

2 skyrius. Užduotis modelinės multiprograminės operacinių sistemų projektui

2.1. Modelinio projekto tikslas

1 skyriuje buvo suminėti pagrindiniai multiprograminės operacinių sistemų bruožai ir funkcionavimo principai. Pastarojo skyriaus paskirtis - suprojektuoti MOS modelį, kuris bent iš dalies iliustruotų 1 skyriuje įvestas sąvokas ir leistų įsigilinti į MOS vidinę sandarą.

Kodėl pasirenkamas tokis MOS gilesnės analizės metodas, c ne, sakykime, nagrinėjama egzistuojančios operacinių sistemos, pvz., aprašytos [1], architektūra? Autoriaus nuomone, kiekviena konkreči realizacija turi keletą esminių trūkumų, trukdančių abstraktesniam analizei.

Pirma, konkreči OS dažnai labai susijusi su skaičiavimo mašinos, kuriai ji skirta, architektūra, konfigūracija ir kt. Todėl, norint studijuoti tokią OS, reikia susipažinti ir su konkrečia ESM, kas keliamą uždavinį daro sudėtingesnį.

Antra, skaičiavimo mašinos specifika neišvengiamai salygoja ypatingų bruožų atsiradimą operacinių sistemoje. Dažniausiai į pradinę - "gryną" OS koncepciją įvedamos naujos, nenumatytos galimybės arba apribojimai, kurių kartais neįmanoma paaiškinti OS ribose. Tokie papildomi bruožai dažniausiai užgožia esmines OS savybes.

Trečia, egzistuojančios OS yra skirtos profesionaliam naudojimui. Todėl jos yra visapusiškai sudėtingos, jų vidinė sandara yra gerai "paslėpta", o vykstantys procesai sunkiai pastebimi.

Šiais atžvilgiais MOS modelio projektavimas žymiai primtinesnis. Visų pirmą, atsiribojama nuo konkrečios skaičiavimo sistemos, aparatūriniu pagrindu pasirenkant abstrakčią mašiną, turinčią tik esmines komponentes ir savybes. Tuo būdu visą dėmesį galima sutelkti į OS. Antra - mašinos abstraktumas salygoja ir projektuojamos MOS universalumą, ty. ir čia išvengiama didesnės specifikos ir pasilikama prie bendresnių, esminių multiprograminės sistemos bruožų. Neatsiranda jokių nepateisinamų išplėtimų ar apribojinių. Trečia, kadangi anlizuotojas pats yra ir projektuotojas, jis kurs MOS modelį taip, kad Jame būtų aiškiai iliustruojamos pageidaujamos savybės, galės įkūnyti projekte ir jo realizacijoje norimą jų rinkinį. Be to atsiranda eksperimento laisvė: galima tobulinti projektą, išbandyti Jame įvairius algoritmus, tikrinti jų efektyvumą ir panašiai.

Autorius pagrindu pasirenka A. Šou siūlomą MOS modelį [6], kuris yra paprastas ir kartu atitinka aukščiau išdėstytaus reikalavimus. Bendra tokio projektavimo strategija - sukurti suabstraktintos mašinos modelį realios ESM bazėje ir tam modeliui sukonstruoti MOS.

2.2. Virtuali mašina

Jau buvo minėta, kad tai, su kuo dirba vartotojas (kas vykdo jo užduotį), yra menama - virtuali - mašina. Jos savybės - svarbiausios vartotojui. Paméginsime apibréžti esminius reikalavimus, kuriuos turi tenkinti MOS palaikoma virtuali mašina.

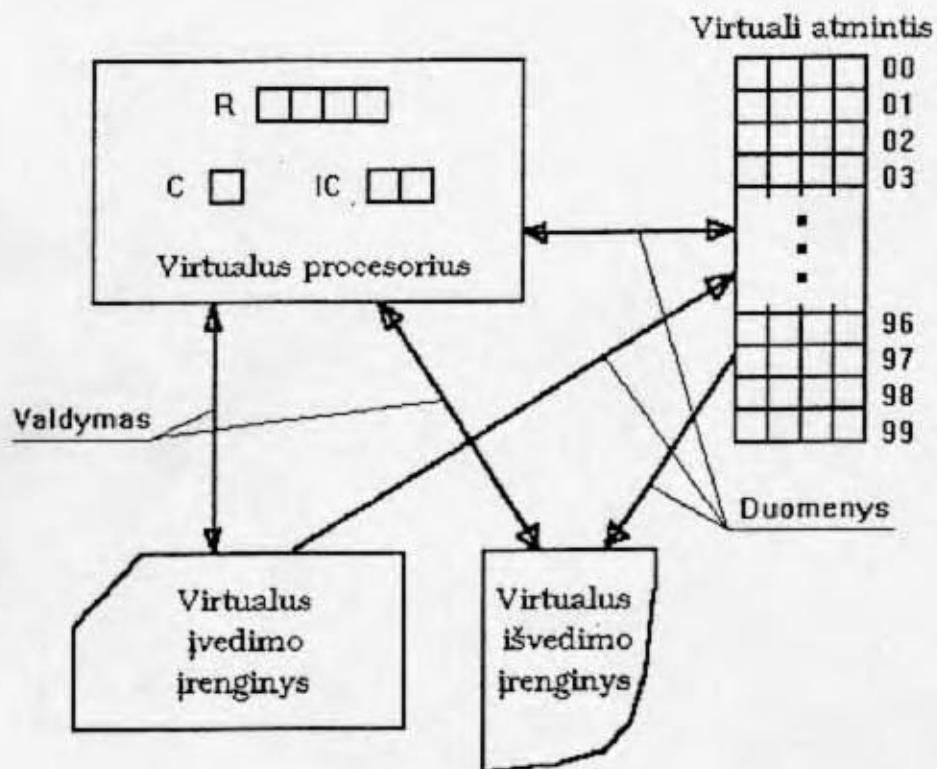
Virtuelus procesorius turi tris pagrindinius registrus:

4 baitų bendrosios paskirties registrą R.

1 baito loginį trigerį C. kuris gali būti vienoje iš dviejų būsenų: "T" (true) arba "F" (false).

2 baitų komandų skaitliuką IC.

Virtueli atmintis yra suskirstyta žodžiais po keturis baitus. Maksimalus atminties ilgis - 100 žodžių, adresuojamų nuo 00 iki 99. Žodis - mažiausias adresuojamas atminties elementas. Baite gali būti bet kuris simbolis, prilimtinės ESM, kurios bazėje konstruojamas modelis. Vyriausiuoju laikomas kairysis žodčio baitas.



5.pieš.

Atminties žodis gali būti interpretuojamas kaip 4 baitų komanda arba kaip duomenų žodis.

Komandoje operacijos kodas užima du vyresniuosius baitus, o operandų adresai į du jaunesniuosius baitus. Komandų formatai ir interpretacijos gali būti tokie:

LR x₁ x₂ - R = [a] - registrą R kopijuojant atminties žodžio su adresu a turinį;

SR x₁ x₂ - a = R - į atminties žodį su adresu a kopijuojant registrą R turinį;

CR x₁ x₂ - if R = [a] then C = 'T' else C = 'F' - pagal tai, ar registro R turinys sutampa su atminties žodžio su adresu a turiniu, ar ne, nustatoma loginio trigerio C reikšmė;

BT x₁ x₂ - if C = 'T' then IC = a - jei loginio trigerio reikšmė true, valdymas perduodamas adresu a;

GD x₁ x₂ - Read ([b+i], i = 0, ..., 9) - perskaito 10 žodžių iš įvedimo srauto ir talpina juos į atminties puslapį, prasidedantį adresu b;

PD x₁ x₂ - Print ([b+i], i = 0, ..., 9) - išveda 10 žodžių iš atminties puslapio su adresu b į išvedimo srautą;

H - Halt - programos vykdymo pabaiga.

Čia x₁, x₂ su reikšmėmis aibėje {0, 1, ..., 9} - operandai, kur x₁ - atminties puslapio numeris, o x₂ - žodžio Jame numeris, t.y. pora (x₁, x₂) yra virtualus adresas. Absoliutus virtualus adresas a = 10^{x₁} + x₂.

Reikia pastebeti, kad komanda GD skaito tik pirmuosius 40 eilutės (perfokortos) simbolių, o komanda PD spausdina naują eilutę išvedimo sraute iš 40 simbolių.

Periferiniai įrenginiai yra virtualus perfokortų (eilučių) skaitymo įrenginys ir virtualus spausdinimo įrenginys.

Tokiai paparastai mašinai galima parašyti nesudėtingų skaičiavimų programų paketą su paprastu įvedimu ir išvedimu. Akivaizdu, kad komandų aibę galima papildyti, įvedant aritmetines ir kitas komandas, prisilaikant pateikto formato ir mašinos sandaros ypatybę, pavyzdžiu:

AD x₁ x₂ - R = R + a - sudeda registro R ir atminties žodžio su adresu a turinius ir rezultatą patalpina į registrą R.

Negali būti atmetama įprastų programinių klaidų galimybė nes MOS turi sugebėti jas apdoroti, t.y. virtualios mašinos darbo eigoje atsiradusi klaida neturi salygoti avarinės situacijos operacinių sistemos lygyje.

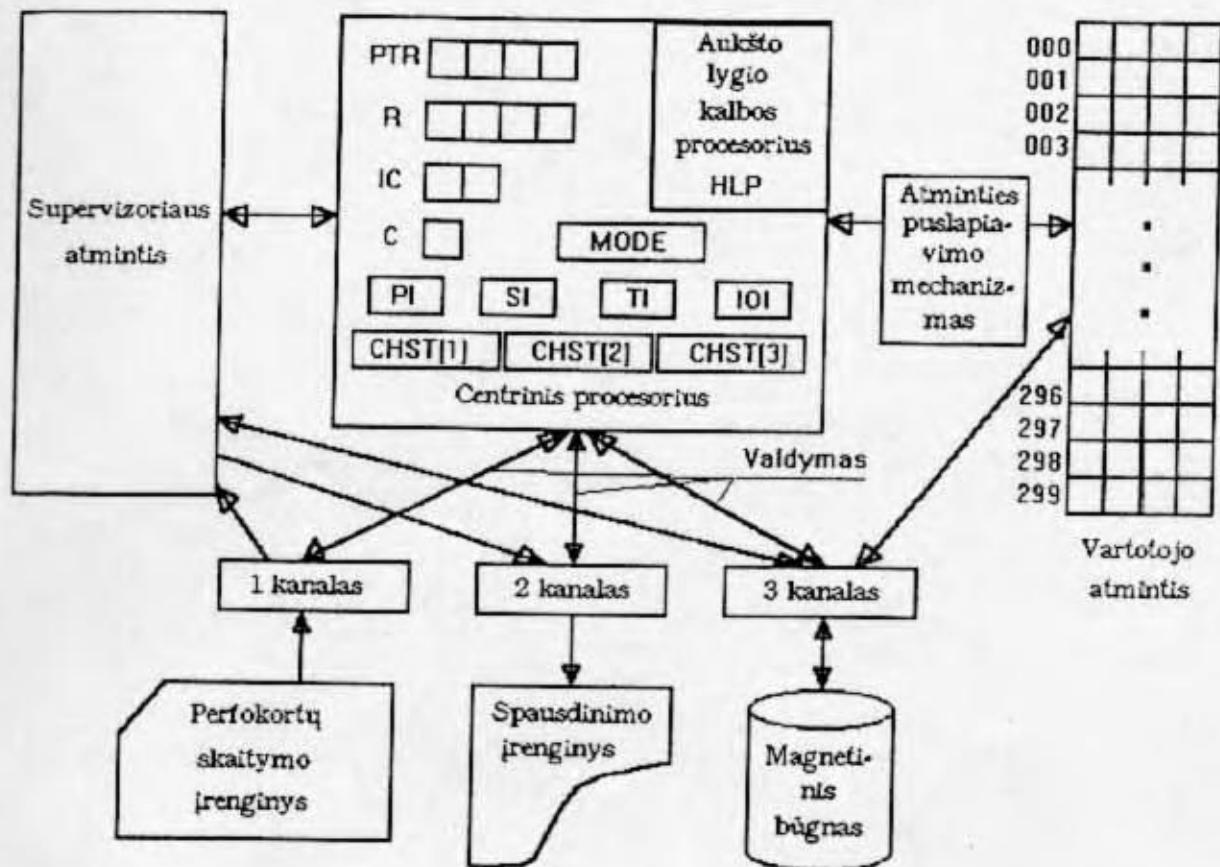
2.3. "Reali" modelinė mašina

Čia ir toliau realia mašina bus vadinama ne ESM, kuri pastrinkta projekto programinei realizacijai, o modelinė skaičiavimo mašina, kuriai kuriama MOS. Tokia mašina turėtų būti sudaryta tik iš esminių komponentų.

Pagrindinės realios mašinos dalys yra centrinis procesorius, atmintis, skaidoma į vartotojo atmintį ir supervizoriaus atmintį, įvedimo (perfokortų skaitymo), išvedimo (spausdinimo) ir išorinės atminties įrenginiai, valdomi kanalu. Vartotojo atmintis procesoriui pasiekiamas per puslapiavimo mechanizmą.

Centrinis procesorius gali dirbti dviem režimais: supervizoriaus arba vartotojo. Supervizoriaus režime komandas iš supervizorinės atminties apdorojamos betarpiskai aukšto lygio kalbos procesoriaus HLP; vartotojo režime HLP interpretuoja "mikroprogramą" supervizorinėje atmintyje tik skaitymui, kuri (programa) imituoja virtualios mašinos procesorių ir prieina prie vartotojo

atmintyje esančiu virtualios mašinos programu per atminties puslapinės organizacijos mechanizmą. Čia HLP - bet kuris prieinamas aukšto lygio kalbos procesorius. (Reikia pastebeti, kad čia turima omenyje programinė įranga bazinėje ESM, palaikanti aukšto lygio programavimo kalbą, kuria rašoma modelio realizacija, ir leidžianti įvykdyti modelio programą. Laikoma, kad supervizorinėje atmintyje yra operacinių sistemos programos komandas, vykdomos supervizoriaus režime, ir programos, interpretuojančios virtualios mašinos komandas. - tokia preilaida reikalinga modelio teoriniam atitikimui realiai funkcionuojančioms skalčiavimo sistemoms nustatyti).



6. pieš.

Centrinio procesoriaus registrai:

1 baito loginis trigeris C,

4 baitų bendrojo naudojimo registratorius R,

2 baitų virtualios mašinos atminties ląstelių skaitiklis IC,
pertraukimų registratoriai PI, SI, IOI, TI,

4 baitų puslapių lentelės registratorius PTR,

kanalų būsenos registratoriai CHST[1], CHST[2], CHST[3],

procesoriaus darbo režimo požymio registratorius MODE.

Vartotojo atmintis susideda iš 300 keturių baitų ilgio žodžių, nuosekliai adresuojamų nuo 000 iki 299. Atmintis suskirstyta į 30 blokus po 10 žodžių.

Supervizorinė atmintis apibrėžiama tokio dydžio, kokio reikia MOS. (Čia vėl reikia pastebeti, kad supervizorinė atmintis modelyje yra gryna sąvoka, neturinti griežtai apibrėžto atitinkmens projekto realizacijoje).

Išorinė atmintis - tai didelio greičio magnetinis būgnas, turintis 100 takelių, kiekviename iš kurių telpa 10 keturių baitų žodžių. Laikoma, kad 10 žodžių persiuntimas į takelį ar iš jo įvykdomas per 1 salyginį laiko vienetą.

Įvedimo - išvedimo įrenginiai (atitinkamai: perfokortų skaitymo įrenginys ir spausdinimo įrenginys) atlieka savo operacijas per 3 salygintus laiko vienetus. Vykdant skaitymą arba rašymą atitinkamai 40 baitų (10 žodžių) persiunčiamą arba iš pirmųjų 40 perfokortos kolonelių, arba į pirmas 10 spaudos pozicijų.

Periferinius įrenginius su supervizorine atmintimi jungia 1-asis ir 2-asis kanalai, o 3-asis kanalas jungia magnetinį būgną su supervizoriaus ir vartotojo atmintimis.

Procesoriui dirbant vartotojo režimu, vartotojiška atmintis adresuojama per puslapinės organizacijos aparatūrą. Registre PTR saugomas einamosios vartotojiškos užduoties puslapių lenteles ilgis ir bazinis adresas. Keturi PTR baitai **a₀ a₁ a₂ a₃** interpretuojami taip: **a₀** - nenaudojamas, **a₁** - puslapių lenteles ilgis minus 1, **10 * a₂ + a₃** - vartotojo atminties bloko, kur yra puslapių lentele, numeris.

Dviženklis adresas (x_1, x_2) virtualioje adresinėje erdvėje aparatuviškai atvaizduojamas į realų adresą vartotojo atmintyje:

$$10 * [10 * (10 * a_2 + a_3) + x_1] + x_2,$$

kur [a] reiškia žodžio su adresu a turinį ir laikoma, kad x_1 nedidesnis už a_1 . Reikalaujama, kad visi vartotojo užduoties puslapiai būtų patalpinti į vartotojo atmintį prieš užduoties vykdymą.

Laikoma, kad kiekviena virtualios mašinos komanda emuliuojama per 1 salyginį laiko vienetą. Visi pertraukimai, įvykstantys procesoriui dirbant vartotojo režime, apdorojami įvykdžius komandą, ir procesorius perjungiamas į supervizoriaus režimą. Operacijos GD, PD, H iššaukia supervizoriinius pertraukimus. T.y. kviečia supervizorių. Programiniai pertraukimai kyla, emulatoriui aptikus vartotojo užduoties programoje klaidingą operacijos kodą arba neleistiną virtualų adresą, kai puslapio numeris x_1 viršija a_1 .

Supervizoriaus režime centrinio procesoriaus darbo pertraukti negalima, tačiau atsitikus įvykiui, sukeliančiam taimerio ar įvedimo-išvedimo pertraukimą, atitinkamas pertraukimo registras įgyja tam tikrą nenulinę reikšmę. Pertraukimo registrus galima apklausti ir išvalyti komanda *Test(x)*, kuri veikia pagal tokį algoritmą:

```
if x = 1 then begin Test := IOI; IOI := 0 end else
if x = 2 then begin Test := PI; PI := 0 end else
if x = 3 then begin Test := SI; SI := 0 end else
if x = 4 then begin Test := TI; TI := 0 end else
if ( IOI + PI + SI + TI ) > 0 then Test := 1
else Test := 0;
```

Čia argumentas x nurodo, ar apklausti ir išvalyti konkretų registrą, ar tiesiog nustatyti, jog pertraukimas apskritai įvyko.

Vartotojo užduoties įvedimo-išvedimo operacijos atliekamos supervizoriaus režime. Įvedimo-išvedimo operacija nurodoma komanda *StartIO(Ch, S, D, n)*, kur *Ch* - kanalo numeris, *S* - nuoroda į išėties blokų masyvą (bloko dydis 10 žodžių), *D* - nuoroda į blokų gaunančią informaciją masyvą, *n* - perduodamų blokų skaičius. Jei *StartIO* duodama užimtam kanalui, tai centrinis procesorius laukia kanalo atlaisvinimo. Kanalu būsenos nustatomos pagal būsenos registrus **CHST[i]**, *i* = 1 . . . , 3. *CHST[i] = 0*, jei kanalas i laisvas, ir *CHST[i] = 1*, jei užimtas.

Procesorius perjungiamas į vartotojo režimą komanda *Slave(ptr, c, r, tc)*, kuri priskiria registrams PTR, C, R ir IC reikšmes atitinkamai *ptr, c, r, tc* ir paleidžia vartotojo užduoties programos komandų interpretatorių.

Laikoma, kad supervizoriaus komandos vykdomos per 0 salyginių laiko vienetų (ty. supervizoriaus režime laikas nėra skaičiuojamas).

Kai komanda *StartIO* duodama i-tajam kanalui, registras *CHST[i]* įgyja reikšmę 1 (užimtas), o įvedimo-išvedimo veiksmas atliekamas, lygiagrečiai dirbant centriniams procesoriui (aparatūrinis paralelizmas). Pasibaigus švedimo-išvedimo veiksmui, *CHST[i]* įgyja reikšmę 0.

Taimerio aparatūra mažina specialios supervizoriaus atminties ląstelės **TM**, vienetu po kiekvienų 10 centrinių procesoriaus darbo salyginio laiko vienetų. Kai *TM = 0*, įvyksta taimerio pertraukimas. *TM* reikšmė gali toliau mažėti ir įgyti neigiamą reikšmę. *TM* reikšmė gali būti nustatyta ir pakeista supervizoriaus režime.

Apibendrinus, pertraukimai gali būti tokie:

- *programiniai*: atminties apsaugos pažeidimo, neegzistuojančio operacijos kodo;

- *supervizoriiniai*: komandų GD, PD, H;

- *įvedimo-išvedimo*: įvedimo-išvedimo operacijos pabaigos;

- *taimerio*: taimerio skaitliuko nulinės reikšmės.

Programiniai ir supervizoriiniai pertraukimai gali įvykti tik vartotojo režime, o įvedimo-išvedimo bei taimerio pertraukimai - ir vartotojo, ir supervizoriaus režimuose.

Kelių tipų pertraukimai gali kilti vienu metu. Įvykus pertraukimams, nustatomos atitinkamų pertraukimų registrų reikšmės, nepriklausomai nuo to, ar leisti, ar uždrausti atitinkamų pertraukimų.

Pertraukimų registrų galiai įgyti tokias reikšmes:

PI: 1 - atminties apsaugos pažeidimas, 2 - neleistinas operacijos kodas;

SI: 1 - komanda GD, 2 - komanda PD, 3 - komanda H;

IOI: 1 - pirmasis kanalas, 2 - antrasis kanalas, 4 - trečiasis kanalas; jei įvyksta keli tokie pertraukimai - reikšmės sudedamos, pavyzdžiu, *IOI = 6* reiškia, kad darbą baigė antrasis ir trečiasis kanalai;

TI: 1 - taimerio skaitliukas lygus 0.

Įvykus pertraukimui vartotojo režime, aparatūra veikia pagal tokį algoritmą:

```
/* išsaugomos registrų reikšmės */  
c := C; r := R; ic := C; ptr := PTR;
```

```

/* procesorius perjungiamas į supervizoriaus režimą */
MODE := 'Master';
/* nustatoma pertraukimo priežastis ir perduodamas valdymas
   atitinkamai pertraukimo apdorojimo programai */
if IOI <> 0 then IOint else
if PI <> 0 then PROGint else
if SI <> 0 then SUPint else
    TIMint;

```

Čia *IOint*, *PROGint*, *SUPint*, *TIMint* - pertraukimų apdorojimo programos supervizorinėje atmintyje.

2.4. Užduoties pateikimo formatas ir taisyklos

Siekiant supaprastinti įvedimo operacijos atlikimo projektavimą ir realizaciją, vartotojo užduotis įsivaizduojama kaip perfokortų (eilučių) paketas, sudarytas iš valdymo, programos ir duomenų kortų. Kortos pakete pateikiamas tokia tvarka

< korta JOB >,< programa >,< korta DATA >,< duomenys >,< korta ENDJOB >.

