TLB veikimo algoritmas. Pagal lokalizacijos principą, dauguma kreipčių į virtualiąją atmintį turi būti išdėstyti dažniausiai naudojamuosiuose puslapiuose. Todėl kreipčių dauguma puslapių lentelės ląsteles įrašinėja į „kešą“, o ne į operatyviąją atmintį. VAX architektūros kompiuterių TLB tyrimai parodė, kad tokia schema žymiai padidina kompiuterizuotos sistemos našumą [12, 13].

Pažymėtina, kad virtualiosios atminties mechanizmas turi sąveikauti su kompiuterio spartinančiąja atmintimi (ne su TLB „kešu“, bet būtent su pagrindinės atminties „kešu“), Tai parodyta 7.23 pav. Bendruoju atveju, loginis adresas egzistuoja kaip puslapio numerio ir postūmio kombinacija. Pradžioje atminties sistema tikrina, ar yra TLB buferyje pažymėta puslapių lentelės ląstelė. Jeigu ji ten yra, realusis (fizinis) adresas generuojamas apjungiant kadro numerį ir postūmį. Jeigu reikiamos ląstelės TLB buferyje nėra, ji skaitoma iš puslapių lentelės pagrindinėje atmintyje. Kai realusis adresas pagal „tego“ ir likučio formą

[1] Supainiojimo funkcija atvaizduoja 0 … *M* ruožo skaičius į 0 … *N* ruožą, M > N. Supainiojimo funkcijos išėjimas (rezultatas) naudojamas kaip supainiotosios lentelės indeksas. Kadangi kiekvieną funkcijos išėjimą atitinka daugiau negu vienas įėjimas, įėjimo reikšmę galima atvaizduoti į jau užimtą supainiotosios lentelės ląstelę. Šiuo atveju, naujoji reikšmė turi persilieti {overflow} į kitą supainiotosios lentelės poziciją. Paprastai, naujoji reikšmė patalpinama į pirmą pasitaikiusią tuščią lentelės vietą, o rodyklė iš originalios pozicijos nukreipiama taip, kad sujungti abi ląsteles kartu. Detaliau apie supainiotąsias lenteles galima pasiskaityti [11].



7.22 pav. „Puslapiavimo“ operacija ir peržiūros iš šalies transliacijos buferis

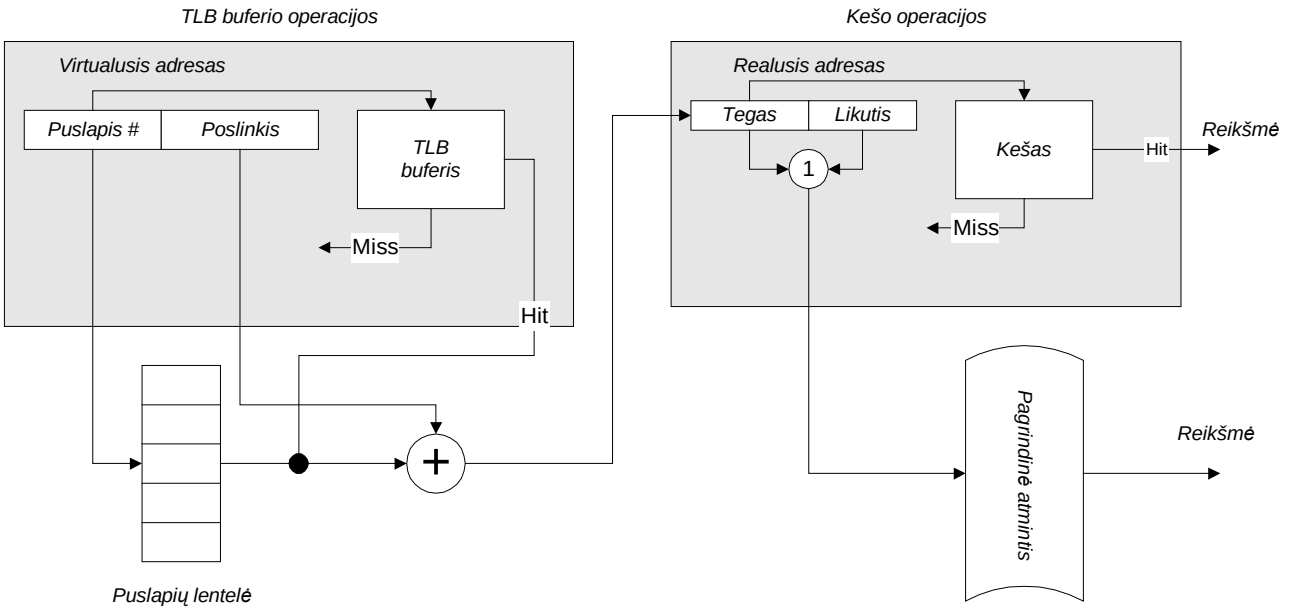
sugeneruotas, tikrinamas „kešas“, ar jame yra duomenų blokas su reikiamu žodžių? Jeigu šis žodis ten yra, jis siunčiamas procesoriui. Jeigu reikiamo žodžio „keše“ nėra jis imamas iš pagrindinės atminties.

Būtina gerai suprasti kokį sudėtingą darbą turi atlikti CPĮ, kad įvykdyti paprastą kreities į atmintį. Loginis adresas transliuojamas į realųjį adresą. Šiame etape vykdomas kreipimasis į puslapių lentelę, kuri gali būti TLB buferyje, pagrindinėje atmintyje arba diskiniame įrenginyje. Reikiamas žodis, savo ruožtu, taip pat gali būti arba keše, arba pagrindinėje atmintyje, arba diskiniame įrenginyje. Pastaruoju atveju puslapis su reikiamu žodžiu turi būti perkeltas į pagrindinę atmintį, o tam tikras duomenų blokas patalpintas į kešą. Be to, šį puslapį atitinkantį puslapių lentelės ląstelė turi būti atnaujinta.

Yra kitas adresuojamos atminties padalijimo būdas, ji vadina *segmentavimu*. Tuo tarpu, kai puslapių sudarymas programuotojui yra nematomas ir taikomas tam, kad suteikti programuotojui didesnę adresų erdvę, segmentacijai programuotojui paprastai yra matoma ir suteikia jam galimybę racionaliau organizuoti duomenis ir programas atmintyje, o taip pat suteikti jiems tam tikrus privilegijų ir apsaugos atributus.

Segmentavimas programuotoją įgalina į atmintį žiūrėti kaip sudaryta iš adresų erdvių arba segmentų aibės. Segmentai iš tikrųjų yra dinamiškai kintantys dydžiai. Programuotojas arba operacinė sistema programoms ir duomenims paprastai priskiria skirtingus segmentus. Skirtingiems programų tipams ir duomenims gali būti numatyti skirtingi segmentai (kiekvienam tipui atskiras). Kiekvienam segmentui galima suteikti priėjimo ir naudojimo teises. Kreipimasis į atmintį susidaro iš adreso formos (segmento numeris ir poslinkis).

Tokia organizacija lyginant su nesegmentuota adresų erdve programuotojo požiūriu turi daug privalumų:



7.23 pav. TLB buferio ir „kešo“ operacijos.

