VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Magistro darbas

Mokomoji operacinė sistema

Atliko: II mag. kurso, 2 grupės studentas
Giedrius Šiaulys (parašas)

Darbo vadovas:
dr. Antanas Mitašiūnas (parašas)

Vilnius
2003

Įžanga

Šio darbo tikslas – parengti mokomosios operacinės sistemos projektą, kurį būtų galima pateikti kaip pavyzdį OS kurso pratybų metu. Projekto rašymo metu buvo nagrinėjami kiti panašių tikslų siekiantys darbai, rasti internete ar spausdinti knygose. Buvo išnagrinėti šių darbų trūkumai ir privalumai, ir gautos išvados panaudotos magistro darbe.

Dabar trumpai apie tai, kodėl šiam tikslui įgyvendinti buvo pasirinktas operacinės sistemos modelis, o ne, sakykime, kokia nors realia (egzistuojančia) aparatūra besiremianti operacinė sistema?

Iš esmės operacinė sistema atlieka dvi funkcijas - paslepia aparatūrą ir skirsto resursus. Tačiau slėpdama aparatūrą operacinė sistema neįučia labai prisiriša prie jos. Taigi, nagrinėjant operacinę sistemą, nori nenori teks nagrinėti ir mašiną, kurią padengia mūsų operacinė sistema. Tokiu būdu mums būtų geriau, kad ji būtų paprasta. Realios aparatūros yra pakankamai sudėtingos, todėl operacinė sistema taip pat bus sudėtinga. Be to, egzistuoja daug kompiuterių architektūrų, kurių kiekviena turi savo privalumus, trūkumus, specifiką ir sudėtį. Nagrinėjant aparatūros modelį, mes galėsime išvengti visų techninių detalių ir žinių, be kurių būtų neįmanoma dirbti. Be to, mes galėsime apibrėžti tokį aparatūros modelį, kokį norėsime – paprastą ir aiškų. Tokį, kuris lengvai leistų įsisavinti operacinės sistemos darbą.

Mes norime, kad operacinę sistemą būtų lengva stebėti, galėtume sekti jos būseną, esant reikalui sustabdyti, vėliau vėl paleisti. Taigi, mes norime tapti stebėtojais. Egzistuojančių operacinių sistemų paskirtis yra kita – jos skirtos aptarnauti vartotoją. Be to, jos yra gana sudėtingos ir atlieka daugybę papildomų funkcijų, kurios mums tik trukdytų. Dėl to jos stebėjimams netinka.

Šio darbo rezultatas turi būti multiprograminės operacinės sistemos modelio projektas. Pirmieji žingsniai į operacinės sistemos modelį bus realios mašinos modelio (sutrumpintai - reali mašina) apibrėžimas. Vėliau, remdamiesi turima mašina, kursime operacinės sistemos modelį ir žiūrėsime, kaip jis veikia. Tačiau iš pradžių apie multiprogramines operacines sistemas.

Multiprograminės operacinės sistemos modelis

1 Sąvokos

Multiprograminės operacinės sistemos – viena operacinių sistemų rūšių. Šio tipo operacinės sistemos užtikrina kelių užduočių lygiagretų vykdymą. Multiprograminės operacinės sistemos (MOS) privalumai yra akivaizdūs. Vartotojui vienu metu paprastai neužtenka vienos aktyvios programos. Tai ypač akivaizdu,kai programa vykdo ilgus skaičiavimus ir tik kartais prašo įvesti duomenis. Tuo metu vartotojas yra priverstas stebėti užduoties vykdymą ir tampa pasyviu.

Paprastiems vartotojams MOS yra gerokai patrauklesnė. Tačiau ar MOS yra pakankamai efektyvi? Sakykime, kompiuteryje yra vienas procesorius. Taigi, vienu metu realiai yra vykdoma tik viena užduotis, kitos pasiruošusios vykdymui laukia savo eilės. Atrodo, nieko efektyvaus čia nėra. Tačiau, sakykime, užduotis pareikalavo perkopijuoti duomenis iš vartotojo atminties į išorinę. Tegu mūsų kompiuteryje yra įrenginys, kuris moka šį darbą atlikti be pagrindinio procesoriaus pagalbos. Taigi, užduotis, reikalaudama duomenų apsikeitimo, pradėjo laukti darbo pabaigos. Tuo metu MOS blokuoja šią užduotį, ima kitą, paruoštą vykdyti užduotį, ir, atėmusi procesorių iš pirmosios, atiduoda jį antrajai užduočiai. Tokiu būdu dabar vykdoma antra užduotis. Pirmoji šiaip ar taip negali būti vykdoma, nes jai reikalingi duomenys dar neperkopijuoti. Efektyvu. Aišku, kaip matyti iš pavyzdžio, šis efektyvumas yra sąlyginis, bet tai nesukliudė MOS tapti populiariausiu šio laikmečio operacinių sistemų tipu.

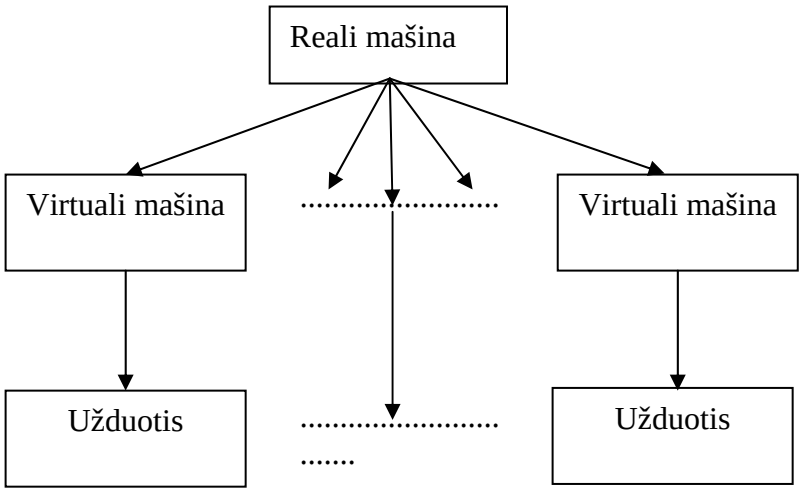
Pabaigai keletas sąvokų. Užduotimi toliau vadinsime programą su vykdymo parametrais ir startiniais duomenimis. Vykdoma užduoties programa tampa procesu. Pati programa yra tik kodas, skirtas procesoriui vykdyti.

2 Reali ir virtuali mašinos

Reali mašina - tai kompiuteris. Užduotis kaip minėjome susideda iš programos, startinių duomenų ir vykdymo parametrų. Rašyti programą realiai mašinai būtų sudėtinga ir nepatogu. Todėl vienas iš operacinės sistemos tikslų yra paslėpti realią mašiną ir pateikti mums virtualią. Užduoties programą vykdo ne reali, o virtuali mašina. Virtuali mašina - tai tarsi virtuali realios mašinos kopija. Šiame apibrėžime yra du žodžiai kurie reikalauja paaiškinimo - "virtuali" ir "kopija".

Kodėl virtuali? Virtuali reiškia netikra. Mes tarsi surenkame reikalingas realios mašinos komponentes, tokias kaip procesorius, atmintis, įvedimo/išvedimo įrenginiai, suteikiame jiems paprastesnę nei reali vartotojo sąsają ir visa tai pavadiname virtualia mašina.

Kodėl kopija? Tai nėra tiksli realios mašinos kopija. Vienas iš virtualios mašinos (VM) privalumų yra programų rašymo palengvinimas, todėl realios mašinos komponentės, turinčios sudėtingą arba nepatogią vartotojo sąsają, virtualioje mašinoje yra supaprastintos. Virtuali mašina dirba su operacinės sistemos pateiktais virtualiais resursais, kurie daugelį savybių perima iš savo realių analogų ir pateikia kur kas paprastesnę vartotojo sąsają. Tai lengvina programavimą.



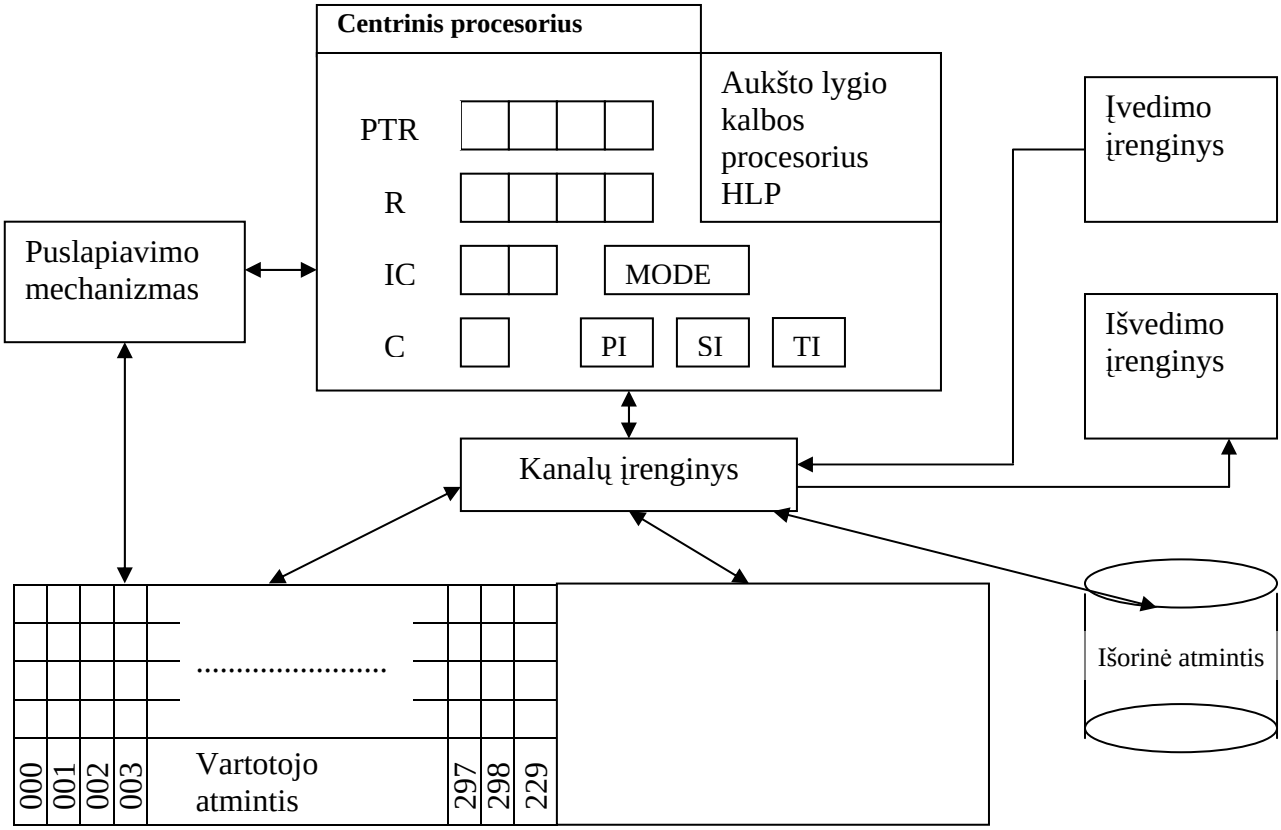
1 Pav. Virtualios mašinos pateikimas užduotims multiprograminės operacinės sistemos atveju.

Kiekviena užduotis turi savo virtualią mašiną, kurios, iš tikrųjų, ir konkuruoja dėl realaus procesoriaus. Vienas esminių virtualios mašinos privalumų yra tas, kad užduotis, kurią vykdo virtuali mašina, elgiasi lyg būtų vienintelė užduotis visoje mašinoje. Tai yra didelė parama programuotojui. Dabar jam tenka rūpintis tik pačios programos rašymu. Visa tai atsispindi 1 paveiksliuke.

2.1 Realios ir virtualios mašinos modeliai

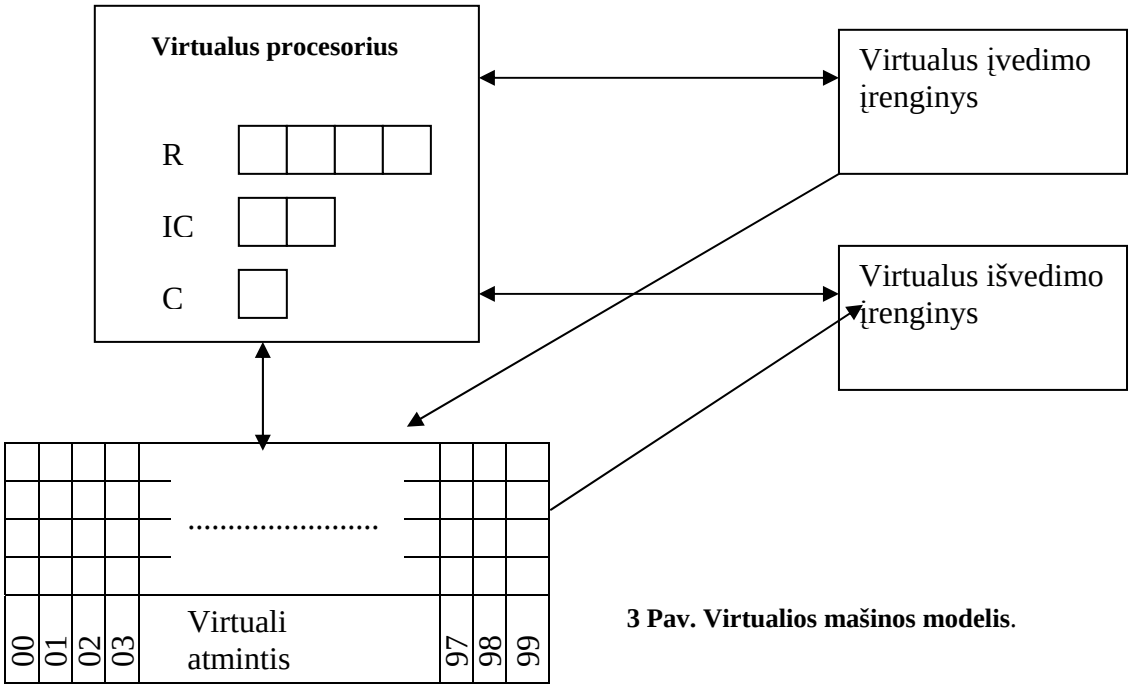
Kaip jau buvo minėta, reali mašina - tai kompiuteris. Realios mašinos modelis (toliau reali mašina) - tai virtualus kompiuteris. Kodėl mums geriau naudoti modelį nei egzistuojančią architektūrą, mes jau apžvelgėme anksčiau. Kokia gi bus mūsų reali mašina?

Iš esmės tai yra laisvas pasirinkimas. Kompiuterių architektūros aukščiausiame lygyje iš esmės yra panašios. T.y. vyrauja tokios sudėtinės dalys kaip procesorius , atmintis , įvedimo , išvedimo įrenginiai. Šio darbo pagrindinis prioritetas yra OS veikimo stebėjimas ir valdymas. Taigi aparatūrai reikalavimai nėra dideli, svarbu kad ji būtų logiška ir patogi mūsų mokomajai OS. Šiame projekte bus pateikta viena iš galimų realios mašinos architektūrų, kuri iš esmės remiasi Šou modeliu. Schematiškai Šou reali mašina pateikta paveiksliuke Nr. 2:



2 Pav. Realios mašinos modelis.

Virtualios mašinos modelis kiek primins realios mašinos modelį. Jis pateiktas trečiame paveiksliuke:



3 Pav. Virtualios mašinos modelis.

Dabar kiek smulkiau aptarsime kiekvieną realios bei virtualios mašinos komponentę ir jų skirtumus.

2.2 Realios mašinos centrinis procesorius

Procesoriaus paskirtis - skaityti komandą iš atminties ir ją vykdyti (interpretuoti). Procesorius gali dirbti dviem režimais – supervizoriaus arba vartotojo. Supervizoriaus režime komandos, iš supervizorinės atminties, yra apdorojamos betarpiškai aukšto lygio kalbos procesoriaus HLP. HLP – bet kuris aukšto lygio kalbos procesorius (programavimo kalbos). Vartotojo režime HLP vykdo užduoties programą. Šiuo atveju HLP imituoja virtualios mašinos procesorių. Dabar apžvelgsime procesoriaus registrus:

- PTR – 4 baitų puslapių lentelės registras.
- R – 4 baitų bendro naudojimo registras.
- IC – 2 baitų virtualios mašinos programos skaitiklis
- C – 1 baito loginis (reikšmės – true “T” arba false “F”) trigeris.
- MODE – registras, kurio reikšmė nusako procesoriaus darbo režimą (vartotojas, supervizorius)
- PI – programinių pertraukimų registras
- SI – supervizorinių pertraukimų registras
- TI – taimerio registras

Registrų paskirtis detaliau išaiškės kiek vėliau, o dabar peržvelgsime virtualios mašinos procesorių.

2.3 Virtualios mašinos centrinis procesorius

Kaip buvo matyti iš virtualios mašinos modelio schemos, centrinis virtualus procesorius yra gerokai paprastesnis. Virtualios mašinos procesoriaus paskirtis - vykdyti programą, kuri yra virtualioje atmintyje. Kiekvienas procesas turi savo virtualų centrinį procesorių, tačiau modelyje sisteminių procesų programas vykdys aukšto lygio kalbos procesorius. Taigi realiai mūsų projekte virtualius procesorius turės tik procesai – virtualios mašinos. Virtualus procesorius turi tris pagrindinius registrus:

- R – 4 baitų bendro naudojimo registras.
- IC – 2 baitų virtualios mašinos programos skaitliklis
- C – 1 baito loginis (reikšmės – true “T” arba false “F”) trigeris.

Prieš pradėdami skyrelį apie atmintis, apžvelgsime virtualaus procesoriaus komandas.

2.4 Virtualios mašinos procesoriaus komandos

Atmintyje esantis 4 baitų žodis gali būti traktuojamas kaip duomenys arba komanda. Pirmieji du baitai (vyresnieji) laikomi operacijų kodu, likę naudojami kaip operandai, paprastai tai bus atminties adresas. Mikroprogramos, interpretuojančios virtualaus procesoriaus komandas bus vykdomos HLP. Pačios mikroprogramos turi būti tik skaitymui pažymėtoje supervizorinės atminties dalyje.