< korta JOB > sudaryta iš keturių laukų:

SAMJ kortos 1 - 4 pozicijose (A Multiprogramming Job).

< užduoties vardas > kortos 5 - 8 pozicijose - unikalus keturių simbolių užduoties vardas.

< laikas > kortos 9 - 12 pozicijose - keturženklis skaičius, reiškiantis maksimalų užduoties vykdymo laiką.

< išvedimo eilučių skaičius > kortos 13 - 18 pozicijose - keturženklis skaičius, reiškiantis maksimalų išvedamų eilučių skaičių užduočiai.

Pastarieji du laukai reikšmingi, esant nekorektiškai užduočiai, kai dėl pavyzdžiui, "amžino" ciklo gali būti piktnaudžiuojama ESM resursais.

Kiekvienos kortų paketo < programa > kortos 1 - 40 pozicijose pateikiama po 10 vartotojo programos žodžių. I-ojoje korte saugoma pradinė vartotojo atminties žodžių su adresais $10 * (i-1)$, $10 * (i-1) + 1$, ..., $10 * (i-1) + 9$, $i = 1, \dots, n$, būsena. Čia n - kortų skaičius pakete < programa >. Kiekviename žodyje gali būti virtualios mašinos komanda arba keturi duomenų baitai. Kortų skaičius n apibrėžia vartotojo atminties sritis, skiriamos šiai užduočiai, dydį, ty. n kortų apibrėžia $10 * n$ ($n \leq 10$) žodžių.

< korta DATA > yra tokio formato:

SDTA kortos 1 - 4 pozicijose reiškia, kad toliau sekus duomenų kortos,

kortų pakete < duomenys > informacija saugoma kortų 1 - 40 pozicijose. Tai yra vartotojo duomenys, virtualioje mašinoje dirbančiai programai pateikiami, vykdant komandą GD.

< korta ENDJOB > turi du laukus:

SEND kortos 1 - 4 pozicijose.

< užduoties vardas > kortos 5 - 8 pozicijose atitinka tokį pat lauką kortoje < korta JOB >.

< korta DATA > praleidžiama, jei nėra duomenų kortų.

2.5. Pagrindiniai reikalavimai operacinei sistemai

Pagrindinis MOS tikslas - efektyvus vartotojiskų užduočių paketo apdorojimas. Šis tikslas pasiekiamas vartotojiskų ir sisteminių procesų darbo išlygiagretinimu.

Vartotojo užduotis turi praeiti tokias stadijas:

- *Įvedimo eilė (spooling)*. Užduotis įvedama per perfokortą skaitymo įrenginį ir pasiunčiama į magnetinį būgną.

- *pagrindinis apdorojimas*. Programinė užduoties dalis perkeliama iš būgno į vartotojo atmintį. Dabar užduotis jau paruošta vykdymui ir yra paverčiama procesu. Kol procesas nesibaigs (normaliai arba dėl klaidos), jo būsena gali keistis taip:

a) *pasiruošimo būsena* - procesoriaus resurso laukimas.

b) *darbas* - proceso programos vykdymas centriniame procesoriuje,

c) *blokavimo būsena* - įvedimo-išvedimo užklausimo patenkinimo laukimas.

Virtualios mašinos įvedimo ir išvedimo operacijos imituojamos įvedimo-išvedimo operacijomis su magnetiniu būgnu:

- *išvedimo eilė (spulingas)*. Užduoties rezultatų resursų naudojimo statistikos, sisteminių pranešimų ir pradinės programos išvedimas. Spausdinama iš magnetinio būgno per supervizorinę atmintį.

Bendru atveju pagrindinio apdorojomo būsenoje vienu metu bus daugelis užduočių.

MOS turi būti suprojektuota kaip tarpusavyje interaktyvių procesų rinkinys. Tipiniame projekte galėtų būti numatyti tokie pagrindiniai procesai:

Readin_Cards : skaityti perfokortas į supervizorinę atmintį;

Job_to_Drum : sukurti užduoties valdymo bloką ir persiusti užduotį į magnetinį būgną;

Loader : pakrauti užduoties programą į vartotojo atmintį;

Get_Put_Data : apdoroti įvedimo-išvedimo komandas virtualiai mašinai;

Lines_from_Drum : skaityti eilutes, skirtas spausdinimui, iš magnetinio būgno ir perkelti į supervizoriaus atmintį;

Print_Lines : spausdinti eilutes.

Operacinė sistema aktyvuojama pertraukimais, kylančiais, dirbant vartotojo režime. Paprastai pertraukimo programos programos po to, kai aptarnauja pertraukimą, iškviečia planuotoją (centrinio procesoriaus resurso paskirstytoją).

Pagrindinė MOS užduotis - aparatūrinų ir programinių resursų valdymas. Pagrindiniai resursai: vartotojo atmintis, išorinė atmintis, trečiasis kanalas, buferai ir centrinio procesoriaus laikas.

MOS turi rinkti statistinę informaciją apie aparatūros naudojimą ir užduočių charakteristikas:

- resursų naudojimas - bendro laiko dalis, kurią užimtas kiekvienas kanalas; bendro laiko dalis, kai užimtas centrinis procesorius (vartotojo režime); vartotojo atminties naudojimo vidurkis; išorinės atrninties naudojimo vidurkis;

- užduočių charakteristikos - vidutinis vykdymo virtualioje mašinoje laikas; vidutinis būvimo sistemoje laikas, vidutinis reikalaujamos vartotojo atminties dydis, vidutiniai įvedimo ir išvedimo duomenų ilgliai.

Tokie statistiniai duomenys spausdinami, pasibaigus užduočių paketo apdorojimui.

2.6. Reikalavimai projektui

Turi būti suprojektuoti ir realizuoti trys programinių modulių rinkiniai:

- aparatūros imitatoriai:

- a) pertraukimų sistema,
- b) taimeris.
- c) kanalai,
- d) įvedimo ir išvedimo įrenginiai,
- e) išorinė atmintis,
- f) vartotojo atmintis,
- g) vartotojo atminties puslapiaivimo sistema;

Pastaba. Laikoma, kad supervizorinė atmintis bazinėje skaičiavimo mašinoje yra pasiekiamas tiesiogiai.

- mikroprograma, emuliuojanti virtualią mašiną;
- multiprograminė operacinė sistema.

Šios dalys turi būti aiškiai atskirtos. Turi būti įmanoma nesunkiai keisti laikinus parametrus ir aparatūros, ypač vartotojo ir išorinės atminties, dydžius, įvedimo-išvedimo, komandų vykdymo laiko ir taimerio "dažnio" parametrus.

Projekte turi būti pateiktai pagrindinių sistemos procesų ir jų sąveikos, resursų skirstymo ir valdymo metodų, pagrindinių duomenų struktūrų aprašymai.

Taip pat turi būti parengtas bandomasis užduočių paketas.

3.1 Projektavimo įrankiai ir principai

Kadangi modelis yra mokomojo, bendrojo pobūdžio, tai projektavimas neturėtų būti susietas su ypač sudėtingomis daugeliui sunkiai suprantamomis kalbinėmis ir programinėmis priemonėmis. Iškyla problema, kokio aukšto lygio kalbos procesoriaus notaciją naudoti algoritmų užrašymui. Galutinės programos bazinei ESM rašymas neišvengiamai apribotų projektuotoją joje esančio konkrečios aukšto lygio kalbos transliatoriaus taisyklemis ir galimybėmis, arba atvirkšciai, įvestų galimybių, kurios galbūt žymiai suefektyvins galutinės programos darbą, tačiau sudarkys išeities tekstus, padarys juos nebesuprantamais, užgoš pradinio algoritmo esmę ir galu gale nebus pasiektais tikslas sukurti pažintinį, mokomąjį modelį. Projektuoant svarbu suvokti, kad tikslas yra ne parašyti efektyvią programą bazinei ESM, o vaizdžiai realizuoti pagrindinius multiprogramavimo principus. Projekto programinė realizacija taip pat turės ne pramoninę, o demonstracinę ir mokomąjā paskirtį ir reikšmę.

Be viso to, projektavimo etapas turi pasižymeti notacijos ir sprendimų bendrumu.

Sekant išdėstytais principais, autorius projektavimui nesirenka konkrečios aukšto lygio programavimo kalbos. Algoritmų formaliam užrašymui naudojamas supaprastintas PASCAL kalbos poaibis su angliskais baziniais žodžiais. Tačiau rašant algoritmus, nesistengiant griežtai laikytis kalbos apribojimų, vaizdumo dėlei įvedant keletą savybių, kurių PASCAL kalbos standartas nenumato.

Komplikuotesnės sudėties algoritmai notuojami struktūrinų blokinių schemų pagalba.

Toks algoritmų užrašymas orientuotas į jų skaitymą ir analizę. Tačiau norint realizuoti projektą konkrečioje ESM, neturėtų būti sunkumų, perrašant programas kompiuteryje realizuota kalba.

3.2 Modelinės "realios" mašinos aparatinės komponentės

3.2.1 Pertraukimų mechanizmas

ESM aparatūroje pertraukimas kyla specialioje jungtyje pasikeitus atmpati, t. y. logiškai fiksuojamas įvykio įvykimo faktas ir pobūdis. Požymį fiksuoja įvykijų sukėlusį priežastis, pavyzdžiu, komandų interpretatoriui aptikus neleistiną kodą programoje, susidaro situacija "neleistinas operacijos kodas" ir fiksuojamas atitinkamas pertraukimo požymis. Sistemai aptikus pertraukima, nutraukiama vartotojo programos vykdymas, o valdymas perduodamas pertraukimo apdorojomo programai, kuri paprastai yra "tik skaitymo" atmintyje. Pastaroji programa, atlikusi darbą, valdymą grąžina operacinei sistemoi pertraukimo vietoje. Ką toliau daryti, sprendžia MOS.

Modelyje kiekvieno pertraukimą sukeliančio įvykio metu bus nustatomi atitinkami pertraukimų registrai, kaip aprašyta 2.3 poskyryje: IOI, SI, PI, TI priskiriamos atitinkamos reikšmės.

Pertraukimą reaguoja tik vartotojo režime, nes supervizoriaus režime pertraukimai uždrausti. Taigi vartotojo režime paprastai įvykdomas programos virtualioje mašinoje komandos interpretavimo ciklas ir po to duodama supervizorinę komandą *Test(x)* (žr. 2.3) su parametru, priklausančiu aibei {1, 2, 3, 4}. Taip nustatoma, ar neįvyko pertraukimas.

Pertraukimui įvykus, išsaugoma virtualios mašinos būseną, procesorius perjungiamas į supervizoriaus režimą ir valdymas atiduodamas atitinkamą pertraukimo apdorojimo programai.

Pertraukimo programos nustato pertraukimą sukelusią priežastį ir išvalo pertraukimo registrą, kviesdama komandą *Test* su atitinkamu parametru. Po to reaguoja į pertraukimą.

Bendra taisyklė pertraukimų programų bendravimui su likusia sistemos dalimi turėtų būti tokia: duomenimis keičiamasi ne tiesiogiai, o per sutartus kintamuosius supervizorinėje atmintyje, kurių adresai yra iš anksto žinomi ir nesikeičia. Tai būtina, norint maksimaliai atriboti žemo lygio pertraukimų apdorojimo aparatūrą nuo aukšto lygio operacinės sistemos.

Formalizuosime pertraukimų apdorojimo programų darbo algoritmus.

IOInt : pranešama sistemai, kad atsilaisvino kažkuris (-ie) kanalas (-ai).

```
procedure IOInt;
begin
    case Test(1) of
        1: CHST[1] := 0;
        2: CHST[2] := 0;
        3: begin, CHST[1] := 0; CHST[2] := 0 end;
        4: CHST[4] := 0;
        5: begin CHST[1] := 0; CHST[3] := 0 end;
        6: begin CHST[2] := 0; CHST[3] := 0 end
    end
end;
```

PInt : baigiamas einamosios vartotojo užduoties vykdymas, kaip sutikus programinę klaidą. Suformuojamas atitinkamas sisteminis pranešimas.

```
procedure PInt;
begin
    /* užduoties vykdymas sustabdomas */
    continue_job := false;
    /* fiksuojamas klaidos įvykimo faktas */
    error := true;
    if Test(2) = 1 then
        /* nustatomas sisteminio pranešimo teksto bloko numeris */
        sys_msg_number := illegal_address_msg_number
    else
        sys_msg_number := unknown_opcode_msg_number;
end;
```

Laikoma, kad supervizoriaus atmintyje yra sisteminiai kintamieji, kurie nusako einamosios vartotojo užduoties vykdymo pratęsimo galimybę, kaičios įvykimo faktą ir sisteminio pranešimo supervizorinėje atmintyje "tik skaitymu" bloko numerį:

```
continue_job, error: boolean;  
sys_msg_number: integer;
```

SInt : nurodoma sistemai aktyvuoti vartotojiškos užduoties įvedimo-išvedimo operacijų aptarnavimą arba baigtis vartotojo užduotį (normali pabaiga)

```
procedure SInt;  
begin  
/* užduoties vykdymas sustabdomas */  
    continue_job := false;  
    case Test(3) of  
        /* nustatomas požymis sistemai, kad reikia aptarnauti virtualios mašinos  
        įvedimo operaciją */  
        1: virtual_input_required := true;  
        /* nustatomas požymis sistemai, kad reikia aptarnauti virtualios mašinos  
        išvedimo operaciją */  
        2: virtual_output_required := true;  
        3: sys_msg_number := normal_end_msg_number  
    end  
end
```

Laikoma, kad supervizorinėje atmintyje yra sisteminiai kintamieji:

```
/* požymis apie virtualios mašinos įvedimo operacijos aptarnavimo  
reikalangumą */  
virtual_input_required,  
/* požymis apie virtualios mašinos išvedimo operacijos aptarnavimo  
reikalangumą */  
virtual_output_required: boolean;
```

TIMInt : reakcija nustatoma, konfigūruojant sistemą. Galima tiesiog tuščia reakcija:

```
procedure TIMInt;  
begin  
    Test(4)  
end;
```

Tačiau prasmingiau yra įvesti laiko ribojimą, siekiant sustabdyti užsieklinusiais programas ir apsaugoti nuo nereikalingo resursų švaistymo:

```
procedure TIMInt;  
begin
```

```

    Test(4);
/* užduoties vykdymas sustabdomas */
    continue_job := false;
/* fiksuojamas klaidos įvykimo faktas */
    error := true;
    sys_msg_number := time_limits_exceeded_msg_number
end;

```

Pastebėkime, kad pertraukimų apdorojimo atveju kalbama apie aparatueros darbą žemame lygyje, todėl jokio paralelizmo nėra: vienu metu reaguojama tik į vieną pertraukimą. Todėl būtina pertraukimų registrų tikrinimą ir reagavimą į pertraukimus įtraukti į ciklą, kartojamą, kol komanda *Test* negražins nulinės reikšmės:

```

procedure Interrupt_test;
begin
    while Test(5) > 0 do
        if IOI <> 0 then IOInt else
        if PI <> 0 then PInt else
        if SI <> 0 then SInt else
            TIMInt
end;

```

Pertraukimus aptarnaujančios programos yra labai paprastos. Jos tiesiogiai negeneruoja naujų procesų, o tik pakeičia sistemos būseną, į ką vėliau MOS reaguoja aukštesnio lygio operacijomis.

3.2.2. Taimeris

Taimeris - procesoriaus laiko skaitiklis. ESM generuojami pastovių dažniu įtampos impulsai - taktai, kurių pasėkoje gali dirbti procesorius. Prosesorius komandas vykdo per sveiką taktų skaičių. Taimeris skaiciuoja taktus ir kas tam tikrą jų skaičių keičia laiko skaitiklio kintamojo reikšmę.

Modelinėje realioje skaičiavimo mašinoje laikas skaiciuojamas tik vartotojo režime. Virtualios mašinos komandos vykdomos per 1 taktą. Taimeris skaiciuoja taktus ir, praėjus dešimčiai taktų, sumažina laiko skaitiklio reikšmę vienetu.

Kadangi laikas skaiciuojamas kiekvienoje virtualioje mašinoje atskirai, tai galima laikyti, kad kiekvienoje virtualioje mašinoje *m* yra savi taktų ir laiko vienetų skaitiklių kintamieji:

```

vm[m].clock,
vm[m].time: integer;

```

Emulatoriui interpretuojant komandas, keičiamama *clock* reikšmė:

```

vm[m].clock := vm[m].clock + <taktų skaičius>

```

Po komandos ciklo taimeris tikrina taktų skaitliuko reikšmę ir jeigu *clock* 10, sumažina *time* reikšmę vienetu. Jei *time* = 0, pertraukimo registrui TI reikiama nenulinė reikšmė:

```
procedure Timer;
begin
    if vm[m].clock ≥ 10 then
        begin
            vm[m].clock := vm[m].clock mod 10;
            vm[m].time := vm[m].time - 1
        end;
    if vm[m].time = 0 then TI := 1;
/* Kreipinys į sisteminio taimerio procedūrą (žr. žemiau) */
    systime (0);
end;
```

Pradinė *vm[m].time* reikšmė įmama iš užduoties antraštės kortos arba gal sistemos nutylėjimą.

Tačiau laikas skaičiuojamas ne tik virtualios mašinos viduje, bet ir realoje realioje mašinoje apskritai. Ši laiko skaitliuką galima laikyti sisteminiu neriui, kuris skaičiuoja sistemos darbo laiką, t. y. sumuoja sisteminių ir oratūrių komandų vykdymo trukmę sąlyginiais laiko vienetais: sisteminės mandos, nesusiję su įvedimo-išvedimo veiksmais atliekamos per 0 laiko netų, virtualios mašinos programos komandos interpretavimas vartotojo įme atliekamas per 1 laiko vieną, apsikeitimas duomenimis su magnetiniu gnu trunka 1 laiko vieną, o skaitymo iš įvedimo įrenginio ir rašymo į edimo įrenginį operacijos užima po 3 laiko vienetus. Visų operacijų trukmę ma lygi skaičiavimo sistemos darbo laikui. Ši laiką reikia žinoti, pateikiant ištinę informaciją užduočių paketo apdorojimo pabaigoje. Be to tam reikia žoti, kiek laiko vieną dirbo kanalai (buvo aptarnaujami periferiniai įrenginiai) centrinis procesorius vartotojo režime. Šiuo atveju reikalingi papildomi skaitliukai, kurių reikšmės didėtų lygiagrečiai su sisteminio taimerio skaitliuko šme, bet tik tuo atveju kai atliekama atitinkama operacija.

Sisteminio taimerio procedūra galėtų atrodyti taip:

```
procedure systime (type:integer);
    var a:integer;
begin
    case type of
    0: begin
        busycpu := busycpu + 1;
        a := 1
    end
    1: begin
        busylcn := busylcn + 3;
        a := 3
    end
    2: begin
        busy2cn := busy2cn + 3;
        a := 3
    end
end;
```

```

    end
3: begin
    busy3cn := busy3cn + 1;
    a := 1
end
end;
stime := stime + a;
end;

```

Čia laikoma, jog *stime*, *busycpu*, *busylcn*, *busy2cn*, *busy3cn* yra aukščiau minėtų skaitliukų kintamieji, būtent sisteminio taimerio bendro laiko, laiko, kurį procesorius dirba vartotojo režime ir laiko, kurį užimtas pirmasis, antrasis ir trečiasis kanalas:

```
stime, busycpu, busylcn, busy2cn, busy3cn: integer;
```

Sisteminio taimerio darbas derinamas su emulatoriaus bei kanalu darbu, todėl algoritmizujant natūralu kreipinius į jo procedūrą patalpinti virtualios mašinos laiko skaiciavimo procedūroje *Timer* bei kanalu darbą aprašančiose procedūrose (žr. 3.23 skyreli).

3.2.3. Kanalai

Kanalai yra ne ryšio linijos, kaip galima suprasti iš pirmo žvilgsnio, o specialūs procesoriai, valdantys įvedimo-išvedimo ir išorinės atminties įrenginių darbą. Jie reikalingi tam, kad nuo palyginti lėtų skaitymo ir rašymo operacijų būtų išlaisvintas centrinis procesorius, tuo būdu efektyvinant ESM darbą. Kanalai ir centrinis procesorius dirba lygiagrečiai (aparatūrinis parallelizmas).

Kanalai duomenis tvarko aukštėsniame loginiame lygyje, nei įvedimo-išvedimo įrenginiai. Skirtingai nuo pastarųjų jie gali operuoti duomenų bloku įvairaus ilgio masyvais.

Pirmasis kanalas priima duomenis iš persokortų skaitymo įrenginio ir perduoda juos į supervizorinę atmintį.

Sakykime, kad blokų, kur bus talpinami perskaityti duomenys, numeriu masyvas yra *blocks*, kintamojo *blocks_number* reikšmė rodo, kiek masyvo elementų yra panaudota, o *buff* - bloko dydžio buferis, į kurį įvedimo įrenginys talpina perskaitytus duomenis. Tegul supervizorinė atmintis formalizuojama masyvu *supervisor_memory*:

```

blocks: array [1..N] of integer;
blocks_number: integer;
buff: array [1..10] of array [1..4] of char;
supervisor_memory: array[1..MAX] of array[1..4] of char;

```

Tada pirmojo kanalo darbo algoritmas yra toks:

```

procedure Chan_1;
var i, j:integer;

```

```

begin
/* 1-asis kanalas dirba */
    CHST[1] := 1;
    for i := 1 to blocks_number do
        begin
/* įvedamas blokas duomenų iš perfokortų sakitimo įrenginio */
        Reader;
/* blokas kopijuojamas iš buferio į supervizoriaus atmintį */
        for j := 0 to 9 do
            supervisor_memory[blocks[i]+j]:=buff[j]
        end
/* padidinama pertraukimo registro reikšmė */
    IOI := IOI + 1;
/* kanalas laisvas */
    CHST[1] := 0;
end;

```

Panašiai dirba ir *antrasis kanalas*, tvarkantis duomenų srautą, einantį iš supervizoriaus atminties į spausdintuvą.

Sakykime, kad *buff* - buferis, skirtas išvedimui į spausdintuvą.

```

procedure Chan_2;
    var i, j: integer;
begin
    CHST[2] := 1;
    for i := 1 to blocks_number do
        begin
/* blokas duomenų kopijuojamas iš supervizorinės atminties į buferį */
        for j := 0 to 9 do
            buff[j]:=supervisor_memory[blocks[i]+j];
/* buferio turinys išvedamas į spausdintuvą */
        Printer
        end;
/* padidinama pertraukimo registro reikšmė */
    IOI := IOI + 2;
    CHST[2] := 0;
end;

```

Trečiasis kanalas atsakingas už apsikeitimą duomenimis tarp supervizorinės atminties ir išorinės atminties įrenginio - magnetinio būgno. Duomenys gali judėti tiek viena (rašymas), tiek kita (skaitymas) kryptimi.