1. Supaprastina didžiulės duomenų struktūros valdymą. Jeigu programuotojas iš anksto nežino kokio dydžio gausis tam tikra duomenų struktūra, šito ir nebūtina prognozuoti. Šiai duomenų struktūrai suteikiamas jos nuosavas segmentas, o operacinė sistema jį padidins arba sumažins kiek reikia.
2. Tai leidžia nepriklausomai modifikuoti ir perkompiliuoti programas, nereikalaujant, kad programų paketas būtų iš naujo sutrasuotas {relinked} ir įkrautas {reloaded}. Visa tai daroma taikant aibę segmentų.
3. Tai natūraliai įgalina vykdyti kelis procesus kartu. Programuotojas taikomąją programą arba reikalingą duomenų lentelę gali patalpinti į segmentą, kuris gali būti adresuojamas kitais procesais.
4. Tai elementariai užtikrina apsaugą. Kadangi segmentas gali būti sukonstruotas taip, kad savyje turėti tiksliai apibrėžtas programas arba duomenis, programuotojas, arba sistemos administratorius gali suteikti tam tikras priėjimo privilegijas.

Visų šitų pranašumų negali užtikrinti programuotojui nematomas puslapių sudarymo procesas. Iš kitos pusės mes įsitikiname, kad puslapiavimas užtikrina efektyvias atminties valdymo formas. Kad apjungti abiejų priėjimų galimybes, kai kuriose kompiuterizuotose sistemose numatyta speciali techninė ir operacinės sistemos programinė įranga užtikrinanti ir vieną ir kitą.

7.4. Atminties tvarkymas Pentium® kompiuterizuotoje sistemoje

32 bitų architektūros mikroprocesoriuose taikomos labai sudėtingos atminties valdymo schemos, kurių ištakos yra kilusios iš vidutinių ir super didelių ESM. Daugumos atveju mikroprocesoriai pralenkia savo super didelius pirmtakus. Mikroprocesorių schemotechnikai savo gaminius projektuoja taip, kad jie tiktų kuo plačiausiam operacinių sistemų asortimentui, todėl sakoma, kad jie puikiai atitinka kompiuterio bendrosios paskirties koncepciją. Labai geru pavyzdžiu gali būti Pentium® mikroprocesoriuje taikoma tvarkymo schema. Pentium® atminties tvarkymo schema su tam tikrais patobulinimais yra tokia pati, kaip ir i80386 bei i80486 procesoriuose.

7.4.1. Adresų sritis

Pentium® mikroprocesoriuje numatytos schemotechninės priemonės atminties padalijimui į segmentus ir į puslapius. Tačiau, abu šie mechanizmai gali būti išjungti, tokiu būdu vartotojui leidžiant pasirinkti vieną iš keturių atminties tvarkymo būdų:

- *Atmintis nedalinama į segmentus ir puslapius.* Šiuo atveju, virtualieji adresai atitinka fizinius adresus. Toks būdas gerai tinka, pavyzdžiui, nesudėtinguose, tačiau labai našiuose kontroleriuose.
- *Nesegmentuota į puslapius padalinta atmintis.* Atmintis čia atrodo kaip linijinių adresų padalintų į puslapius sritis. Šiuo atveju atminties tvarkymas ir apsaugojimas organizuoti per puslapius. Šis būdas yra taikomas kai kuriuose operacinėse sistemose, pvz., Berkeley UNIX.
- *Segmentuota į puslapius nedalinta atmintis.* Čia atmintis atrodo kaip loginių adresų sričių rinkinys. Šiuo būdo privalumus lyginant su padalinimu į puslapius – reikalui esant, apsaugojimas atskirų bitų lygmenyje. Dar labiau, skirtingai nuo padalinimo į puslapius, tuo atveju kai segmentas yra atmintyje, garantuota, kad atitinkanti transliavimo lentelė (segmentų lentelė) bus patalpinta tam tikroje atminties mikroschemoje. Tokiu būdu, segmentuotos į puslapius nedalintos atminties atveju, duomenų pasiekimo laikas yra gerai prognozuojamas.
- *Segmentuota į puslapius padalinta atmintis.* Segmentavimas taikomas, kad nustatyti loginės atminties skyrių {partitions subject} kreipimosi į atmintį metu, o padalinimas į puslapius – kad atmintį tvarkyti sektorių {partitions} ribose. Šis būdas taikomas tokiose operacinėse sistemose, kaip UNIX System V.

7.4.2. Atminties padalinimas į segmentus

Kai kompiuteryje taikomas segmentavimas, kiekvienas virtualusis (angl. virtual – faktiškas; nenominalus) adresas (Pentium® dokumentacijoje jį vadina loginiu) susideda iš 16 bitų segmento numerio ir 32 bitų postūmio. Iš 16 segmento numerio bitų 2 bitai naudojami apsaugai (klaidoms aptikti), paliekant 14 bitų pačiam segmentui nurodyti. Tokiu būdu, jeigu atmintis nesegmentuota, vartotojo virtualioji atmintis yra $2^{32} = 4$ Gbaitai. Segmentuotoje atmintyje bendras vartotojui pasiekiamas virtualios atminties dydis yra $2^{46} = 64$ terabaitai (64 TB). Fiziškas adresas – 32 adresų bitai, įgalina kreiptis ne daugiau kaip į $2^{32} = 4$ GB atminties.

Virtualios atminties dydis faktiškai gali būti didesnis negu 64 Tbaitai. Taip gali būti todėl, kad procesoriaus virtualaus adreso interpretacija priklauso nuo vykdomo proceso pobūdžio. Viena virtualios atminties pusė (8 K segmentu $\times$ 4 GB) yra globali ir ją naudoja bendrai visi procesai; likusioji dalis vadinama lokaliąja ir ji gali būti suteikta kiekvienam procesui.

Kiekvienas segmentas apsaugotas dviem būdais: pagal privilegijos lygį ir priėjimo atributus. Privilegijų lygių yra keturi – nuo labiausiai apsaugoto (0 lygis) iki mažiausiai apsaugoto (3 lygis). Segmento, kuriame saugomi duomenys, privilegijos lygis yra jo savotiška "klasifikacija"; segmento, kuriame saugomas programos kodas, privilegijos kodas yra jo vadinamas "švarumas" {"clearance"}. Vykdomoji programa gali kreiptis tik į tuos duomenų segmentus, kurių privilegijos lygis yra aukštesnis (labiau privilegijuotas) arba lygus (lygios privilegijos) programinio segmento "švarumo" lygiui.

Schemotechninėje procesoriaus dalyje nenumatyta kaip šie privilegijų lygiai bus naudojami; tai priklauso tik nuo operacinės sistemos. Pentium® procesoriuje buvo numatyta, kad 1-ąjį privilegijos lygį naudos visos operacinės sistemos, 0-nį lygį – nedidelė dalis operacinių sistemų, kuriose numatytas atminties tvarkymas {management}, apsauga ir priėjimo kontrolė. Tokiu būdu, taikomosioms programoms suteikiamas žemiausias – 3-iasis privilegijos lygis, 2-asis palikdamas nenaudojamu. Pastarąjį – 2-ąjį privilegijos lygį gali naudoti specializuoti taikomieji posistemiai, kuriuose numatytas nuosavas saugumo mechanizmas. Tokių posistemių pavyzdžiais gali būti duomenų bazių valdymo sistemos, ofiso automatizuota sistema, o taip pat inžinerijos programinė įranga.