Dabar pateiksime mūsų virtualios mašinos procesoriaus komandas su paaiškinimais:

- **LRxy**. Atminties ląstelės, kurios adresas x*10 + y turinio kopijavimas į registrą R. Trumpiau: R:= [x*10 + y]
- **SRxy**. Registro R reikšmė įrašoma į atminties ląstelę, kurios adresas x*10 + y. Trumpiau [x*10 + y]:= R

- **ADxy** Prie registro R reikšmės pridedama atminties ląstelės, kurios adresas x*10 + y reikšmė. Trumpiau: R:= R + [x*10 + y]
- **CRxy.** Jei R = [x*10 + y], tai registrui C priskiriama reikšmė “T”, kitu atveju C:= “F”
- **BTxy.** Jei registro C reikšmė lygi “T”, tai registrui IC priskiriama reikšmė [x*10 + y].
- **GDxy.** Iš įvedimo srauto perskaito 10 žodžių ir įrašo juos į ląsteles [x *10 + i], kur i = 0.. 9. Operandas “y” reikšmės neturi. Formaliau tai galima užrašyti taip:

for i:= 0 to 9 **do** [x *10 + i]:= buffer[i]. Buffer – dešimties žodžių masyvas, kur yra laikomi įvedimo srauto rezultatai.

- **PDxy** Išsiunčia išvedimui 10 žodžių srautą iš atminties ląstelių [x *10 + i], kur i = 0.. 9. Operandas y reikšmės neturi.
- **HALT –** Vartotojo programos vykdymo pabaiga.

Kaip jau buvo kalbėta, visos komandos yra 4 baitų ilgio. Jei šių komandų neužtenka kokiam nors uždaviniui išspręsti, žinoma galima pridėti keletą naujų, laikantis taisyklingo komandų formato. Tai yra įmanoma mūsų modelyje, nes komandų interpretatorių mums teks rašytis patiems.

Komandos GD ir PD kaip matyti iš jų apibrėžimo dirba tik su 40 simbolių (baitų). Komanda HALT žymi virtualios mašinos darbo pabaigą.

Naudojantis šiomis komandomis ir bus rašomos užduočių programos, kurias vykdys virtualios mašinos. Demonstracinė programa bus pateikta kiek vėliau, kai susipažinsime su užduoties struktūra. Aišku programos, kurias jūs parašysite, gali veikti nekorektiškai. Svarbiausia, kad vykdant komandas ir sutikus neapibrėžtą situaciją nebūtų nutrauktas operacinės sistemos darbas.

2.5 Realios mašinos atmintys

Atmintis – įrenginys informacijai saugoti. Mūsų reali mašina turi trijų rūšių atmintis: supervizorinę, vartotojo ir išorinę.

Dabar trumpai apibūdinsime **supervizorinę** atmintį. Jau buvo minėta, kad supervizorinė atmintis yra skirta pačios operacinės sistemos poreikiams. Supervizorinėje atmintyje laikomi sisteminiai procesai, sisteminiai kintamieji, resursai, mikroprogramos, interpretuojančios virtualaus procesoriaus komandas.Tačiau taip yra tik realiose operacinėse sistemose. Modelio atveju supervizorinė atmintis tampa tik sąvoka, kuri realiai nėra realizuojama. Supervizorinėje atmintyje turėjusiais būti komandas ir resursus valdys aukšto lygio kalbos procesorius HLP. Supervizorinės atminties verta atsisakyti vardan paties modelio paprastumo, o tai buvo vienas iš mūsų tikslų. Vis dėlto ši sąvoka, nors realiai nerealizuota ir toliau bus vartojama.

Vartotojo atmintis skirta virtualių mašinų atmintims bei puslapių lentelėms laikyti. Mes apibrėšime vartotojo atmintį taip: lentelės dydis – 300 žodžių po 4 baitus. 10 žodžių laikysime bloku (takeliu). Taigi vartotojo atmintis lygi 30–čiai blokų, sunumeruotų nuo 0 iki 29, arba 300 žodžių, sunumeruotų nuo 0 iki 299.

Takelis\Žodis										
00										
01										
02										
……**Vartotojo atmintis** nuo 00 iki 29……...										
29										
29										
30										
31										
32										
Supervizorinė atmintis. Laikomos sisteminės programos, kintamieji, ir kiti OS naudojimui skirti resursai										
MAX – 1										
MAX										

3 lentelė. Atminties padalijimas į vartotojo ir supervizorinę.

Vartotojo ir supervizorinės atmintys dalijasi realios mašinos atmintį. Kaip matome iš apibrėžimo, vartotojo atmintis bus atminties pradžioje. Visą likusią dalį užims supervizorinė atmintis, kuri, kaip jau buvo minėta, realiai nebus realizuota.

Visa tai pavaizduota trečioje lentelėje.

Išorinė atmintis bus realizuota failu kietame diske. Tarkime laikysime kad faile yra 500 blokų arba 5000 žodžių. Schematiškai išorinę atmintį galima pavaizduoti analogiškai atminčiai, tik ji nėra dalijama į kitas atmintis. Darbą su išorine atmintimi atliks HLP.

Procesorius darbą su atmintimis valdo naudodamas kanalų įrenginį. Apie kanalų įrenginį bus plačiau, o dabar apie virtualios mašinos atmintį.

2.6 Virtualios mašinos atmintis

Kiekvienai virtualiai mašinai yra skiriama 10 vartotojo atminties blokų. Tuose dešimtyje blokų (100 žodžių) turi tilpti užduoties programa. Kiekvienas virtualios atminties blokas turi virtualų ir realų adresą. Virtualiais adresais operuoja virtuali mašina, realiais – reali mašina. Ryšiai tarp virtualaus ir realaus adreso nusakomi puslapių lentelėmis. Tai bus nagrinėjama skyrelyje “**puslapiavimo mechanizmas”**, pavyzdžiui, virtuali atmintis gali atrodyti taip:

Blokas\Žodis	0	1	2	3	4	5	6	7	8	9
0	LR70	AD91	CR92	NT07	PD80	GD50	HALT	SR70	PD70	GO00
1	HALT									
2										
3										
4										
5										
6										
7	0000									
8	Susk	aici	uota							
9	0000	0001	0999							

4 lentelė. Virtualios atminties pavyzdys

Lentelėje galima pastebėti keletą nematytų komandų. Jos buvo realizuotos papildomai. Jų paaiškinimas:

- **NTxy**. Jei C <> “T tai IC:= xy
- **GOxy**. Besąlygiškai priskiriame registrui IC xy reikšmę. t.y. IC:= xy

Dabar kalbėsime apie ryšius tarp virtualaus ir realaus atminties adreso.

2.7 Puslapiavimo mechanizmas

Realios mašinos vartotojo atmintis siekia 30 takelių (arba blokų). Kiekvienai naujai sukurtai virtualiai mašinai reikia skirti 10 takelių iš tų 30. Jie gali būti parinkti bet kokiu būdu. Klausimas: kaip virtuali mašina gali sužinoti kokio nors jai priklausančio takelio realų adresą? Tam naudosime puslapiavimo mechanizmą.

Puslapiavimo mechanizmo esmė: sakykime, kuriama nauja virtuali mašina. Jai reikia dešimt takelių atminties. Mes parinkome takelius su numeriais: 1, 3, 5, 7, 9,11, 13, 15, 17, 19. Šiais takeliais naudosis virtuali mašina. Pati virtuali mašina šiuos takelius mato sunumeruotus nuo 0 iki 9. t.y. 1 takelis jai yra nulinis, 5 takelis jai yra antras, o 19 takelis – devintas. Kaip išlaikyti sąryšius tarp realių ir virtualių takelių adresų? Naudosime puslapių lentelę. Puslapių lentelė – tai vienas takelis (t.y. 10 žodžių). Kiekvieno žodžio eilės numeris atitiks virtualios mašinos takelio numerį, ir jame (žodyje) bus laikomas realus to takelio numeris. Pavyzdžiui, prieš tai pateikto pavyzdžio puslapių lentelė bus tokia:

Virtualus takelio numeris	0	1	2	3	4	5	6	7	8	9
Realus takelio numeris	1	3	5	7	9	11	13	15	17	19

5. Lentelė. Puslapių lentelės pavyzdys. Pastaba: Puslapių lentelė būtų tik antra (realus takelio numeris) eilutė.

Taigi, dabar virtuali mašina, norėdama sužinoti realų takelio adresą, kreipiasi į savo puslapių lentelę ir nuskaito reikšmę, esančią žodyje su takelio numeriu. Pvz. penkto virtualios mašinos takelio realus numeris yra 11. Dabar turi išaiškėti procesoriaus registro PTR prasmė. Juk puslapių lentelė taip pat yra atminties takelis. Ir šis takelis, be abejo, taip pat turi savo realų numerį. Todėl registro PTR reikšmė – puslapių lentelės adresas. Kiekviena virtuali mašina, prieš pradėdama darbą, nustatys šį registrą jai reikalinga reikšme.

Dabar galime apžvelgti patį registrą PTR. Tai 4 baitų registras. Simboliškai pažymėsime PTR reikšmę $a_0a_1a_2a_3$, kur a_i yra baitai. Baitas a0 yra nenaudojamas, a_1 puslapių lentelės ilgis atėmus 1 (arba nenaudojamas. Tai priklauso nuo puslapiavimo mechanizmo realizacijos), o štai $a_2 * 10 + a_3$ žymi puslapių lentelės adresą. Dabar galime pateikti formulę, kuri virtualiam adresui x_1x_2 gražina realų adresą:

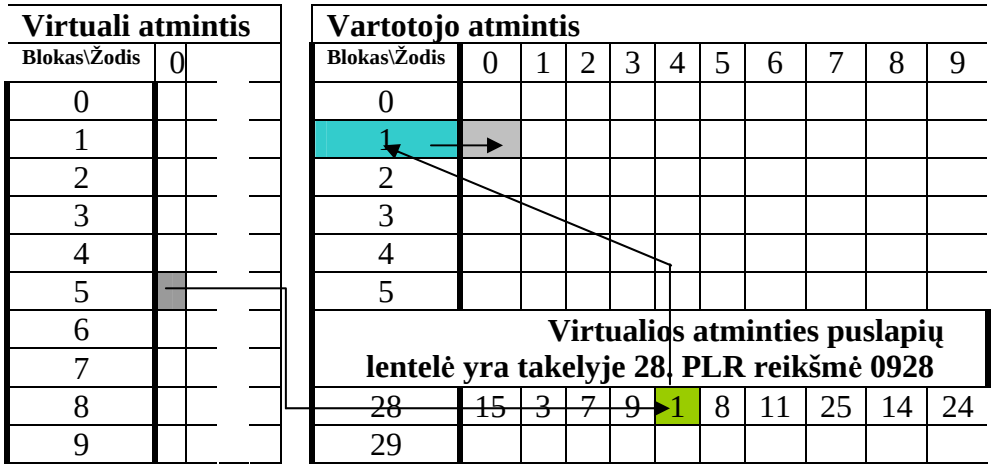
$$\text{Realus adresas} = 10*[10*(10 * a_2 + a_3) + x_1] + x_2$$

Virtualios mašinos kūrimo ir veiklos scenarijus žvelgiant iš atminties pusės būtų toks:

1. Pradėta kurti virtuali mašina.
2. Virtuali mašina reikalauja 10 takelių atminties savo reikmėms.
3. Yra išskiriama 10 takelių virtualiai mašinai.
4. Yra išskiriamas 1 takelis virtualios mašinos puslapių lentelei.
5. Puslapių lentelė (t.y. tas takelis)yra užpildomas išskirtų 10 takelių realiais adresais.
6. Virtualios mašinos virtualaus registro PTR reikšmei priskiriamas puslapių lentelės takelio realus adresas.
7. Virtuali mašina baigiama kurti.
8. Virtuali mašina gauna procesorių.

9. Virtualiai mašinai prireikė paversti virtualų adresą 50 (t.y. penktas takelis, parinkta atsitiktinai) realiu.
10. Virtuali mašina iš puslapių lentelės nuskaito 5 žodį. Tai ir yra realus adresas.
11. Taip toliau

Schema:pateikta 6 lentelėje.



6 lentelė. Puslapiavimo mechanizmo pavyzdys. Mus domina virtualaus adreso 50 realus adresas, kuris kaip matyti iš lentelės yra 10.

2.8 Taimerio mechanizmas

Šis mechanizmas atsakingas už geresnį užduočių išlygiagretinimą. Yra laikoma kad ta pati užduotis negali būti vykdoma ilgiau nei N laiko momentų. Mūsų mokomosios OS atveju šis mechanizmas bus truputi kitos – bus vykdoma ne daugiau N einamosios užduoties taktų.

Laikysim kad įvedimo/išvedimo instrukcijos atliekamos per 3 taktus, visos kitos per 1 taktą. Dabar apie veikimo principą.

Pradedant virtualios mašinos užduoties vykdymą TI registro reikšmė nustatoma tam tikrai reikšmei. Tarkim N = 10. Įvykdžius eilinę instrukciją TI reikšmė mažinama priklausomai nuo to per kiek taktų ši instrukcija yra atliekama. Kuomet TI reikšmė yra lygi nuliui, mikrokomanda Test () aptinka taimerio pertraukimą. Kiek plačiau apie tai galima paskaityt skyrelyje : 2.9 Pertraukimų mechanizmas.

2.9 Pertraukimų mechanizmas

Pertraukimas – tai signalas apie įvykusį įvykį. Kiekvienas pertraukimas turi savo identifikaciją (sistema turi turėti galimybę atskirti pertraukimų tipus). Pertraukimas savaime nepertraukia sistemos darbo, kaip kad gali pasirodyti iš pirmo žvilgsnio. Pertraukimą sistema turi aptikti ir atitinkamai sureaguoti. Pertraukimas - tai tik pakeista kompiuterio būsena. Kompiuterio būseną keičia tą būseną sukėlusi priežastis, pavyzdžiui, jei virtuali mašina vykdydama vartotojo programą sutinka neteisingą adresą ar operacijos kodą, tai komandų interpretatorius fiksuoja požymį “neteisingas adresas”(priskiria pertraukimo registrui tam tikrą reikšmę). Pertraukimus aptinka procesoriaus komanda test, kuri apklausia reikalingus registrus. Ir tik **sistemai aptikus** pertraukimą, yra nutraukiamas vartotojo programos vykdymas. Tai atliekama pasinaudojant pertraukimų vektorių lentele (kur yra nuorodos į pertraukimus apdorojančias programas). Valdymas yra perduodamas pertraukimą apdorosiančiai programai, kuri dažniausiai yra atmintyje su požymiu “tik skaitymui”. Įvykdžiusi savo darbą, pertraukimo apdorojimo programa valdymą gražina operacinei sistemai pertraukimo vietoje. Ką daryti toliau, sprendžia OS.

Modelyje bus realizuoti trijų tipų pertraukimai – programiniai, supervizoriniai ir taimerio. Programinių pertraukimų registras yra PI, supervizorinių pertraukimų registras – SI,

taimerio - TI. Programiniai pertraukimai kyla vykdant virtualią mašiną, bandant įvykdyti kokį nors neleistiną veiksmą arba nuskaičius neleistiną reikšmę. Supervizoriniai pertraukimai kyla virtualiai mašinai norint įvykdyti veiksmą, kuris gali vykti tik supervizoriaus režime. Pertraukimai gali būti aptikti tik vartotojo režime. Supervizoriniame režime centrinio procesoriaus darbo pertraukti negalima. Apie taimerio pertraukimą jau buvo kalbėta ankstesniame skyrelyje.

Pertraukimai kils šiais būdais:

- Operacijos GD, PD ir HALT iššauks supervizorinius pertraukimus. SI = 1 - komanda GD, SI = 2 - komanda PD, SI = 3 – komanda HALT.
- Programiniai pertraukimai:PI = 1 – neteisingas adresas, PI = 2 – neteisingas operacijos kodas, PI = 3 – neteisingas priskyrimas, PI = 4 – perpildymas (overflow)
- Esant TI = 0 bus fiksuojamas taimerio pertraukimas.

Esant situacijai SI = 0 ir PI = 0 ir TI <> 0, pertraukimų sistema neaptiks.

Pertraukimai nustatomi paprasčiausiai virtualaus procesoriaus registrams priskiriant atitinkamas reikšmes (pavyzdžiui, komandų interpretatoriui vykdant komandą GD, jis priskiria SI:= 1) Kiekvieną kartą komandų interpretatoriui įvykdžius programą, kviečiama komanda test(), kuri apklausia registrus, ir, jei kilo pertraukimas, gražina informaciją apie tai.

Komanda test() gali būti tokia:

If ((PI + SI) > 0) or (TI = 0) then return 1 else return 0;

Nustačius įvykusį pertraukimą (t.y. test() <> 0) virtualios mašinos procesoriaus registrų reikšmės yra išsaugomos, procesorius perjungiamas į supervizoriaus režimą, nustatomas pertraukimo pobūdis (pavyzdžiui tiesiogiai apklausiant registrus) ir kviečiama pertraukimą apdorosianti paprogramė. Vėliau, kaip buvo minėta, valdymas grįžta į virtualią mašiną, registrai atstatomi, procesorius perjungiamas į vartotojo režimą, ir operacinė sistema sprendžia, ką daryti toliau.

2.10 Kanalų įrenginys

Kanalų įrenginys leidžia dirbti su atmintimis. Priklausomai nuo nustatytų registrų kanalų įrenginys gali vykdyti apsikeitimą duomenimis visomis galimomis kryptimis.Veiksmai su kanalų įrenginiu atliekami tik supervizoriaus režime. Dabar bus pateikta kanalų įrenginio vartotojo sąsaja:

Kanalų įrenginio **registrai**:
SB: Takelio, iš kurio kopijuosime numeris.
DB: Takelio, į kurį kopijuosime numeris
ST: Objekto, iš kurio kopijuosime, numeris

1. Vartotojo atmintis;
2. Supervizorinė atmintis;
3. Išorinė atmintis;
4. Įvedimo srautas;

DT: Objekto, į kurį kopijuosime, numeris

1. Vartotojo atmintis;
2. Supervizorinė atmintis;
3. Išorinė atmintis;
4. Išvedimo srautas;

Kartu kanalų įrenginys turi komandą **XCHG**, tačiau neturi procesoriaus, kuris galėtų ją įvykdyti. Šią komandą vykdo centrinis procesorius, taigi, šis kanalų įrenginys nėra lygiagrečiai su centriniu procesoriumi veikianti aparatūra.