Sakykime, kad masyve *tracks* saugomi takelių po 10 žodžių, į kuriuos bus vedami (arba iš kurių bus skaitomi) duomenys, numeriai, o kinatamasis *ioflag* dikuoja duomenų srauto kryptę: *true* - rašymas į magnetinį būgną, *false* - skaitymas iš jo:

```

tracks: array [1..N] of integer;
ioflag: boolean;

```

Tada trečiojo kanalo darbas formaliai atrodo taip:

```

procedure Chan_3;
  var i: integer;
begin
  CHST[3] := 1;
  for i := 1 to blocks_number do
    /* bloko dydžio duomenų paketu apsiętiama su magnetiniu būgnu */
    Drum(tracks[i], ioflag, blocks[i]);
  /* padidinama pertraukimo registro reikšmė */
  IOI := IOI + 4;
  CHST[3] := 0;
end;

```

Reikia pastebeti, kad kanalų ir centrinio procesoriaus darbo sinchronizacijai ir yra skirti įvedimo-išvedimo pertraukimai. Kanalas, baigęs darbą, informuoja apie tai centrinį procesorių, atitinkamai padidindamas pertraukimo registro IOI reikšmę.

3.2.4. Įvedimo ir išvedimo įrenginiai

Įvedimo ir išvedimo įrenginiai yra periferiniai įrenginiai, skirti apsiętimui duomenimis su išore.

Tarkime, kad įvedimo įrenginys - perfokortų skaitytuvas, o išvedimo įrenginys - eilutinis spausdintuvas. Jie turi 10 žodžių dydžio buferius duomenų apsiętimui su ESM.

Perfokortų skaitymo įrenginys skaito perfokortas (10 simbolių eilutes) po vieną į buferį. Užpildęs pastarąjį jis laukia, kol kanalas jo turinį perkels į supervizorinę atmintį ir leis toliau dirbti.

Paprastumo dėlei laikysime, jog apsiętiama tik visiškai užpildytais buferiais. Todėl jei dėl kokių nors priežasčių įvedimo srautas nutrūksta, esant nebaigtam pildyti buferiui, skaitymo įrenginys užpildo jį fiktyviais simboliais, reiškiančiais srauto pabaigą, pavyzdžiu, "?".

```

procedure Reader;
  var i: integer;
begin
  for i := 1 to 10 do
    if not end_of_input then read(buff[i]);
    else buff[i] := '????'
end;

```

Čia funkcija *end_of_input* grąžina reikšmę *true*, jei įvedimo srautas baigesi, ir *false* - priešingu atveju. Prėcedūra *read* skaito 4 simbolių eilutę - 1 žodį:

```

function end_of_input: boolean;
procedure read(var buff: array[1..10] of array[1..4] of
char);

```

Išvedimo įrenginys spausdina 40 simbolių eilutę (10 žodžių buferį) popieriuje. Atspaustintas eilutę spausdintuvas laukia, kol kanalas atnaujins buferio turinį:

```
procedure Printer;
    var i: integer;
begin
    for i := 1 to 10 do write(buff[i]);
    writeln
end;
```

Čia procedūra *write* išveda į spausdintuvą 4 simbolių eilutę, o procedūra *writeln* perkelia spausdinimą į sekančios eilutės popieriuje pradžią.

3.2.5. Išorinės atminties įrenginys

Magnetinis būgnas - didelio apsikeitimo duomenimis greičio informacijos ugojimc įrenginys. Į jį gali būti rašoma ir iš jo skaitoma. Galima laisvai sirinkti takeli - tiesioginis priėjimas.

Magnetinių būgnų modeliuosime failu bazinėje ESM, leidžiančiu laisvą našu išrinkimą, o takelius - to failo įrašais.

Išorinės atminties įrenginiui nurodomas takelio numeris *track*, pervizorinės arba vartotojo atminties bloko numeris *block* ir duomenų dėjimo kryptis *ioflag* (*true* - rašymas, *false* - skaitymas). Magnetinis būgnas apsikeičia duomenų bloku su vidine atmintimi ir laukia kanalo leidimo kančiam apsikeitimui:

```
procedure Drum (var track: integer; ioflag: boolean;
lock: integer);
    var i: integer;
begin
    seek(track);
    if ioflag then
        /* duomenų blokas iš vidinės atminties kopijuojamas į išorinę atmintį */
        for i := 1 to 10 do
            fwrite(supervisor_memory[10*block+i])
    else
        /* duomenų blokas iš išorinės atminties kopijuojamas į vidinę atmintį */
        for i := 1 to 10 do
            fread(supervisor_memory[10*block+i]);
end;
```

Čia procedūra *seek* pozicionuoja magnetinio būgno skaitymo-rašymo ivutę ties takeliu *track*, procedūra *fread* - perskaito 4 simbolius (žodį) omenų iš būgno, o procedūra *fwrite* - įrašo 4 baitus duomenų į būgną:

```
procedure seek(track: integer);
procedure fread (var memory_block: array [1..10] of
ray [1..4] of char);
```

```
procedure fwrite ( memory_block: array [1..10] of
array [1..4] of char);
```

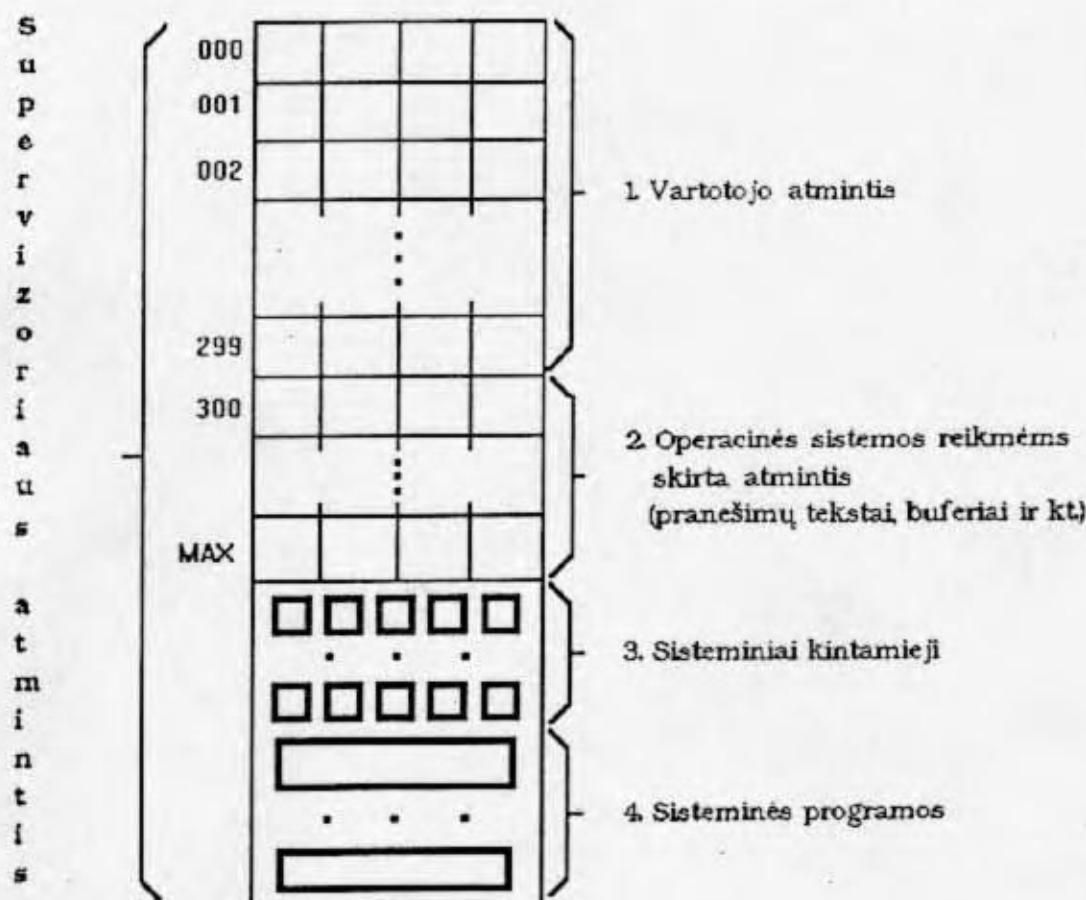
3.2.6. Vartotojo ir supervizoriaus atmintis

Vartotojo atmintį konstruosime taip, kad ji būtų supervizoriaus atminties pradžioje. Taip daryti tikslinga dėl adresavimo vientisumo ir atminties homogeniškumo (žr. 7 pieš.).

Tai atitinka situaciją ESM, kur vartotojo ir sisteminės paskirties atmintis sudaro vientisą operatyviosios atminties erdvę ir skiriasi tik logiškai - panaudojimo pobūdžiu.

Išreikštiniu pavidalu projektuosime tik 1 ir 2 atminties sritis, kur 1 sritis bus adresuojama nuo 000 iki 299 (vartotojo atminties ilgis - 300 žodžių), o 2 sritis - nuo 300 iki MAX (MAX parenkamas, realizuojant sistemą, taip, kad būtų patenkintos sistemos reikmės):

```
supervisor_memory: array [0..MAX] of array [1..4] of
char;
```



7. pieš.

3 ir 4 sritys išreikštiniu pavidalu nebus projektuojamos. Tiesiog laikoma, kad sisteminiai kintamieji lentelės ir programos bus saugomos supervizoriaus

atmintyje, jos detaliau nestruktūrizuojant Aukščiau aprašyta dalinė struktūrizacija reikalinga, mašinos modelį darant vaizdesniu, nes būtent struktūrizuotoji dalis valdins svarbų vaidmenį realizuojant multiprogramavimo principus.

3.2.7. Vartotojo atminties puslapiavimo sistema

1.3. poskyryje buvo aprašytas virtualios atminties modeliavimo principas: tam tikru metodu ji atvaizduojama į realią (vartotojo atmintį).

Naudosime patį paprasčiausią puslapiavimo mechanizmą be segmentacijos: realaus adreso nustatymui pakaks vieno kreipinio į atmintį - *puslapių lentelę*. Kadangi modelyje virtualios mašinos atmintis gali būti iki 10 puslapių ilgio, tai bet kuriai virtualiai atminčiai susieti su vartotojo atmintimi pakaks standartinės 10 žodžių (1 bloko) dydžio lentelės.

Dėl to, kad puslapių lentelė yra vartotojui tiesiogiai "nežinoma" ir nepasiekiamą ir atlieka grynai sistemos palaikymo vaidmenį, netikslinga būtų ją tarpinti į vartotojo atminties sritį - puslapių lentelei bus išskiriamaas supervizorinės atminties blokas iš sisteminės paskirties srities.

Į puslapiavimo mechanizmą jeina ir *puslapių lentelės registras PTR*. Jis, išlaikant vieningą realios mašinos simbolinių duomenų vaizdavimo formatą, yra 4 simbolinių baitų ilgio: $a_0 a_1 a_2 a_3$. Remiantis 2.3 poskyryje pateikta koncepcija, a_0 - ignoruojamas, a_1, a_2, a_3 - gali įgyti reikšmes "0" - "9", kurios interpretuoojamos taip: a_1 - puslapių lentelės ilgis (virtualios atminties dydis puslapiais) minus 1, $10 \cdot \text{dig}(a_2) + \text{dig}(a_3)$ - bloko supervizoriaus atmintyje, kur yra puslapių lentelė, numeris. Čia funkcija *dig* transformuoja simbolinį skaitmenį į sveikajį skaičių:

```
function dig( digit: char ): integer;
```

Kiekvienai vartotojo užduočiai atlikti bus sukuriama virtuali mašina su virtualia atmintimi, kuriai sistema iš anksto sukuria puslapių lentelę ir atitinkamą PTR reikšmę, patalpindama pastarąjį užduoties valdymo bloke. Vartotojiskam procesui tapus einamuoju, ši reikšmė patalpinama į PTR. Dirbdamas vartotojo režime, procesorius virtualų adresą (x_1, x_2) , kur x_1 - puslapio numeris, x_2 - žodžio Jame numeris, į realų adresą - vartotojo atminties žodžio numerį - keičia taip:

a) pagal PTR reikšmę nustatomas blokas, kuriame yra priskirtoji puslapių lentelė. Formaliai PTR aprašomas taip:

```
/* PTR[1] - a0, PTR[2] - a1, PTR[3] - a2, PTR[4] - a3 */  
PTR: array [1..4] of char;
```

b) iš puslapių lentelės išrenkamas įrašas su numeriu $\text{dig}(x_1)$, t. y. atitinkantis x_1 puslapį virtualioje atmintyje.

c) pagal įrašo turinį nustatomas puslapį atitinkančio vartotojo atminties bloko numeris.

d) bloke išrenkamas žodis su numeriu $\text{dig}(x_2)$.

Tegul *block_nr* - simbolinis žodis, į kurį laikinai patalpinsime puslapių lentelės bloko adresą simboliniame pavidale, *virt_addr* - virtualus adresas (x_1, x_2), *realaddr* - realus adresas (žodžio vartotojo atmintyje numeris):

```
block_nr: array [1..4] of char;
/* virt_addr[1] - x1, virt_addr[2] - x2 */
virt_addr: array [1..2] of char;
realaddr: integer;
```

Naudojant įsivestus pažymėjimus, galima išreikšti realų adresą *realaddr* taip:

```
block_nr := supervisor memory[10*(10*dig(PTR[3]) +
dig(PTR[4]))+dig(virt_addr[1])];
realaddr := 10*numb(block_nr)+dig(virt_addr[2]);
```

Funkcija *numb* konvertuoja skaičių iš simbolinio žodžio pavidalo į sveikajį skaičių:

```
function numb(char_word:array [1..4] of char):integer;
```

Toks adreso transformavimo mechanizmas būtų pakankamas, jei virtuali atmintis visuomet turėtų maksimalų ilgį - 10 puslapių. Kadangi ji gali būti mažesnė, tai vartotojo programa atsitiktinai arba sąmoningai gali suformuoti neleistiną virtualų adresą, t. y. kreiptis į puslapį su numeriu, kuris viršytų virtualią atmintį sudarančių puslapių skaičių. To pasekmės būtų nenuuspējamos, galimas kitų vartotojų informacijos sugadinimas. Tam ir reikalinga atminties apsauga, kuri šiuo atveju yra elementari: pakanka patikrinti, ar $dig(x_1) \leq dig(a_1)$. Jei nelygybė netenkinama, įvyko programinė klaida - suformuotas neleistinas virtualus adresas.

Todel bendras virtualaus adreso transformavimo ir tikrinimo algoritmas turėtų būti toks:

```
procedure test_transf_addr ( virt_addr: array [1..2] of
char; var realaddr: integer);
    var block_nr: array [1..4] of char;
begin
    dig
    if numb(virt_addr[1]) > numb(PTR[2]) then
        begin
            /* jei pažeista atminties apsauga, tai
               - nustatoma atitinkama programinio pertraukimo registro reišmė,
            */
            PI := 1;
            /* - valdymas perduodamas išoriniu adresu ERROR */
            go to external ERROR
        end
    else
        begin
            /* nustatomas bloko, kuriamo yra puslapių lentelė numeris */
```

```

block_nr := supervisor_memory[10*(10*
dig(PTR[3])+dig(PTR[4]))+
dig(virt_addr[1])];

/* apskaičiuojamas realus adresas */
realaddr := 10*numb(block_nr)+  

dig(virt_addr[2])
end;

```

Matome, kad klaidos atveju programinio pertraukimo registrui priskiriamas nenulinis reikšmė ir pereinama prie programinio fragmento ERROR. Pertraukimo požymis inicijuos vartotojo programos vykdymo nutraukimą ir pertraukimo apdorojimą (žr. 3.2.1 skyrelį). Plačiau apie adreso transformavimo mechanizmo ir procesoriaus darbo vartotojo režime suderinimą kalbama 3.3 poskyryje.

3.2.8 Aparatūros modeliavimo apžvalga

Šio poskyrio 1 - 7 skyreliuose buvo trumpai išdėstyti pagrindinių aparatūros dalij veikimo principai ir algoritmai. Nebuvo detalizuotas tik procesorius. Iš esmės tai ir nebuvvo būtina, nes procesoriaus veikimo principas paprastas - išrinkti iš atminties komandą ir ją suinterpretuoti. A. Šou formuliuodamas užduotį modelio projektui, ir nesistengė pernelyg detalizuoti procesoriaus.

Nurodyti komponentai - registrai ir komandų aibė reikalingi tik dirbant vartotojo režime. Supervizoriaus režime procesorius dirba kaip HLP - aukšto lygio kalbos procesorius. Tai modelio projektuotojui paranku dviem atžvilgiais: pirma, modeliuojant virtualios mašinos darbą žemiausiaame lygyje, nesunku parodyti, kaip jis vyksta, antra, modeliuojant sudėtingus procesus, nereikalaujama smulkmenų, ne šiuo atveju svarbu ne *kaip vyksta* procesas, o tai, kad *jis vyksta*. Méginiamas detalizuoti pastarąjį realaus procesoriaus darbo dalį tik pridėtų papildomą sunkumą, o, antra vertus, užgoštų pagrindinius proceso darbo žingsnius. Mokomojo pobūdžio projekte tai nėra pageidautina.

Nagrinéjant periferinių įrenginių ir juos valdančių kanalų modelius, gali susidaryti išvada, jog kanalai čia patenka kaip "parazitiniai" programinių fragmentai, dubliuojantys periferinius įrenginius. Tačiau reikia prisiminti, jog pagrindinis šio projekto tikslas ne parašyti efektyviai programą, o išsiaiškinti pagrindinius aparatūros ir programinės įrangos funkcionavimo principus. Demonstracinė programa - projekto realizacija - turėtų tik apjungti pateiktus fragmentus (galbūt su neesminėmis modifikacijomis) ir įrodyti faktą, kad tas ar kitas algoritmas veikia.

Plečiant projektą, galima keisti kai kuriuos parametrus: taimerio "dažnį", vartotojo atminties dydį. Tačiau keičiant pastarąjį, reikia atsižvelgti adresaciją. Virtualioje mašinoje adresinė erdvė ribota iki 10 puslapių po 10 žodžių, nes tokia yra maksimali adresinė erdvė. Kurią galima nusakyti pora (x_1, x_2), kur x_1 ir x_2 įgyja reikšmes iš aibės {0, ..., 9}. Norint didinti virtualios mašinos atmintį, būtina įvesti virtualios atminties segmentaciją ir ją palaikančią mechanizmą. Nedidinant

virtualios atminties, o tik plečiant vartotojo atmintį, būtina atsižvelgti į tai, kad bus "nustumiami tollyn" supervizorinės atminties sisteminės paskirties srities adresai. Kadangi pastarojoje srityje talpinamos puslapių lentelės, o jų užimamų blokų numeriai nurodomi dviženkliu skaičiumi $10^* \text{dig}_2 + \text{dig}_3$, kur PTR = $a_0a_1a_2a_3$, tai blokų pasiekiamą per PTR numeriai negali viršyti 99 (bendra adresinė erdvė - nuo 000 iki 999). Priešingu atveju gali taip pat tekti įvesti atminties segmentaciją ir jos palaikymo mechanizmą.

Galima ir kita išeitis - radikalesnė - didinti atminties žodį. Tačiau tokiu atveju tektų keisti ir tokius parametrus, kaip bendrojo registro R ir puslapių lentelės registro PTR dydis, bloko-takelio-kortos dydis simboliais, virtualios mašinos komandų formatas ir panašiai. Iš esmės toks mašinos rekonstravimo būdas pateisinamas tik tuomet, kai siekiama padaryti sudėtingesnę vartotojo darbo aplinką. Demonstraciniam - mokomajam projektui tai nėra būtina.

3.3. Virtualios mašinos komandų interpretatorius

Virtualios mašinos komandų interpretatorius praktiškai yra virtualus procesorius arba realaus procesoriaus darbo dalis vartotojo režime. Virtualios mašinos komandų interpretavimo ciklai sudaro žemiausio lygio procesą - vartotojo užduoties vykdymo procesą. Šiame realaus procesoriaus darbo režime galimi pertraukimai, skaičiuojamas darbo laikas - dirba taimeris.

Interpretatoriaus darbo schema pateikiama 8 piešinyje.

I interpretoriaus ciklą įeina, kai vartotojiškas procesas tampa einamuoju ir gauna procesorių. Tai inicijuojama, duodant komandą Slave(ptr, c, r, lc), kurią galima interpretuoti taip:

```
procedure Slave ( ptr: array [1..4] of char;
                  c: char;
                  r: array [1..4] of char;
                  ic: array [1..2] of char);
begin
    /* nustatoma registrų reikšmės */
    PTR := ptr;
    C := c;  R := r;  IC := ic;
    /* procesorius perjungiamas į vartotojo režimą */
    MODE := 'Slave'
end;
```

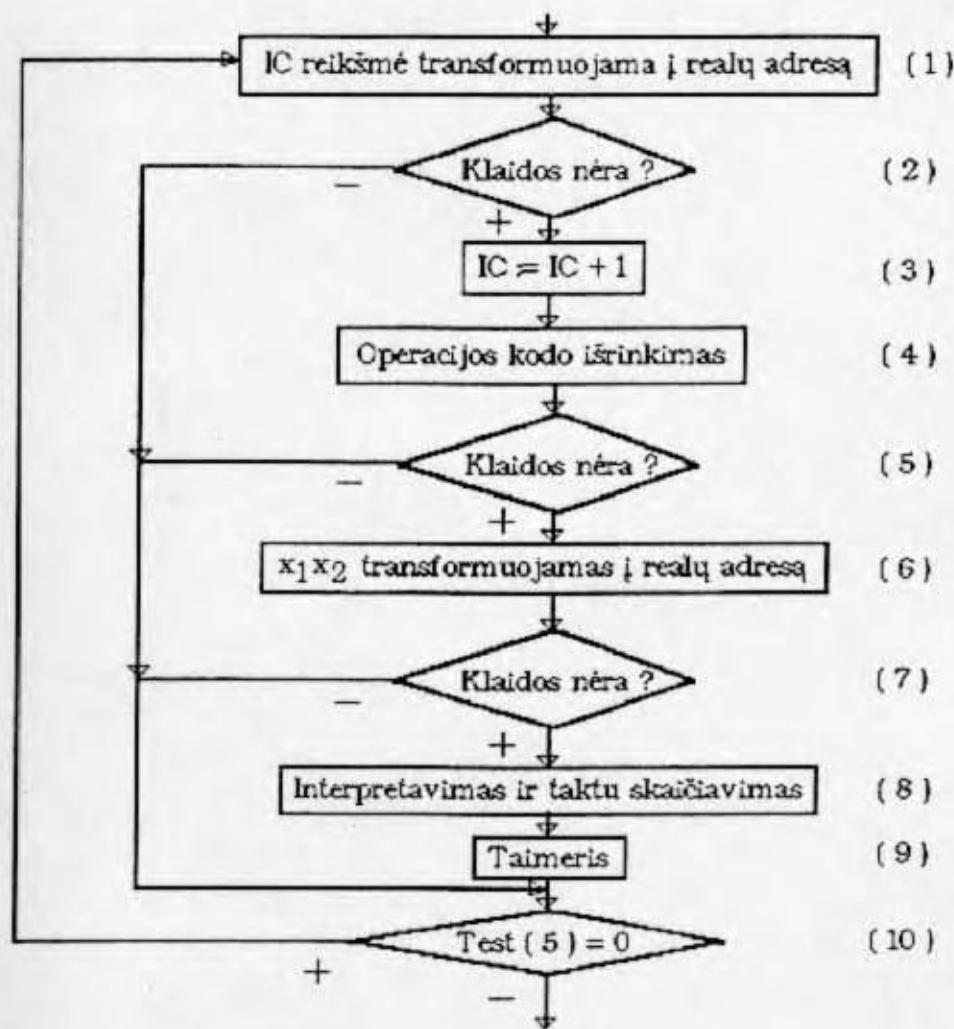
Cikle atliekami tokie veiksmai:

1) simbolinė IC reikšmė - virtualus adresas - transformuojamas į realų adresą;

2) jei virtualus adresas neleistinas, nustatomas atitinkamas programinis pertraukimas ir pereinama į ciklo pabaigą;

3) komandų skaitliukas padidinamas vienetu. Inkrementacijai ši vieta parinkta, norint išvengti konfliktų su komanda BT, kuri betarpiskai keičia IC reikšmę, ir papildomo mechanizmo įtems spręsti kūrimo;

4) operacijos kodo identifikavimas;



8. pieš.