Be aptarto priėjimo prie duomenų segmentų reguliavimo, privilegijų mechanizmas riboja tam tikrų instrukcijų taikymą. Kai kurios instrukcijos, sakykime tos, kurios susijusios su atminties sutvarkymo registrais, gali būti vykdomos tik 0-iame lygmenyje. Į/I instrukcijos gali būti vykdomos aukštesniame, operacine sistema nustatytame lygyje; paprastai tai būna 1 lygis.

Priėjimo prie duomenų segmento atributai specifikuoja koks būtent priėjimas leistinas: ar skaitymas ir rašymas, ar tik skaitymas. Programinių segmentų atveju, priėjimo atributai gali specifikuoti arba skaitymo ir vykdymo, arba tik skaitymo atributus.

Tuo atveju, jeigu atmintis yra segmentuota, adreso transliavimo mechanizmas, be kito ko, vykdo virtualiojo kreipimosi adreso atvaizdavimą į linijinį adresą (7. 24 pav., b). Virtualusis adresas susideda iš 32 bitų postūmio ir 16 bitų segmento selektoriaus (7. 24 pav., a). Segmento selektoriuje yra tokie laukai:

- *Lentelės indikatorius (LI).* Rodo kuri iš lentelių bus naudojama transliacijoje, ar globaliųjų segmentų ar lokaliųjų segmentų.
- *Segmento numeris.* Šis numeris naudojamas kaip tam tikras indeksas segmentų lentelėje.
- *Reikiamas privilegijos lygis (RPL).* Koks privilegijos numeris reikalingas šiam priėjimui.

7. 24 pav., c parodyta, kad kiekvienas segmentų lentelės įrašas susideda iš 64 bitų. Atitinkami įrašo laukai apibūdinti 7. 6 lentelėje.

7. 6 lentelė. Pentium® procesoriaus atminties tvarkymo parametrai

Segmento deskriptorius (Segmentų lentelės įrašas)	
Base	**Bazė.** Nustato segmento pradinį adresą 4 Gbaitų linijinių adresų erdvėje
D/B bit	Kodo segmente tai yra D bitas, jis rodo kokio ilgio operandas ir adresas – 16 ar 32 bitų
Descriptor Privilege Level (DPL)	**Deskriptoriaus privilegijos lygis.** Specifikuoja duoto segmento privilegijos lygį (0…3)
Granularity bit (G)	**"Grūdėtumas".** Rodo kaip bus interpretuojamas lauko vienetas: ar kaip vienas baitas, ar kaip 4 Kbaitai.
Limit	**Limitas.** Nurodo segmento dydį. Procesorius, priklausomai nuo "grūdėtumo" bito reikšmės lauko dydį gali interpretuoti dvejopai: vieno baito vienetais, šiuo atveju segmento dydžio limitas 1 Mbaitas, arba 4 Kbaitų vienetais – segmento dydžio limitas 4 Gbaitai.
S bit	Nustato koks yra gautas segmentas: ar tai sistemos segmentas, ar duomenų arba kodo segmentas
Segment Present bit (P)	**Segmento pristatymo bitas.** Taikomas tik tuo atveju, kai pagrindinė atmintis nepadalinta į puslapius. Rodo kaip segmentas pateikiamas pagrindinėje atmintyje. Atminties padalintos į puslapius atveju, šis bitas visada 1.
Type	**Tipas.** Varijuojamas priklausomai nuo segmento tipo ir rodo priėjimo atributus.

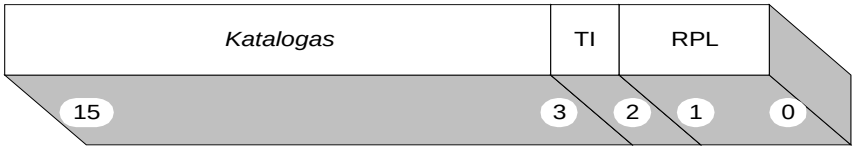
7.4.3. Atminties padalinimas į puslapius

Segmentavimas yra pasirinktinis mechanizmas – jo gali ir nebūti. Tuo atveju, kai segmentavimas vis dėl to taikomas, adresai, kurie naudojami programose yra virtualūs adresai ir jie konvertuojami į linijinius adresus ką tik aprašytu būdu. Kai segmentavimas nenaudojamas, programose naudojami linijiniai adresai. Ir vienu ir kitu atveju, linijinis adresas į realų 32 bitų adresą konvertuojamas per kelis žingsnius.

Kad suprasti linijinio adreso struktūrą, reikia žinoti, kad Pentium® procesoriaus atminties dalinimo į puslapius mechanizmas faktiškai yra dviejų lygmenų lentelės peržiūros operacija. Pirmas lygmuo tai puslapio katalogas kuriame yra iki 1024 sekcijų (įėjimų). Tai leidžia suskaidyti 4 Gbaitų linijinės atminties erdvę į 1024 puslapių grupes, kurių kiekvienos dydis – 4 baitai ir kurių kiekviena turi nuosavą puslapių lentelę. Kiekviena puslapio lentelė taip pat susideda iš 1024 sekcijų (įėjimų); kiekvienas įėjimas atitinka 4 Kbaitų puslapį. Atminties tvarkymo procese yra opcijos įgalinančios naudoti vieną puslapių katalogą visiems procesams arba vieną puslapių katalogą kiekvienam procesui arba šių dviejų kombinaciją. Esamo uždavinio puslapių katalogas visada yra pagrindinėje atmintyje. Tuo tarpu puslapių lentelės gali būti ir virtualioje atmintyje.

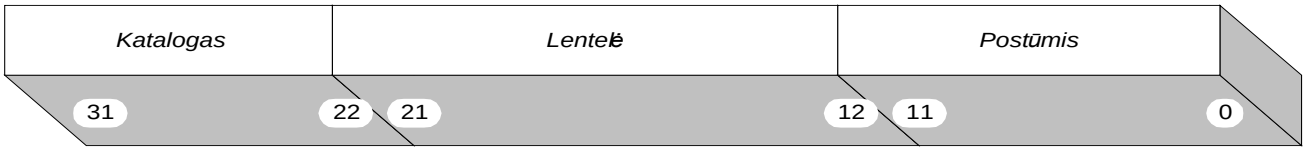
7. 24 pav., d ir e parodyti puslapių katalogo sekcijos ir puslapių lentelės formatai. Atitinkami laukai aprašyti 6 lentelės tęsinyje. Reikia pažymėti, kad priėjimo kontrolės mechanizmas gali būti sudarytas vieno puslapio arba puslapių grupės pagrindu.

Adresų transliacijai Pentium® procesorius taip pat gali taikyti specialų {lookaside} buferį. Šiame buferyje gali būti patalpinti 32 puslapio lentelės įėjimai. Kiekvieną kartą kai puslapių katalogas keičiamas, minėtas buferis išvalomas.