Procesas, norėdamas pasinaudoti kanalų įrenginiu, turi nustatyti kanalų įrenginio registrus ir tada įvykdyti komandą **XCHG**.

2.11 Užduoties formatas

Tai paskutinė realios ir virtualios mašinos modelio tema. Tolesniuose skyriuose jau bus kalbama apie pačią MOS. Užduotis, kaip jau buvo minėta, - tai programa, su savo vykdymo parametrais ir duomenimis. Užduotys bus laikomos failuose, su kuriais dirbs aukšto lygio kalbos procesorius. Norint sukurti naują užduotį, užtenka sukurti ir teisingai užpildyti naują tekstinį failą modelio išorėje. Bendras užduoties pavidalas bus iš šių dalių:

- Parametrai
- Programa
- Pabaigos žymė

Parametrams skirta 40 baitų (0.. 39) ir ši dalis susideda iš trijų laukų:

1. "**$AMJ**" (A Multiprogramming JOB). Pirmasis laukas visada turi turėti šią reikšmę. Ji užima pirmus keturis baitus
2. Maksimalus išvedimo eilučių skaičius. Jis skirtas sustabdyti dėl amžino ciklo užstrigusias programas. Užima antrus keturis baitus. (4..7)
3. Užduoties vardas. Šiam laukui skirti visi likę baitai (8..39)

Programos dalis. Šiai daliai skirta 100 * 4 = 400 baitų. Kad būtų paprasčiau vesti programas reikėtų laikytis sistemos:

Prieš kiekvieną naują programos ar duomenų bloką, turi būti simbolių rinkinys $*x*, kur x kinta intervale [0..9], ir x reikšmė reiškia bloko, į kurį bus rašomi duomenys, numerį. * reiškia nenaudojamą baitą. Pavyzdžiui:

- $020
- PD40
- HALT
- $040
- ALIO

Taigi, dvidešimtame virtualios mašinos žodyje bus laikoma reikšmė PD40, dvidešimt pirmame žodyje laikoma reikšmė HALT, o keturiasdešimtame žodyje reikšmė bus ALIO. Beje, ši programėlė į išvedimo srautą pasiųstu žodį ALIO.

Programos dalį seka **pabaigos žymė**, kuriai skirti 4 baitai ir ten turėtų būti reikšmė **"$END**".

O dabar demonstracinis užduoties pavyzdys. Tai kaip ši užduotis atrodo jau virtualioje atmintyje, galima pamatyti lentelėje Nr. 3.

$AMJ	PD80	0000
1000	GD50	**$080**
Demonstracija	HALT	Suskaiciuota
$000	SR70	**$090**
LR70	PD70	000000010999
AD91	GO00	**$END**
CR92	HALT	
NT07	**$070**	

3 Operacinės sistemos modelis

Visos šiuolaikinės operacinės sistemos gali atlikti kelias užduotis tuo pačiu metu. Multiprograminėje operacinėje sistemoje centrinis procesorius yra perjungiamas iš vieno proceso į kitą. Kiekvieną iš procesų vykdo tik kelias dešimtąsias ar šimtąsias sekundės dalis. Todėl net ir sekundės bėgyje procesorius sugeba aptarnauti keletą procesų, ir tai suteikia vartotojui lygiagretaus veikimo įspūdį. Tačiau kaip tas mechanizmas yra organizuojamas? Kaip ir kas perdavinėja procesorių užduotims? Apie visa tai plačiau.

3.1 Procesai

Procesas – tai vykdoma programa, kartu su esamomis registrų reikšmėmis ir savo kintamaisiais. Kiekvienas procesas turi savo virtualų procesorių Žiūrėti: *2.3* Virtualios mašinos centrinis procesorius). Nors skirtumas tarp programos ir proceso nėra didelis, bet jis svarbus. Procesas – tai kokioje nors veiklumo stadijoje esanti programa. Tuo tarpu programa – tai tik tam tikras baitų rinkinys. Veiklumo stadiją apibūdina proceso aprašas – deskriptorius. Apraše ir yra laikomi visi procesui reikalingi parametrai; tokie kaip virtualaus procesoriaus registrų reikšmės, ar jam reikalingi kintamieji. Taigi, galimas ir kitoks požiūris į procesą – t.y. kaip į proceso aprašą. Kiek vėliau bus naudojama sąvoka “procesų sąrašas”. Procesų sąrašo elementai - procesų aprašai.

Procesų aprašai – dinaminiai objektai. Tai reiškia, kad jie gali būti sukurti ar sunaikinti jau sistemos veikimo metu. Procesus kuria procesai, iš to seka, kad turi būti vienas pagrindinis procesas, kuris sukurs visus kitus.

Paprastai procesus galima suskirstyti į vartotojiškus ir sisteminius. Sisteminių procesų paskirtis – aptarnauti vartotojiškus. Tuo tarpu vartotojiško proceso paskirtis yra vykdyti vartotojo programą. Kiek vėliau apie procesus bus detaliau, o dabar apie proceso būsenas.

3.1.1 Procesų būsenos

Procesas gali gauti procesorių tik tada, kai jam netrūksta jokio kito resurso. Procesas gavęs procesorių tampa vykdomu. Procesas, esantis šioje būsenoje, turi procesorių, kol sistemoje neįvyksta pertraukimas arba einamasis procesas nepaprašo kokio nors resurso (pavyzdžiui, prašydamas įvedimo iš klaviatūros). Procesas blokuojasi priverstinai (nes jis vis tiek negali tęsti savo darbo be reikiamo resurso). Tačiau, jei procesas nereikalauja jokio resurso, iš jo gali būti atimamas procesorius, pavyzdžiui, vien tik dėl to, kad pernelyg ilgai dirbo. Tai visiškai skirtinga būsena nei blokavimasis dėl resurso (neturimas omeny resursas - procesorius). Taigi, galime išskirti jau žinomas procesų būsenas:

- **Vykdomas** - turi procesorių
- **Blokuotas** - prašo resurso (išskyrus procesorių)
- **Pasiruošęs** – vienintelis trūkstamas resursas yra procesorius.

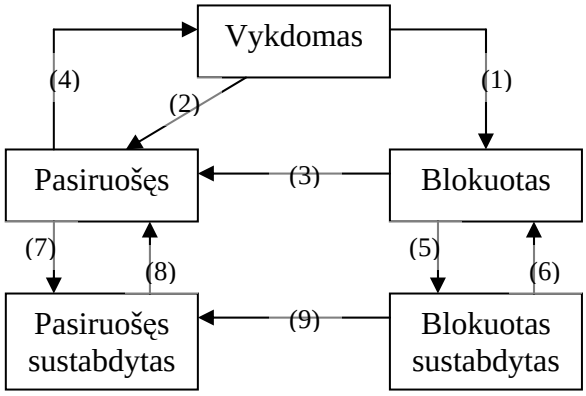
Tačiau šių būsenų gali neužtekti. Gali susiklostyti tokia situacija, kad tam tikram procesui negalima leisti gauti procesorių, nors jis ir yra pasiruošęs. Toks procesas vadinamas sustabdytu.Galime papildyti būsenų sąrašą:

- **Sustabdytas** – kito proceso sustabdytas procesas.

Kaip matyti procesorius yra ypatingas resursas. Kuo jis skiriasi nuo kitų?

Procesoriaus resursas yra reikalingas visiems procesams. Nė vienas procesas netaps vykdomu (taigi, ir nedirbs savo darbo), neturėdamas procesoriaus. Todėl jo skirstymas yra ypatingas, ir mes jį apžvelgsime kitame skyrelyje.

O dabar tęsiame apie procesų būsenas. Norėtųsi žinoti kaip gali procesas pakliūti į tam tikrą būseną ir kaip iš jos išeiti. Tai turėtu pademonstruoti 1 diagrama:



1 diagrama. Proceso būsenos ir ryšiai tarp jų.

Kaip matyti iš diagramos yra devyni perėjimai. Juos trumpai pakomentuosime:

1. Vykdomas procesas blokuojasi jam prašant ir negavus resurso.
2. Vykdomas procesas tampa pasiruošusiu atėmus iš jo procesorių dėl kokios nors priežasties (išskyrus resurso negavimą).
3. Blokuotas procesas tampa pasiruošusiu, kai yra suteikiamas reikalingas resursas.
4. Pasiruošę procesai varžosi dėl procesoriaus. Gavęs procesorių procesas tampa vykdomu.
5. Procesas gali tapti sustabdytu blokuotu, jei einamasis procesas jį sustabdo, kai jis jau ir taip yra blokuotas.
6. Procesas tampa blokuotu iš blokuoto sustabdyto, jei einamasis procesas nuima būseną sustabdytas.
7. Procesas gali tapti pasiruošusiu sustabdytu, jei einamasis procesas jį sustabdo, kai jis yra pasiruošęs.
8. Procesas tampa pasiruošusiu iš pasiruošusio sustabdyto, jei einamasis procesas nuima būseną sustabdytas
9. Procesas tampa pasiruošusiu sustabdytu iš blokuoto sustabdyto, jei procesui yra suteikiamas jam reikalingas resursas.

Dabar apie ypatingo resurso - procesoriaus skirstymą procesams

3.1.2 Planuotojas

Trumpai kalbant, planuotojo paskirtis yra atimti procesorių iš proceso, peržvelgti pasiruošusių procesų sąrašą, išrinkti pasiruošusį procesą, kuris planuotojo manymu yra tinkamiausias, ir perduoti procesorių jam.

Gali iškilti klausimas – kam apskritai reikalingas planuotojas? Galbūt procesai gali perdavinėti valdymą patys? Multiprograminėje operacinėje sistemoje atsakymas būtų paprastas – negali. Negali, nes procesas nežino ir iš anksto (programavimo metu) negali žinoti kuris procesas bus pasiruošęs vykdymui, kuris blokuotas, jo veikimo momentu. Be to, procesams perdavinėjant valdymą patiems, būtų neišvengta tiesiškumo.

Reikia atkreipti dėmesį į tai, kad procesų programos yra ciklinės. Taigi, iš pirmo žvilgsnio gali atrodyti, kad procesas, gavęs procesorių, pasiims jį visam laikui, bet tai netiesa. Procesų išlygiagretinimas ir valdymo perdavimas dabar vyksta per resursų primityvų mechanizmą (apie tai bus vėliau). Tam, kad būtų aiškiau, tiesiog bus išvardinti planuotojo tikslai, kuriuos pasiekti, kai procesai patys naudotų valdymo perdavimą, būtų beveik neįmanoma:

- Užtikrinti, kad kiekvienas procesas pagristą laiko tarpą (nei per daug, nei per mažai) turėtų procesorių;
- Laikyti procesorių užimtą netoli 100 %;
- Sumažinti iki minimumo atsakymo laiką vartotojams;
- Maksimizuoti darbų skaičių nuveiktą per valandą.

Pats planuotojas, grubiai tariant, yra algoritmas, skirtas šių tikslų įgyvendinimui. Tokių algoritmų yra nemažai ir šiame modelyje mes remsimės vienu iš jų – grįstu prioritetais.

3.1.3 Prioritetais besiremiantis modelis

Kaip jau buvo minėta, planuotojas turi iš pasiruošusių procesų sąrašo išrinkti vieną, kuris taps vykdomu. Kaip jis tai padarys – algoritmo reikalas. Mūsų algoritmas remsis procesų prioritetais.

Proceso prioritetas – tai proceso svarba, įvertinta kokioje nors skalėje, sakykime, nuo 1 iki 100. 1 – tai pati žemiausia proceso svarba, tuo tarpu 100 – pati aukščiausia. Procesas, turintis didesnį prioritetą, atsiduria arčiau procesų sąrašo pradžios. Visi procesų sąrašai yra rūšiuojami pagal proceso prioritetą, ir arčiau pradžios esantis procesas turi didesnę galimybę tapti vykdomu (pasiruošusių procesų sąrašo atveju) arba pasiruošusiu (blokuotų procesų sąrašo atveju).

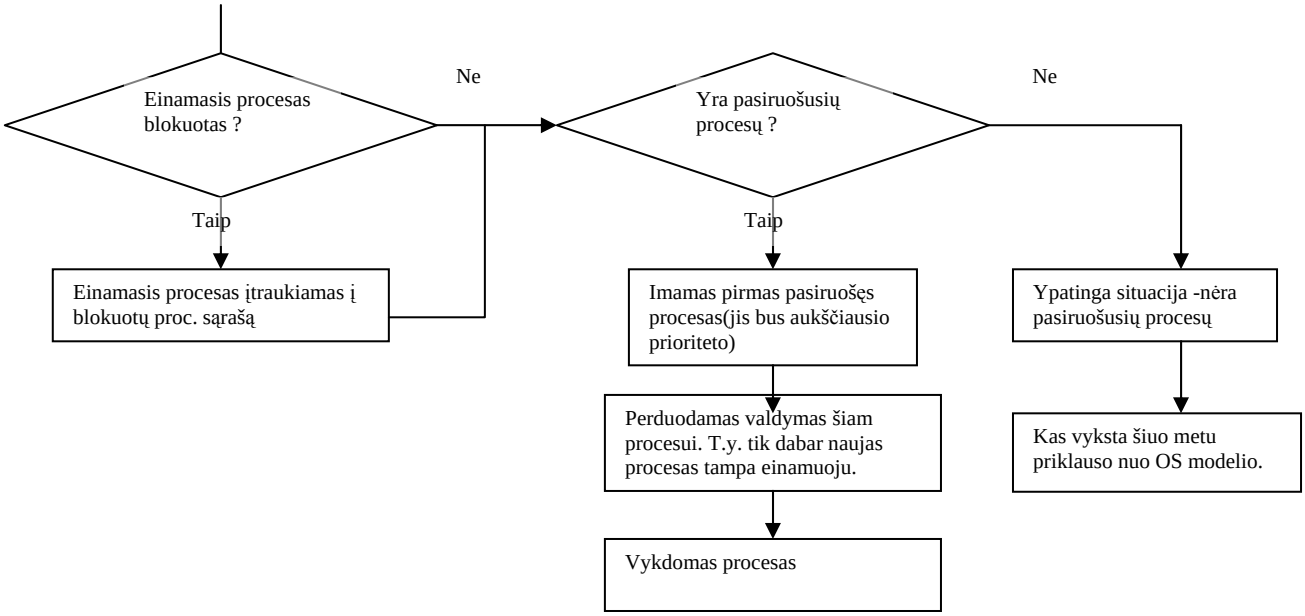
Sisteminių procesų paskirtis – aptarnauti vartotojiškus. Praktika rodo, kad, aptarnaujant sisteminius procesus prieš vartotojiškus, sistemos efektyvumas auga, todėl paprastai vartotojiški procesai turi kur kas mažesnį prioritetą nei sisteminiai. Procesų prioritetų priskyrimas yra gana atsakingas darbas. Nelogiškai paskirsčius prioritetus, sistemos darbas gali tapti nestabilus, ar net neįmanomas.

Siekiant sistemos efektyvumo, galima patobulinti šį paprastą algoritmą. Pavyzdžiui, tam, kad koks nors procesas negautų procesoriaus pernelyg dažnai, galima kiekvieną kartą, jam suteikus procesorių, jo prioritetą sumažinti vienetu.

3.1.4 Planuotojo veiksmų seka

Planuotojas kviečiamas kai norima pakeisti einamąjį procesą kitu (susiklosčius tinkamoms aplinkybėms einamasis procesas gali ir nepasikeisti, ypač, jei jo prioritetas aukštas). Planuotojo veiksmų seką gali pademonstruoti 2 diagrama.

Ši diagrama reikalauja paaiškinimų. Labiausiai reikia pabrėžti, kad einamasis procesas tampa paprastu tik valdymo perdavimo momentu. Visą laiką iki to momento, nesvarbu, ar jis blokuotas ar koks kitoks, jis yra einamasis.



2 diagrama. Planuotojo schema

Pirmas planuotojo žingsnis yra einamojo proceso būsenos tikrinimas. Jei einamasis procesas nėra blokuotas, jis įtraukiamas į pasiruošusių procesų sąrašą. Sekantis planuotojo žingsnis yra tikrinimas, ar yra pasiruošusių procesų. Nesvarbu, koks yra procesų rinkinys jūsų modelyje, jame turi būti numatytas atvejis, kada yra užsiblokavę visi vartotojo ir visi aptarnaujantys vartotojiškus procesus procesai. Sprendimų yra nemažai : pavyzdžiui, galima pridėti keletą procesų su mažiausiu įmanomu prioritetu. Jie gautą laiką skirs sistemos stabilumui tikrinti, ar tvarkyti ją. Jei kompiuterio architektūra leidžia, jį galima “užmigdyti”.

Jeigu pasiruošusių procesų sąrašas nėra tuščias, tai imamas pirmasis sąrašo elementas - proceso aprašas. Tai bus procesas su aukščiausiu prioritetu. Sekantis žingsnis – valdymo perdavimas. Proceso apraše laikoma virtualaus procesoriaus būsena priskiriama realiam procesoriui, išsaugoma einamojo proceso aplinka (stekas), užkraunama pasirinktojo proceso aplinka. Naujasis procesas pažymimas pažymimas kaip einamasis.Viskas. Pakeistoje proceso aplinkoje bus ir grįžimo į naująjį einamąjį procesą adresas. Taigi, valdymas netiesiogiai perduotas. Baigdama savo darbą, planuotojo funkcija grįš ne ten, iš kur atėjo, o ten, kur rodo užkrauta einamojo proceso aplinka.