5) programinė klaida - nežinomas operacijos kodas. Inicijuojamas programinis pertraukimas ir pereinama į ciklo pabaigą;

6) operandai - pora (x₁, x₂) - transformuojami į realų adresą;

7) analogiškas (2);

8) interpretuojama komanda ir padidinama taktų skaitiklio count reikšmę;

9) taimerio aparatūra nustato laiko pokytį ir jei reikia, inicijuoja taimerio pertraukimą. Tai iš esmės nėra interpretatoriaus darbo dalis. Tačiau kadangi procesorius ir taimeris dirba tuo pat metu, valdomi taktų generatoriaus duodamų impulsų, tai algoritmizuojant šį procesą, natūralu taimerio darbą pateikti šioje vietoje. Logiška įtraukti taimerį į virtualią mašiną ir dėl to, kad jis skirtas būtent jos darbo laikui skaičiuoti;

10) ciklo pabaiga. Čia galimi pertraukimai (pertraukimai apdorojami tik komandinio ciklo pabaigoje), ir todėl komanda Test mėginama nustatyti, ar jie buvo. Jei ne - grįztama į ciklo pradžią ir viskas kartojama. Priešingu atveju procesorius persijungia į supervizoriaus režimą ir išeina iš interpretatoriaus ciklo.

Pasitelkę formalius žymėjimus, gautume tokią procedūrą:

```
procedure VM_Proc;
    var realaddr:integer;
        opc, opr:array [1..2] of char;
begin
    /* komandos interpretavimo ciklas */
    VM_proc_loop:
    /* nustatomas realus adresas: galimas perėjimas į žymę ERROR (3.2.7) */
        test_transf_addr(IC, realaddr);
    /* IC reikšmės padidinimas vienetu */
        increment(IC);
    /* operacijos kodo ir operandų išskyrimas iš komandos */
        opc := concat(supervisor_memory[realaddr][1],
                      supervisor_memory[realaddr][2]);
        opr := concat(supervisor_memory[realaddr][3],
                      supervisor_memory[realaddr][4]);
    /* komandos interpretavimas */
        case opc of
            'LR', 'SR', 'CR', 'GD', 'PD':
                begin
                    /* operando reikšmė transformuojama į realų adresą: galimas perėjimas į žymę ERROR (3.2.7) */
                        test_transf_addr(opr, realaddr);
                        case opr of
                            'LR': R:=supervisor_memory[realaddr];
                            'SR': supervisor_memory[realaddr]:=R;
                            'CR': if R=supervisor_memory[realaddr] then
                                    c := 'T'
                                else c := 'F';
                            'GD':begin
                                    input_page := opr[1];
                                    SI := 1;
                                    end;
                            'PD':begin
                                    output_page := opr[1];
                                    SI := 2;
                                    end
                                end;
                            'BT': IC := opr;
                            'H ': SI := 3;
                        else PI := 2
                        end;
                    /* veikia taimerio aparatūra */
                    count := count + 1;
                    Timer;
                ERROR:
                    /* nesant klaidai ar kitiems pertraukimams, kartojamas interpretavimo ciklas */
                    if Test(5) = 0 then goto VM_proc_loop;
```

```

    /* išsaugomos registrų reikšmės ir procesorius per jungiamas į
supervizoriaus režimą */
    ptr := PTR; r := R; c := C; ic := IC;
    MODE := 'Master'
end;

```

Čia procedūra *increment* padidina skaičių simbolinio žodžio pavidale vienetu o funkcija *concat* - iš dviejų baitų suformuoja pusžodi:

```

procedure increment(var counter:array[1..2]of char);
function concat(bytel,byte2:char):array[1..2]of char;

```

Įvykus pertraukimui valdymas perduodamas pertraukimų apdorojimo programomis (žr. 3.2.1 skyr.).

Kaip matyti iš algoritmo - realiai virtuali mašina gali dirbti iki pirmo pertraukimo. Būtent pertraukimai leidžia logiškai išlygiagretinti vartotojiškų ir sisteminių procesų darbą, pirmiesiems gavus procesorių, nes tik įvykus pertraukimui iš jų galima ji "atimti". Apdorojus pertraukimus, valdymą vėl perims operacine sistema, konkrečiai resursų paskirstytojai ir procesų planuotojas, kuris atiduos procesorių pirmajam pasiruošusiam darbui procesui. Pertrauktoji virtuali mašina galės pratęsti darbą tik vėl atsidūrusi pasiruošusių procesų sąrašo pradžioje (jei pertraukimas neinicijavo jos darbo pabaigos).

Detaliau apie procesus rašoma 3.4 poskyryje.

3.4. Modelinė multiprograminė operacinė sistema

3.4.1. Procesai ir resursai

Ši projekto dalis - pati svarbiausia, nes čia konstruojama pati MOS, kurioje realizuosis multiprogramavimo principai. Reikia įsidėmėti, jog čia turimas omenyje multiprogramavimas plačiąja prasme, o ne tik siauraja - vartotojo požiūriu. Tai reiškia, kad laisvai varžytis dėl centrinio procesoriaus ir atminties resursų galės ne tik virtualių mašinų procesai, bet ir procesai, sudarantys pačią operacинę sistemą (pastarieji dar gali varžytis ir dėl kitų resursų). Sukonstruoti MOS kaip tarpusavyje sąveikaujančių lygiagrečių procesų rinkinį - toks yra šio etapo uždavinys.

Konstruojant multiprograminę operacинę sistemą, reikia atsisakyti tokios įprasto programavimo sąvokos, kaip valdymo perdavimas. Procesai negali perduoti arba gauti valdymą, nes toks perdavimas nėra apibrėžtas. Kalbant apie valdymo perdavimą, žinoma, iš kur valdymas yra gautas ir kam jis perduodamas. Be to, tai padaroma iš anksto numatytu momentu, vienam moduliui kviečiant kitą. Tuo tarpu procesas "nežino" iš ko jis "gaus" valdymą, kada jis "gaus", kam jis atiteks po to. Loginiame lygyje procesai yra lygiagretūs, t. y. "turi valdymą" tuo pat metu. Taigi čia apie valdymo perdavimą kalbėti nėra prasmės. Panašiai yra ir realiame lygyje: procesas gauna procesorių atsitiktiniu momentu, negali pats nurodyti, kam procesorius atiteks po jo ir iš anksto

"nežino", kuriuo metu tai įvyks. Taigi klasikinė valdymo perdavimo sąvoka šluo atveju netinka.

Jau buvo užsiminta, kaip galima "atimti" procesorių iš vartotojų ško proceso (virtualios mašinos) būtent pertraukimų ir jų apdorojimo pagalba. Taigi virtualių mašinų darbą leidžia išlygiagretinti pertraukimų mechanizmas. Jis netinka sisteminių procesų išlygiagretinimui, nes procesorius supervizoriaus režime negali būti pertraukiama. Šiuo atveju į pagalbą ateina resurso sąvoka.

Procesas gali gauti procesorių tik tuomet, kai pasiruošęs darbui, t. y. turi visus kitus reikiamus resursus. Vadinas, sisteminis procesas gali pradėti dirbti tik tada, kai sistemoje taps laisvi arba atsiras reikiami resursai. Be to proceso programa yra iš esmės ciklinė, kurios kiekviename cikle galbūt kažkaip eikvojami turimi resursai. Tada gali susidaryti situacija, kai jų pritruks. Bendru atveju iš proceso tokioje situacijoje bus atimtas procesorius, o pats procesas bus pastatytas į eilę prie trūkstamo resurso. Tuo tarpu procescrius gali būti atiduotas kažkuriams kitam pasiruošusiam darbui procesui. Be tokios tipinės situacijos galimos ir kitos: procesas tiesiogiai neeikvoja jokio resurso, bet tam tikri resursai gali pasidaryti reikalingi darbo eigoje, pavyzdžiui: periferiniai įrenginiai, apdorojant kokią nors įvedimo-įšvedimo operaciją; reikiamas resursas nėra užimtas, bet jo dar tiesiog nėra sistemoje ir panašiai. Visų šių situacijų rezultatas vienodas - aktyvus procesas pereina į blokavimo būseną tol, kol bus patenkinti resurso poreikiai, o procesorių gauna kitas pasiruošęs darbui procesas.

Tokiu būdu - per blokavimo dėl resursų trūkumo mechanizmą - galima išlygiagretinti sisteminių procesų darbą.

Tiek resursai, tiek procesai yra dinaminiai objektai. Jie gali būti sukuriami ir naikinami iš esmės bet kuriuo sistemos darbo metu. Šiuo požiūriu yra aktyvūs tik proceai: jie gali kurti kitus procesus ir resursus. Resursai to daryti nesugeba. Visi procesai ir resursai turi būti sukurti kažkurio proceso. Jie negali atsirasti "iš niekur". Yra šiuo atveju išimtis - pradinis procesas, inicijuojantis sistemos generaciją. Galima laikyti, kad jis sukuriamas automatiškai, įjungiant mašiną, ir iš karto tampa einamuoju. Pradinis procesas yra atsakingas ne tik už kitų pagrindinių procesų generaciją, bet ir už statinių resursų, tokių kaip atmintis, periferiniai įrenginiai, sukūrimą. (Nesunku suprasti, kad realiai šie resursai jau egzistuoja, tačiau juos vis tiek reikia "sukurti", t. y. pranešti operacinei sistemai, kad jie yra ir juos galima valdyti). Sukurtieji procesai kuria žemesnio rango procesus bei resursus ir t. t. Gaunamas tam tikras procesų ir resursų medis, kuriame ryžiai yra *tėvas* - *sūnus* tipo. Šis medis turi prasmę tik generavimo ir naikinimo metu (prosesą ar resursą bendru atveju gali sunaikinti tik jo "tėvas").

Darbo metu "tėvystės" ryšiai niekaip nedaro įtakos procesų veikimui - "tėvas" negali valdyti nei sukurto proceso, nei resurso. Procesas - "tėvas" ir procesas - "sūnus" logiškai dirba lygiagrečiai ir bendru atveju turi tas pačias teise į resursus.

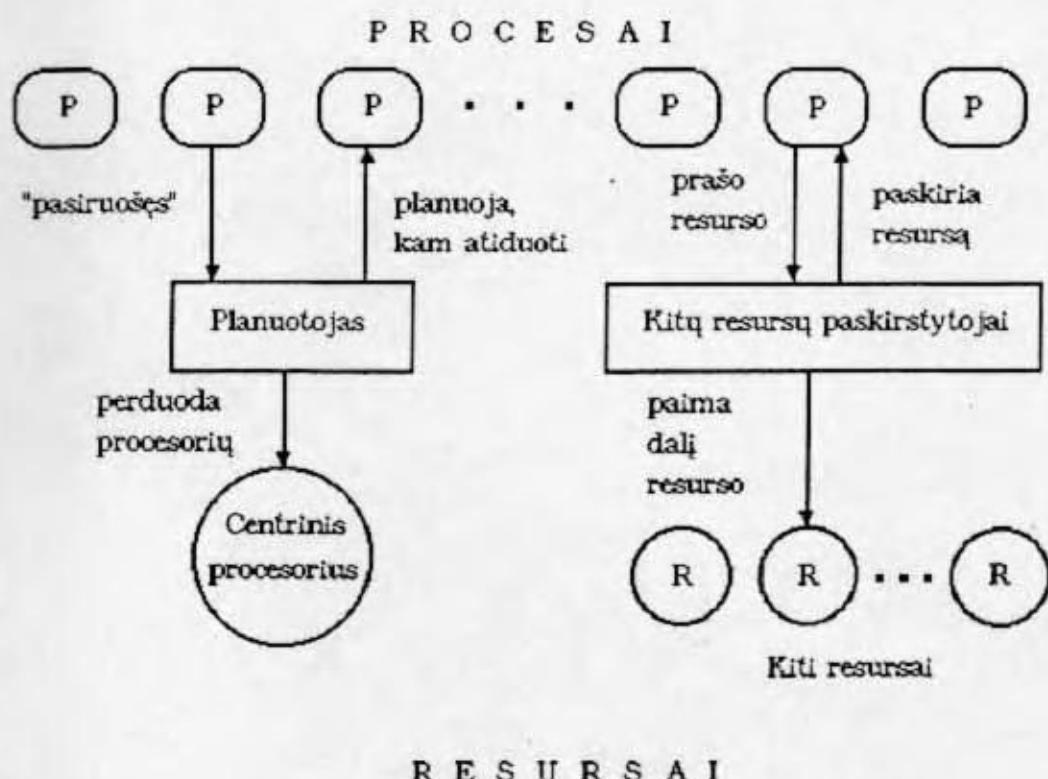
3.4.2. Resursai ir jų skirstymas

Operacinės sistemos naudingo darbo metu (t. y. atmetus sistemos generaciją ir degeneraciją, prieš išjungiant mašiną) galima skirti dvi resursų rūšis:

- statiniai resursai,
- dinaminiai resursai.

Statiniai resursai - tie resursai, kurie yra sukuriami, generuojant sistemą, ir naikinami tik jai baigus darbą, t. y. degenaracijos metu. Jų dydis sistemos darbo metu pasilieka pastovus. Šie resursai "uždengia" skaičiavimo sistemos aparatūrą: centrinį procesorių, atmintį, kanalus. Taip pat šiai klasei priklauso sistemos pranešimų tekstai.

Dinaminiai resursai gali būti sukurti ir sunaikinti bet kuriuo sistemos darbo momentu. Šie resursai turi gryna loginė pobūdį - bendru atveju tai kokios nors informacijos blokai: programų tekstai, aprašai, procesų tarpusavio pranešimai.



9.pieš.

Procesai patys resursų pasiimti negali. Jie taip pat negali tiesiogiai vienas su kitu sąveikauti. Resursai savo ruožtu yra visiškai pasyvūs. Kas skirsto procesams resursus?

MOS iš tikrujų yra sudaryta ne iš dviejų, o iš trijų tipų komponenčių procesų, resursų ir resursų paskirstytojų.

Resursų paskirstytojai yra tarpinė grandis tarp procesų ir resursų. Būtent paskirstytojai organizuoja procesų darbo palaikymą, pateikdami jems reikalingus resursus, tame tarpe ir centrinių procesorių. Tai yra savarankiški programiniai fragmentai. Juos taip pat galima laikyti dinaminiais objektais, nes jie pradeda funkcionuoti ir apskritai turi prasmę tik tuomet, kai atsiranda atitinkami resursai. Vis dėlto patogiau juos laikyti statiniais objektais (tai bus paaiškinta vėliau).

Bendru atveju kiekvienas resursas turi savo paskirstytoją. Paskirstytojas gali aptarnauti bet kurį procesą, prašantį jo valdomo resurso. Išskirtinę padėtį užima centrinio procesoriaus resurso paskirstytojas, dar vadintamas planuotoju. Jis sprendžia kuris aktyvus procesas iš pasiruošusio darbui tampa einamuoju.

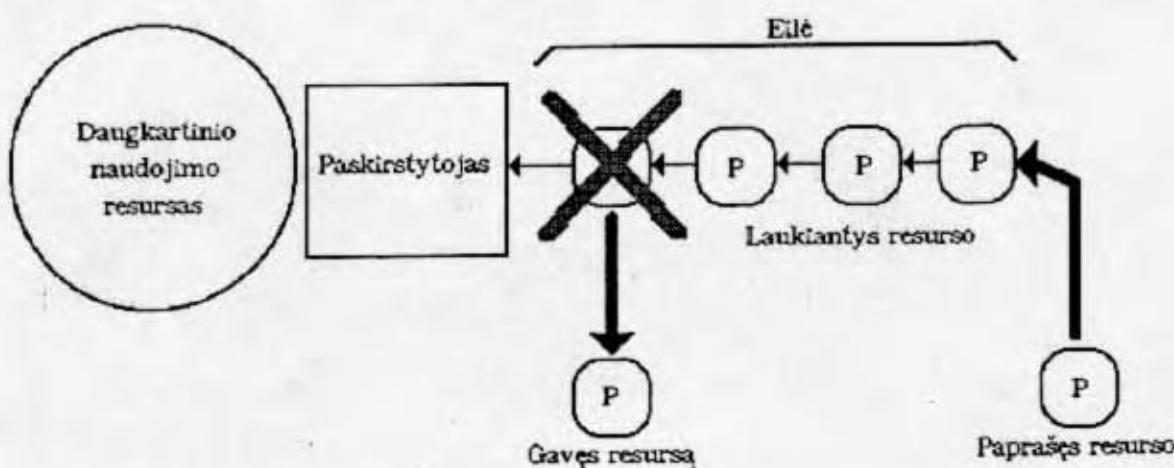
MOS sandaros ir funkcionavimo schema pateikiama 9 piešinyje.

Resursus galima klasifikuoti ir kitu aspektu - jų naudojimo kartotinumo: 1) daugkartinio naudojimo resursai, 2) vienkartinio naudojimo resursai.

Daugkartinio naudojimo resursai - procesorius, bendros paskirties atmintis, sisteminių pranešimų tekstai, kanalai, buferiai ir kt. Juos sistemos darbo eigoje galima naudoti kiek reikia kartų.

Vienkartinio naudojimo resursai - tai specializuotos paskirties resursai - tam tikra informacija, kurios laukia ir kurią gali apdoroti tik vienas procesas. Šiemis resursams priklauso įvairių pavidalu informacija apie vartotojo užduotį, procesų tarpusavio pranešimai.

Procesai privalo panaudotus neberekalingus resursus atlaisvinti, t. y. leisti jais naudotis kitiems procesams. Iš esmės taip yra su daugkartinio naudojimo resursais. Tuo tarpu vienkartiniai resursai yra sunaudojami visiškai juos gavusių procesų ir todėl apie jų atlaisvinimą kalbetti nėra prasmės.



10 pieš.

Nesunku pastebėti, jog aprašytieji resursai yra skirtingo sudėtingumo. Iš to seka, kad ir jų paskirstytojai yra skirtingo sudėtingumo. Sudėtingiausias yra daugkartinio panaudojimo resursų paskirstymas, nes tai yra resursai, dėl kurių konkuruojama, kadangi procesų reikalaujamą dalį sume viršija tikrąjį resursų apimtį. Galimos įvairios strategijos (žr. 12 poskyri). Patogu pasirinkti eilės organizaciją su kiekvienu resursu susiejamas jo laukiančiu procesų sąrašas. Anksčiau iš jų patekės, t. y. anksčiau paprašęs resurso procesas anksčiau jį ir gauna. Pirmiausia resursą gauna pirmasis eilės narys, kuris po to iš eilės išbraukiamas. Kiekvienas naujas resurso prašytojas talpinamas į eilės pabaigą (sudėtingesniu atveju, įvedus prioritetus, kiekvienam iš pastarujuju sudaroma atskira eilė; žr. 10 pieš.). Sudėtingesniu atveju to paties prioriteto procesai gali konkuruoti ir eilės viduje: aptarnaujamas nebūtinai pirmasis eilėje, o tas, kuriam pirmajam pakanka turimo resurso. Taigi šiuo atveju, pavyzdžiui, daug atminties reikalaujanti užduotis turės "praleisti į priekį" užduotis, kurių poreikiai mažesni, - tai pagreitina užduočių srauto apdorojimą.

Paprastesnis yra vienkartinio naudojimo resursų skirstymas. Čia paskirstytojai stebi, ar neatsirado reikiamu resursu, ir jei taip, tai informuoja apie tai atitinkamą laukiantį procesą, nurodydamas resurso parametrus.

Iš bendros taisykles išskrinta pranešimo tipo resursai. Projektuojant patogu yra laikyti visus pranešimus vieno pobūdžio resursu ir su juo susieti pranešimo laukiančiu procesų grupę (ne eilę). Atsiradus pranešimams, paskirstytojas turi aptarnauti visus jų laukiančius procesus, nesvarbu, kokia tvarka jie paprašė pranešimo resurso.

Projektuojant modelį patogu logiškai sujungti visus paskirstytojus į vieną bloką. Jame paskutinis turėtų eiti planuotojas, o visų kitų resursų paskirstytojai - anksčiau, laisvai pasirinkta tvarka (turėtų būti pirma aptarnaujami visų įmanomų gauti resursų (išskyruis centrinio procesoriaus) poreikiai, o tik po to tikrinamas pasiruošusių procesų sąrašas ir kažkam atiduodamas procesorius).

Toks apjungimas natūralus ir koncepciniame lygyje: paskirstytojų bloką galima taip pat laikyti cikliniu procesu, kuris gauna procesorių, kai atsiranda resurso poreikio situacija, ir yra blokuojamas, kai daugiau poreikių, negu aptarnauta, patenkinti neberekia.

Vis dėlto paskirstytojų bloką patogiau laikyti statiniu paprogramių rinkiniu. Sistemos lygyje jis nėra apiforminamas kaip procesas. Tai yra tiesiog tarybinė įranga, be kurios negalėtų funkcionuoti procesų ir resursų aparatas. Be to, jei ir paskirstytojus apiformintume kaip procesą, nebūtų aišku, kas turėtų aptarnauti juos pačius.

Paskirstytojai paprastai nėra išreikštiniu būdu kviečiami iš proceso programos. Procesas tiesiog paprašo reikalingo resurso, o atitinkamas paskirstytojas iškviečiamas, aptarnaujant resurso prašymą žemesniame lygyje.

3.4.3. Sisteminiai ir vartotojiški procesai

MOS apdorojant vartorojo užduočių paketą, joje vyksta dviejų rūšių procesai: sisteminiai ir vartotojiški. Nors jų funkcionavimo principai yra tie patys, tačiau statusą jie turi skirtą. Ši statusą apibūdina jų generavimas,

egzistavimo trukmė. prioritetas procesoriaus atžvilgiu. santykis su kitais resursais ir panašiai.