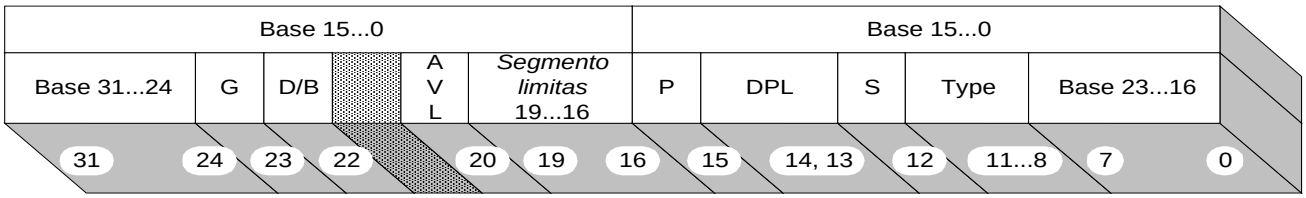


TI - Table Indicator *(Lentelės indikatorius)*
RPL - Requestor Privilege Level *(Būtinas Privilegijos lygis)*

a) Segmento selektorius

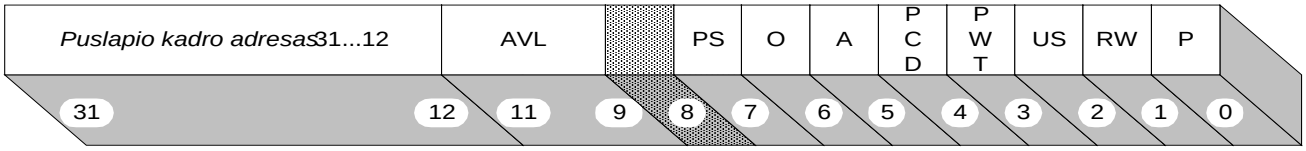


b) Linijinis adresas



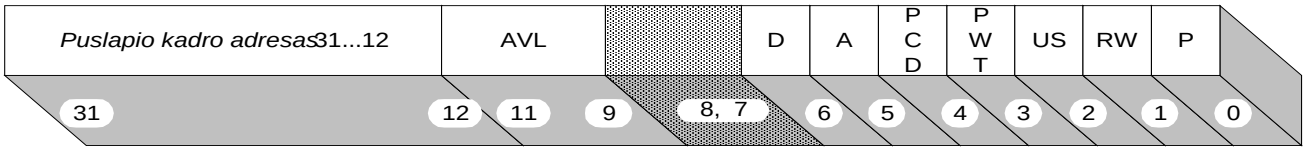
AVL - Available *(Prieinamas sisteminiam programuotojui)*
Base - Segment Base Address *(Segmento bazinis adresas)*
D/B - Default Operation Size *(Operacijos dydis pagal nutylėjimą)*
DPL - Descriptor Privilege Level *(Diskriptoriiaus privilegijos lygis)*
G - Granularity *(Grūdingumas)*
P - Segment Present *(Segmento pateikimas)*
Type - Segment Type *(Segmento tipas)*
S - Discriptor Type *(Diskriptoriaus tipas)*

c) Segmento diskriptorius (segmento lentelės sekcija)



AVL - Available *(Prieinamas sisteminiam programuotojui)*
PS - Page Size *(Puslapio dydis)*
A - Accessed *(Prieinamas)*
PCD - Cache Disable *(Puslapis nėra kešuojams)*
PWT - Write Through *(Rašyti ištisai ("kiaurai") leistina)*
US - User/Supervisor *(Vartotojas/Supervizorius)*
RW - Read/Write *(Skaityti/Rašyti)*
P - Present *(Puslapio pateikimas)*

d) Puslapio katalogo sekcija



D - Dirty *(Išnaudotas)*

e) Puslapio lentelės sekcija

7. 24 pav. Pentium® procesoriams atminties sutvarkymo formatai

7. 25 pav. parodyta segmentavimo ir dalinimo į puslapius mechanizmų kombinacija. Aiškumo dėlei lookaside buferis ir atminties kešavimo mechanizmai neparodyti.

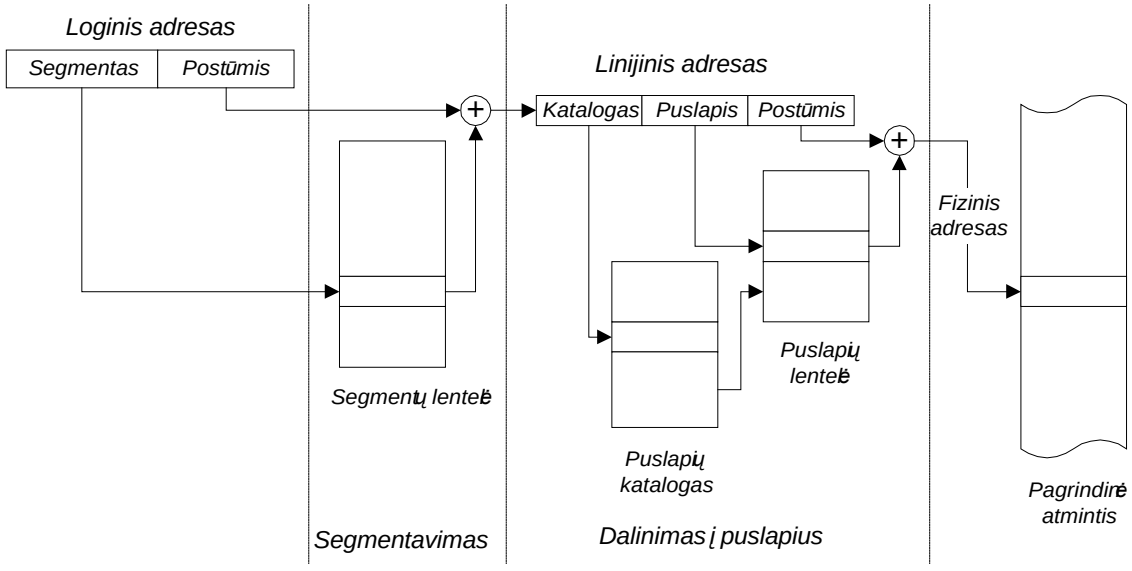
7. 6 lentelės tęsinys

Puslapių katalogo ir puslapio lentelės sekcijos	
Accessed bit (A)	**Priėjimo bitas.** Šį bitą procesorius nustato į 1 abiejuose puslapių lentelių lygmenyse jeigu atitinkamame puslapyje vykdoma skaitymo arba rašymo operacija
Dirty bit (D)	**Panaudojimo bitas.** Šį bitą procesorius nustato į 1 kai į atitinkamą puslapį vykdoma rašymo operacija.
Page Frame Address	Jeigu pristatymo bitas {Present bit (P)} yra 1, puslapis atmintyje surandamas pagal savo fizinį adresą. Šiuo atveju, kadangi puslapio dydis apribotas 4 Kbaitais, jaunesnieji adreso 12 bitai yra 0 ir tik 20 vyresniųjų bitų yra sekcijoje (įėjime). Puslapių kataloge adresas yra toks pats, kaip ir puslapių lentelėje
Page Cache Disable bit (PCD)	**Atminties puslapis nekešuojamas.** Rodo kurio puslapio duomenys yra kešuojami, o kurio ne
Page Size bit (PS)	**Puslapio dydis.** Rodo koks yra esamo puslapio dydis – 4 Kbaitai ar 4 Mbaitai
Page Write Through bit (PWT)	**Puslapio rašymo "kiaurai" (tiesioginis rašymas) bitas.** Rodo koks rašymo metodas naudojamas kešuojamame puslapyje
Present bit (P)	**Pristatymo bitas.** Rodo kas dabar patalpinta pagrindinėje atmintyje – ar puslapio lentelė ar pats puslapis
Read/Write bit (RW)	**Skaitymo/rašymo bitas.** Vartotojo lygmens puslapyje rodo ar puslapis turi tik skaitymo priėjimų; arba vartotojo lygmens programoje – ar puslapis turi ir skaitymo ir rašymo priėjimą.
User/Supervisor bit (US)	**Vartotojo/Supervizoriaus bitas.** Rodo ar į esamą puslapį gali kreiptis tik operacinė sistema (supervizoriaus lygmuo) ar ir operacinė sistema ir taikomosios programos (vartotojo lygmuo)

Pentium® procesoriuje yra inovacija, kurios nebuvo i80386 arba i80486 procesoriuose – dviejų puslapio dydžių nustatymas. Jeigu PSE {Page Size Extension} bitas ketvirtajame valdymo registre nustatytas į 1 būseną, atminties dalinimo į puslapius įrenginys suteiks operacinės sistemos programuotojui galimybę nurodinėti skirtingus puslapio dydžius – arba 4 Kbaitai, arba 4 Mbaitai.