3.1.5 Proceso ir su juo susijusių klasių aprašai

Proceso aprašu vadinama fiksuoto dydžio duomenų struktūra, kuri saugo informaciją apie proceso einamąjį stovį. Formalūs proceso, procesų sąrašo, branduolio, proceso būsenų, virtualaus procesoriaus registrų, pertraukimų registrų ir atminties vienetų apibrėžimai:

```
TProcess = class // proceso apibrėžimas
        FList: TProcessList;
        FID: Integer;
        FSavedRegisters: TSavedRegisters;
        FProcessor: TProcessor;
        FCreatedRes: TResourceList;
        FOwnedRes: TElementList;
        FState: TProcessState;
        FPriority: Integer;
        FParent: TProcess;
        FChildren: TProcessList;
      FKernel:TKernel;
        FUserName: AnsiString;
        FStack: Pointer;
        FStackSize: Integer;
      end;
```

```
TProcessList = class //procesų sąrašo ap.
   FKernel: TKernel;
   FProcesses: array[ ]of TProcess:
   FCount: Integer ;
  end;

TProcessState = (sRun, sReady, sBlock,
sReadyStop, sBlockStop);//Proceso būsenos ap.

TSavedRegisters = class (TObject) registrų ap.
   R: TDWord;
   C: TByte;
   IC: TWord;
   PI: TPIReg;
   SI: TSIReg;
   IOI: Byte;
   PTR: TDWord;
   Mode: TByte;
  end;

TByte = Char;

TWord = array [0..1] of Char;

TDWord = array [0..3] of Char;

TPIReg = (piNone, piInvalidOpCode,
piInvalidAddress, piOverflow,piIllegalAssignment); //
PI reg. ap.

TSIReg = (siNone, siGetData, siPutData, siHalt);
// SI registro apibrėžimas

TKernel = class //Branduolio apibrėžimas
   FProcesses: TProcessList;
   FResources: TResourceList;
   FReadyProc: TProcessList;
   FRunProc: TProcessList;
   FRealMachine: TRealMachine;
end
```

Dabar trumpi aprašų komentarai. Patogumo dėlei buvo įvesti atminties vienetai TByte, TWord, ir TDWord , kurie komentarų, matyt, nereikalauja.

Pertraukimų registrų struktūros:

Programinių pertraukimų registras **TPIReg**: piNone – nėra pertraukimo; piInvalidOpCode – neteisingas operacijos kodas; piInvalidAddress – neteisingas adresas,;piOverflow – atminties žodžio arba registro perpildymas; piIllegalAssignment – neteisingas priskyrimas.

Supervizorinių pertraukimų registras **TSIReg:** siNone – nėra pertraukimo; siGetData – įvedimo pertraukimas; siPutData – išvedimo pertraukimas; siHalt – vartotojo programos pabaigos pertraukimas.

Proceso būsenos apibrėžimas **TProcessState**: sRun – einamasis procesas; sReady – pasiruošęs procesas; sBlock – blokuotas procesas; sReadyStop – pasiruošęs sustabdytas procesas; sBlockStop – blokuotas sustabdytas procesas.

Proceso aprašas **TProcess**:

- *FList: TProcessList;* - procesų sąrašas, kuriam priklauso procesas. Tai gali būti pasiruošusių procesų sąrašas arba blokuotų, dėl kokio nors resurso, procesų sąrašas.
- *FID: Integer;*- Vidinis proceso vardas. Kiekvienas sisteminis objektas turi savo unikalų numerį.
- *FSavedRegisters: TSavedRegisters;* - Proceso išsaugota procesoriaus būsena. Naudojama perduodant valdymą procesui.
- *FProcessor: TProcessor;* nuoroda į procesorių.
- *FCreatedRes: TResourceList;* - Proceso sukurti resursai.
- *FOwnedRes: TElementList;* - Procesui kūrimo metu perduoti resursai.
- *FState: TProcessState;* - Proceso būsena.

- *FPriority: Integer;* - Proceso prioritetas. Pasirinkta sistema: kuo didesnė reikšmė, tuo procesas laikomas svarbesniu .
- *FParent: TProcess;* - nuoroda į procesą–tėvą.
- *FChildren: TProcessList;* - procesų–vaikų sąrašas.
- *FKernel:TKernel;* - Nuoroda į branduolį.
- *FUserName: AnsiString;* - Išorinis vardas. Naudojamas patogesniam proceso identifikavimui OS stebėjimo metu.
- *FStack: Pointer;* - proceso aplinka – stekas. Kiekvieną kartą perduodant valdymą procesui yra gražinama jo aplinka. Kiekvieną kartą, iš proceso atimant procesorių, jo aplinka yra išsaugoma.
- *FStackSize: Integer*; - proceso aplinkos – steko dydis.

Procesų sąrašo aprašas TProcessList:

- *FKernel: TKernel*; - Nuoroda į branduolį
- *FProcesses: array[ ]of TProcess*: - Dinaminis procesų sąrašas.
- *FCount: Integer* ; Procesų skaičius procesų masyve.

Operacinės sistemos branduolys TKernel

- *FProcesses: TProcessList;* - Bendras visų, esančių sistemoje, procesų sąrašas
- *FResources: TresourceList ;*-Bendras visų, sistemoje esančių, resursų sąrašas
- *FReadyProc: TProcessList;* - Pasiruošusių procesų sąrašas.
- *FRunProc: TProcessList;* - Einami procesai. Jų gali būti ne vienas, jei realioje mašinoje yra keli procesoriai.
- *FRealMachine: TRealMachine*; -Nuoroda į realią mašiną. Jos sandaros mes nenagrinėsime. Ji remiasi mūsų nagrinėtu modeliu.

3.1.6 Procesų primityvai

Procesų primityvų paskirtis – pateikti vienodą ir paprastą vartotojo sąsają darbui su procesais. Darbui su procesais skirti 4 primityvai:

1. **Kurti procesą**. Šiam primityvui perduodama nuoroda į jo tėvą, jo pradinė būsena, prioritetas, perduodamų elementų sąrašas ir išorinis vardas. Pačio primityvo viduje vyksta proceso kuriamasis darbas. Jis yra registruojamas bendrame procesų sąraše, tėvo-sūnų sąraše, skaičiuojamas vidinis identifikacijos numeris, sukuriamas jo vaikų procesų sąrašas (tuščias), sukurtų resursų sąrašas ir t.t.
2. **Naikinti procesą**. Pradedama naikinti proceso sukurtus resursus ir vaikus. Vėliau išmetamas iš tėvo sukurtų procesų sąrašo. Toliau išmetamas iš bendro procesų sąrašo ir, jei reikia, iš pasiruošusių procesų sąrašo. Galiausiai naikinami visi jam perduoti resursai ir proceso deskriptorius yra sunaikinamas.
3. **Stabdyti procesą**. Keičiama proceso būsena iš blokuotos į blokuotą sustabdytą arba iš pasiruošusios į pasiruošusią sustabdytą. Einamasis procesas stabdomas tampa pasiruošusiu sustabdytu.
4. **Aktyvuoti procesą.** Keičiama proceso būsena iš blokuotos sustabdytos į blokuotą, ar pasiruošusios sustabdytos į pasiruošusią.

Pastaba: Procesai labai aktyviai naudojasi resurso primityvais “prašyti resurso” ir “atlaisvini resursą”. Nereikia jų painioti su procesų primityvais.

Kiekvieno primityvo programos gale yra kviečiamas planuotojas.

3.1.7 Sisteminiai ir vartotojiški procesai

Jau buvo minėta, kad procesus galima suskirstyti į sisteminius ir vartotojiškus. Sisteminis procesas – tai procesas, atliekantis tam tikras sistemines funkcijas, pavyzdžiui, vartotojiško proceso palaikymą.Vartotojiškas procesas skirtas vartotojo užduočiai atlikti.

Sisteminiai procesai yra kuriami paleidžiant sistemą, o naikinami – naikinant sistemą. Taigi visą sistemos gyvavimo laiką jie yra pasiruošę dirbti jiems skirtą darbą. Paprastai paleidus sistemą, visi sisteminiai procesai anksčiau ar vėliau užsiblokuoja. Bet tai nereiškia, kad jie yra neveiklūs. Jie laukia, kol jie galės atlikti savo darbą, kuris paprastai būna susijęs su vartotojiška užduotimi.Pavyzdžiui, atėjus signalui iš vartotojo sąsajos apie naują užduotį, yra atlaisvinamas tam tikras resurso elementas. Jo dėka atsiblokuoja vienas iš sisteminių procesų, kuris nuveikia tam tikrą darbą ir atlaisvina vieną ar kelis resursus. Taigi, sisteminiai procesai vienas po kito atsiblokuoja, kol galų gale sukuriama vartotojo užduotis. Vartotojo užduočiai baigus darbą, neužsiblokavę procesai ne už ilgo užsiblokuoja, ir vėl prasideda laukimas.

Vartotojiški procesai yra sukuriami sisteminių procesų jau veikiant sistemai. Kartu su vartotojišku procesu gali būti sukurti vienas ar keli sisteminiai procesai, skirti aptarnauti vartotojišką procesą.

Jau buvo minėta, kad paprastai sisteminiai procesai turi turėti aukštesnį prioritetą nei vartotojiški. Centrinio procesoriaus resursas yra vienintelis, dėl kurio varžosi tiek sisteminiai tiek vartotojiški procesai.Modelyje vartotojiški procesai varžosi tarpusavyje (nes prioritetai neleidžia jiems varžytis su sisteminiais) tik dėl procesoriaus resurso.

3.2 Resursai

Resursas yra tai, dėl ko varžosi procesai. Dėl resursų trūkumo procesai blokuojasi, gavę reikiamą resursą, procesai tampa pasiruošusiais. Resursus galima skirstyti į:

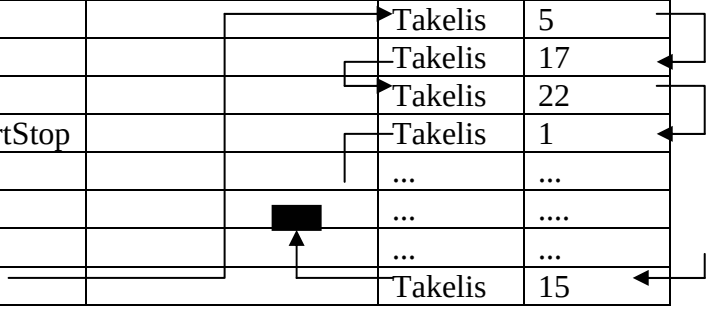
- Statinius resursus. Kuriami sistemos kūrimo metu. Tai mašinos resursai, tokie kaip procesorius, atmintis ar kiti resursai, kurie sistemos veikimo metu nėra naikinami. Šie resursai gali būti laisvi, kai nė vienas procesas jų nenaudoja, arba ne, kada juos naudoja vienas ar keli, jei tą resursą galima skaldyti, procesai.
- Dinaminius resursus. Kuriami ir naikinami sistemos darbo metu. Šie resursai naudojami kaip pranešimai. Kartu su jais gali ateiti naudinga informacija. Kartais šio tipo resursas pats yra pranešimas. Pavyzdžiui, esantis laisvas kanalo resursas žymi, kad bet kuris procesas gali naudotis kanalu. Jei jo nėra, procesas priverstas laukti, kol šis resursas taps prieinamu (bus atlaisvintas).

Iškart reiktų pastebėti, kad resursas iš tikrųjų tėra tik aprašas (deskriptorius) ir nuoroda į resurso elementus. Prašyti resurso – tai prašyti resurso elemento. Lengviausia būtų tai pavaizduoti atminties resurso atveju. Tarkime, vartotojo atmintis susidedanti iš 30 takelių.

Resursas “atmintis” turėtų kintamuosius, tokius kaip resurso elementų kiekis, laisvų elementų kiekis, galiausiai nuorodą į pačius resurso elementus. Resurso elementai turėtų lauką, kuriame būtų takelio numerio reikšmė. Pavyzdys 7 lentelėje.

Kiekvienas resursas turi laukiančių procesų sąrašą (jis gali būti ir tuščias). Kiekvienas procesas prašęs, ir negavęs resurso elemento, yra ne tik užblokuojamas, bet ir įdedamas į resurso laukiančių procesų sąrašą.

Resursas Atmintis			**Resurso elementai**	
Laukas	**Reikšmė**		**Laukas**	**Reikšmė**
Kiekis	30		Takelis	5
Laisvi	10		Takelis	17
ID	25		Takelis	22
Tėvas	StartStop		Takelis	1
...	...		...	...
...	...		...	....
...	...		...	...
Elementai			Takelis	15



7 Lentelė. Resursas ir jo elementai

3.2.1 Resurso primityvai

Resursas turi keturis primityvus:

1. **Kurti resursą**. Resursus kuria tik procesas. Resurso kūrimo metu perduodami kaip parametrai: nuoroda į proceso kūrėją, resurso išorinis vardas. Resursas kūrimo metu yra: pridedamas prie bendro resursų sąrašo, pridedamas prie tėvo suskurtų resursų sąrašo, jam priskiriamas unikalus vidinis vardas, sukuriamas resurso elementų sąrašas ir sukuriamas laukiančių procesų sąrašas.
2. **Naikinti resursą**. Resurso deskriptorius išmetamas iš jo tėvo sukurtų resursų sąrašo, naikinamas jo elementų sąrašas, atblokuojami procesai, laukiantys šio resurso, išmetamas iš bendro resursų sąrašo, ir, galiausiai naikinamas pats deskriptorius.
3. **Prašyti resurso.** Šį primityvą kartu su primityvu “atlaisvinti resursą” procesai naudoja labai dažnai. Procesas, iškvietęs šį primityvą, yra užblokuojamas ir įtraukiamas į to resurso laukiančių procesų sąrašą. Sekantis šio primityvo žingsnis yra kviesti resurso paskirstytoją.
4. **Atlaisvinti resursą**. Šį primityvą kviečia procesas, kuris nori atlaisvinti jam nereikalingą resursą arba tiesiog perduoti pranešimą ar informaciją kitam procesui. Resurso elementas, primityvui perduotas kaip funkcijos parametras, yra pridedamas prie resurso elementų sąrašo. Šio primityvo pabaigoje yra kviečiamas resursų paskirstytojas.

3.2.2 Resurso paskirstytojas

Kaip kad procesorius yra skirstomas planuotojo, kiekvienas resursas taipogi yra skirstomas tam tikro paskirstytojo. Prašydamas resurso ar norėdamas jį atlaisvinti, procesas kreipiasi į atitinkamą resursų paskirstytoją, kuris privalo jį aptarnauti.

Resursų paskirstytojo paskirtis – suteikti paprašytą resurso elementų kiekį procesui. Resursų paskirstytojo algoritmas gali būti sudėtingas. Pavyzdžiui, turi būti numatyta galimybė atiduoti resurso elementą konkrečiam procesui, arba prašyti kelių resurso elementų. Resurso paskirstytojas peržvelgia visus laukiančius šio resurso procesų sąrašą, ir, sutikęs galimybę aptarnauti procesą, perduoda jam reikalingus resurso elementus ir pažymi jį pasiruošusiu. Paskirstytojo pabaigoje yra iškviečiamas planuotojas.

Pastaba: demonstraciniame darbe resursų paskirstytojas buvo integruotas į resurso primityvus, tačiau taip neturėtų būti. Norėta parašyti vieną resursų paskirstytoją visiems resursų tipams, tačiau realizuotas paskirstytojas tapo labai painus.

3.2.3 Resurso ir su juo susijusių klasių aprašai

Resurso aprašu vadinama fiksuoto dydžio duomenų struktūra, kuri saugo informaciją apie resurso einamąjį stovį. Formalūs resursų sąrašo, resurso, resurso elemento ir elementų sąrašo apibrėžimai:

```
TResource = class
    FID: Integer ;
    FCreator: TProcess;
    FList: TElementList;
    FWaitingProc : TProcessList;
    FWaitingCount: TIntegerList;
    FKernel: TKernel ;
    FResourceList: TResourceList;
    FWaitingProcPoint: TList;
end;

TResourceList = class
    Count: Integer ;
    Kernel: TKernel ;
    Resources: array[ ] of  TResource ;
end
```

```
TResElement = class
    FElementList: TelementList
    FReceiver: TProcess;
    FSender: TProcess;
    FRetList: Boolean;
end;

TElementList = class
      FResource: TResource;
      FCount: Integer;
      FElement: array[ ] of TResElement;
 end;

.
```

Dabar trumpi aprašų komentarai:

Resurso aprašas **TResource**:

- *FID: Integer ;* - vidinis resurso vardas. Kiekvienas sisteminis objektas turi savo unikalų numerį.
- *FCreator: TProcess;* -Nuoroda į procesą, sukūrusį šį resursą.
- *FList: TElementList;* - Nuoroda į resurso elementų sąrašą.
- *FWaitingProc : TProcessList;* -Nuoroda į šio resurso laukiančių procesų sąrašą.
- *FWaitingCount: TIntegerList;* -Nuoroda į šio resurso laukiančių procesų paprašytų resurso kiekių sąrašą.
- *FKernel: TKernel ;* - Nuoroda į branduolį.
- *FResourceList: TResourceList;*- nuoroda į visų resursų sąrašą
- *FWaitingProcPoint: TList;* - nuoroda į šio resurso laukiančių procesų resurso elementų rodyklių sąrašą.

Įsivaizduokime situaciją. Procesas A prašo 10 vartotojo atminties takelių ir jų negauna. Jis perduoda kaip parametrą norimą elementų kiekį – 10 ir nuorodą, kur tuos elementus padėti. Taigi į FWaitingProc sąrašą įtraukiamas pats procesas A, į FwaitingCount sąrašą įtraukiamas norimo resurso kiekis 10 ir, galiausiai, į FwaitingProcPoint sąrašą įtraukiamas perduotas adresas, kur reikės padėti elementus.

Resursų sąrašo aprašas **TResourceList**

- *Count: Integer ;* - sąraše esančių resursų kiekis.
- *Kernel: TKernel ;* - nuoroda į branduolį.

- *Resources: array[ ] of TResource ;* - dinaminis masyvas, kurio elementai – resursai.

Resurso elemento aprašas **TResElement**

- *FElementList: TelementList*- nuoroda į resurso elementų sąrašą, kuriame yra šis resurso elementas.
- *FReceiver: TProcess;*- procesas, kuris turi gauti šį resurso elementą. Jei šio lauko reikšmė lygi **nil,** tai laikoma, kad šį elementą gali gauti bet kuris procesas.
- *FSender: TProcess;* - procesas, atlaisvinęs šį resurso elementą
- *FRetList: Boolean;* - šio lauko reikšmė žymi, ar šio tipo resurso elementai bus gražinami kaip sąrašas ar kaip paprastas elementas. Pavyzdžiui atminties atveju šio lauko reikšmė turėtų būti lygi 1, nes grąžinti reikės elementų, su takelių numeriais, sąrašą.