Sisteminių procesų - tai procesas, atliekantis su MOS darbu susijusias funkcijas - duomenų įvedimą-išvedimą, išorinių įrenginių aptarnavimą, duomenų apdorojimą žemesniame lygyje (ne vartotojo), vartotojo užduoties atlikimo palaikymą ir kt.

Vartotojo procesas - tai procesas, užimtas išimtinai vartotojo užduoties atlikimu, t. y. programos su pradiniais duomenimis (ar be jų) interpretavimu ir rezultatų pateikimu.

Paprasciausiamame projekto variante sisteminių procesų yra permanentiniai sistemos darbo metu, t. y. jie sukuriami, generuojanč (paleidžiant) sistemą, o naikinami - ją išjungiant. Sisteminių procesų generuojami anksčiau negu vartotojiški, nes pastaruosius jie turi būti pasiruošę aptarnauti. Visą sistemos darbo laikotarpį (nesvarbu, ar apdorojamas užduočių paketas, ar ne) sisteminių procesų yra aktyvūs: blokuoti, pasiruošę darbui arba einamuoju momentu dirbantys. Išimtį sudaro sisteminių procesų, atliekantys interfeiso tarp vartotojiško proceso ir sistemės aplinkos vaidmenį - jie inicijuoja vartotojiškus procesus, tvarko su jais susijusius duomenis, kolekcionuoja rezultatus ir sisteminius pranešimus, ruošia listingą ir kt. Šie procesai turi prasmę tik apdorojant vartotojo užduotis ir todėl kuriami betarpiskai prieš vartotojiško proceso atsiradimą bei naikinami, jam baigus darbą.

Vartotojiški procesai - virtualios mašinos - sistemos darbo trukmės požiūriu yra laikini: jie gali būti sukurti ir sunaikinti bet kuriuo iš anksto nenusakomu momentu. Vartotojiški procesai yra sukuriami sisteminių procesų (bet ne atvirkščiai).

Natūralu yra įvesti bent dviejų lygių prioritetus: aukštesnį - sisteminiams procesams, žemesnį - vartotojo procesams. Sisteminių procesų turėtų turėti aukštesnį prioritetą todėl, kad jie skirti aptarnauti vartotojiškiems procesams (MOS yra skirta efektyviai apdoroti vartotojo užduotims). Ši teiginjį paauskina tai, kad norint pagreitinti vartotojiškų procesų darbą bendru atveju yra svarbu greitinti jų aptarnavimą - įvedimą-išvedimą, užduoties paruošimą ir t. t. Kuo greičiau bus aptarnautas vartotojo procesas, tuo greičiau jis veiks.

Toks prioritetų mechanizmas šiame projekte reikalinas, skirstant procesoriaus laiką - sisteminių procesų turi gauti procesorių anksčiau, negu vartotojiški. Centrinio procesoriaus resursas - vienintelis, dėl kurio varžosi ir sisteminių, ir vartotojiški procesai. Daugiau tokų bendrų resursų nėra: išorinius įrenginius (tiksliau - kanalus) naudoja tik sisteminių procesų, o vidinės atminties vartotojo ir sisteminių paskirties sritys yra griežtai atribuotas.

Vartotojiški procesai varžosi tik dėl vieno resurso - procesoriaus. Dėl atminties tiesioginio varžymosi nėra, nes vartotojo užduotis tampa procesu tik tada, kai atsiranda jos programą vykdanti virtuali mašina, t. y. kai vartotojo atmintis jau yra išskirta. Sudėtingesniame projekto variante galima įvesti ir varžymąsi dėl vartotojo atminties tarp virtualių mašinų, kai virtualiai mašinai išskirta atmintis yra mažesnė už galimus poreikius, ir jų nėra fiksuoto dydžio - t. y. darbo eigoje gali kisti. Nagrinėjamame variante to nėra padaryta.

Vartotojiškas procesas pats nekuria nei procesų, nei resursų. Jo duomenis ir rezultatus kaip resursus apipavidalina specialūs sisteminiai procesai.

Sisteminiai procesai, be procesoriaus laiko, gali varžytis dėl išorinės atminties, trečiojo kanalo, buferių, kitos pagalbinės supervizorinės atminties. Svarbu šioje vietoje suvokti, kad varžomasi tik dėl pagalbinės - *darbinės* - atminties, o ne atminties reikalingos pačio proceso programai. Sisteminio proceso programa visa yra supervizoriaus atminties fiksuootoje srityje (sudētingesniame variante galima įvesti dinamiškumą). Atmintis, dėl kurios varžomasi, reikalinga tik kuriant vieną ar kitą resursą ir panašiai.

3.4.4. Prosesų ir resursų valdymo blokai (deskriptoriai)

Proceso ar resurso valdymo blokas, dar vadinamas deskriptoriu, yra fiksuoto formato duomenų struktūra, sauganti informaciją apie proceso ar resurso einamąjį stovę. Remiantis informacija *proceso deskriptoriuje*, korektiškai pratesiamas blokuoto proceso vykdymas, procesams skirtomas centrinio procesoriaus laikas ir kiti resursai. *Resurso deskriptoriuje* nurodomas jo užimtumo laipsnis, laisvas kiekis ir kt. - šia informacija naudojasi duotojo resurso paskirstytojas.

Resurso arba proceso sukūrimas - atitinkamos deskriptorinės struktūros užpildymas.

Formaliai apibrėžime deskriptorių duomenų struktūras:

```
/* procesų sąrašo apibrėžimas */
type proclist = ^record
    proc:integer;
    next:proclist
end;

/* resurso dailies aprašymas */
res_part = record
    ammount:integer;
    defin:array [1..MAXIT] of integer;
    messid:array [1..MAXNAME] of char
end;

/* resursų sąrašo apibrėžimas */
reslist = ^record
    rec:integer;
    part:res_part;
    next:reslist
end;

/* procesoriaus būsenos aprašymo apibrėžimas */
cpustate = record
    mode:array [1..6] of char;
    entry:integer;
    r:array [1..4] of char;
    ic:array [1..2] of char;
    c:char;
    ptr:array [1..4] of char
    clock:integer;
```

```

        time:integer;
      end;
/* resurso laukiančių procesų eilės (ir grupės) apibrėžimas */
queue = record
  procid:integer;
  wanted:res_part;
  next:queue
end;
/* proceso valdymo bloko apibrėžimas */
pdeskr = record
  id:array [1..MAXNAME] of char;
  cpu:cpustate;
  state:array [1..6] of char;
  relations:record
    parent:integer;
    sons:proclist
  end;
  wplist:queue;
  given_res:reslist;
  maked_res:reslist;
  priority:integer;
  procedure programme;
end;
/* resurso pasiekiamumo aprašymo apibrėžimas */
acc = record
  message:boolean;
  list:array [1..MAXIT] of integer;
  number, receiver:integer
end;
/* resurso valdymo bloko apibrėžimas */
rdeskr = record
  header:record
    id:array [1..MAXNAME] of char;
    status:boolean;
    creator:integer
  end;
  accessibility:acc;
  waiting:queue;
  procedure distributor
end;

```

Proceso valdymo blokas sudarytas iš tokių pagrindinių laukų:

1) proceso (arba užduoties) išorinis vardas *td*. Tai simbolinis proceso identifikatorius, reikalingas procesui skirti nuo kitų išoriškai - sisteminio operatoriaus ar sisteminio programuotojo lygyje. Pagal šį vardą operatorius gali identifikuoti, kurie procesai yra aktyvūs, kuris iš jų yra einamasis ir t.t (jei leidžia sistemos interfeisinės priemonės), o programuotojas gali organizuoti sistemoje statinius objektų tarpusavio ryšius (pavyzdžiu, proceso ir jam skirto specialaus pranešimo).

2) procesoriaus einamosios būsenos aprašymas *cpustate* dirbant su konkrečiu procesu. Pagal šiame lauke esančią informaciją procesorius gali

korektiškai pratęsti darbą su nutrauktuoju procesu. Einamosios būsenos išprašymo informacija yra struktūruota:

2.1) procesoriaus darbo režimas *mode*. Tai simbolinė reikšmė, nurodanti, kokiame režime procesorius turi vykdyti proceso programą, t. y. koks yra procesas: sisteminis ("Master") ar vartotojo ("Slave");

2.2) jėjimo taškas *entry*. Šis sublaukas yra apibrėžtas tik sisteminiam procesui. Kadangi laikoma, jog sisteminį proceso programos ašomas aukšto lygio kalba, proceso darbo pratęsimo taškus patogu žymėti tos žalbos priemonėmis - pavyzdžiu, sveikaisiais skaičiais;

2.3) laukai procesoriaus registrų reikšmėms saugoti *r*, *lc*, *c*. Šie laukai yra prasmę tik virtualioms mašinoms, kai procesorius dirba vartotojo režime. Ypač galima atstatyti virtualios mašinos būseną ir korektiškai pratęsti vartotojo užduoties programos vykdymą;

2.4) laukas puslapio lentelės registro reikšmei saugoti *ptr*. Apibrėžtas iš vartotojiškam procesui;

2.5) virtualios mašinos lokalaus laiko ir sutartinės darbo trukmės kaitliukai *clock* ir *time*.

Proceso darbo pradžioje laukai inicijuojami pradinėmis reikšmėmis. Procesui esant einamuoju, lauką *cpustate* reikia laikyti neapibrėžtu. Kai įtertraukiama proceso darbas, einamoji procesoriaus būsena - režimas, ypač jėjimo taškas sisteminiam procesui arba procesoriaus registrų reikšmės vartotojo proceso - įrašomi į atitinkamus laukus. Procesui vėl tapus einamuoju, pagal šias reikšmes nustatomas procesoriaus darbo režimas ir registrų reikšmės.

3) proceso būsenos pavadinimas *state*. Čia fiksuojama einamoji proceso būsena: ar jis blokuotas ("Blocked"), ar pasiruošęs darbuti ("Ready"), ar einamasis lėrbantis ("Running");

4) proceso "tėvystės" ryšiai nusakomi lauke *relations*. Proceso "tėvas" nurodomas jo eilės numeriu proceso masyve - vidiniu vardu *parent* (žr. šiame skyrelyje žemiau). Nuoroda į proceso "sūnų" sąrašą talpinama lauke *sons*;

5) laukiančių proceso sąrašas, kuriam duotu momentu priklauso ištarasis procesas, kai jis yra blokuotas, nurodomas lauke *wplist*;

6) nuoroda į procesui suteiktą sukūrimo metu arba po paprašymo esusų sąrašą saugoma lauke *given_res*;

7) nuoroda į proceso sukurtų resursų sąrašą sugoma lauke *maked_res*;

8) proceso prioriteto numeris nurodomas lauke *priority*;

9) proceso programos jėjimo taškas nurodomas lauke *programme*.

Akivaizdu, jog esant ribotiem realios mašinos resursams, joje veikiančių proceso nebūs daugiau už tam tikrą pastovų skaičių. Todėl patogu laikyti, kad proceso deskriptoriai yra surašyti į masyvą, kurio panaudotos dalies ilgis (t. y. iki šių metų sistemoje yra dirbančių proceso) žinomas kiekvienu momentu. Kiekviename masyvo elementui priskiriama skalčius - to elemento indeksas masyve. Jis vadinamas *vidiniu vardu*. Vidinis vardas proceso suteikiama linamiskai, proceso atsiradus, ir nepriklauso bendru atveju nuo proceso tipo ar atsiradimo laiko. Šis identifikatorius neturi jokios prasmės vartotojui arba sistemininkui. Tai grynai sistemos vidiniams tikslams naudojamas vardas.

Procesų aprašymas vienodos struktūros deskriptorių pavidalu, ir tiesiogiai nuo proceso nepriklausančių vidinių vardų suteikimas leidžia pasiekti pakankamai aukštą proceso abstrakcijos laipsnį. Tai labai naudinga planuotojo darbe, kai procesoriaus resursas turi būti pateiktas bet kuriam procesui.

Analogiškai "svienodinami" ir resursai. Kiekvienas resursas aprašomas fiksuotos struktūros valdymo bloku:

1) resurso aprašymo antraštės laukas *header* yra tokios struktūros:

1.1) resurso išorinis vardas *id*. Tai simbolinis resurso identifikatorius. Jo paskirtis analogiška atitinkamo lauko proceso deskriptoriuje paskirčiai:

1.2) resurso daugkartinio naudojamumo požymis *status*. Tai loginę reikšmę įgyjantis laukas: ji teigama, kai resursas yra daugkartinio panaudojimo, ir neigama, kai resursas vienkartinio panaudojimo. Praktinė šio lauko reikšmė yra tokia: jei resursas daugkartinio panaudojimo, ji reikia atlaisvinti, kai jis neberekalingas.

1.3) resurso kūrėjo - proceso - vidinis vardas fiksuojamas lauke *creator*. Pagal šį lauką galima nustatyti, pavyzdžiui, kas pasiuntė pranešimą, jei į jį reikia specializuotai atsakyti.

2) lauke *accessibility* pateikiama nuoroda į resurso pasiekiamumo aprašymą. Šiame lauke aprašomas visas duotu momentu laisvas resursas.

Pastarojo lauko struktūra priklauso nuo projektuojamo modelio sudėtingumo, aukšto lygio kalbos procesoriaus HLP bazinėje mašinoje teikiamų galimybių duomenų struktūroms aprašyti ir panašiai. Bendru atveju sunku rekomenduoti ką nors konkretaus. Tačiau iš principo šis laukas turi būti struktūruotas, turintis aukštą bendrumo laipsnį, nes Jame turi būti įmanoma aprašyti visų tipų resursus, kokie gali atsirasti projektuojamoje operacinėje sistemoje. Nagrinėjamam modelio variantui galima pasiūlyti struktūrą su alternatyviniais laukais:

2.1) pranešimo požymis *message*. Praktiškai visus sistemoje egzistuojančius resursus galima suskirstyti į dvi grupes: išmatuojamus ir neišmatuojamus (atskirais atvejais šios grupės gali būti dalies persidengti). *Išmatuojami resursai* - tai vidinė ir išorinė atmintis bei į jas panašūs agregatai, kaip "programa", "užduotis" ir t.t. *Neišmatuojamieji resursai* - pranešimo tipo resursai - tai pranešimai ir kanalai. Pranešimo išmatuoti negalima - galima tik konstatuoti, kad jis yra arba jo néra. Analogiškai yra ir su kanalais - jie arba užimti, arba laisvi - taigi visai logiška juos aprašyti taip pat, kaip pranešimus.

Jei požymis *message* turi teigiamą reikšmę, tai resursas yra pranešimo tipo, jei neigiamą - tai išmatuojamasis:

2.2) laukas *receiver* bendru atveju turi prasmę ir yra apibrėžtas tik tuomet, kai požymis *message* turi teigiamą reikšmę, t.y. resursas yra pranešimo tipo - tiksliau - pranešimas. Šiame lauke nurodomas pranešimo adresato - proceso, kuriam skirtas pranešimas - vidinis vardas (atskiru atveju šis laukas gali būti užpildomas ir tada, kai resursas yra išmatuojamasis, bet skirtas tik konkrečiam gavėjui);

2.3) lauke *list* pateikiamas išmatuojamojo resurso elementų - blokų ar takelių - numeriu masyvas (sarašas). Jis yra traktuojamas kaip sutvarkytas sarašas, t.y. elementai išrenkami būtent ta tvarka, kaip jie eina saraše (tai turi

prasmę ne visada: laisvą atmintį galima imti bet kuria tvarka, - bet tokiam resursui, kaip užduoties kortų paketas, loginė tvarka yra būtina). Laukas *list* bendru atveju yra apibrežtas tik tada, kai požymis *message* turi neigiamą reikšmę:

2.4) laukas *number* yra tiesiogiai susijęs su lauku *list*. Jis fiksuoja, kiek sąraše yra elementų, t. y. kiek masyvo elementų yra užimti, ir yra apibrežtas tik išmatuojamam resursui;

3) lauke *waiting* saugoma nuoroda į resurso laukiančių procesų sąrašą (eilę arba grupę):

Laukiančių procesų eilės elementas yra struktūrizuojamas, o struktūros sudėtingumas priklauso nuo tų pačių faktorių, kaip ir resurso pasiekiamumo aprašymo struktūros sudėtingumas. Pavyzdžiu, galima tokia struktūra:

3.1) lauke *procid* saugomas resurso paprašiusiojo ir laukiančiojo proceso vidinis vardas;

3.2) reikalinga resurso dalis nurodoma lauke *wanted*. Šis laukas skaidytinas į polaukius:

3.2.1) lauke *amount* nurodomas reikiamas išmatuojamo resurso kiekis;

3.2.2) išmatuojamos resurso dalies apibrežimui (pasiekiamumui) nurodyti yra skirtas laukas *defin*. Ši lauką užpildo resurso paskirstytojas, perduodamas reikalingo dydžio dalį procesui. Laukas apibrežtas tik išmatuojamam resursui;

3.2.3) laukiamo pranešimo tipo resurso išorinis vardas nurodomas lauke *messid*. Tai simbolinis laukiamo pranešimo arba reikiamo kanalo identifikatorius. Vidinis vardas šiam tikslui netinka, nes bendru atveju procesai "nežino", kas "slepiasi" po vidiniu vardu (tiksliau, proceso programą rašančiajam asmeniui žinomi tik fiksoti išoriniai varda).

3.3) lauke *next* saugoma nuoroda į sekantį sąrašo elementą;

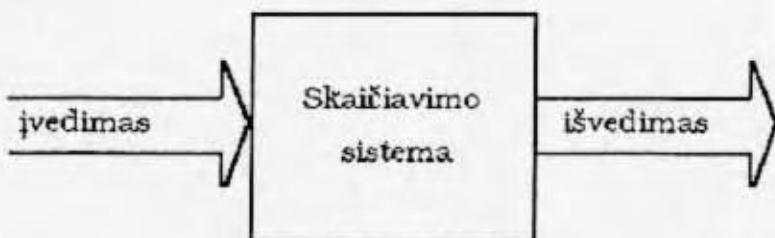
4) lauke *distributor* nurodomas resurso paskirstytojo programos adresas.

Panašiai, kaip ir procesų resursų maksimalus kiekis (čia turimas omenyje ne resurso dydis) sistemoje yra žinomas. Todėl patogu resursų deskriptorius patalpinti į vienmatį masyvą, tuo pačiu suteikiant kiekvienam resursui vidinį vardą - masyvo elemento numerį. Tuo būdu analogiškai, kaip ir procesų atveju gauname patogią resurso abstrakciją, kuri leidžia panašiai traktuoti bet kurio tipo ir struktūros resursą, esantį sistemoje.

Negalima laikyti šiame skyrelyje pateiktų valdymo blokų aprašymų išbaigtais. Tai iš esmės yra metmenys, o deskriptoriaus sudėtį konkretizuoti gali tik realizacijos metu iškylantys konkretūs reikalavimai.

3.4.5. Vartotojo užduoties kitimas sistemoje

Pačiame aukščiausiam abstrakcijos lygyje skaičiavimo sistemą, galima laikyti "juoda dėže" su vienu įėjimu (įvedimo srautas) ir vienu išėjimu (išvedimo srautas), kaip parodyta II piešinyje.



II pieš.

Įvedimo srautas - tai vartotojo užduočių paketas, skirtas apdorojimui, o *išvedimo srautas* - to paketo apdorojimo rezultatai - listingas.

Užduočių paketą sudaro elementai - užduotys. Jos apdorojimo metu pereina per eilę būsenų:

- 1) fizinė užduotis - tekstas (kortų paketas)
- 2) užduotis - resursas,
- 3) užduotis - procesas,
- 4) užduoties listingas.

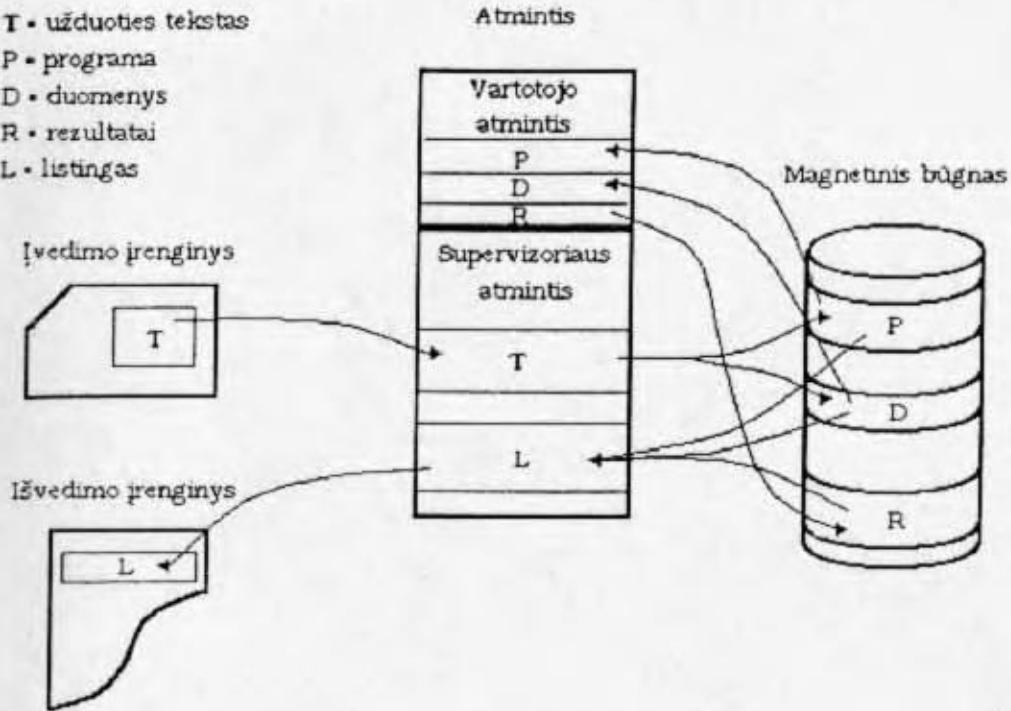
Fizinė užduotis - tai tokis užduoties pavidalas, kuriuo užduotis suprantama įvedimo sraute. Faktiškai tai yra tekstai - programos, duomenų aprašo su valdančiaisiais parametrais, - užrašyti (perforuoti ir sutvarkyti) tam tikru formatu. Tai yra tiesiog įvedimo duomenys sistemai.

Patekusi į sistemą, fizinė užduotis interpretuojama - išskiriama programa, duomenys, nustatomi vykdymo parametrai. Tolesniams apdorojimui užduotis paverčiama *resursu* - tuo, ko laukia užduoties apdorojimo procesai, t. y. suformuojamas atitinkamas resurso valdymo blokas (deskriptorių). Šiame etape užduotis iš esmės padaroma žinoma sistemai kaip tokia.

Kai užduotis žinoma ir yra visi jos vykdymui reikalingi resursai, pereinama prie jos atlikimo: generuojamas užduoties programą vykdantis procesas - virtuali mašina. Taigi čia užduotis tampa *procesu*. Tai aukščiausias vartotojiškos užduoties pasiekiamas loginis lygis.

Sistemai atlikus vartotojo užduoties procesą, šis naikinamas, ir užduotį dabar identifikuoja jos vykdymo rezultatai - *listingas*: programos, duomenų rezultatų bei su vykdymu susijusių sistemos pranešimų vartotojui tekstai. Tokiame pavidale užduotis patenka į išvedimo srautą ir tampa jo elementu.