Kai puslapių dydis 4 Mbaitai yra tik vienas puslapių lentelės peržiūros operacijos lygmuo. Šiuo atveju, kai aparatiniame lygmenyje vyksta kreipimasis (priėjimas) į puslapių katalogą, šio katalogo sekcijos PS bitas nustatytas į 1 (7. 24 pav., d). Tada bitai nuo 9 iki 21 ignoruojami, o bitai nuo 22-o iki 31-o rodo 4 Mbaitų puslapio bazinį adresą atmintyje. Tokiu būdu, gaunasi tik viena puslapių lentelė.

4 Mbaitų dydžio puslapiai mažina atminties išlaidas pačiam atminties tvarkymo mechanizmui (kai pagrindinė atmintis pakankamai didelė). Jeigu pagrindinė atmintis būtų maksimalaus – 4 Gbaitų dydžio, 4 Kbaitų puslapiams esant, puslapių lentelėms sudaryti reikėtų apie 4 Mbaitus šios atminties. Su 4 Mbaitų puslapiais, atminties tvarkymo mechanizmui pakaktų vienos 4 KB dydžio puslapių lentelės.



7. 25 pav. Atminties adreso transliavimo mechanizmas Pentium® procesoriuje

7.5. Atminties tvarkymas PowerPC mikroprocesoriaus kompiuterizuotoje sistemoje

PowerPC šeimos mikroprocesoriai aprūpinti labai išvystytu atminties adresavimo mechanizmu.

32 bitų architektūros mikroprocesoriuose realizuota dalinimo į puslapius schema su paprastu segmentavimo mechanizmu, o 64-ių bitų architektūros mikroprocesoriuose – dalinimo į puslapius schema su efektyvesniu segmentavimo mechanizmu. Be to, ir 32-jų bitų ir 64-ių bitų skaičiavimo mašinose yra dar ir kitas schemotechninis adresavimo mechanizmas – vadinamasis adresų blokų transliavimas. Lakoniškumo dėlei, galima sakyti, kad bloko adresavimo schema buvo sukurta tam, kad įveikti vieną dalinimo į puslapius mechanizmo trūkumą. Dalinimo į puslapius atveju, vykdomoji programa gali dažnai kreiptis į didelį puslapių skaičių. Pavyzdžiui, taip gali daryti programos, kurios naudoja operacinių sistemų lenteles arba grafinių kadrų buferius {graphics frame buffers}. To rezultate, dažnai naudojami atminties puslapiai bus pastoviai iškviečiami ir sugražinami atgal. Adresavimas blokais mikroprocesorių įgalina aprėpti keturis didelius atminties blokus su instrukcijomis ir keturis didelius blokus su duomenimis taip, kad puslapių "sklaidymas" bus užblokuotas.

Blokų adresavimo šiame paragrafe nenagrinėsime. Susikoncentruosime ties atminties dalinio į puslapius ir segmentus mechanizmais PowerPC šeimos mikroprocesoriuose.

7.5.1.Atminties tvarkymas 32 bitų PowerPC mikroprocesoriuje

PowerPC mikroprocesorius gali naudoti 32 bitų efektyvųjį adresą (7. 26 pav., a). Šiame adrese yra 16 bitų puslapio identifikatorius ir 12 bitų baito selektorius. Tokiu būdu, puslapių dydis yra $2^{12} = 4$ Kbaitai (4096 baitai), o viename segmente gali būti iki $2^{16} = 64$ K (65536) puslapių. Likusieji 4 bitai leidžia nurodyti vieną iš 16 segmentų registrą. Šių registrų turinys kontroliuojamas operacine sistema. Kiekviename segmentų registre yra kreipimosi kontrolės bitai ir 24 bitų identifikatorius, tokiu būdu, 32 bitų efektyvusis adresas atvaizduojamas į 52 bitų (12 + 16 + 24 = 52) virtualųjį adresą (7. 27 pav.).

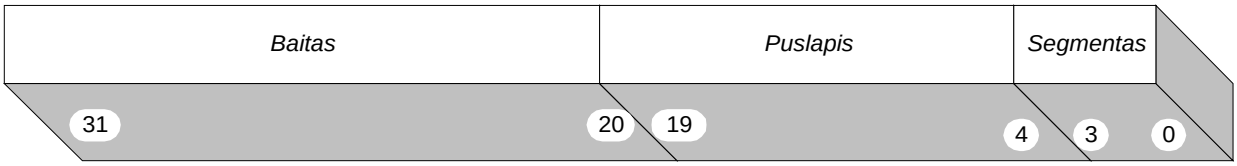
PowerPC mikroprocesorius gali naudoti vieną invertuotą puslapio lentelę. Indeksuojant puslapį virtualusis adresas vartojamas taip. Pirma skaičiuojamas randomizuotas kodas {hash code}:

$$H(0\ldots19) = SID(5\ldots23) \oplus VPN(0\ldots18).$$

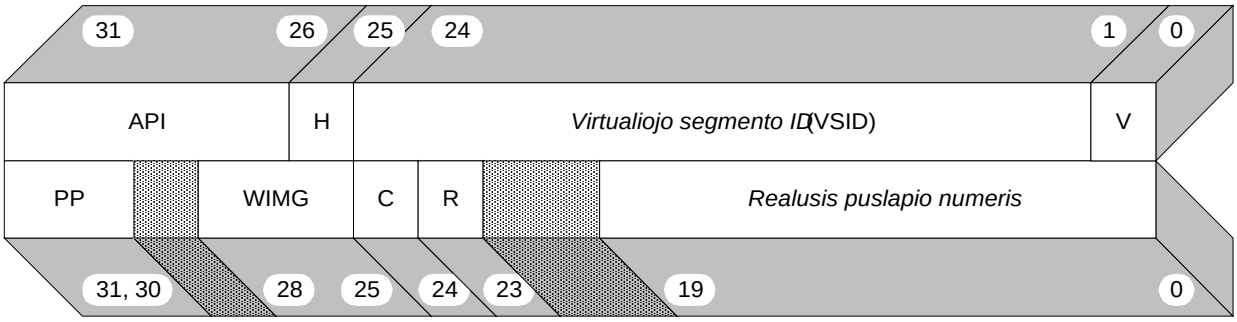
Virtualiajame adrese puslapio virtualiojo numerio pridedami trys binariniai nuliai taip, kad jis taptų 19 bitų numeriu (pats virtualusis numeris pasistumia į kairę, t.y. į reikšmingesnę pusę). Tada, taikant atitinkamiems sektoriaus identifikatoriaus {SID} ir virtualiojo puslapio numerio {VPN} bitams *išskirtinio arba (sudėties pagal modulį 2)* operaciją, gaunamas minėtas 19 bitų randomizuotas kodas. Puslapių lentelė sudaryta iš *n* grupių po 8 ląstelių kiekvienoje. Vienai iš grupių pasirinkti puslapių lentelėje naudojama nuo 10 iki 19 randomizuoto kodo bitų (priklausomai nuo lentelės dydžio). Tada, atminties tvarkymo schemotechninė įranga skenuoja visas aštuonias ląstelių grupes, tikrinant jų suderinamumą su virtualiuoju adresu.