Pastaba: čia tik bendros visiems resurso elementams savybės. Kiekvieno tipo resursui resurso elementai yra kuriami pagal šį šabloną (paveldint) ir pridedant kelias papildomas reikalingas reikšmes.

Resurso elementų sąrašas **TElementList**

- *FResource: TResource;* - Nuoroda į resursą, kuriam priklauso šis elementų sąrašas
- *FCount: Integer;* - Elementų skaičius sąraše. Šis skaičius nurodo, kiek resurso elementų yra laisvų (t.y. prieinamų procesams) iš viso.
- *FElement:array [ ] of TResElement;* - dinaminis elementų sąrašas. Kreipiantis su tam tikru indeksu pasiekiamas atitinkamas elementas.

3.3 Vartotojo užduoties būsenos kitimas sistemoje

Vartotojo užduotis jos apdorojimo metu pereina keletą būsenų:

- Fizinė užduotis – tai toks užduoties pavidalas, kokiu ji yra suprantama įvedimo sraute.
- Užduotis - resursas. Sisteminiai procesai, apdorodami informaciją iš įvedimo srauto, formuoja iš jos resursą, kurio prašo vis kiti sisteminiai procesai. Taip apdorojama vartotojo užduotis nukeliauja iki sekančio savo gyvavimo etapo.
- Užduotis – procesas. Sisteminiams procesams surinkus visus vartotojo užduoties procesui (virtualiai mašinai) reikalingus resursus, procesas yra sukuriamas ir įtraukiamas į pasiruošusių sąrašą. Šioje stadijoje užduotis vykdo savo vartotojišką programą tol, kol darbas nėra baigiamas. Vartotojo užduočiai proceso stadija yra paskutinė. Toliau seka tik tos užduoties naikinimas. Už tai yra atsakingi sisteminiai procesai.

3.4 Procesų paketas

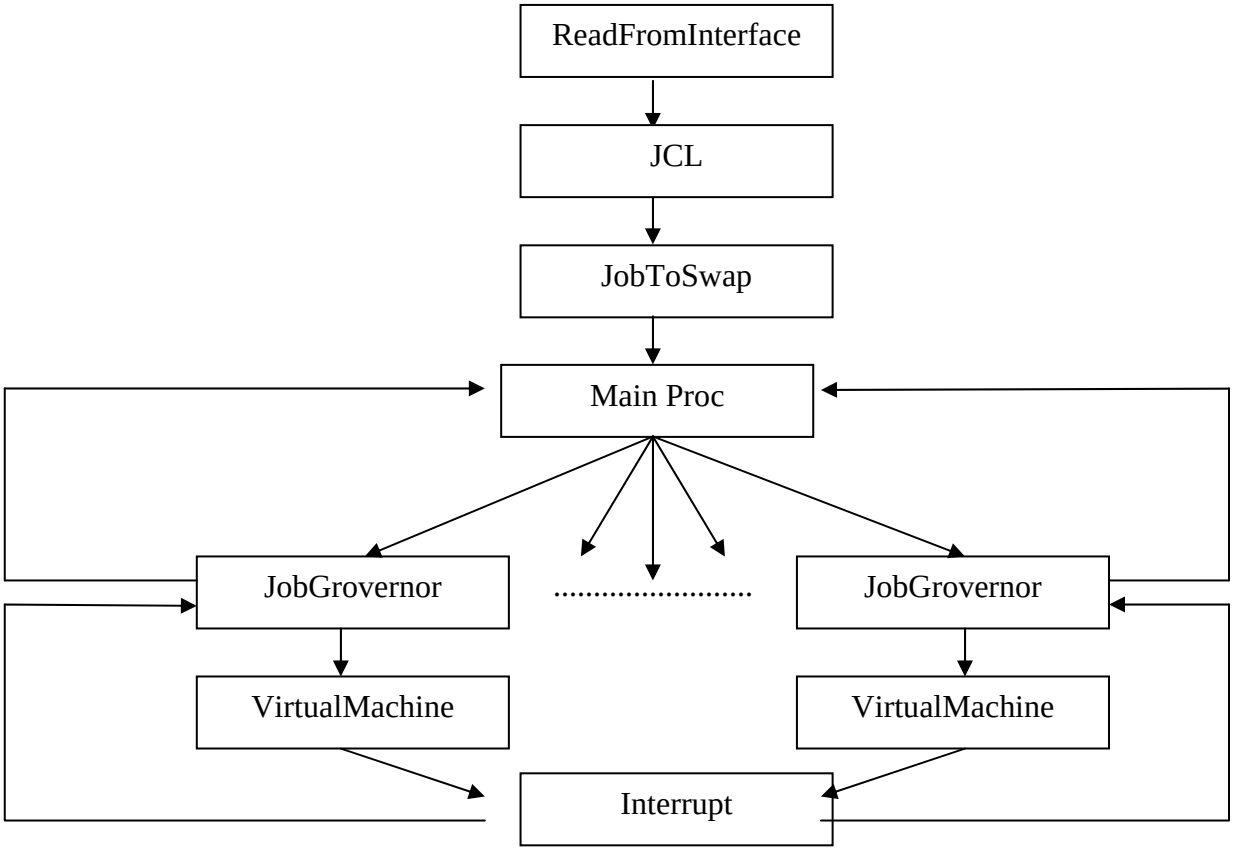
Mūsų modelyje užduoties keliui palaikyti bus naudojami šie procesai:

- **StartStop** – šakninis procesas, sukuriantis bei naikinantis sisteminius procesus ir resursus.
- **ReadFromInterface** – užduoties nuskaitymo iš įvedimo srauto procesas
- **JCL** – užduoties programos, jos antraštės išskyrimas iš užduoties, ir jų organizavimas kaip resursus..
- **JobToSwap** – užduoties patalpinimas išorinėje atmintyje
- **MainProc** – Procesas valdantis JobGorvernor procesus.
- **JobGorvernor** – virtualios mašinos proceso tėvas, tvarkantis virtualios mašinos proceso darbą
- **Loader** – iš išorinės atminties duomenys perkeliami į vartotojo atmintį

- **Virtual Machine** – procesas atsakantis už vartotojiškos programos vykdymą.
- **Interrupt** – procesas, apdorojantis virtualios mašinos pertraukimą sukėlusią situaciją.
- **PrintLine** – į išvedimo įrenginį pasiunčiama eilutė iš supervizorinės atminties.

Beveik visi procesai yra sukuriami sistemos darbo pradžioje proceso *StartStop*. *StartStop* nekuria tik 2 procesų – *JobGorvernor* (kiekvienai naujai vartotojo užduočiai *MainProc* kuria po naują procesą *JobGorvernor*) ir *VirtualMachine*, kurį kuria *JobGorvernor*.

Taigi, dabar galima peržvelgti bendrą procesų schemą iš vartotojo užduoties pusės:



4 paveiksliukas. OS procesai iš vartotojiškos užduoties perspektyvos

O dabar apie kiekvieną procesą detaliau.

3.4.0 Įvedimas ir išvedimas

Ši tema yra gana aktuali. Savaime aišku, vykdant vartotojo užduotis, būtina turi turėti įvedimo ir išvedimo galimybes. Ši būtina sąlyga – tai reikalavimas projekto realizacijai. Kol kas buvo tenkinamasi gana miglotais terminais “laukiama įvedimo srauto” ar “eilutė perkelta į išvedimo srautą”. Kas slypi už šių žodžių? Mūsų modelyje už šių žodžių slypės vartotojo sąsaja. Nebus jokių tarpinių grandžių, tokių kaip klaviatūros portų nuskaitymai, ar virtualus monitorius ir panašių grynai techninių realios mašinos pusių. Bus laikoma, kad yra galimybė iš vartotojo sąsajos pasiųsti pranešimą (atlaisvinti resursą). Šis veiksmas atstos įvedimo srautą. Be to,

pasiųsta į išvedimo srautą eilutė, bus kokiu nors būdu atvaizduota vartotojo sąsajoje. Taigi, šie veiksmai yra atliekami tarsi “anapus” modelio. Vartotojo sąsaja – tai būdas stebėti ir įtakoti modelį. Įvedimo ir išvedimo srautai jungia modelį su vartotojo sąsaja.

Dabar bus pateikta viena iš galimų vartotojo sąsajos realizacijų.Tegu pagrindinis procesas StartStop bei virtualios mašinos turi langą.

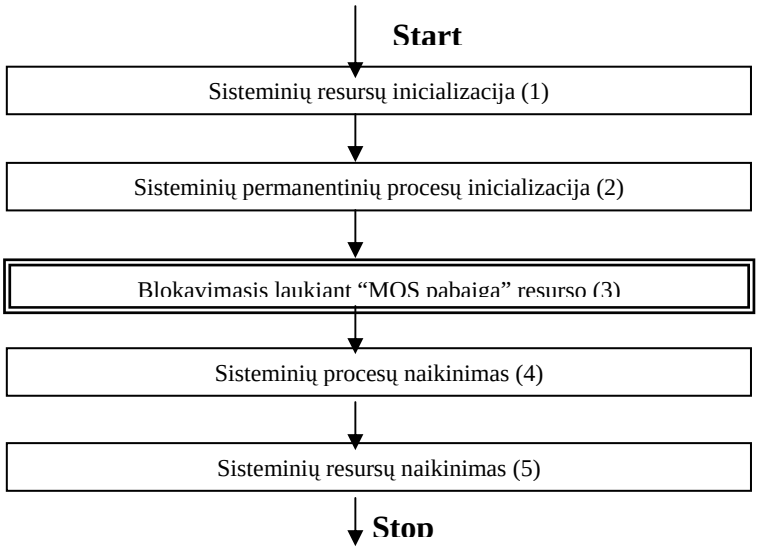
Langas – tai būdas atlaisvinti įvedimo resursą iš vartotojo sąsajos bei vieta, kur išvedama informacija, modelio pasiųsta į išvedimo srautą.Vienas iš lango pavyzdžių – konsolė, kurioje matomi kažkokie sistemos pranešimai, bei yra galimybė perduoti komandas sistemai. Lango sudedamosios dalys – komandinė eilutė ir išvedimų vieta. Komandinė eilutė leidžia įvesti bet kokią simbolių seką, kuri, paspaudus mygtuką <enter>, yra pasiunčiama į įvedimo srautą(atlaisvinamas resursas). Išvedimų vieta kaupia gautą iš išvedimo srauto informaciją.

StartStop langas – pagrindinis. Jame bus išvedami operacinės sistemos lygio pranešimai, bei įvedamos komandos sistemai (pavyzdžiui, paleisti tam tikrą vartotojo programą). Virtualios mašinos langai – tai būdas virtualiai mašinai gauti įvedimą iš išorės bei parodyti jos išvedamą informaciją.

Tai tik vienas iš daugelio galimų pavyzdžių, tačiau šis turėtų būti pakankamai paprastas.

3.4.1 Procesas StartStop

Šis procesas atsakingas už sistemos darbo pradžią ir pabaigą. Įjungus kompiuterį šis procesas pasileidžia automatiškai. Šio proceso paskirtis – sisteminių procesų ir resursų kūrimas. Šio proceso schema yra pateikta 3 diagramoje.



3 diagrama. Procesas StartStop

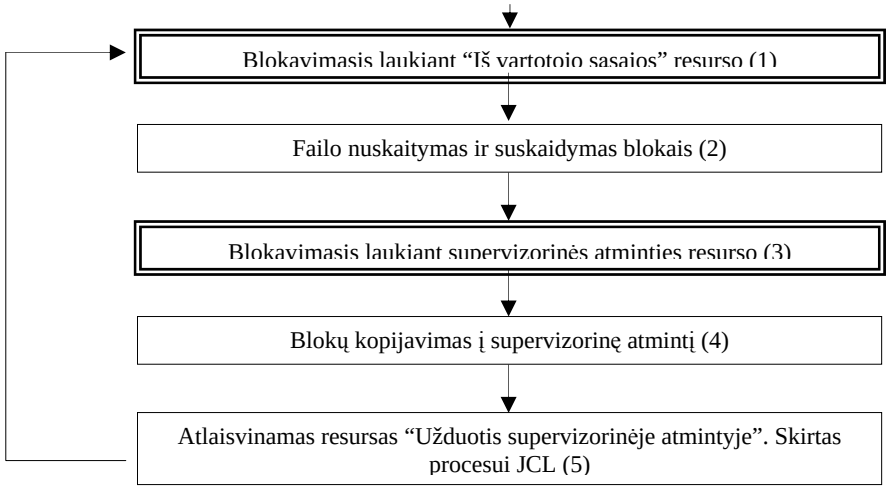
Procesas *StartStop*, gavęs procesorių, savo darbą pradeda sukurdamas visus sisteminius resursus (1). Laikoma, kad procesoriaus resurso kurti nereikia, jis kuriamas ir naikinamas įjungiant ir išjungiant kompiuterį. Sukūręs resursus, *StartStop* kuria permanentinius procesus (2), t.y. tuos procesus, kurie bus aktyvūs visą MOS gyvavimo laiką. Mūsų modelyje tokie procesai bus: *ReadFromInterface*, *JCL*, *JobToSwap*, *Loader*, *MainProc*, *Interrupt* ir *PrintLine*.

Sekantis *StartStop* etapas yra prašyti resurso “MOS pabaiga”(3). Šioje vietoje *StartStop* blokuojasi ir laukia kol bus atlaisvintas pranešimas apie MOS darbo pabaigą. Priklausomai nuo prioriteto anksčiau ar vėliau *StartStop* bus atblokuotas ir tęs darbą. Sekantis etapas yra sisteminių procesų naikinimas (4). Jeigu procesų primityvas “naikinti procesą” yra realizuotas korektiškai,

bus sunaikinti visi sistemoje esantys procesai (t.y. *StartStop* vaikų vaikai ir t.t.). Galiausiai yra naikinami sisteminiai resursai (5). Tuo MOS darbas ir pasibaigia.

3.4.2 Procesas ReadFromInterface

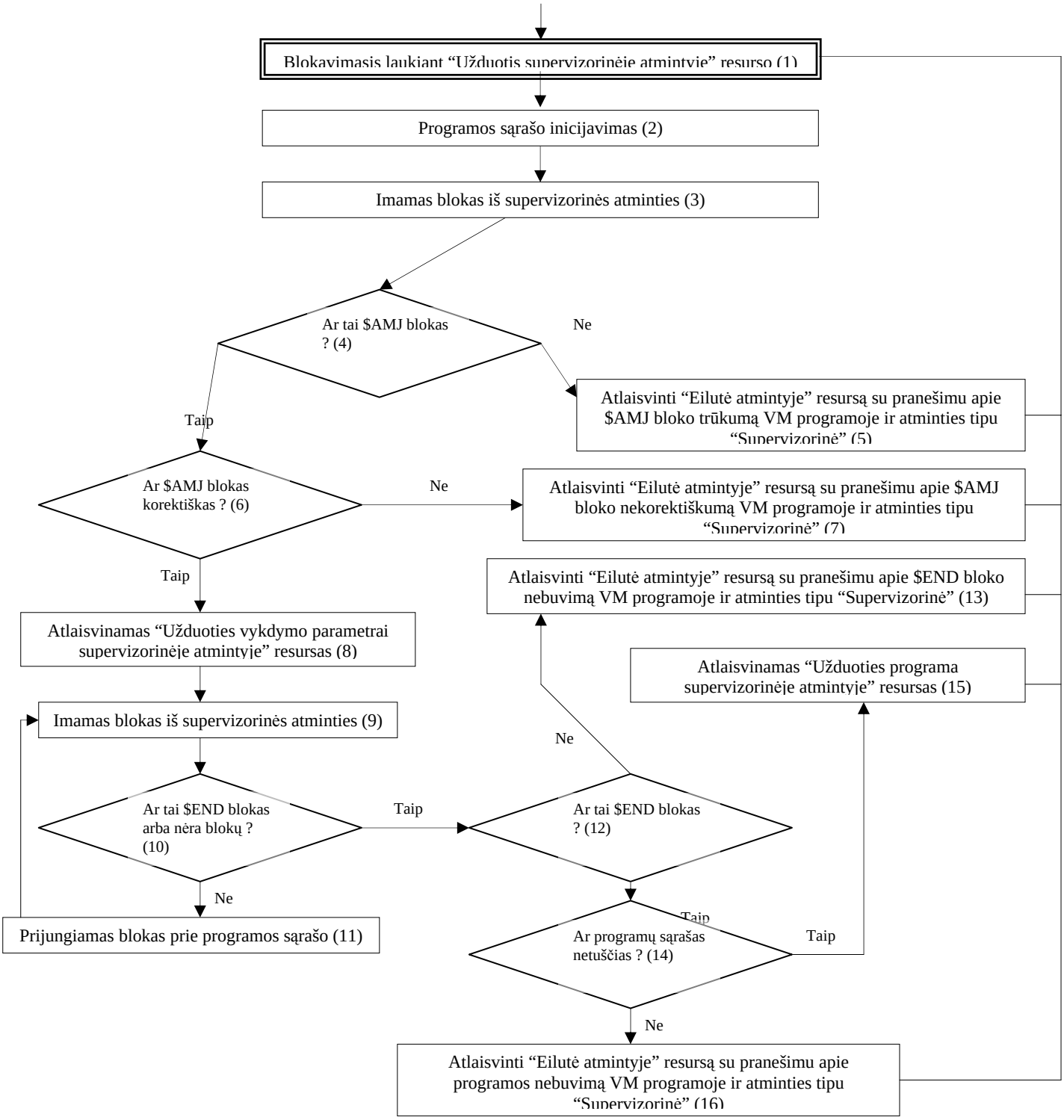
Šį procesą kuria ir naikina procesas *StartStop*. Šio proceso paskirtis – gavus informaciją iš įvedimo srauto ir atlikus pirminį jos apdorojimą, atiduoti informaciją tolesniam apdorojimui, kurį atliks procesas *JCL*. Proceso schema pateikta 4 diagramoje. *ReadFromInterface* pirmasis žingsnis – laukti įvedimo srauto (1). Realizuojant modelį reikėtų nuspręsti, kas įvedimo sraute turi ateiti. Realizuotame projekte įvedimo srautu ateina išorinio (modelio atžvilgiu) failo pavadinimas, kuriame ir yra vartotojiška programa.