T - užduoties tekstas
 P - programa
 D - duomenys
 R - rezultatai
 L - listingas



12 pieš.

Toks yra loginis užduoties keliai. Pagal šį projektą fizinis užduoties keliai yra tokis (schema žr. 12 pieš.):

1) perfokortų rinkinys (programa ir duomenys su valdančiaja informacija);

2) supervizorinė atmintis (tas pat kaip (1));

3) išorinė atmintis (tas pat kaip (1));

4) vartotojo atmintis (programa po pakrovimo);

5) vartotojo atmintis (duomenys po virtualios įvedimo operacijos);

6) išorinė atmintis (rezultatai);

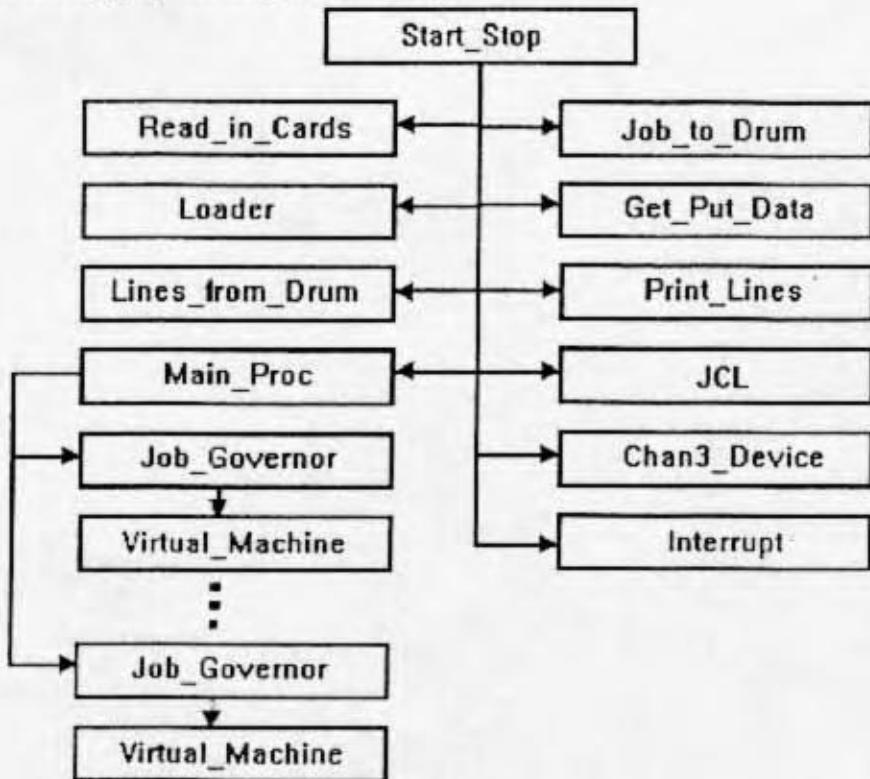
7) supervizorinė atmintis (listingas);

8) listingas popieriuje (užduoties tekstas, rezultatai ir sisteminiai pranešimai).

3.4.6. Procesų paketas

Keliui, kurį turi praeiti užduotis, palaikyti reikalingi atitinkami sisteminiai procesai: perfokortų skaitymo į supervizorinę atmintį procesas *Read_in_Cards*, užduoties siuntimo į išorinį kaupiklį procesas *Job_to_Drum*, programos pakrovėjas į vartotojo atmintį *Loader*, virtualaus duomenų įvedimo -išvedimo aptarnavimo procesas *Get_Put_Data*, listingo perkėlimo iš magnetinio būgno į supervizorinę atmintį procesas *Lines_from_Drum* ir listingo išvedimo į spausdintuvą procesas *Print_Lines*. Be to reikalingas užduoties valdymo procesas *Job_Governor*, tvarkantis virtualios mašinos darbą, bei jį reikiamu momentu

(atsiradus užduočiai) generuojantis ir likviduojantis (užduotį įvykdžius) procesas *Main_Proc*. Taip pat reikalingi trečiojo kanalo valdymo procesas *Chan3_Device*, užduočių valdymo kalbos, kuria pateikiama užduotis įvedimo sraute, interpretatorius *JCL*, ne dėl įvedimo - išvedimo atsirandančių pertraukimų sukeltų situacijų apdorotojas *Interrupt*.



13. pieš.

Generuojant sistemą, svarbūs anksčiau minėtų procesų "tėvystės" ryšiai, t. y. reikia žinoti schemą, pagal kurią sistemoje procesai kuriami, ją generuojant, ir pagal kurią jie turėtų būti naikinami, ją degeneruojant (korektiškai išjungiant). Sudėtingiausia šioje vietoje yra parinkti vadinamąjį šakninį procesą (t. y. pradinį, kuris generuoja mas automatiškai), kuris savo ruožtu būtų procesų hierarchijos pradžia. Šis procesas turi, reikalui esant, reaguoti į sistemos degeneravimą inicijuojančią situaciją: komandą, pranešimą ar sistemos būseną. Galima būtų parinkti kažkurį iš jau turinčių kitą tiesloginę paskirtį procesų, kurie nėra labai apkrauti savo pagrindinėmis funkcijomis bei nenaudoja daugelio rūšių resursų, unifikuojant juos blokuojančio - deblokuojančio resurso tipą (pavyzdžiu, pareikalaujant, kad procesas "lauktų" tam tikros rūšies pranešimo resurso, kuris viduje proceso galėtų būti interpretuojamas keleropai, t. y. resurso rūši sudarytų keli porūšiai, į kuriuos procesas skirtingai reaguotų). Tačiau tai būtų gana dirbtinė schema, nepateisinamai komplikuojanti procesą, kuriam jau numatyta aiški sisteminė paskirtis.

Paprasčiausias ir logiškiausias sprendimas šiuo atveju būtų įvesti specializuota generacijos - degeneracijos procesą *Start_Stop*, kuris būtų

kuriamas automatiškai, įjungiant skaičiavimo sistemą, ir kurio pagrindinė ir vienintelė užduotis būtų sekti, ar sistemoje nesusiklostė situacija, reikalaujanti baigti darbą, ir atitinkamai į tai reaguoti.

Kadangi visi kiti sisteminiai procesai, išskyrus *Job_Governor*, nagrinėjamame projekte yra iš esmės lyglaverčiai, būtų ne visai logiška kurti kokią nors sudėtingą jų "tėvystės" hierarchiją. Natūralu pareikalauti, kad visi jie būtų kuriami startinio proceso laisvai pasirinkta tvarka. Prieš generuodamas procesus, startinis procesas dar turėtų sukurti statinius resursus (t. y. sukurti jų valdymo blokus) vidinę bei išorinę atmintis, kanalus (ivedimo-išvedimo srautus), buferius, o sistemai baigiant darbą - juos korektiškai sunaikinti.

Vartotojišką procesą *Virtual_Machine* - virtualią mašiną - turėtų kurti ja valdantis procesas *Job_Governor*.

Bendra "tėvystės" ryšių schema pateikiama 13 paveiksle.

Sekančiuose skyreliuose bus detaliau aptariamos šių procesų funkcijos ir darbo algoritmai.

3.4.7. "Šakninis" procesas *Start_Stop*

Šis procesas yra atsakingas už sistemos darbo korektišką pradžią ir pabaigą. Jis sukuriamas ir naikinamas automatiškai, įjungiant skaičiavimo sistemą ir ją išjungiant. Jo užduotis - sukurti statinius MOS darbo eigoje sisteminius procesus bei resursus.

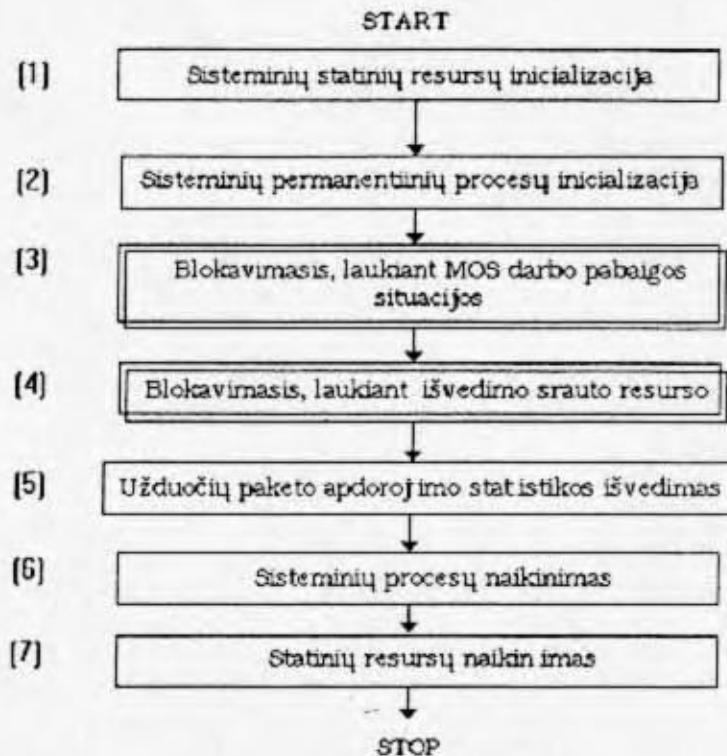
Proceso darbo schema pateikiama 14 paveiksle.

Start_Stop, gavęs procesorių (einamuoju šis procesas tampa iškart po atsiradimo), pirmiausia sukuria statinius sisteminius resursus (1) vidinę bei išorinę atmintį, įvedimo ir išvedimo srautus, 3 kanalo resursą, buferius. Laikoma, jog procesoriaus resursas kuriamas ir naikinamas automatiškai, įjungiant ir išjungiant skaičiavimo sistemą (panašiai, kaip ir procesas *Start_Stop*). Po to generuojami pagrindiniai sisteminiai procesai (2) užduočių vykdymo aptarnavimo (*Read_in_Cards*, *Job_to_Drum*, *JCL*, *Loader*, *Lines_from_Drum*, *Main_Proc*, *Print_Lines*), virtualaus įvedimo - išvedimo aptarnavimo (*Get_Put_Data*), trečiojo kanalo aptarnavimo (*Chan3_Device*), virtualios mašinos ne su įvedimu-išvedimu susijusių pertraukimų pavertimo pranešimais valdančiajam procesui (*Interrupt*).

Po to šakninis procesas blokuojasi, laukdamas situacijos, kai reikės korektiškai užbaigti sistemos darbą (3). Tai gali būti koks nors pranešimas, atsirandantis įvykus išoriniam įvykiui, situacija, kai visas vartotojo užduočių paketas jau apdorotas ir t. t. Tačiau tai jau yra realizacijos metu sprendžiamas uždavinys.

Blokavimo būsena šiam procesui labai tinka: iš vienos pusės jis yra aktyvus (stebi, ar nereikalaujama darbo nutraukimo), iš kitos pusės - nereikalauja jokių kitų resursų, išskaitant ir procesorių, taigi - netrukdo efektyviams MOS darbui.

Po deblokavimo procesas blokuojasi, reikalaudamas išvedimo srauto resurso (4), gavęs ji, išveda paketo apdorojimo statistiką (5) ir atlieka tvarkingą MOS degeneraciją, sunaikindamas savo sukurtus procesus (6) bei resursus (7).



14. pieš.

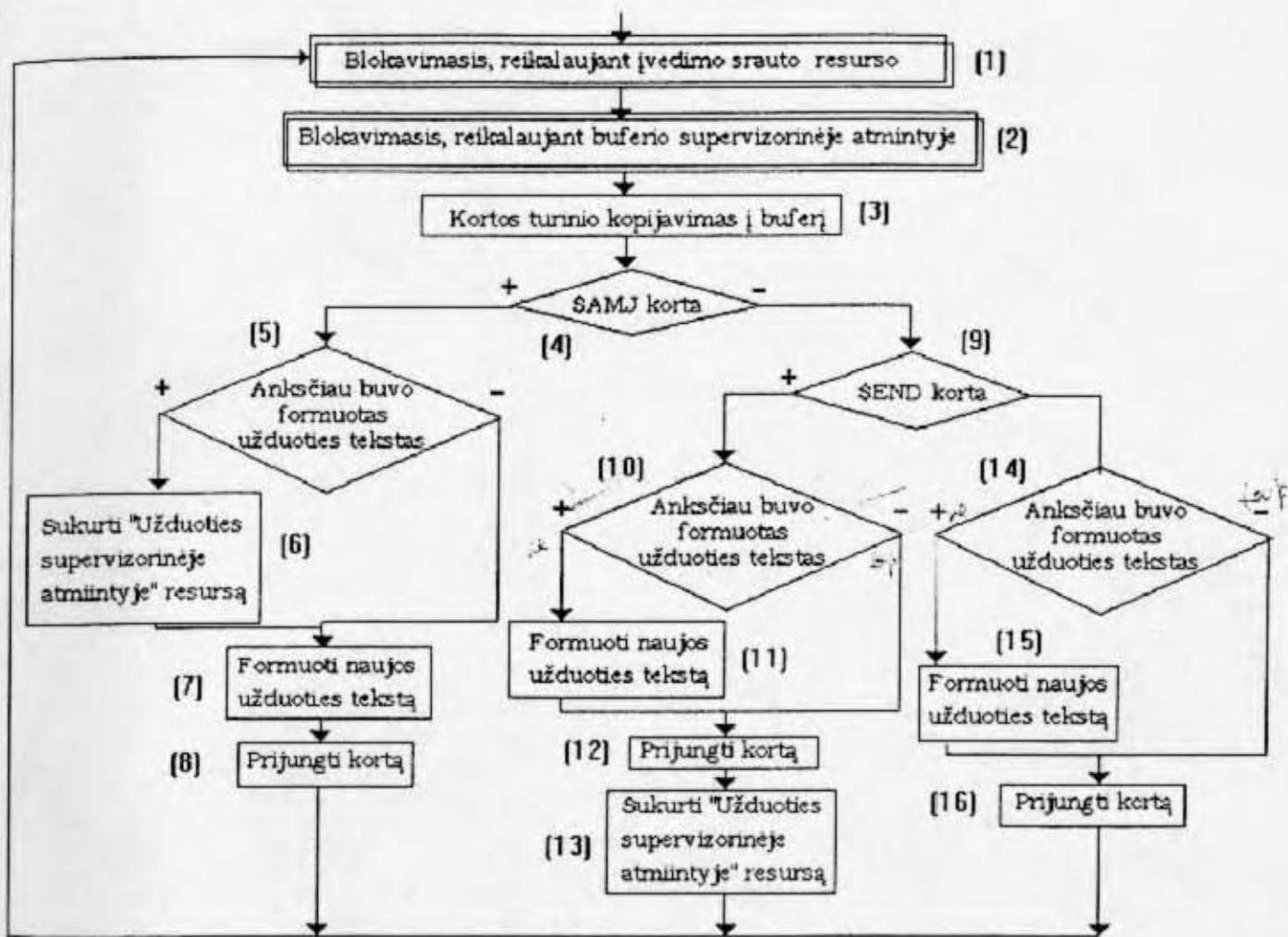
3.4.8. Įvedimo srauto turinio įvedimo į supervizorinę atmintį procesas *Read_in_Cards*

Ši proceso sukuria ir naikina "šakninis" procesas *Start_Stop*. Jis gyvuoja nuo sistemos generacijos iki jos degeneracijos. Pagrindinės šio proceso funkcijos yra dvi: aptarnauti įvedimo srautą, kurio atitinkmuo aparatūroje yra įvedimo procesorius - pirmasis kanalas, ir preliminariai skaidyti srautą į vartotojo užduočių tekstus. Proceso darbo schema pateikiama 15 paveiksle.

Read_in_Cards, gavęs procesorių, pareikalauja įvedimo srauto, t. y. pirmojo kanalo, resurso (1). Nors iš pirmo žvilgsnio čia blokavimas néra visai logiškas - dėl 1-ojo kanalo daugiau niekas nesivaržo, tačiau iš tikrujų jis yra būtinis: nenumatytais išorinis įvykis gali nutraukti įvedimo srautą (pavyzdžiu, gali sugesti arba būti išjungta perfokortų įvedimo aparatūra) arba gali tiesiog pasibaigti vartotojo užduočių paketą sudarančios kortos.

Užsistikrinč srauto egzistavimą, procesas pareikalauja supervizoriaus atminties bloko, į kurį būtų galima nukopijuoti įvedimo sraute esančios kortos turinį (2).

Jei gauti abu reikalingi resursai, procesas kopijuojant perfokortos turinį į supervizoriaus atmintį, t. y. inicijuoja aparatūrinio pirmojo kanalo darbą su atitinkamais parametrais.



15 pieš.

Pastaba. Nors pirmasis kanalas sugeba apsikeisti keletu blokų informacijos, MOS parankiau tai daryti tik po vieną bloką - perfokortą, nes taip lengviau kontroliuoti užduoties teksto įvedimą į operatyviajų atmintį.

Po to procesas pradeda vykdyti savo antrają funkciją - grupuoja kortas pagal preliminarią priklausomybę užduoties tekstu. Grupavimas vyksta pagal SAMJ ir SEND kortas. Tačiau šiame užduoties apdorojimo etape iš esmės dar nėra jokios gilesnės interpretacijos ir klaidų teškojimo - tai tiesiog duomenų operacinei sistemai įvedimo procesas.

Įvedės eilinę kortą, procesas tikrina, kokia ji yra: SAMJ korta, SEND korta ar bet kuri kita korta (identifikuojama pagal pirmąjį bloko žodį).

Jei korta yra SAMJ (3), tai pasitikrinama, ar jau buvo formuojamas užduoties tekstas (kortas atitinkančių supervisorinių atminties blokų sąrašas) (4). Jei taip, tai laikoma, kad ankstesnioji užduotis baigėsi ir sukuriamas "Užduoties supervisorinėje atmintyje" resursas, nurodant blokų sąrašą (5). Po to inicijuojamas naują užduotį sudarantis blokų sąrašas (6) ir prie jo prijungiamas apdorojamoji korta (7) (pastarieji du veiksmai atliekami betarpiskai, jei anksčiau užduoties tekstas neformuotas arba suformuotas iki galio).

Jei korta yra SEND (8), tai vėl pasitikrinama, ar anksčiau neformuotas užduoties tekstas (9). Jei taip, tai korta prijungiamai prie jo (11) ir sukuriamas "Užduoties supervizorinėje atmintyje" resursas (12). Jei ne, tai prieš pastaruosius du veiksmus dar inicijuojamas naujos užduoties teksto formavimas (10).

Jei korta nėra ribojanti vieną užduotį nuo kitos, tai atliekamas analogiškas patikrinimas dėl formuojamo teksto egzistavimo (13). Jei tekstas jau formuojamas, tai korta tiesiog prijungiamai prie jo (15), o jei ne, tai prieš tai dar inicijuojamas naujas tekstas (14).

Visų šiu šakų pabaigoje cikliškai grįztama į pradinę padėtį.

Laikoma, kad ciklo žingsnio metu procesas visiškai išnaudoja gautą atminties resursą, o pirmojo kanalo resursą atlaisvina.

3.4.9. Užduočių valdymo kalbos interpretavimo procesas JCL

Ši procesą sukuria šakninis procesas *Start_Stop*. *JCL* pskirtis - interpretuoti užduoties tekštą supervizorinėje atmintyje, išskirti programos tekštą, duomenis ir valdančiuosius parametrus, užfiksuoti klaidas užduoties formulavimo lygyje.

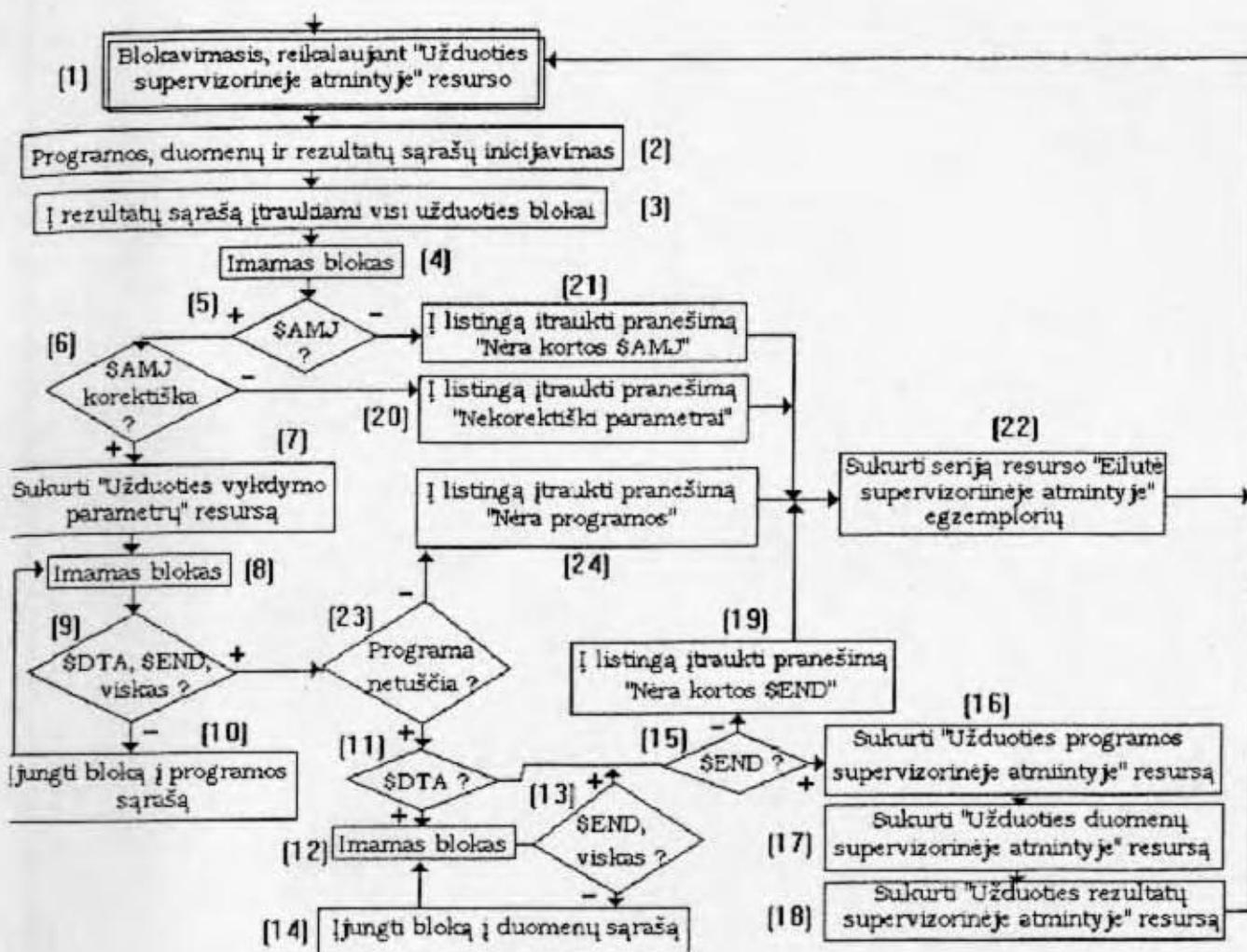
Proceso darbo schema pateikta 16 piešinyje.