Suderinamumo dėlei į kiekvieną puslapių lentelės ląstelę įvedami virtualiojo segmento identifikatorius {VSID} ir 6 kairieji virtualiojo puslapio numerio bitai, kurie vadinami *sutrumpintu* puslapio indeksu {API–Abbreviated Page Index} (mažiausiai 10 iš 16 virtualiojo puslapio numerio bitai visada taikomi randomizacijoje puslapio lentelės ląstelių grupei pasirinkti, todėl atitinkamam suderinimui su virtualiuoju adresu į puslapių lentelės ląstelę pakanka įvesti tik sutrumpintą virtualiojo puslapio numerį). Po to, kai aptinkama suderinta ląstelė, 32 bitų realiajam kreipimosi adresui suformuoti, apjungiami realiojo adreso 20 bitų puslapio numerio ir 12 bitų efektyviojo adreso.

Jeigu ne viena iš 8 ląstelių nesiderina su virtualiuoju adresu, tada randomizuotas kodas yra papildomas, kad gautųsi naujas puslapio lentelės indeksas, kuris yra analogiškoje pozicijoje kitame lentelės gale. Ši ląstelė vėl skenuojama suderinamumui tikrinti. Jeigu suderinamumas vėl neaptinkamas, generuojama puslapio klaidos pertrauktis {page fault interrupt}.

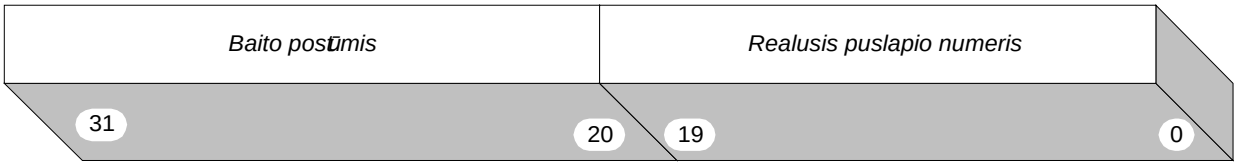


a) Efektyvusis adresas



V - Entry Valid Bit *(Įėjimo galiojimo bitas)*
H - Hash Function Identifier *(Randomizavimo funkcijos identifikatorius)*
API - Abbreviated Page Index *(Puslapio indeksas sutrumpa)*
R - Referenced Bit *(Referavimo - kreipimosi bitas)*
C - Changed Bit *(Pakeitimo bitas)*
WIMG - Cache and Storage Accesss Control Bit *(Kešavimo ir saugojimo kontrolės bitai)*
PP - Page Protection Bits *(Puslapio apsaugos bitai)*
▒ - *Rezervuotas*

b) Puslapio lentelės ląstelės



c) Realusis adresas

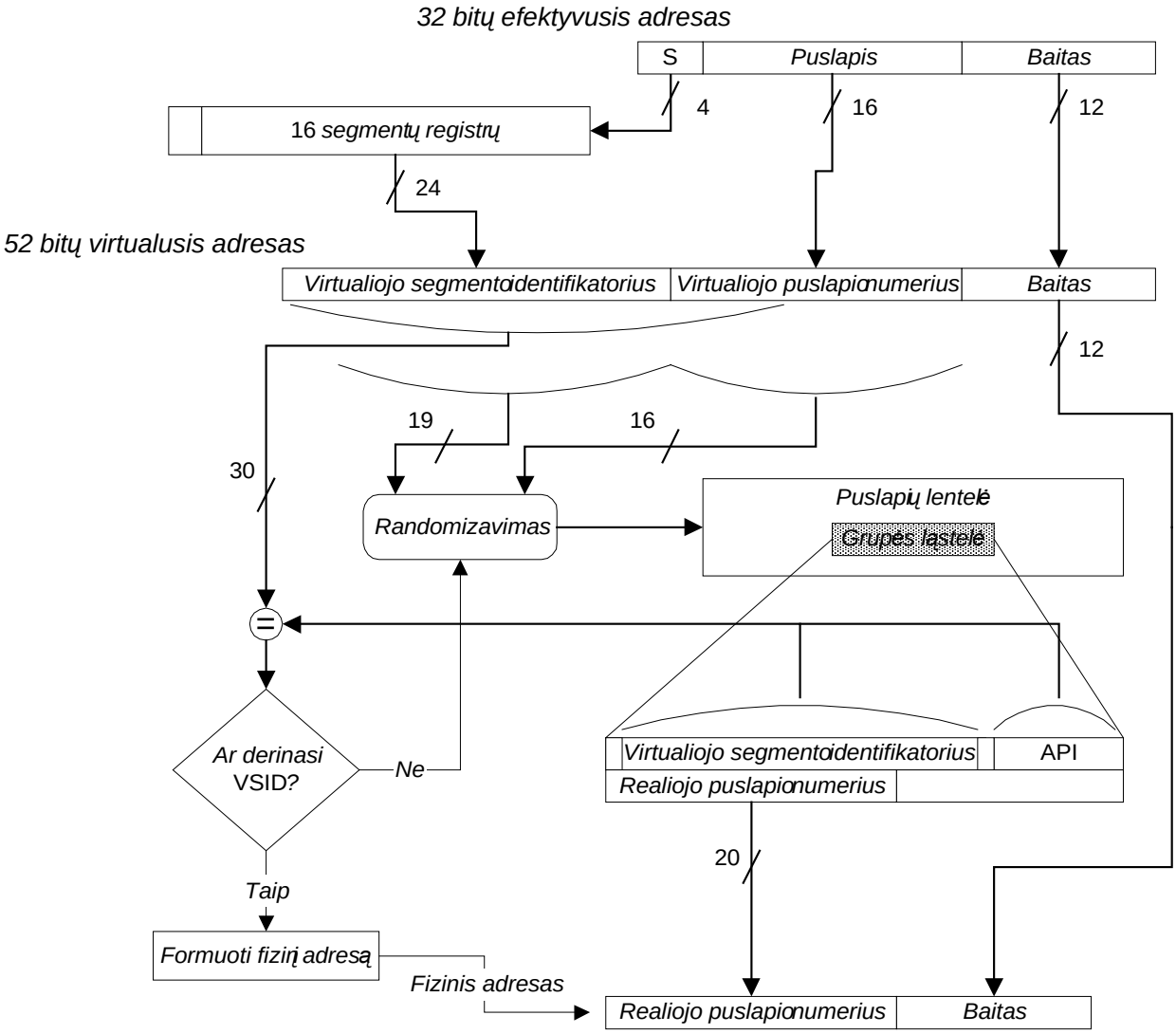
7. 26 pav. PowerPC® 32-jų bitų mikroprocesoriams atminties tvarkymo formatai

7. 27 pav. parodyta adresų transliavimo mechanizmo logika, 7. 26 pav. išdėstyti efektyviojo adreso, puslapių lentelės ląstelės ir realiojo adreso formatai. 7. 7 lentelėje nurodyti puslapių lentelės ląstelės parametrai.