4 diagrama. Procesas ReadFromInterface

Nors šiuo atveju vartotojiška programa atsiranda tarsi iš niekur (modelio išorės), tačiau realistiškesnis variantas reikalautų failų sistemos įvedimo modelyje. Taigi, antrasis etapas yra failo skaitymas ir skaidymas blokais (40 baitų). Apdorojimo rezultatas – blokų sąrašas yra laikomas supervizorinėje atmintyje, todėl mes jos prašome (3) ir kopijuojame blokus atmintin (4). Galiausiai yra sukuriamas ir atlaisvinamas resursas, skirtas procesui *JCL*, kuriame yra informacija apie blokų padėtį atmintyje (5). Proceso programa yra ciklinė, taigi, *ReadFromInterface* vėl užsiblokuoja laukdamas

3.4.3 Procesas JCL

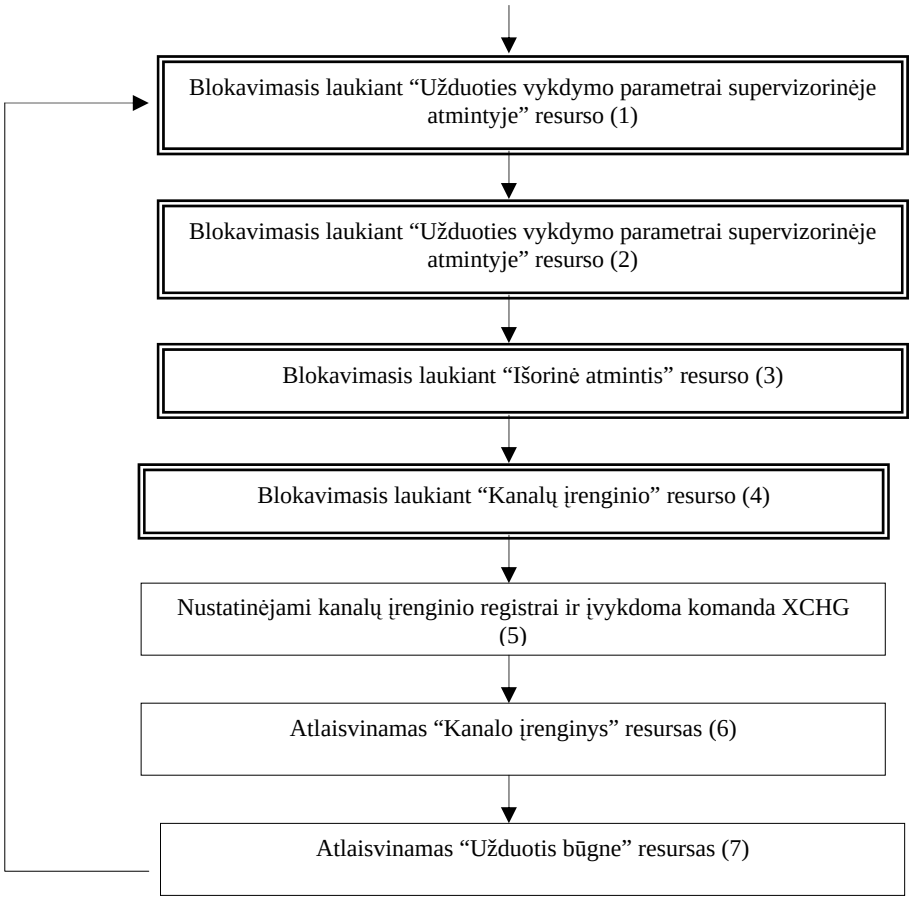


5 diagrama. Procesas JCL

Procesą JCL kuria ir naikina procesas *StartStop*. Proceso *JCL* paskirtis – gautus blokus iš *ReadFromInterface* suskirstyti į antraštės blokus ir programos blokus, ir atidavus juos procesui *JobToSwap*, vėl blokuotis laukiant pranešimo iš ReadFromInterface. Proceso schema pateikta 5

diagramoje. Procesas pradeda savo darbą blokuodamasis, laukdamas pranešimo iš *ReadFromInterface.* Sulaukus šio pranešimo, procesas pasiruošia darbui ir inicijuoja programos blokų sąrašą (2). Dabar reikėtų prisiminti skyrių apie užduoties formatą, ir tiesiog atskirti antraštės blokus nuo programos blokų, tačiau bus pateikta bendra schema. Imamas pirmas blokas. Žiūrima ar pirmi 4 baitai yra “$AMJ”. Jei ne, yra kuriamas ir atlaisvinamas resursas “Eilutė atmintyje”, su parametrais: eilute “Trūksta $AMJ bloko vartotojo programoje” ir informacijos, kad ši eilutė yra supervizorinėje atmintyje (5). Atlaisvinęs pranešimą procesas vėl blokuojasi (1) Jei $AMJ blokas yra, tikrinama ar $AMJ blokas yra korektiškas. Korektiškas – ar teisingas nurodytas maksimalus išvedimų skaičius ir ar yra vartotojiškos programos pavadinimas (6). Jei nekorektiška antraštė yra elgiamasi analogiškai (5) žingsniui, tik pranešimas yra apie nekorektišką $AMJ bloką. Jei $AMJ blokas yra korektiškas, atlaisvinamas programos antraštę, kaip parametrą, turintis resursas (8). Toliau yra pildomas programos blokų sąrašas tol, kol nesutinkamas $END blokas arba blokų nebėra (10,11). Šioje vietoje vėlgi atkreipkite dėmesį, į vartotojiškos programos struktūrą.Taigi, jei bloko $END nėra, elgiamasi kaip (5) žingsnyje tik keičiamas pranešimas į “Nėra $END bloko” (12,13). Jei programų sąrašas yra tuščias(14), vėl elgiamės kaip (5) žingsnyje, tik pranešimas bus - “Nėra vartotojo programos”(16). Jei programos blokų sąrašas nėra tuščias, atlaisvinamas resursas, skirtas *JobToSwap* procesui su nuoroda į programos blokų sąrašą (15). Dabar JCL vėl blokuojasi laukdamas pranešimo iš *ReadFromInterface* (1).

3.4.4 Procesas JobToSwap

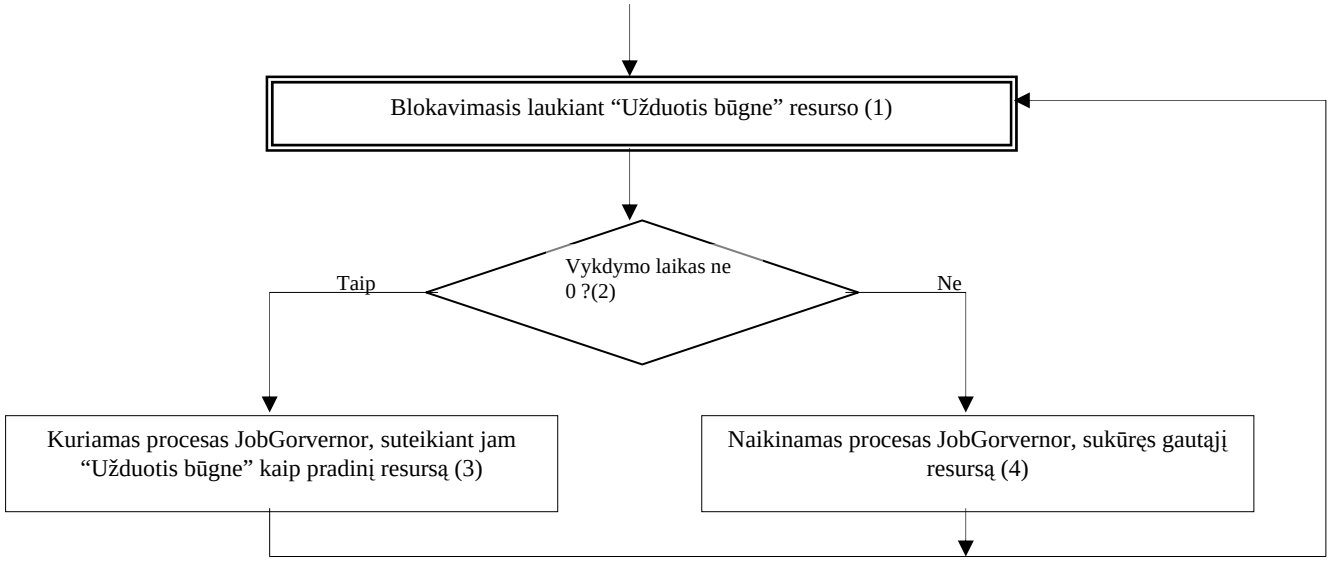


6 diagrama. Procesas JobToSwap

Šį procesą kuria procesas *StartStop*. Šio proceso paskirtis – perkelti užduoties programos blokus iš supervizorinės atminties į išorinę. Šio proceso schema pateikta 6 diagramoje. Procesas pradeda savo veiklą laukdamas pranešimo iš *JCL* proceso, kuriame bus programos antraštė(1), gavęs jį, laukia resurso su programos blokais. (2). Trečiu žingsniu procesas prašo tiek išorinės atminties takelių, kiek blokų yra vartotojo programoje (3). Šiame modelyje laikoma, kad jų yra dešimt. Toliau procesas blokuojasi laukdamas leidimo dirbti su kanalų įrenginiu (4), atblokuotas jis nustato kanalo duomenų perdavimo kryptį ir adresus bei įvykdo komandą XCHG (5). Tai kartojama kiekvienam blokui. Rezultate visa vartotojiška programa atsiduria išorinėje atmintyje.

Atlaisvinęs pranešimą leidžiantį dirbti su kanalu(6) jis visus atlaisvina resursą skirtą procesui *MainProc*(6), kuriame laikomi visi išorinės atminties takelių numeriai, kuriuose yra laikoma vartotojo programa. Toliau procesas kartoja (1) žinsnį.

3.4.5 Procesas MainProc



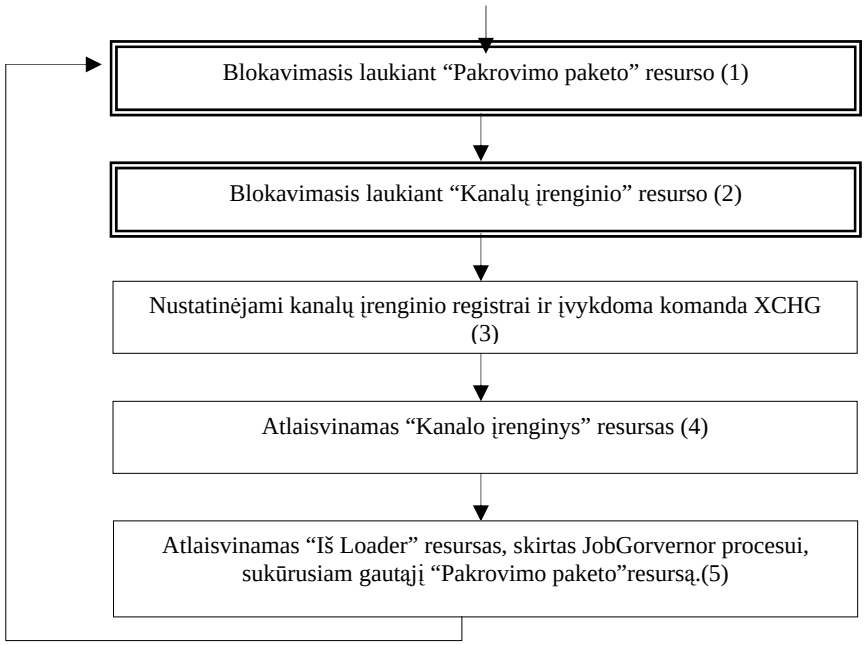
7 diagrama. Procesas MainProc

Procesą MainProc kuria ir naikina procesas *StartStop*. Šio proceso paskirtis - kurti ir naikinti procesus *JobGorvernor*.*MainProc* schemą galima rasti 7 diagramoje. Šis procesas pradeda savo veiklą prašydamas “Užduotis būgne” resurso, kurį atlaisvina procesas *JobToSwap.* arba *JobGorvernor*. Gavęs reikalingą resursą, *MainProc* žiūri ar jo vykdymo laikas nėra 0 (2). Nenulinė reikšmė – naujo *JobGorvernor* kūrimo ženklas (3), kai tuo tarpu nulinė reikšmė yra *JobGorvernor* naikinimo ženklas (4). Kurdamas JobGorvernor procesas suteikia jam “užduotis būgne” kaip pradinį resursą. Naikinamas būtent tas JobGorvernor, kuris atsiuntė “Užduotis būgne” resursą, su nuliniu vykdymo laiku. Atlikęs savo darbą MainProc vėl blokuojasi laukdamas “Užduotis būgne resurso”

3.4.6 Procesas Loader

Procesą *Loader* kuria ir naikina procesas *StartStop*. Šio proceso paskirtis – išorinėje atmintyje esančius blokus perkelti į vartotojo atmintį. *Loader* proceso schema yra pavaizduota 8 diagramoje.

Procesas *Loader* pradeda savo darbą laukdamas “Pakrovimo paketo”resurso (1). Pakrovimo pakete yra laikoma informacija apie išorinės atminties takelius, (ką reikės perkelti) ir vartotojo atminties takelius (kur reikės perkelti).Pakrovimo paketą šiame MOS modelyje siunčia *JobGorvernor* procesas. Gavęs šį resursą procesas blokuojasi laukdamas “Kanalų įrenginys” resurso (2). Toliau *Loader* dirba su kanalų įrenginiu. Nustatęs atitinkamus kanalo įrenginio registrus, jis įvykdo komandą XCHG kiekvienam takeliui (3). Vėliau “Kanalų įrenginio” resursas yra atlaisvinamas (4) ir sukuriamas bei atlaisvinamas pranešimas *JobGorvernor* apie darbo pabaigą. Pranešimas yra skiriamas būtent tam *JobGorvernor*, kuris atsiuntė “Pakrovimo paketo” resursą. Įvykdęs savo darbą procesas *Loader* vėl blokuojasi laukdamas “Pakrovimo paketas ” resurso.



8 diagrama. Procesas Loader

3.4.7 Procesas JobGorvernor

Procesą (galima sakyti ir procesus – jų vienu metu gali būti keli) *JobGorvernor* kuria procesas *MainProc*. Proceso *JobGorvernor* paskirtis – kurti, naikinti ir padėti procesui *Virtuali mašina* atlikti savo darbą. “Padėti – tai atlikti veiksmus, kurių *Virtuali mašina,* procesoriui dirbant vartotojo režimu, nesugeba atlikti. Vienas *JobGorvernor* aptarnauja vieną virtualią mašiną. Detali šio proceso schema yra 9-oje diagramoje.

Procesas *JobGorvernor* pradeda darbą laukdamas varotojo atminties (1), kur bus perkelta vartotojo užduoties programa.Prašoma tiek takelių, kiek yra blokų. Gavęs “Vartotojo atmintis” resursą procesas atlaisvina resursą “Pakrovimo paketas” (2), skirtą *Loader* ir blokuojasi laukdamas pranešimo apie *Loader* darbo pabaigą(3). Perkėlus vartotojo užduoties programą į vartotojo atmintį, reikia atlaisvinti išorinę atmintį, kur ji buvo iki šiol(4). Dabar paprašoma dar vieno takelio vartotojo atminties, kur bus laikoma virtualios mašinos puslapių lentelė (5). Dabar puslapių lentelė yra užpildoma išskirtosios vartotojo atminties adresais. Dabar yra pasiruošta virtualios mašinos kūrimui, taigi sekantis žingsnis yra kurti procesą *Virtuali mašina*. Dabar

JobGorvernor blokuosis(7) tol, kol negaus pranešimo iš proceso *Interrupt* apie įvykusį pertraukimą proceso *Virtuali mašina* metu.

Gavęs pranešimą apie pertraukimą, JobGorvernor stabdo procesą *Virtuali mašina*(8). Yra tikrinama ar tai įvedimo - išvedimo pertraukimas (9). Jei tai nėra įvedimo – išvedimo pertraukimas, naikinamas procesas "*Virtuali mašina*"(16), atlaisvinama jos užimta vartotojo atmintis (17) ir duodamas ženklas procesui *MainProc* apie tai, kad šis *JobGorvernor* jau baigė darbą ir turi būti sunaikintas. Ženklas - sukurtas ir atlaisvintas fiktyvus resursas "Užduotis būgne" su nuliniu vykdymo laiku (18). Kadangi šis procesas savo darbą jau baigė, tačiau dar nėra sunaikintas, jis blokuojasi laukdamas "Neegzistuojantis" resurso(19). Šio resurso *JobGorvernor* negaus, taigi jis blokuosis tol kol nebus sunaikintas.