JCL blokuojamas, jam paprašius "Užduoties supervizorinėje atmintyje" resurso, kurį sukuria *Read_in_Cards* (1). Gavęs šį resursą, procesas inicijuoja tris blokų (kortų), sąrašus: programos, duomenų ir listingo (rezultatų) (2). Į listingo sąrašą iš karto įtraukiamas visas gautų blokų sąrašas (3), nes reikalaujama, kad būtų išvedamas ir išeities tekstas.

Po to pradedama interpretacija.

Imamas blokas (4). Jei jis nėra SAMJ korta (5), sukuriamas pranešimas "Nėra SAMJ kortos" ir įtraukiamas į listingo sąrašą (21). Jei yra pradinė korta, tikrinama, ar nepažeisti reikalavimai jos formatui (5) (jei nurodytas nulinis vykdymo laikas, jis automatiškai padaromas lygus 1 - užduotį reikės vykdyti). Jei pažeidimų yra, fiksuojama kлаida "Nekorektiški SAMJ parametrai" ir atitinkamas pranešimas vartotojui įtraukiamas į listingo sąrašą (20). Jei ne - sukuriamas "Užduoties vykdymo parametru" resursas su nuoroda į SAMJ bloką (7) ir imamas sekantis blokas (8). Tolesni blokai jungiami į programos sąrašą (10), kol nesutinkama korta SDTA, SEND arba nesibaigia užduoties kortų paketas (9). Jei iš karto po kortos SAMJ kortos eina SDTA, SEND arba baigiasi kortų paketas - fiksuojama kлаida "Nėra programos" ir atitinkamas pranešimas prijungiamas prie listingo sąrašo (24). Tuo tikslu pasitikrinama, ar programos sąrašas nėra tuščias (23).

Jei sutikta korta SDTA (11), imami sekantys blokai (12) ir talpinami į sąrašą "duomenys" (14), kol nesutinkama korta SEND arba nesibaigia kortų paketas (13).



16 pieš.

Jei sutikta korta SEND (15), sukuriamas "Programos supervizorinėje atmintyje" resursas su nuoroda į programos blokų sąrašą, "Duomenų supervizorinėje atmintyje" resursas su nuoroda į duomenų blokų sąrašą bei "Rezultatų supervizorinėje atmintyje" resursas su nuoroda į listingo sąrašą ((16) - (18)). Po to cikliškai grįztama į pradinę būseną.

Jei paketas baigėsi, nesutikus SEND, fiksuojama klaida "Nėra kortos SEND" ir atitinkamas pranešimas įtraukiamas į listingo sąrašą (19).

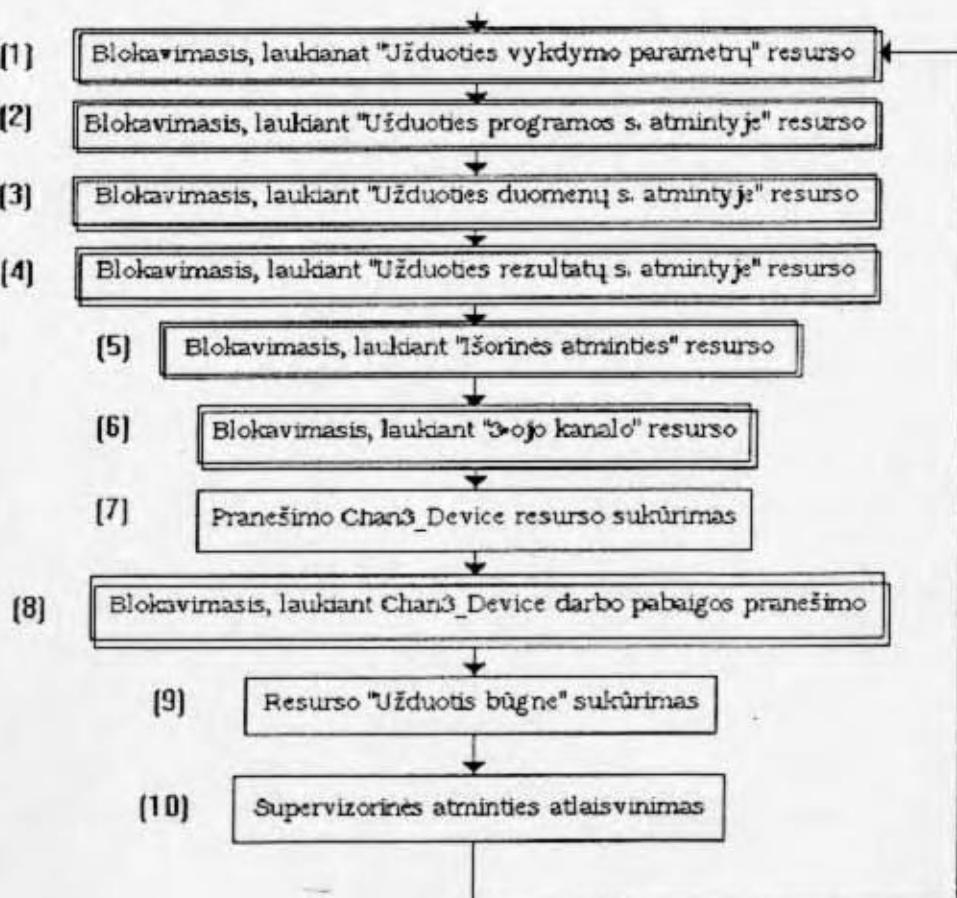
Visos klaidingos situacijos atveda į užduoties listingo supervizorinėje atmintyje sukūrimą - sukuriama eilė resurso "Eilutė supervizorinėje atmintyje", kurio laukia procesas *Print_Lines*, egzempliorių (22). Po to cikliškai grįztama į pradinę padėti.

Taigi galimi du ciklo žingsnio rezultatų atvejai - arba užduotį aprašančiu resursų paketas (natūralu į šiuos resursus įtraukti nuorodas į užduoties identifikatorių, kad jie toliau nebūtų painiojami su analogiškais resursais, gautais iš kitų užduočių tekstu) arba listingo resursas, kuris laukia spausdinimo procesas *Print_Lines*. Pastaruoju atveju užduoties vykdymas tuo ir baigiasi.

3.4.10. Užduoties komponentų siuntimo į būgną procesas *Job_to_Drum*

Ši procesą generuoja pradinis procesas *Start_Stop*. *Job_to_Drum* paskirtis - persiųsti užduoties programos, duomenų bei listingo tekstuus į būgną ir informuoti MOS apie užduotyje esančios informacijos parengimą pagrindiniam apdorojimui - sukurti užduoties kaip tokios resursą.

Proceso darbo schema pateikta 17 piešinyje.



17 pieš

Job_to_Drum pirmiausiai blokuojamas. jam reikalaujant užduotį aprašančio "Užduoties parametru" resurso (1). Iš jo paimami užduoties identifikatorius ir vykdymo parametrai. Pagal identifikatorių toliau pareikalaujama tos užduoties programos, duomenų ir rezultatų supervizoriinėje atmintyje resursų ((2) - (4)). Gavus šią informaciją, pagal jos suminę apimtį nustatoma reikalinga išorinės atminties būgne erdvė ir pareikalaujama šio resurso (5). Jei sužinoma, kad yra pakankamai laisvos išorinės atminties, pareikalaujama trečiojo kanalo resurso (6). Pastarasis gali būti užimtas, t. y. su disku gali dirbti kitas procesas: *Loader*, *Get_Put_Data*, *Lines_from_Drum*. Kai

trečiasis kanalas atsilaisvina. Jį valdančiam procesui *Chan3_Device* siunčiamas pranešimas apie tai, kad reikia išvesti užduoties programą, duomenis ir rezultatus į magnetinį būgną. T. y. sukuriamas pranešimo resursas su nuorodomis į siuntėją, blokų sąrašą supervizorinėje atmintyje ir takelių sąrašą išorinėje atmintyje (7). Po to blokuojamasi, laukiant *Chan3_Device* pranešimo apie darbo pabaigą, t. y. atitinkamo resurso su nuoroda į *Job_to_Drum* (8).

Kai užduoties komponentai sekmingai išvesti į išorinę atmintį, sukuriamas "Užduoties būgne" resursas su nuorodomis į atitinkamus takelių sąrašus (9). I šio resurso aprašymą įtraukiama ir užduoties vykdymo parametrai.

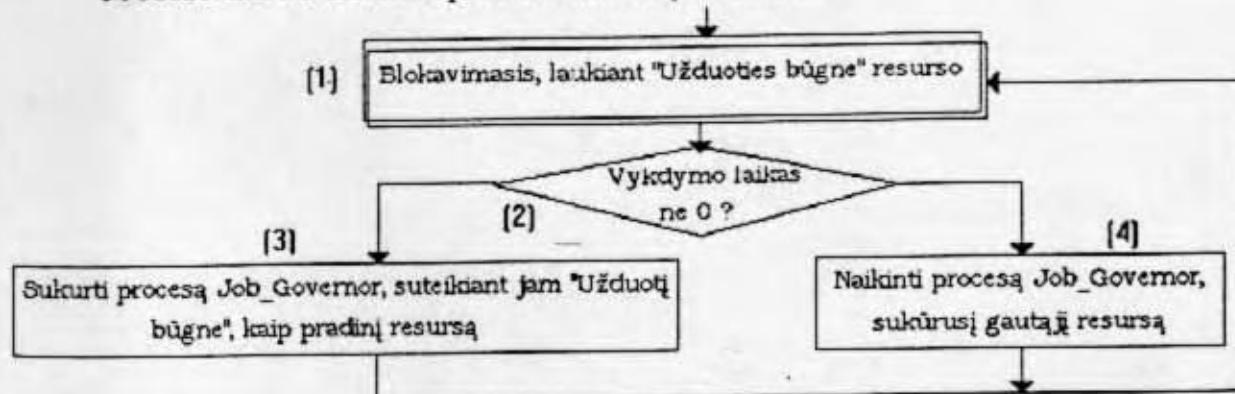
Po to atlaisvinama visa supervizorinė atmintis, kurią užémė *JCL* sukurtas užduoties tekstas: programa, duomenys ir valdančiosios kortos (10). Laikoma, kad resursai "Užduoties programa supervizorinėje atmintyje", "Užduoties duomenys supervizorinėje atmintyje", "Užduoties rezultatai supervizorinėje atmintyje" ir "Užduoties parametrai" yra sunaudojami visiškai.

Atlikus šiuos veiksmus, cikliškai grįztama į pradžią.

3.4.11. Užduoties apdorojimo individualios aplinkos inicializacijos procesas *Main_Proc*

Ši procesą sukuria pradinis procesas *Start_Stop*. *Main_Proc* paskirtis - atsiradus sistemoje pagrindiniams apdorojimui parengtai užduočiai, inicijuoti Jos vykdymo aplinką, t. y. sukurti valdantįį procesą *Job_Governor*. Taip pat jis turi sekti, kad atidirbus užduotims, neliktų parazituojančių jų aplinką, t. y. reikiamu momento turi naikinti valdančiuosius procesus.

Proceso darbo schema pateikiama 18 paveiksle.



18 pieš.

Main_Proc ciklo žingsnio pradžioje blokuojamas, jam laukiant "Užduoties būgne" resurso (1), kurį sukuria *Job_to_Drum*. Jei toks resursas sistemoje yra, pasitikrinama, ar jis néra fiktyvus, t. y. ar vykdymo laikas néra nulinis (2).

Pastaba. Fiktyvaus resurso galimybė čia įvedama *Main_Proc* reikalingu resursų unifikavimui.

Jei laikas nenulinis, tai atsiradusi užduotis reikalauja pagrindinio apdorojimo. Tuo tikslu sukuriamas apdorojimo aplinką užtikrinantis užduoties

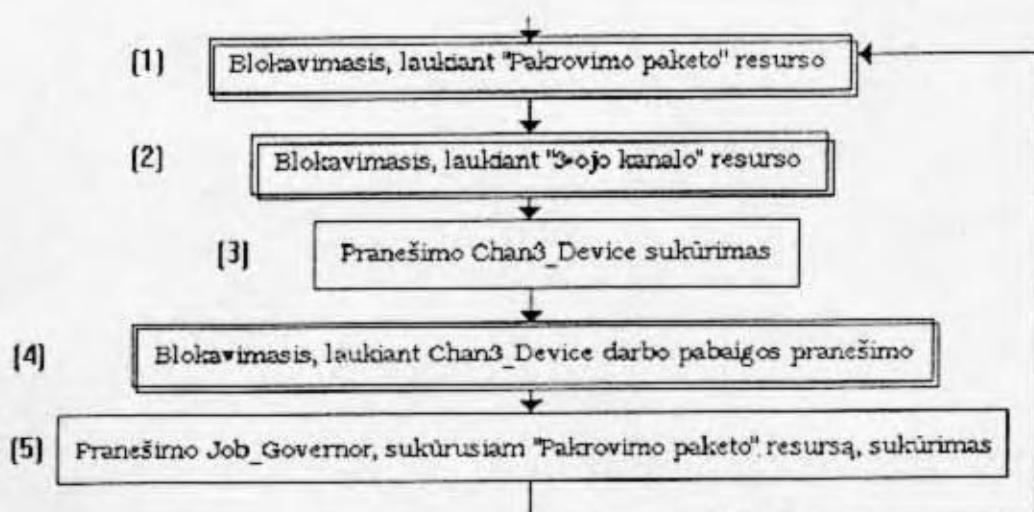
individualus valdymo procesas *Job_Governor* (3), jam suteikiant gautąjį "Užduoties būgne" resursą, kaip pradinį resursą. Po to ciklas užsidaro.

Jei "Užduoties būgne" resursas yra fiktyvus, sukurtas ne *Job_to_Drum*, o *Job_Governor* (žr. 3.4.14 skyр.), tai šis faktas interpretuojamas kaip pranešimas, kuris reiškia, jog reikia sunaikinti jo siuntėją - valdantįį procesą, kurio užduotis yra jau atidirbusi (4). Po to cikliškai grįztama į pradžią.

3.4.12. Užduoties programos pakrovimo iš išorinės atminties į vartotojo atmintį procesas *Loader*

Ši procesą sukuria šakninis procesas *Start_Stop*. *Loader* paskirtis - kopijuoti vartotojo užduoties programos tekstą iš magnetinio būgno į vartotojo atmintį.

Proceso darbo schema pateikta 19 paveiksle.



19_pieš.

Loader ciklo žingsnio pradžioje pareikalauja "Pakrovimo paketo" resurso (1), kurį sudaro nuorodos į takelių išorinėje atmintyje ir vartotojo atminties blokų sąrašus. Gavus šiuos resursus, pareikalaujama trečiojo kanalo resurso (2) ir ji gavus, kuriamas pranešimas procesui *Chan3_Device* su nuorodomis į takelių bei blokų sąrašus ir atliktiną operaciją (ivedimas ar išvedimas) (3). Po to *Loader* blokuojasi laukdamas *Chan3_Device* darbo pabaigos pranešimo (4). Kai šis pranešimas gaunamas, pakrovimo procesas kuria pranešimą apie savo darbo pabaigą procesui *Job_Governor*, sukūrusiam "Pakrovimo paketo" resursą ir grįžta į ciklo pradžią.

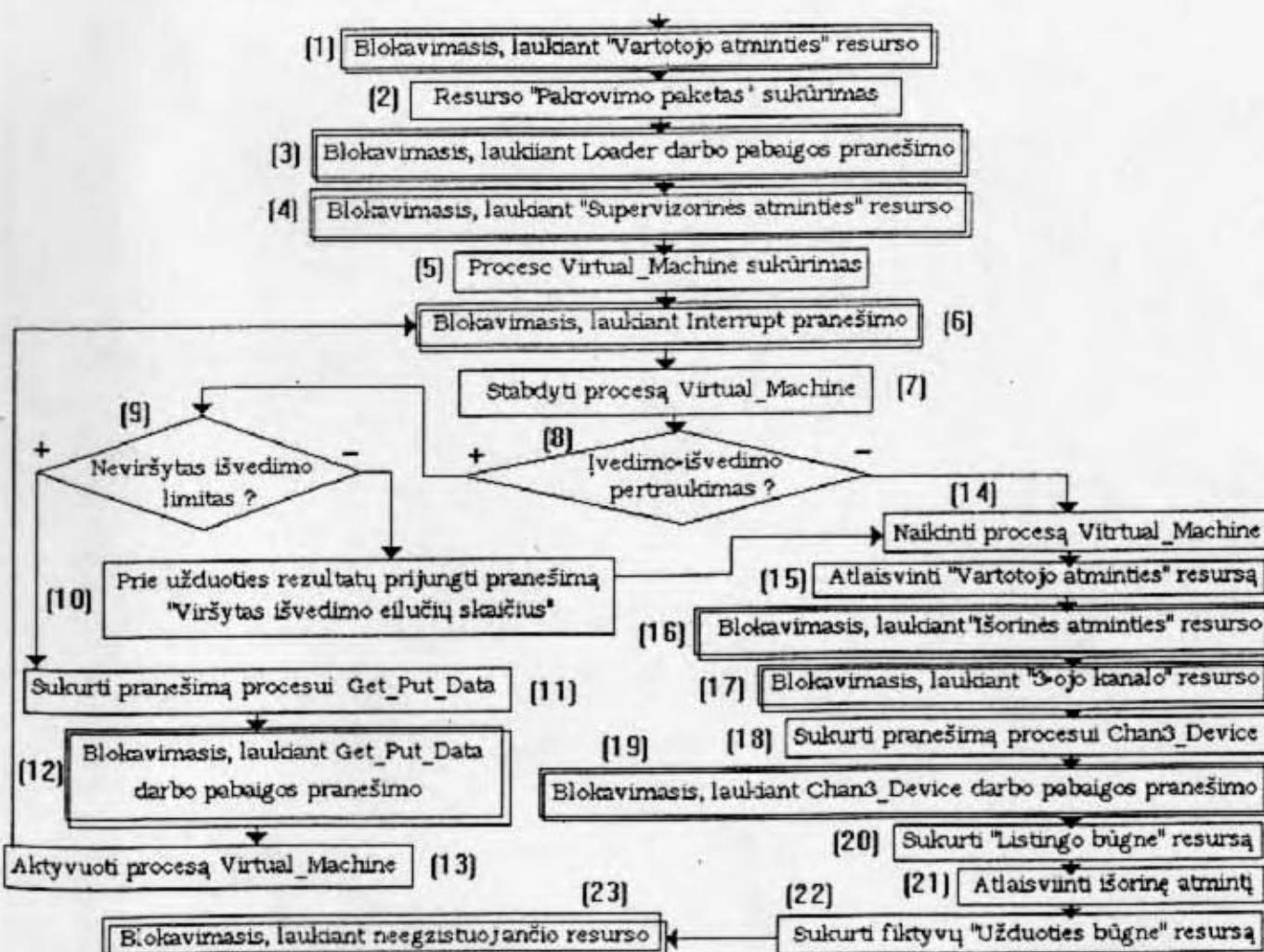
Laikoma, kad ciklo žingsnio metu visiškai sunaudojamas "Pakrovimo paketas", o trečiasis kanalas po įvedimo atlaisvinamas.

3.4.13. Užduoties valdymo procesas Job_Governor

Ši procesą kuria užduoties pagrindinio apdorojimo aplinkos inicializavimo procesas *Main_Proc*. *Job_Governor* - laikinas procesas MOS bendro darbo atžvilgiu. Jis kuriamas individualiai kiekvienai užduočiai. Šis individualumo ryšys nustatomas pagal "tėvystės" sąryšį tarp *Job_Governor* ir jo sukurtos virtualios mašinos (žr. žemiau).

Job_Governor paskirtis - sukurti vartotojišką procesą - virtualią mašiną - konkrečios užduoties programos atlikimui. Taip pat jis turi sekti virtualios mašinos darbą, laiku duoti nurodymus procesui *Get_Put_Data* dėl virtualios mašinos įvedimo ir išvedimo operacijų aptarnavimo, nutraukti virtualios mašinos darbą ir ją naikinti. Įvykus kladai vartotojo programos interpretavimo metu ar viršijus leistą išvedimo eilučių kiekį, rinkti sistemos pranešimus apie programos vykdymą, kurti vartotojo programos listingą bei papildyti sistemos kaupiamą statistiką apie vartotojo užduočių vykdymą.

Proceso darbo schema pateikiama 20 piešinyje.



Resursą "Užduotis būgne" šis procesas jau yra gavęs kaip pradinį. Pagal programos dydį pareikalaujamas atitinkamas kiekis "Vartotojo atminties" resurso programos pakrovimui (1) ir kuriamas "pakrovimo paketas" procesui *Loader* (2), kuriame pateikiamos nucrodos į vartotojo atminties blokų, kur reikės talpinti vartotojo užduoties programą, ir magnetinio būgno takelių, kur ji šiuo metu yra, sąrašus. Perdavus tokį paketą, blokuojamas, laukiant *Loader* darbo pabaigos pranešimo (3). Programą pakrovus į vartotojo atmintį, pareikalaujama supervizorinės atminties bloko virtualios mašinos puslapių lantelėi (4). Gavus ir šį resursą, sukuriams vartotojiškas procesas *Virtual_Machine* - virtuali mašina (5).

Visi aukščiau išvardinti žingsniai yra iš esmės virtualios mašinos aplinkos inicializavimas. Turėdamas užduoties resursą, *Job_Governor* žino, kur yra išorinėje bei vartotojo atmintyje programa, kur išorinėje yra duomenys jai ir kur turės būti kaupiami jos rezultatai (virtualūs įvedimo ir išvedimo įrenginiai), ir gali jais operuoti.

Kuriant virtualią mašiną, nustatoma jos pradinė būsena (registru ir laiko skaitliuko) bei užpildoma puslapių lentelė. Visi reikalingi parametrai įtraukiami į vartotojiško proceso valdymo bloką (deskriptorių). Po to virtuali mašina pastatomą į pasiruošusiu procesu eilę.

Virtualios mašinos procesas dėl savo paprastumo turi būti valdomas "griežčiau" nei kiti sistemoje vykstantys procesai. Tėvystės ryšys tarp *Job_Governor* ir *Virtual_Machine* čia įgyja didesnę reikšmę, nei analogiški ryšiai tarp kitų procesų. *Job_Governor* paliekama išmtinė teisė sustabdyti (blokuoti) virtualią mašiną ir ją vėl paleisti (deblokuoti). Toks priverstinis blokavimas būtinas todėl, kad vartotojo procesas nereikalauja jokio kito resurso išskyrus procesoriaus, ir tuo būdu pats blokuotis negali.