7. 7 lentelė. PowerPC mikroprocesoriaus atminties tvarkymo parametrai

Parametras	Aprašas
Segmento lentelės ląstelė	
Effective Segment ID	**Efektyvusis segmento identifikatorius.** Rodo vieną iš 64 G (2^{36}) efektyviųjų segmentų; taikomas segmentų lentelėje ląstelei nustatyti
Entry Valig (V) bit	**Ląstelė galiojanti**
Segment Type (T) bit	**Segmento tipas.** Šis bitas rodo ką atitinka segmentas – ar atmintį ar Į/I
Supervisor Key (K_s)	**Supervizoriaus raktas.** Šis bitas taikomas su virtualiojo puslapio numeriu puslapio lentelėje ląstelei nurodyti
Puslapio lentelės ląstelė	
Entry Valid (V) bit	**Ląstelė galiojanti.** Šis bitas rodo ar duomenys esantys duotoje ląstelėje yra tikri
Hesh Function Identifier (H)	**Randomizavimo funkcijos identifikatorius.** Rodo kelintą kartą

	randomizuojama ląstelė – pirmą ar antrą
Abbreviated Page Index (API)	**Sutrumpintas puslapio indeksas.** Taikomas unikaliam virtualiojo adreso suderinimui {mach}
Referenced (R) bit	**Referavimo bitas.** Šis bitas procesoriaus nustatomas į 1 jeigu į atitinkamą atminties puslapį buvo vykdoma skaitymo arba rašymo operacija.
Changed (C) bit	**Pakeitimo bitas** procesoriaus nustatomas į 1jeigu į atitinkamą atminties puslapį buvo įvykdyta rašymo operacija.
WIMG bits	**Kešo ir saugojimo priėjimo kontrolės bitai.** *W* = 0 – taikomas "rašymo vėliau" {write-back} metodas; *W* = 1 – taikomas "rašymo iš karto" {write-through} metodas; *I* = 0 – kešavimas neuždraustas; *I* = 1 – kešavimas uždraustas; *M* = 0 – atmintis kolektyviškai {shared} nenaudojama; *M* = 1 – atmintis naudojama kolektyviškai; *G* = 0 – atmintis neapsaugoma; *I* = 1 – atmintis apsaugoma {guarded};
Page Protection (PP) bits	**Puslapio apsauga.** Priėjimo kontrolės bitai taikomi kartu su segmento registro K bitais arba segmentų lentelės ląstele priėjimo teisėms nustatyti.



7. 27 pav. PowerPC® mikroprocesoriams atminties tvarkymo formatai

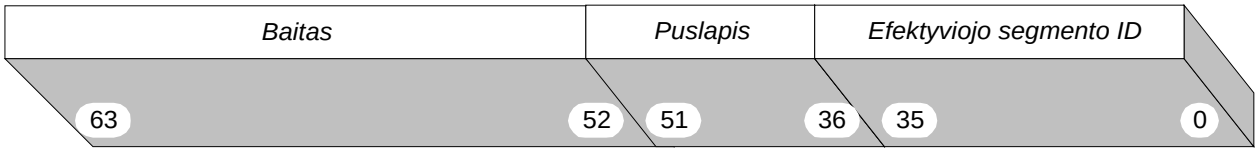
7.5.2. Atminties tvarkymas 64 bitų PowerPC mikroprocesoriuje

64 bitų atminties tvarkymo schema sukurta taip, kad būtų užtikrintas suderinamumas ir su 32 bitų atminties vadybos schema. Iš esmės, visi efektyvieji adresai, bendrosios paskirties registrai ir šakojimosi adresų registrai išplėsti į kairiąją pusę iki 64 bitų. Atminties atvaizdavimo reikalavimams užtikrinti įprastas segmentų registrų taikymas yra pakeistas randamizuotų segmentų lentelės struktūra, panašia į prieš tai aptartą puslapių lentelės struktūrą. Segmentų lentelėje (jos ląstelių grupėse) yra po aštuonias segmentų lentelės ląstelių; kiekvienoje ląstelėje yra 36 bitų efektyvusis segmento identifikatorius, 52 bitų virtualusis segmento identifikatorius, o taip pat keli kreipimosi apsaugos ir Į/I erdvės selektoriaus bitai (7. 28 pav.), kurie taip pat yra ir 32 bitų atminties vadybos schemoje.

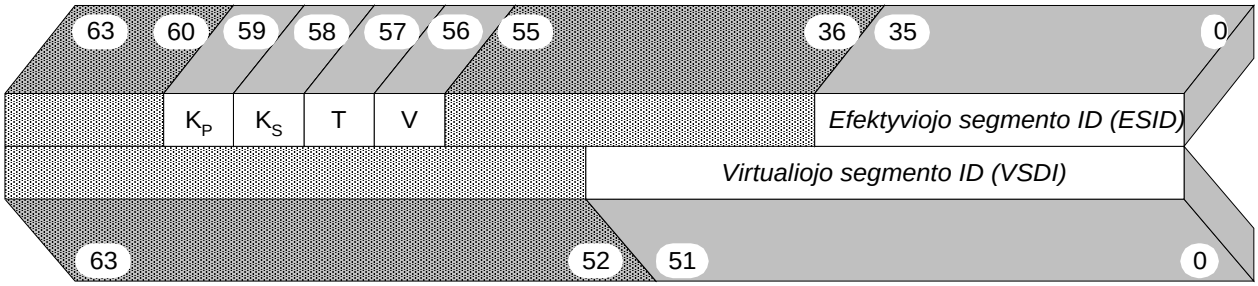
7. 29 pav. parodytas 64 bitų efektyviojo adreso atvaizdavimas į 80 bitų virtualųjį adresą. Kaip ir 32 bitų vadybos schemoje efektyviajame adrese yra 12 bitų baito selektorius ir 16 bitų puslapio numeris. Likusieji 36 bitai formuoja efektyvųjį segmento identifikatorių. Kadangi realizuoti 2^{36} segmentų registrą neįmanoma taikoma vadinamoji supainiotoji {hashed} segmentų lentelė.

Segmentų lentelėje yra 256 ląstelės (“stambiosios”), kurių kiekvienoje yra 32 grupės sudarytos iš 8 ląstelių (“smulkiosios”). Vienai iš 32 ląstelių grupių pasirinkta atminties tvarkumo schemotechninė įranga taiko efektyviojo segmento identifikatoriaus 5 jaunesniuosius bitus. Tuo atveju, jeigu grupėse lyginant segmento identifikatorius su duotuoju aptinkamas suderinamumas {mach}, formuojamas 80 bitų virtualusis adresas. Pastarasis susideda iš 52-jų bitų virtualiojo segmento identifikatoriaus apjungto su efektyviojo adreso puslapio ir baito numerio bitais. Priešingu atveju, jeigu segmentų identifikatorių suderinamumas neaptinkamas – 5 indekso bitai pridedami prie indekso, kuris jau yra lentelėje kitam kartui. Jeigu suderinamumas vėl nebus aptiktas, generuojama puslapio praradimo pertrauktis {page fault interrupt}.