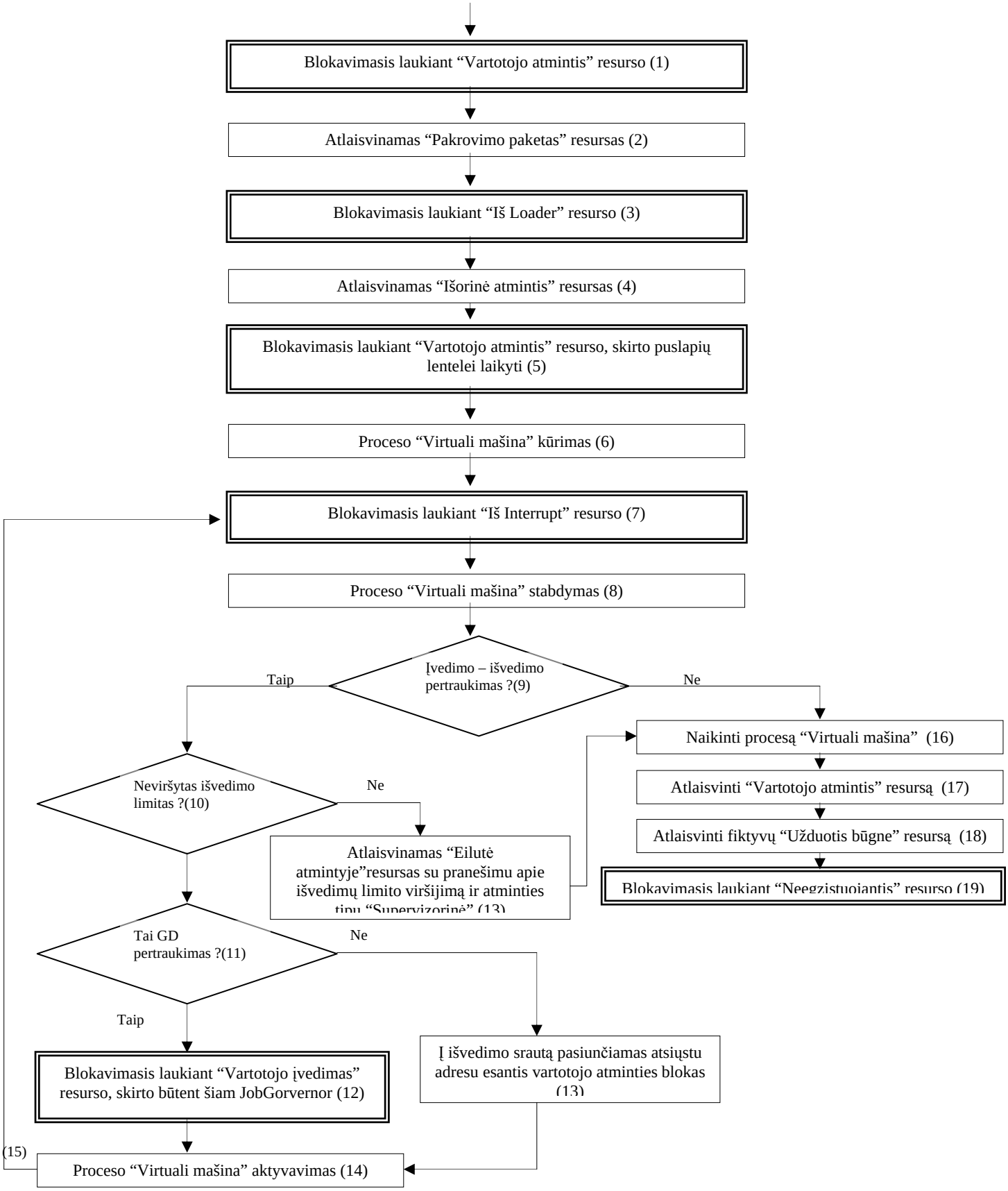
Anksčiau ar vėliau procesas *MainProc* gaus *JobGorvernor* atlaisvintą fiktyvų "Užduotis būgne" resursą ir sunaikins šį JobGorvernor procesą.

Jeigu patikrinimas (9) rodo kad tai yra įvedimo – išvedimo pertraukimas, yra tikrinama ar neviršytas išvedimo limitas (10). Limitą viršijus yra atlaisvinamas "Eilutė atmintyje"resursas su pranešimu apie išvedimų limito viršijimą ir atminties požymiu "Supervizorinė" (13). Toliau yra atliekami veiksmai apie kuriuos jau kalbėta(16,17,18,19). Neviršijus išvedimų limito yra tikrinama ar tai įvedimo pertraukimas(11). Jeigu tai įvedimo pertraukimas, procesas blokuojasi laukdamas "Vartotojo įvedimas"resurso,skirto būtent šiam *JobGorvernor*(12). Šioje vietoje *JobGorvernor* prašo resurso, kurį įves "anapus" modelio esantis vartotojas į virtualios mašinos langą. Taigi vartotojas mato, kaip jo programa laukia jo įvedimo, tuo tarpu vykdydama kitas užduotis.

Gavęs reikalaujamą resursą, procesas *Virtuali mašina* yra aktyvuojamas (14) ir procesas *JobGorvernor* ciklišk ai grįžta toliau blokuotis dėl "Iš interrupt" resurso (15,7).

Jei tai buvo išvedimo pertraukimas, analogiškai ankstesniems atvejams, naudodamiesi kanalų įrenginiu, pasiunčiame į išvedimo srautą informaciją, esančią vartotojo atmintyje(13). Reikalingas adresas ateina kartu su "Iš Interrupt" resursu – kaip vienas jo parametrų. Nors išvedimo srautas yra tik vienas, laikoma,kad modelio išorėje esantis mechanizmas "sugeba" rasti tą langą, kuriame bus išvedama ši informacija.

Atlikus šiuos veiksmus yra aktyvuojama virtuali mašina(15), ir procesas užsiblokuoja laukdamas "Iš Interrupt" resurso(15,7)

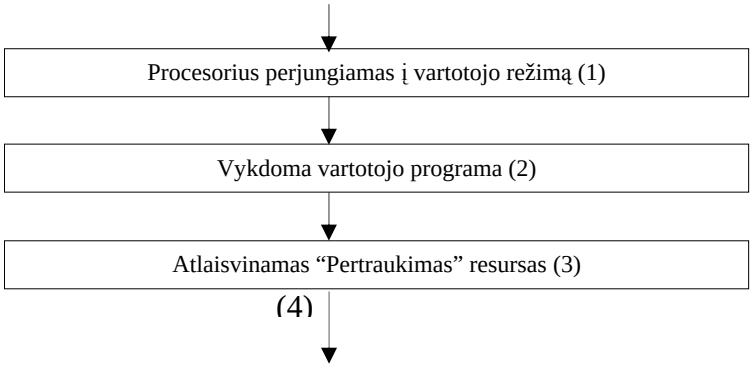


9 diagrama. Procesas JobGorvernor

3.4.8 Procesas Virtual Machine

Procesą *Virtuali mašina* kuria ir naikina procesas *JobGorvernor*.Virtualios mašinos paskirtis yra vykdyti vartotojo užduoties programą. Procesų *Virtuali mašina* yra tiek, kiek yra procesų *JobGorvernor*.

Virtualios mašinos darbo pradžioje procesorius yra perjungiamas į vartotojo režimą.(1). Vėliau vykdoma virtualaus procesoriaus komandų interpretatoriaus programa(2). Ši programa veikia tol, kol sistemoje neaptinkamas pertraukimas. Tada virtuali mašina išsaugoja savo procesoriaus būseną, perduodamas valdymas pertraukimą apdorosiančioms programoms.Apdorojus pertraukimą valdymui sugrįžus į virtualią mašiną, yra atlaisvinamas resursas “Pertraukimas”, skirtas procesui Interrupt.(3) Procesas virtuali mašina turi mažą prioritetą, todėl procesorių visų pirma gaus procesas Interrupt, kuris identifikuos pertraukimą ir perduos informaciją JobGorvernor, kuris atlikęs nustatytus veiksmus, priklausomai nuo situacijos arba aptarnaus arba sunaikins procesą “Virtuali mašina”

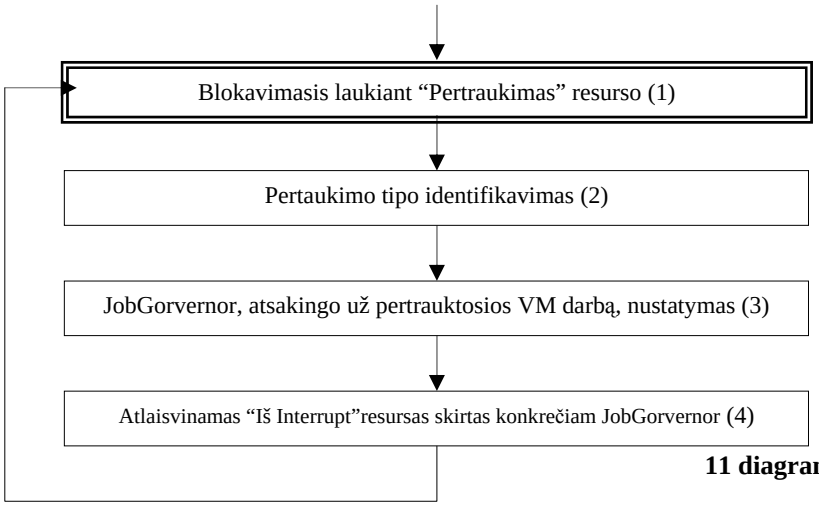


10 diagrama. Procesas Virtual Machine

3.4.9 Procesas Interrupt

Procesą *Interrupt* kuria ir naikina procesas *StartStop*. Šio proceso paskirtis – reaguoti į pertraukimus, kilusius virtualios mašinos darbo metu. Šio proceso schema pateikiama 11 diagramoje.

Procesas *Interrupt* savo darbo pradžioje laukia “Pertraukimas” resurso, kurį siunčia procesas virtuali mašina (1). Dabar procesas nustato pertraukimo tipą apklausinėdamas pertraukimo programų nustatytas sisteminių kintamųjų reikšmes(2).



11 diagrama. Procesas Iterrupt

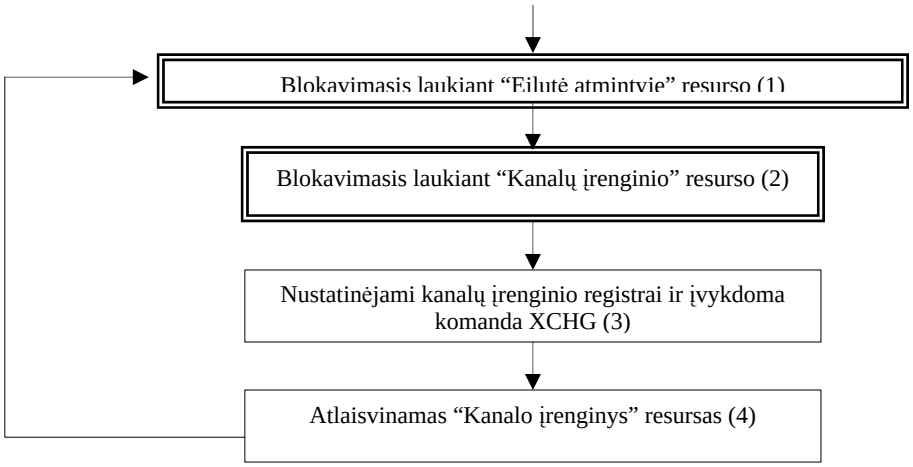
Galiausiai yra kuriamas ir atlaisvinamas “IšInterrupt” resursas, kuris yra skirtas nustatytajam *JobGorvernor* procesui(4). Toliau procesas ciklišKai blokuojasi vėl laukdamas”Pertraukimas” resurso.

Kiekviena virtuali mašina kūrimo metu laiko savo tėvo *JobGorvernor* identifikatorių.Kartu su pranešimu apie pertraukimą yra perduodamas ir tas identifikatorius, kurį *Interrupt* naudoja jam reikalingo *JobGorvernor* atskyrimui iš kitų (3).

3.4.10 Procesas PrintLine

Procesą *PrintLine* kuria ir naikina procesas StartStop. Šio proceso paskirtis – į išvedimo srautą pasiųsti kokioje nors atmintyje esantį pranešimą. Šio proceso schema pateikiama 12 –oje diagramoje.

Proceso darbas prasideda blokavimusi dėl “Eilutė atmintyje”resurso(1).Šis resursas turi parametrą, nusakantį iš kurios atminties reikės pasiųsti eilutę į išvedimo srautą, bei atminties adresą, žymintį bloko numerį.Toliau procesas blokuojasi laukdamas leidimo dirbti su kanalų įrenginiu (2), atblokuotas jis nustato kanalo duomenų perdavimo kryptį ir adresą bei įvykdo komandą XCHG (3). Įvykdęs komandą, procesas atlaisvina pranešimą leidžiantį dirbti su kanalu(4). Procesas ciklišKai blokuojasi pirmame savo žingsnyje – laukdamas “Eilutė atmintyje” resurso.



12 diagrama. Procesas PrintLine

4 Multiprograminės operacinės sistemos projekto realizacija

Ankstesniuose skyriuose mes apibrėžėme realią mašiną ir operacinės sistemos modelį, kuris padengia ją. Darbe nebuvo kreipiamas didelis dėmesys pačio projekto realizacijai, buvo stengiamasi suteikti bendrą atskirų operacinės sistemos dalių veikimo schemą. Kiek stipriau buvo paliestas procesų rinkinys. Taigi galimos įvairios šio modelio realizacijos. Tai jokiu būdu nėra trūkumas. Visos šio projekto realizacijos bus panašios savo realia mašina, resursais ir procesais. Bet ne jų veikimu !

Autorius realizavo operacinės sistemos modelį remdamasis šiuo projektu. Buvo pasirinkta aukšto lygio programavimo aplinka - Delphi. Tačiau neišvengta ir žemo lygio programavimo atliekant pavyzdžiui valdymo perdavimo veiksmus. Delphi - objektiškai orientuota programavimo aplinka. Taigi kiekvienas operacinės sistemos objektas buvo realizuotas Delphi klase. Aukšto lygio programavimo aplinkos dėka, modelio realizacija yra labai aiški, logiška,lengvai suprantamas kodas bei daugiau galimybių turinti vartotojo sąsaja..

Žvelgiant į modelį iš realizacijos pusės, patogu jį suskirstyti į modulius:

•Operacinės sistemos branduolys. Šiame modulyje laikomos visos sistemos branduolio klasės, tokios kaip procesai, resursai, procesų sąrašas, resursų sąrašas, elementas ir t.t. Šiame modulyje būtų realizuoti resursų paskirstytojai bei planuotojas.

•Reali mašina. Šiame modulyje yra laikomos klasės, imituojančios kompiuterio aparatūrą - procesorius, vartotojo atmintis, išorinė atmintis ir kitos.

•Konkrečių resursų ir procesų aprašų bei procesų programų modulis. Šiame modulyje laikomos modelyje minėtų procesų realizacijos bei resursai. Sistemos branduolyje yra laikomi tiek procesų tiek resursų šablonai - pagrindinės klasės.

•Vartotojo sąsajos modulis. Papildomas modulis, kuris yra skirtas patogiam MOS modelio stebėjimui.

Dabar apie kiekvieną modulį kiek plačiau.

4.1 Realios mašinos realizacija

Šiame modulyje yra realizuotos visos realios mašinos klasės. Kiekvienas realios mašinos komponentas yra atstovaujamas klasės šiame modulyje. Čia galima rasti tokias klases:

- *TRealMachine* – pagrindinė klasė – virtualus kompiuteris, apjungiantis visas kitas realios mašinos klases
- *TChannelChip*-klasė, realizuojanti kanalų veikimą. Realizacija atspindi
- *2.10 Kanalų* įrenginys skyrelyje nagrinėtą kanalų įrenginį
- *TProcessorList* – klasė, realizuojanti procesorių sąrašą.Leidžia realiai mašinai turėti daugiau nei vieną procesorių.
- *TProcessor* – klasė, realizuojanti procesorių. Realizacija remiasi *2.2 Realios mašinos centrinis procesorius* skyrelyje nagrinėta medžiaga. Procesoriaus komanda *Translate* paverčia virtualų adresą realiu.
- *TUserMemory* – klasė, realizuojanti varotojo atmintį. Realizacija remiasi *2.5 Realios mašinos atmintys* skyrelyje išdėstyta medžiaga.
- *Išorinė atmintis* realizuota failu modelio išorėje. Kanalų įrenginys, naudodamasis aukšto lygio priemonėmis skaito ir rašo į šį failą.

4.2 Branduolio realizacija

Šiame modulyje galima rasti visas OS koncepcijas, realizuotas klasėmis.Visų šių klasių aprašus galima rasti skyreliuose:*3.1.5 Proceso ir su juo susijusių klasių aprašai* ir *3.2.3 Resurso ir su juo susijusių klasių aprašai.*

TKernel – klasė realizuojanti OS branduolį. Čia laikomi procesų, resursų sąrašai ir kitas OS funkcijas atliekančios klasės.Branduolyje yra realizuotas planuotojas kaip atskira procedūra.

TProcessList – procesų sąrašą realizuojanti klasė.Verta pastebėti, kad šis sąrašas yra rūšiuojamas pagal procesų prioritetus. Procesas su aukščiausiu prioritetu visada bus sąrašo pradžioje.

TProcess – bendrą proceso aprašą realizuojanti klasė. Proceso aprašas jau buvo išnagrinėtas

TResourceList – resursų sąrašą realizuojanti klasė. Resursai sąraše nėra rūšiuojami.

TResource – bendrą resurso aprašą realizuojanti klasė. Resurso aprašas taip pat jau buvo išnagrinėtas

TResElement – bendrą resurso elemento aprašą realizuojanti klasė.

TElementList – resurso elementų sąrašą realizuojanti klasė. Elementai nėra rūšiuojami.

4.3 Konkrečių procesų ir resursų aprašų bei procesų programų realizacija

Šiame modulyje laikomos jau konkrečios procesų ir resursų bei jų elementų aprašų klasės. Kartu šiame modulyje laikomos procesų programos. Realizuotas projekto *3.4 Procesų paketas*

skyrelyje išnagrinėtas procesų rinkinys. Procesai: *TStartStop, TReadFromInterface, TJCL, TJobToSwap, TMainProc, TLoader,TJobGorvernor, TVirtualMachine, TInterrupt* ir *TPrintLine*. Resursų sąrašas:

- MOSEnd: Sends: *Interface*,Gets: *StartStop*
- TaskInSupMemory - Sends: *ReadFromInterface*,Gets: *JCL*
- TaskParamSends: *JCL*, Gets: *JobToDrum*
- StrInSupMemory: Sends: *JCL*, Gets: *PrintLines*
- TaskProgInSupMemory: Sends: *JCL*, Gets: *JobToDrum*
- ExtMemory: Gets: Various, Frees: Various
- TaskInDrumSends: *JobToDrum*, JobGorvernor, Gets: *MainProc*
- InterEvent: Sends: *VirtualMachine*, Gets: *Interupt*
- UserMemory:Gets:Various, Frees: Variuos
- LoadPackage: Sends: *JobGorvernor*,Gets: *Loader*
- FromLoader: Sends: *Loader*,Gets: *JobGorvernor*
- Unreal: Sends: None Gets: *JobGorvernor*
- InterToGov: Sends: *Interupt*, Gets: *JobGorvernor*
- ToReadInCards: Sends: *Interface*, Gets: *ReadFromInterface*
- ChannelSemaphore: Sends: Various,Gets: Various

Procesų programos realizuotos remiantis projekte nagrinėtais procesų algoritmais.