Job_Governor ciklo pradžioje laukia pranešimo iš reakcijos į pertraukimus proceso *Interrupt* (6). Atsiradus tokiam pranešimui, kuriame nurodomas pertraukimo tipas, valdantysis procesas sustabdo savo valdomą virtualią mašiną (7). Po to patikrinama, ar pertraukimas buvo sukeltas virtualios mašinos komandų GD arba PD (8). Jei taip, ir reikalaudama išvedimo, tai papildomai patikrinama, ar neviršytas maksimalus šiai užduočiai leistų išvesti eilučių skaičius (9). Jei jis viršytas, į prie užduoties rezultatų prijungiamas atitinkamas pranešimas vartotojui (10). Jei viskas tvarkoje, suformuojamas atitinkamas pranešimas procesui *Get_Put_Data* (11), nurodant apsikeitimo duomenimis tipą (įvedimas ar išvedimas), vartotojo atminties bloko, kur turi būti įvesti duomenys arba iš kur turi būti išvesti rezultatai, ir magnetinio būgno takelio, iš kur turi būti paimti duomenys, numerius. Po to valdantysis procesas blokuojasi, laukdamas pranešimo apie proceso *Get_Put_Data* darbo pabaigą (12). Pastarajam procesui atlikus savo darbą, vėl paleidžiamas (deblokuojamas) vartotojiškas procesas (13) ir cikliškai pereinama prie pranešimo iš *Interrupt* laukimo.

Jei pertraukimas buvo ne įvedimo - išvedimo aptarnavimo reikalavimo, o vartotojo užduoties programos pabaiga arba avarinis vykdymo nutraukimas arba įvyko jau minėtas išvedimo limito viršijimas, vartotojiškas procesas naikinamas (14), atlaisvinama užimtoji vartotojo atmintis (15). Prie užduoties rezultatų prijungiamas sisteminis pranešimas vartotojui, t. y. jo tekstas

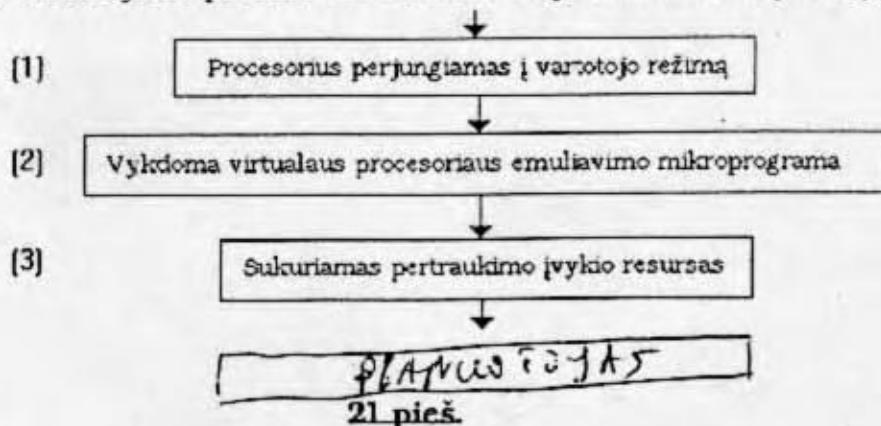
nukopijuojamas į būgną; pareikalaujama vieno takelio išorinėje atmintyje (16) ir trečiojo kanalo resurso (17), suformuojamas pranešimas su reikiamais parametrais procesui *Chan3_Device* (18) ir blokuojamas, laukiant jo darbo pabaigos pranešimo (19). Tada apie užbaigtą užduoties listingą pranešama MOS, suformuojant "Listingo būgne" resursą (20). Po to atlaisvinama išorinė atmintis (21), kurią užémė užduotis būgne, sukuriamas fiktyvus resursas "Užduotis būgne" su nuliniu vykdymo laiko parametru ir tuščiomis nuorodomis į užduoties komponentus (22). Pastarasis resursas iš esmės yra pranešimas apie savo darbo pabaigą procesui *Main_Proc*. Jo pasėkoje pastarasis turėtų naikinti pranešimo autorium - *Job_Governor*. Kad šis procesas prieš sunaikinimą nebenaudotų sisteminių resursų, jis sąmoningai užsiblokuoja, reikalaudamas resurso, kurio sistemoje būti negali (pavyzdžiui, kokio nors neįmanomo pranešimo) (23). Tuo ir užsibaigia ši proceso darbo algoritmo šaka.

3.4.14. Vartotojiškas procesas *Virtual_Machine*

Tai yra procesas, skirtas betarpiskai vykdyti vartotojo užduoties programai. Jis yra sukuriamas individualaus valdymo proceso *Job_Governor* ir egzistuoja tol, kol pastarasis jo nesustabdo ir nesunaikina.

Panašiai, kaip ir valdančiųjų procesų *Job_Governor* atveju *Virtual_Machine* nusako ne vieną procesą, o panašių procesų klasę. Šios klasės procesai turi vieną ir tą pačią programą ir skiriasi tik būseną, kurią nusako kintamujų reikšmės, individualios kiekvienam klasės procesui.

Bendra vartotojiško proceso darbo schema pateikiama 21 piešinyje.



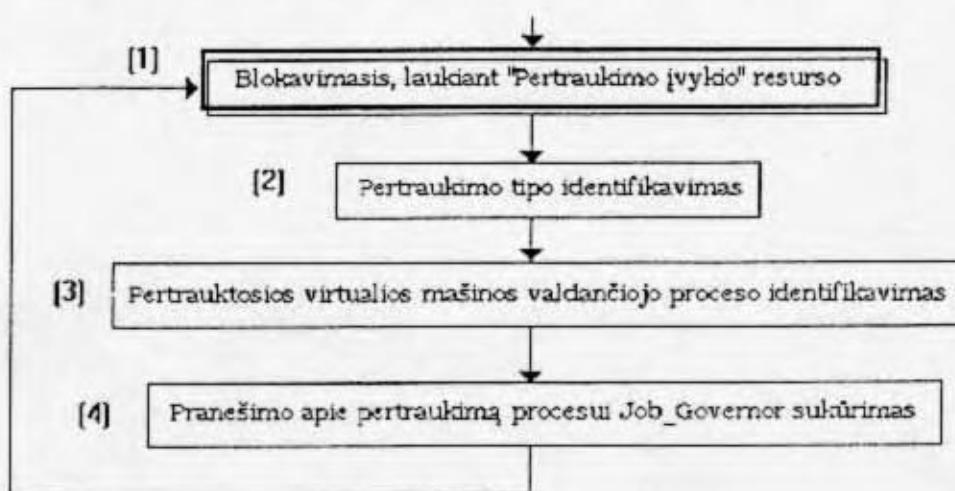
Proceso *Virtual_Machine* darbo pradžioje procesorius perjungiamas į vartotojo režiną (1). Po to valdymas duodamas virtualaus procesoriuas komandų interpretavimo programai (2), kuri cikliškai dirba tol, kol nefiksuojamas pertraukimas. Tada programa baigia darbą, išsaugoma virtualios mašinos būseną ir valdymas atiduodamas pertraukimo apdorojimo programoms. Pastarosios specialiems sisteminiams kintamiesiems suteikia atitinkamas reikšmes. (Paskutiniai du žingsniai vyksta emulatoriaus ir aparatūros lygyje, todėl išreikštiniu būdu į proceso darbo algoritma nepatenka) Po to apie pertraukimo ivykimą informuojama operacinė sistema: sukuriamas pertraukimo

jvykio resursas - pranešimas, kad reakcijos į pertraukimus procesas *Interrupt* turi pradėti darbą (3).

3.4.15. Reakcijos į pertraukimus procesas *Interrupt*

Ši procesą sukuria pradinis procesas *Start_Stop*. *Interrupt* paskirtis - priversti MOS reaguoti į pertraukimus, kilusius virtualios mašinos darbo metu. *Interrupt* privalo sekti, ar neatsirado pertraukimo jvykio resursas, informuojantis, jog reikia kažkaip reaguoti į virtualios mašinos darbo metu jvykusius pakitimus. Šis procesas reikalingas tam, kad *Job_Governor* galetų tinkamai valdyti vartotojišką procesą *Virtual_Machine*.

Proceso darbo algoritmas schematizuojamas 22 piešinyje.



22 pieš.

Kiekvieno savo darbo ciklo pradžioje *Interrupt* laukia resurso "Pertraukimo jvykis" (1), kuris atsiranda, jei vykdant vartotojo programą jvyko pertraukimas. Po to procesas identifikuoja pertraukimo tipą (2): įvedimo reikalavimas, išvedimo reikalavimas, virtualios mašinos stabdymo situacija (normali arba avarinė darbo pabaiga) ar kitas. Tai *Interrupt* nustato pagal sisteminį kintamujų reikšmes, kurias savo ruožtu ten patalpina pertraukimų apdorojimo programos. Toks interfeisas tarp aparatūros ir operacinės sistemos turi du privalumus: pirma, MOS nereikia tiesioginio sąlyčio su aparatūra, o antra, MOS gali "aukščiausiu lygiu" reagucti į žemo lygio jvykius: virtualios mašinos procesoriaus emulatoriaus aptiktas klaidas ir vykdomas komandas.

Interrupt reaguoja tik į situacijas, kurios reikalauja pertraukti virtualios mašinos darbą dėl kokių nors aukštésnio lygio veiksmų, susijusių su vartotojišku procesu, atlikimo būtinumo.

Laikoma, kad virtualios mašino vidinis vardas nuo jos paskelbimo einamaja iki procesoriaus atidavimo kitam vartotojo procesui, saugomas specialiame sisteminame kintamajame. Dėl aukštésnio sisteminį procesų

prioritetu *Interrupt* realiai suspėja pagal ši kintamąjį nustatyti, kuris virtualus procesas buvo pertrauktas ir pagal tai nustatyti, kurį valdantįjį procesą *Job_Governor* apie tai reikia informuoti (3).

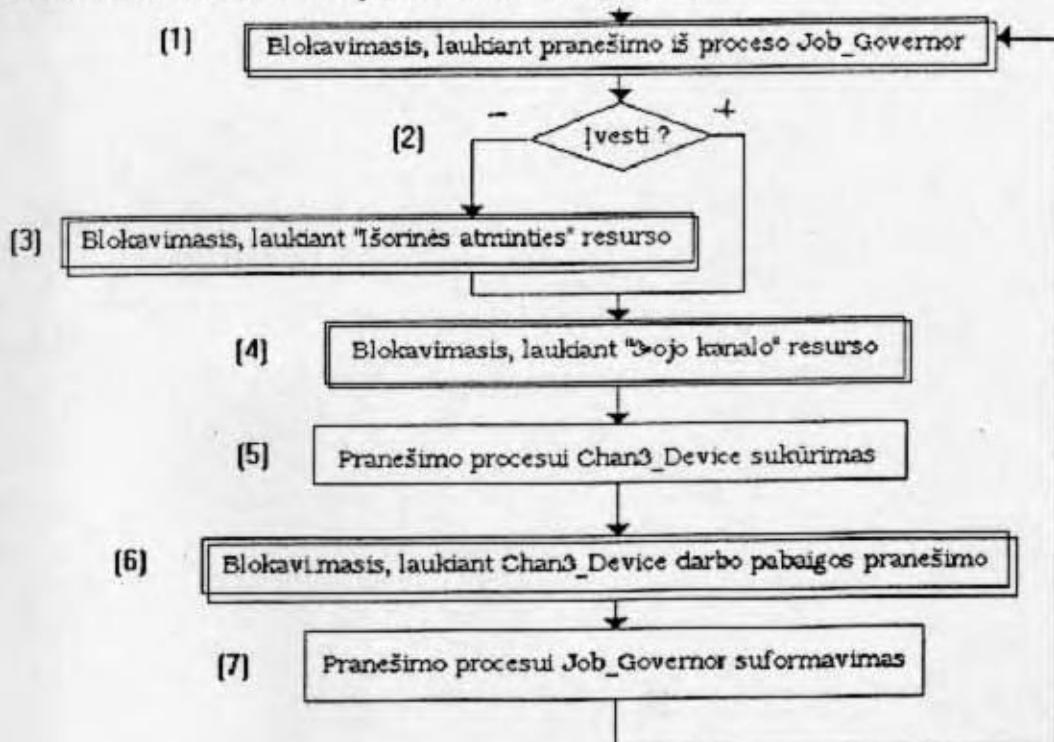
Po to *Interrupt* sukuria pranešimą nustatydam: procesui *Job_Governor* su nuoroda į pertraukimo tipą (4).

Tai atlikęs procesas cikliškai kartoja veiksmus nuo pradžios. Laikoma, jog *Interrupt* visiškai sunaudoja pertraukimo įvykio resursą.

3.4.16. Vartotojo proceso įvedimo ir išvedimo aptarnavimo procesas *Get_Put_Data*

Ši procesą sukuria pradinis procesas *Start_Stop*. *Get_Put_Data* privalo aptarnauti virtualios mašinos komandų GD ir PD vykdymą, t. y. atitinkama kryptimi persiūsti duomenų bloką (10 žodžių) tarp vartotojo atminties ir magnetinio būgno.

Proceso darbo schema pateikiama 23 paveiksle.



23_pieš.

Pirmausia *Get_Put_Data* blokuojasi, laukdamas pranešimo iš bet kurio *Job_Governor* (1). Gavęs jį, nustato operacijos - įvedimo ar išvedimo - tipą (2). Jei reikalingas išvedimas, dar blokuojamas, reikalaujant išorinės atminties vieno takelio dydžio srities. (3). Po to laukiama, kol atsilaisvins trečiasis kanalas, atsakingas už duomenų apsikeitimą su išorine atmintimi (4). Sekančiame etape *Get_Put_Data* suformuoja pranešimą procesui *Chan3_Device*, kuriame nurodo operacijos tipą ir vartotojo atminties bloko bei išorinės atminties takelio

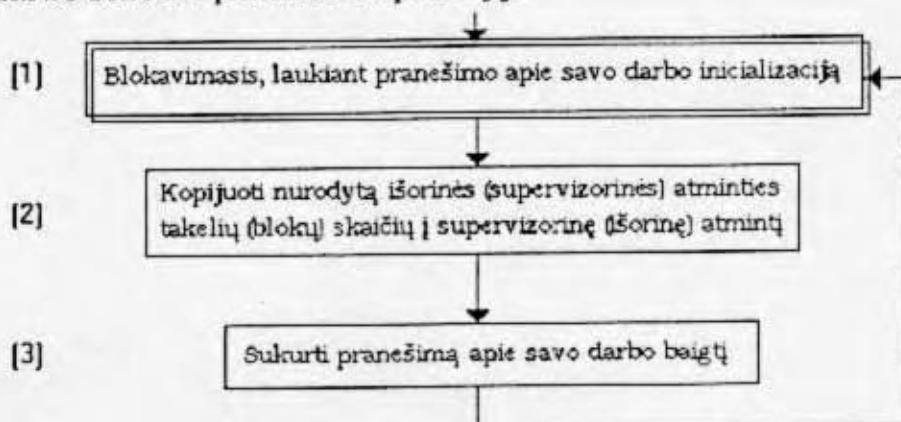
numerius (5) ir užsiblokuoja, laukdamas *Chan3_Device* darbo pabaigos pranešimo (6). Kai baigimas apsikeitimas duomenimis, procesas *Get_Put_Data* suformuoja apie tai pranešimą atitinkamam procesui *Job_Governor*.

Po to proceso darbas kartojas.

3.4.17. Trečiojo aparatūrinio kanalo valdymo procesas *Chan3_Device*

Ši procesą sukuria šakninis procesas *Start_Stop*. *Chan3_Device* paskirtis - "uždengti" aparatūrinį trečiąjį kanalą nuo kitų sisteminių procesų ir atlikti jo valdymą.

Proceso darbo schema pateikta 24 piešinyje.



24 pieš.

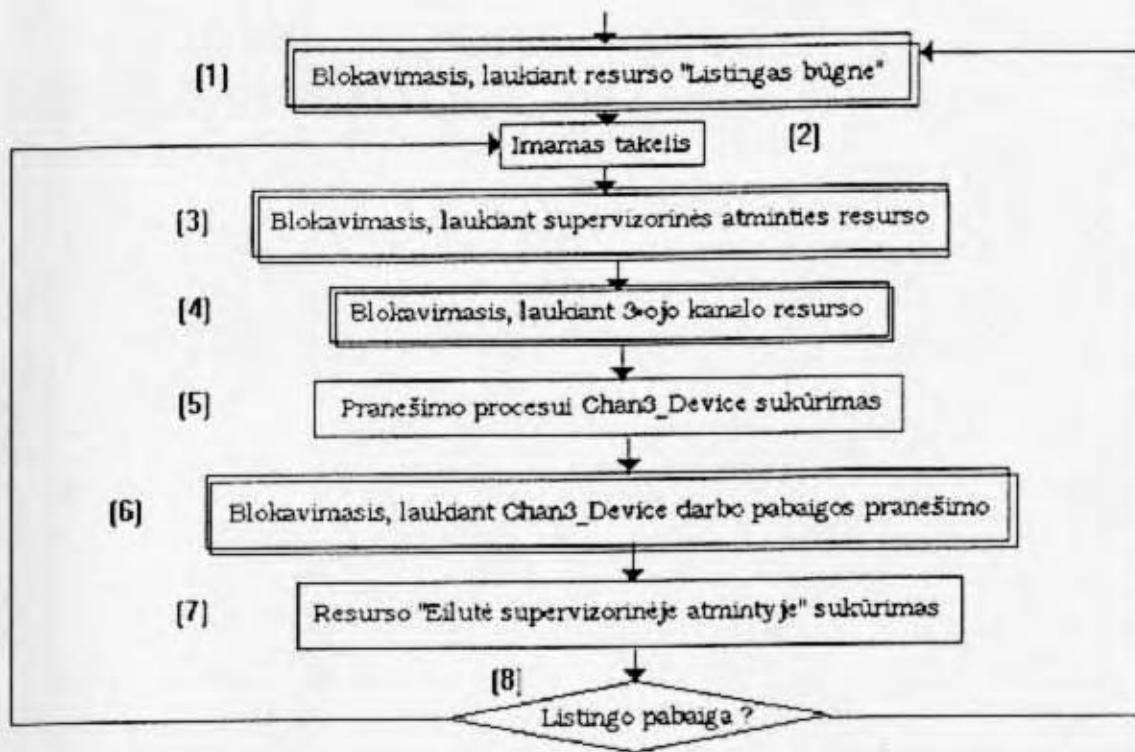
Darbo ciklo pradžioje *Chan3_Device* laukia pranešimo iš bet kurio proceso, kuriam reikia apsikeisti informacija su išorinės atminties įrenginiu (1). Tokiam universalumui palaikyti būtina, kad procesas, siūlyiantis inicializaciją pranešimą, turi pateikti tame ir kokią nors informaciją apie save - siuntėją, kad atlikęs užduotį, *Chan3_Device* galėtų apie tai jį korektiškai informuoti. Tokia informacija yra resurso kūrėjo vidinis vardas.

Gavęs tokį pranešimą, *Chan3_Device* inicijuoja aparatūrinio trečiojo kanalo darbą su atitinkamais parametrais (2), o darbui pasibaigus, apie tai informuoja inicijuojančiąjį pranešimą atsiuntusį procesą, sukurdamas jam pranešimo resursą (3).

3.4.18. Vartotojo užduoties listingo paruošimo išspausdinimui procesas *Lines_from_Drum*

Ši procesą sukuria šakninis procesas *Start_Stop*. *Lines_from_Drum* paskirtis - pradeti vidinį išvedimo srauto formavimą, t. y. paruošti vartotojo užduočių listingus išvedimui į spausdintuvą, perkeliant ju eilutes iš išorinės atminties į supervizorinę atmintį.

Proceso darbo schema pateikiama 25 piešinyje.



25. pieš.

Ciklo pradžioje procesas *Lines_from_Drum* laukia resurso "Listingas būgne" (1), kurį sukuria užduoties valdantysis procesas *Job_Governor*. Po to prasideda listingo eilučių siuntimas į supervisorinę atmintį.

Imamas išorinės atminties takelis su listingo eilutės tekstu (2). Po to laukiama supervisorinės laisvo atminties bloko (3) ir trečiojo kanalo resurso (4). Turėdamas abu šiuos resursus, *Lines_from_Drum* sukuria pranešimą procesui *Chan3_Device* su nuorodomis į atminties vietas ir atliktinos operacijos tipą (5). Po to blokuojamasi, laukiant pranešimo iš *Chan3_Device* (6). Sulaukus pranešimo apie trečiojo kanalo valdymo proceso darbo pabaigą, sukuriamas resursas "Eilutė supervisorinėje atmintyje" (7). Po to atlaisvinamas trečiojo kanalo resursas ir . Jei listingas dar nesibaigė (8), imamas sekantis listingo elementas. Trečiojo kanalo atlaisvinimas reikalingas, kad kiti procesai galėtų juo pasinaudoti tuo atveju, jei *Lines_from_Drum* blokuosis dėl supervisorinės atminties trūkumo.

Jei listingas jau pasibaigė, atlaisvinama jo užimta išorinė atmintis (9) ir visi veiksmai kartojami nuo pradžios.

Laikoma, kad procesas visiškai išeikvoja "Listingo būgne" resursą.

3.4.19. Išvedimo srauto generavimo procesas *Print_Lines*

Ši procesą sukuria pradinis procesas *Start_Stop*. *Print_Lines* paskirtis - siųsti listingo eilutes iš supervisorinės atminties į spausdinimo įrenginį, t. y. generuoti išvedimo srautą.

Proceso darbo schema pateikiamā 26 piešinyje.



26 pieš.

Darbo ciklo pradžioje *Print_Lines* blokuojasi, reikalaudamas "Eilutės supervisorinėje atmintyje" resurso (1). Gavęs šį resursą, *Print_Lines* pareikalauja antrojo kanalo resurso (2). Antrajį kanalą būtina laikyti resursu dėl panašių priežasčių, kaip ir pirmajį kanalą: jis gali būti atjungtas išoriškai. Taigi šiuo atveju spausdinimas neįmanomas, ir reikia užblokuoti procesą *Print_Lines*, kad nebūtų bereikalingo procesoriaus laiko eikvojimo išvedimo aparatūros gedimo atveju.

Jei spausdinimas galimas, *Print_Lines* iniciuoja aparatūrinio antrojo kanalo darbą, nurodydamas išvestinąjį bloką supervisorinėje atmintyje (3). Reikia pastebeti, kad apibrėžtoje realioje mašinoje antrasis kanalas gali operuoti ne tik atskirais blokais, bet ir blokų masyvais. Tai būtų įmanoma ir projektuojamajoje sistemoje, tačiau ši galimybė neišnaudojama, siekiant supaprastinti sisteminių procesų funkcionavimo algoritmus.

Po to atlaisvinamas supervisorinės atminties blokas, kurį užėmė listingo eilutė (4) ir veiksmai kartojami iš pradžių.

Laikoma, kad "Eilutės supervisorinėje atmintyje" yra sunaudojamas visiškai, o antrojo kanalo resursas atlaisvinamas - tam, kad preš sekančios eilutės išvedimą būtų pakartotinai testuojamas išvedimo įrenginys (5).

didesnė įvairovė. Beje pastarasis principas reikalingas, jei siekiama rasti originalių sprendimų.

(FIN :)