7.6. Klausimai savikontrolei



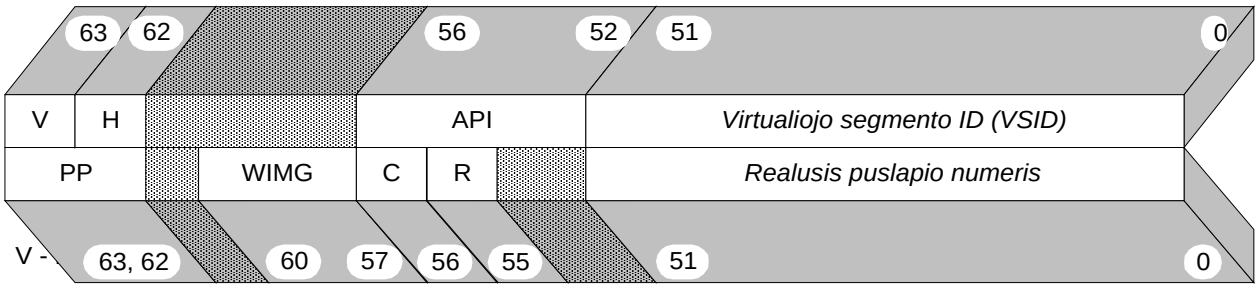
a) Efektyvusis adresas



V - Entry Valid Bit *(Įėjimo galiojimo bitas)*
T - Segment Type *(Segmento tipas)*
K_S - Supervisor Key *(Supervizoriaus raktas)*
K_P - Problem State Storage Key *(Kliutiesstadijos saugojimo raktas)*

- *Rezervuotas*

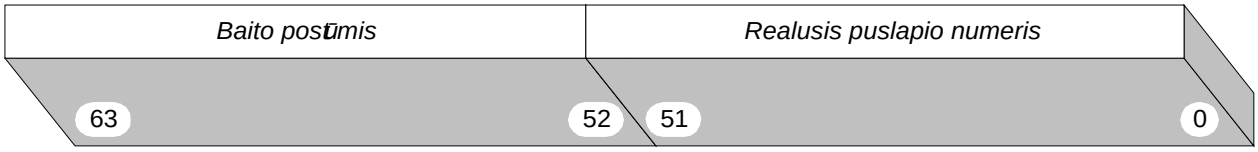
b) Segmento lentelės ląstelės



V - Entry Valid Bit *(Įėjimo galiojimo bitas)*
H - Hash Function Identifier *(Randomizavimo funkcijos identifikatorius)*
API - Abbreviated Page Index *(Puslapio indekso santrumpa)*
R - Referenced Bit *(Referavimo - kreipimosi bitas)*

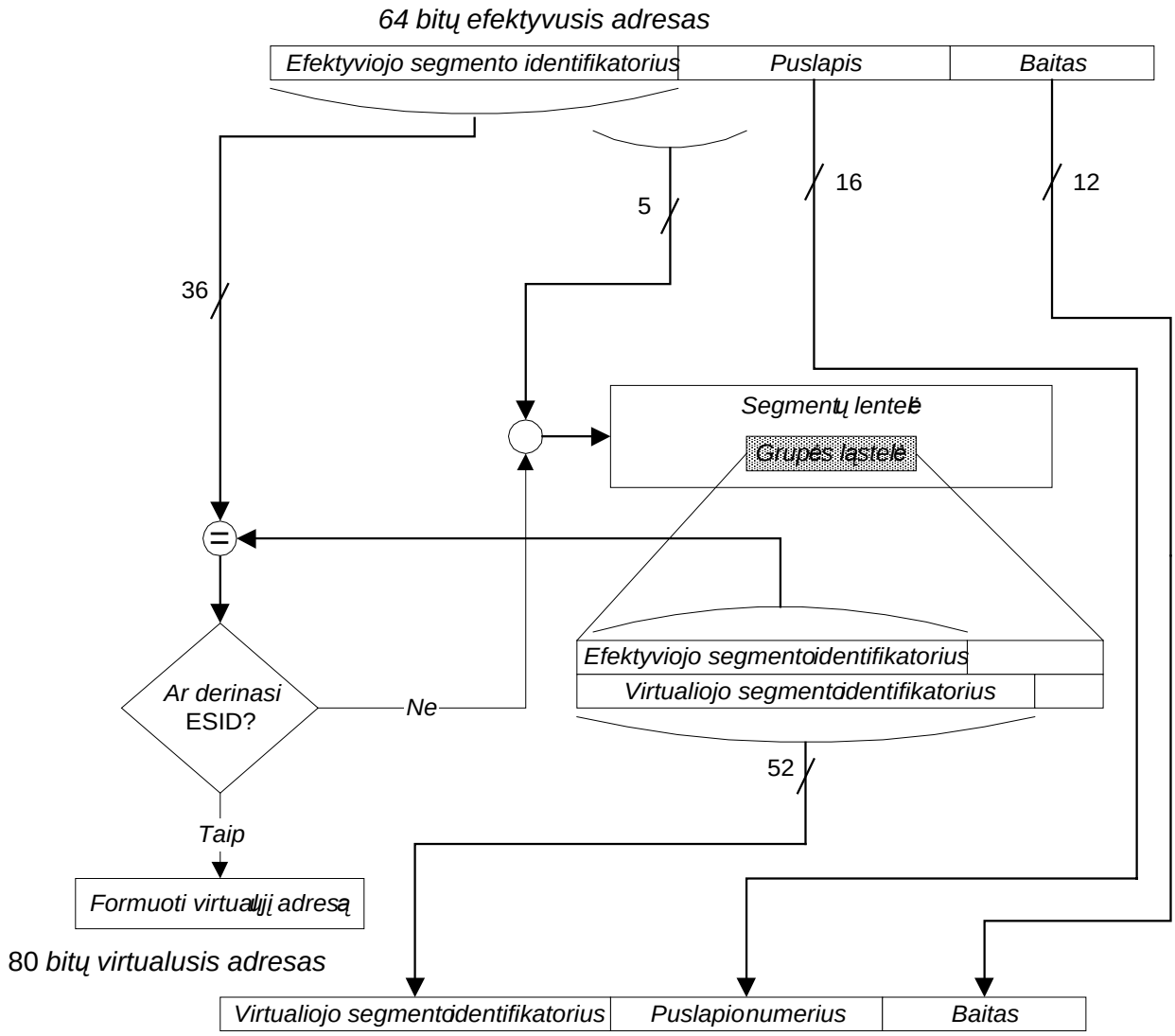
WIMG - Cache and Storage Accesss Contro Bits *(Kešavimo ir saugojimo kontrolės bitai)*
C - Changed Bit *(Pakeitimo bitas)*
PP - Page Protection Bits *(Puslapio apsaugos bitai)*

c) Puslapio lentelės ląstelės



d) Realusis adresas

7. 28 pav. 64 bitų PowerPC mikroprocesoriaus atminties tvarkymo formatai.



7. 29 pav. Efektyviojo 64 bitų adreso transliavimas į virtualųjį PowerPC mikroprocesoriuje.