4.4Vartotojo sąsajos realizacija

Šis modulis yra atsakingas už projekto realizacijos vartotojo sąsają.Vartotojo sąsaja yra modelio išorėje, tačiau kartu tai yra būdas stebėti ir įtakoti modelį (žr. 3.4.0 Įvedimas ir išvedimas). Šio modelio realizacija yra tik šiek tiek paliesta projekte, taigi kiekviena nauja projekto realizacija gali turėti skirtingą vartotojo sąsają.Autoriaus šiame modulyje realizavo pagrindinį langą, vartotojo užduočių langus, modelio valdymo mechanizmą (stabdyti, pažingsniui, aktyvuoti) bei sisteminių pranešimų rodymo mechanizmas, virtualaus kompiuterio būsenos stebėjimo mechanizmas.

Tačiau šio modulio realizacija nėra apibrėžta modelyje, todėl galimos įvairios šio modulio versijos.

Išvados

Vienas šio darbo tikslų - multiprograminės operacinės sistemos modelio projektavimas ir realizavimas. Darbą autorius laiko mokomąja priemone, todėl buvo stengiamasi dėstyti temą ne citatomis iš knygų, bet iš savo patirties vedant OS pratybas. Šio darbo prioritetai: temų nuoseklumas ir ne visiškai formalus bendravimas su skaitytoju.

Projekto kūrimo eigoje buvo apibrėžta reali mašina ir multiprograminė operacinė sistema, padengianti šią mašiną. Šiame projekte nebuvo gilinamasi į iškilusių problemų realizaciją.Buvo pateikiamas sprendimo būdas ir jo motyvacija, taigi projekto realizacija turėtų būti laisva.

Nuo pat projekto užuomazgų autorių lydėjo viena problema - riba tarp modelio ir realios operacinės sistemos. Modelis turi būti paprastas ir aiškus, ir tuo pačiu realus - kitaip jis neteks prasmės.

Autoriaus nuomone, projektas buvo realizuotas gana lanksčiai. Keletas temų šiame modelyje liko nepaliestos, tačiau palikta vietos jo tobulinimams ar išplėtimams, kurie gali būti gana įvairūs.

Norint paskatinti studentą ruošti įvairesnius projektus, priede buvo pateiktos kelios alternatyvos, kurias išplėtojus ir įtraukus į projektą, pateikiama MOS įgauna naujų savybių arba tiesiog pakeičiamas funkcionalumas.

Literatūros sąrašas

[Mit] Antano Mitašiūno paskaitų ciklas, 2000.

[Tan92] Andrew S. Tanenbaum. Modern Operating Systems, pp.5

PRIEDAS NR.1 – Alternatyvos

Alternatyvos yra skiriamos tiems studentams, kurie ruošdami savo MOS projektą norėtų kažkuo išsiskirti nuo šabloninio darbo, tačiau nelabai įsivaizduoja ką ir kaip galima pakeisti. Čia bus pateikta keletas alternatyvų, kurios parodys, kad modifikacija iš principo yra įmanoma ir netgi pageidautina, ir ją riboja tik jūsų fantazija, žinios ir žinoma realizacijos galimybės .

1 Alternatyvus aparatūros apibrėžimas

Ši alternatyva iš esmės liečia tik procesorių bei atminties modelį. Kaip pagrindu bus naudojamasi Šou modeliu – būtent jo atminties bei procesoriaus apibrėžimai bus modifikuoti.

Taigi pereikime prie problemos. Tarkime turime Šou apibrėžtą aparatūrą. Ji popieriuje (ar jos realizacija) gerai veikia ir ją kažkas intensyviai naudoja. Buvo rašomos aplikacijos mūsų virtualiai mašinai. Ir tarkime iš naudotojų pusės buvo išreikštas pageidavimas padidinti virtualios atminties erdvę. Jiems nepakanka suteiktos adresinės erdvės. Ir iš tikro, baziniame Šou modelyje į pateikiamą 100 žodžių dydžio erdvę programuotojas turi sutalpinti savo programos tekstą (kodą), likusią erdvę naudoti kaip duomenų sritį. Taigi didinsime virtualios mašinos atmintį ir stebėsim kas keisis, bei kas tampa neefektyvu. Verta paminėti, kad siūlomi pakeitimų variantai yra vieni iš galimų sprendimų, tačiau toli gražu ne vieninteliai.

Šou standartiniame modelyje virtuali atmintis yra 100(adresuojama 00..99) žodžių dydžio. Reali atmintis – 300 žodžių. Tarkime praplėsime realią atmintį iki 30000 žodžių. Virtualią atmintį apibrėžkime 10000 (adresuojama 0000..9999)dydžio. Kadangi mūsų sistemoje adreso žodžio baito reikšmių aibė yra 10 dešimtainių skaitmenų (0..9) - tai naujam virtualiam adresui nurodyti reikia 4 baitų (anksčiau buvo 2 baitai). Taigi galima pastebėti, kad keisis aparatūros apibrėžimas (pvz. procesoriaus registrų dydis). Dabar detaliau apie tai :

Alternatyvus procesorius

Šiame skyrelyje tiesiog peržvelgsime mūsų naujojo procesoriaus registrus

- PLR – **6** baitų puslapių lentelės registras. Pasikeitė šio registro dydis. Detaliau skyrelyje apie puslapiavimo mechanizmą.
- R – **6** baitų bendro naudojimo registras. Pasikeitė registro dydis. Vėlesniame skyrelyje apie procesoriaus komandas paaiškės kodėl.
- IC – **4** baitų virtualios mašinos programos skaitiklis. Pakito dydis dėl anksčiau minėtų priežasčių(norint nurodyti virtualų adresą, būtini 4 baitai, nes reikšmių aibė yra nuo 0000 iki 9999).
- C – 1 baito loginis (reikšmės – true “T” arba false “F”) trigeris. Šio registro apibrėžimo nereikia keisti.
- MODE – registras, kurio reikšmė nusako procesoriaus darbo režimą (vartotojas, supervizorius). Apibrėžimo keisti nereikia.
- PI – programinių pertraukimų registras. Apibrėžimo keisti nereikia.
- SI – supervizorinių pertraukimų registras. Apibrėžimo keisti nereikia.
- TI – taimerio registras. Apibrėžimo keisti nereikia.

Realios bei virtualios atminties apibrėžimų pokyčiai

Taigi kai iškėlėme pačią problemą, mes jau šiek tiek ir apibrėžėme naujus atminčių parametrus. Visų pirma padidinsim atminties žodį iki 6 baitų. Tai daroma atsižvelgiant į ateitį – jei 4 baitai skiriami adresavimui, tai dar bent 2 reikėtų skirti operacijos kodui. Taigi apsistokime ties 6 baitais. Toliau viskas panašu – blokas iš 10 žodžių, taigi visą atmintį – 30000(sunumeruoti 00000..299999) žodžių galima suskirstyti į 3000 blokų. Virtuali atmintis, kaip jau minėta anksčiau,bus sudaryta iš 10000(0000..9999) žodžių. Pradžiai atminties apibrėžimų pokyčių tiek.

Alternatyvaus procesoriaus komandos

Kaip jau minėta ankstesniame skyrelyje, atminties žodis bus 6 baitų. Laikysime kad tuose 6 baituose gali būti arba duomenys arba instrukcija procesoriui. Jei tai duomenys – viskas analogiškai Šou modeliui. Jei tai instrukcija procesoriui – jos bus kiek kitokios. Jei tiksliau tai padidės adresinė dalis – iki 4 baitų. Taigi jų iš naujo perrašinėti nėra prasmės, tiesiog pateiksiu vieną komandą kaip pavyzdį:

- **LR$x_1x_2x_3x_4$**. Atminties ląstelės, kurios adresas **x_1*1000 +x_2*100 + x_3*10 + x_4** turinio kopijavimas į registrą R. Trumpiau: R:= [**x_1*1000 +x_2*100 + x_3*10 + x_4**]

Puslapiavimo mechanizmo pokyčiai

Įvesti pakeitimai tiesiogiai atsiliepia ir puslapiavimo mechanizmui. Visų pirma apie PLR registrą. Anksčiau tik du paskutiniai baitai buvo naudojami adresui nurodyti, dabar tam bus naudojami 4 paskutiniai baitai. Pirmi 2 baitai nurodo kiek puslapių išskirta virtualios mašinos puslapių lentelei.

Taip pat iškyla ir puslapių lentelės dydžio problema. Jeigu virtualios atminties dydis – 10000 žodžių, t.y. 1000 blokų, tai akivaizdu kad puslapių lentelė turi talpinti iki 1000 žodžių - blokų adresų. 1000 žodžių galima sutalpinti į 100 blokų. Taigi pirmam klausimui atsakymą radom – puslapių lentelė gali siekti iki 100 blokų. Dabar keletas pastabų:

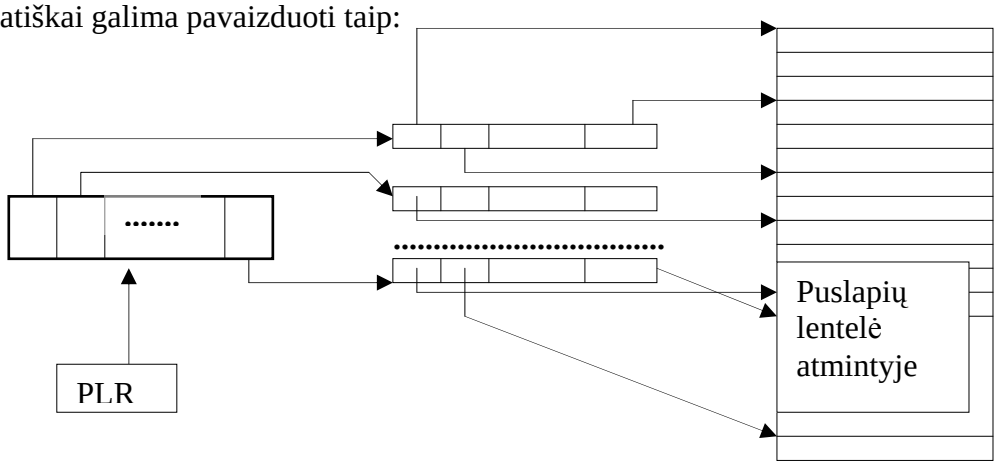
1. Programos virtualiai mašinai ne visada užima visą joms skirtą atmintį. Taigi ne visada verta laikyti tokią milžinišką(100 blokų) puslapių lentelę atmintyje. Verta iš anksto apgalvoti kaip programuotojui leisti apriboti programai išskiriamą atminties kiekį.
2. PLR registras saugo puslapių lenteles adresą. Šiuo atveju puslapių lentelė nėra vienas blokas – jų visas 100. Taigi reikia spręsti problemą kaip vieno adreso pagalba pasiekti informaciją apie 100 blokų adresus.

Dėl **pirmos pastabos** – matyt yra verta įvesti tokias sąvokas kaip segmentai. Galbūt verta pasirinkti patį paprasčiausią modelį – kodų segmentui skirti tiksliai tiek blokų kiek reikia (priklausomai nuo programos dydžio), o duomenų segmento dydį nurodyti papildoma instrukcija virtualios mašinos užduotyje. Lieka vienintelė problema – kaip programuotojui nustatyti duomenų segmento pradžią, kad jis galėtų adresuoti atmintį duomenų srityje.

Galbūt paprasčiausias sprendimas būtų dar viena speciali vartotojo užduoties instrukcija, nurodanti, kad duomenų segmentas prasideda nuo tam tikro bloko. Žinoma šis metodas prastas savo nelankstumu – programuotojas turi sekti kad neprirašytu programos kodo tiek, kad kodų segmento dydis susikirstų su nurodyta duomenų segmento pradžia. Taip pat programuotojui gal kiek neįprasta būtų sekti kad nepanaudotų adreso kuris mažesnis nei nustatyta duomenų segmento pradžia.

Kitas sprendimas(geresnis bet sudėtingesnis) būtų pridėti kelis naujus procesoriaus registrus. Duomenų segmento registrą ir kodo segmento registrą. Naudojimasis jais turbūt akivaizdus, tačiau jų realizacija reikalauja papildomo darbo. Jei įdomu, galima vystyti šią sritį toliau.

Dabar apie **antrą pastabą.** Pats paprasčiausias sprendimas būtų laikyti PLR registre informaciją apie pirmąjį puslapių lentelės bloką. Toliau nuosekliai išdėstyti visus 100 blokų, kuriuose būtų nuorodos į tuos 1000 programai išskirtų blokų. Tačiau tai yra labai netaupu. Jei programa nedidelė ir nereikalauja daug atminties vykdymo metu, tai galbūt jai užtektų žymiai mažiau blokų. Taigi ir puslapių lentelė būtų mažesnė. Tuo tikslu verta PLR registro pirmuosiuose 2 baituose laikyti puslapių lentelės naudojamų blokų skaičių. Tačiau išlieka viena problema. Išskirti nuosekliai einančius tarkim kad ir 50 blokų (jei ne visą 100) yra labai problematiška ir nenaudinga(vartotojui juk nesvarbu kokia tvarka MOS sudės tuos blokus). Taigi galima pasirinkti kiek sudėtingesnį adresavimo būdą. Tarkime PLR registro paskutiniuose 4 baituose laikomas adresas rodo į bloką, kurio kiekvieno žodžio paskutiniuose 4 baituose yra laikomi adresai blokų kuriuose ir yra nuorodos į programos naudojamus puslapių lentelės blokus. Schematiškai galima pavaizduoti taip:



Taigi kaip matyti turėdami adresą PLR registre kreipiamės į pagrindinį bloką, kurio elementai atitinkamai žymi programos naudojamu blokų dešimtis. Pasiėmę atitinkamą blokų dešimtį apibūdinančio bloko adresą galime kreiptis jau į konkretų puslapių lentelės bloką. O turint puslapių lentelės bloką, mes pasiekiame realų bloką, su kuriuo jau galima dirbti. Naudojant šį metodą, kaip matyti, nėra reikalaujamas blokų nuoseklumas, taipogi lengva išskyrinėti ir valdyti mažesnį nei maksimalų blokų skaičių.

Pavyzdys 2 pastabos antram sprendimui. Tarkime PLR registro reikšmė yra 504555. Iš pirmų dviejų baitų nustatome kad puslapių lentelės dydis yra 50 blokų. Paėmę trečią baitą, kreipiamės į pagrindinio bloko 3 elementą (skaičiuojant nuo 0). Paėmę ten esantį adresą keliaujam į dešimčių bloką. Ten naudojamės PLR 4-uoju baitu. Sužinome mūsų konkretaus puslapių lentelės puslapio adresą. Pasiekę jį naudojamės 5-uoju baitu ir gauname mums reikalingo bloko realų adresą. Ten pasinaudoję 6 baitų pasiekiame mus dominusį konkretų žodį.

2 Alternatyvūs procesų rūšiavimo (ar naujo einamojo) parinkimo metodai

Šou modelyje naudojamas prioritetinis procesų rūšiavimo modelis su tam tikrais Round Robin algoritmo elementais. Nors šis metodas yra mūsų MOS pilnai pakankamas, tačiau įdomumo dėlei galima įvesti kiek sudėtingesnį modelį. Apie keletą variantų labai trumpai, tiesiog bendrus principus.

Daugelio eilučių modelis

Šis modelis pasižymi tuo, kad procesui, kuris nuolat išnaudoja jam skirtą procesoriaus laiką, to laiko suteikiama vis daugiau. Šou modelyje laiko procesui ribojimui naudojamas TI registras, kurio reikšmė įvykdžius eilinę komandą yra mažinamas tam tikru dydžiu. Tuomet kai TI tampa lygus nuliui laikoma kad procesas išnaudojo jam skirtą laiko limitą.

Taigi šiuo atveju startuojant procesui jam skiriamas normalus laiko limitas. Pasibaigus pirmam virtualios mašinos vykdymo laiko limitui ir vėliau jai vėl gavus procesorių TI registras jau inicializuojamas 2 kartus didesne reikšme. Šiuo atveju TI = 20. Kitą syki bus inicializuojama TI = 40 ir t.t.

Laikoma kad šis metodas yra efektyvus, nes pasitaikius daug procesoriaus reikalaujančiai programai (pvz. įvairūs skaičiavimai) ji bus įvykdyta greičiau, rečiau bus kviečiamas planuotojas. Žinoma iškyla kita problema – nukenčia vartotojo interaktyvumas su sistema. Nes pasiekus dideles TI reikšmes, ilgai bus vykdomas skaičiavimo procesas ir kita užduotis, reikalaujanti didelio interaktyvumo su vartotoju vartotojui gali pasirodyti “per lėtai reaguojanti”. Tai galima išspręsti taip: kiekvieną kartą didinant TI dvigubai, mažinti to proceso prioritetą. Taip pat TI galima didinti nebūtinai dvigubai arba didinti iki tam tikros ribos.

Valdymas prieš mechanizmą

Pagrindinė šio metodo idėja – galimybė vartotojo užduočiai įtakoti planuotojo veiksmus. Paprasti planuotojo algoritmai dažnai neturi galimybių būti nors minimaliai valdomi programos vykdymo metu. Taigi jeigu vartotojo užduotis turi galimybę kurti procesus – vaikus, ir žino kad tarkim pirmas procesas - vaikas yra svarbesnis ir reikalauja daugiau procesoriaus nei antrasis, turėtų būti galimybės tam realizuoti. Norint įgyvendinti šį procesą šou modelyje, jei jame būtų galimybės kurti procesus – vaikus, užtektų pridėti galimybę nustatyti proceso – vaiko prioritetą.

3 Trumpos pastabos

Tiesiog norisi priminti kad visos šios alternatyvos yra pateikiami kaip pavyzdžiai, norint parodyti, kad tikrai įmanoma, įdomu ir naudinga keisti standartinį projektą. Tačiau reikia atsiminti, kad kai kurie pakeitimai turi tiesioginę įtaką kitoms projekto dalims (pvz. pakeitus aparatūros apibrėžimus keisis ir OS dalis), todėl reikia atidžiai sekti visus atliekamus pokyčius.