Operacinės sistemos modelis

Goda Gutparakytė ir Tomas Baublys

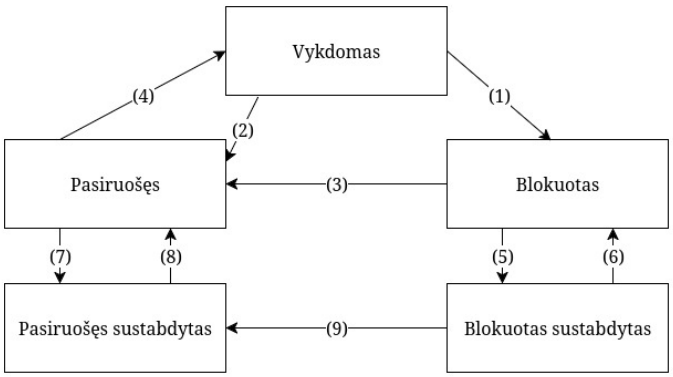
2025 m. spalis

1 Procesai

1.1 Procesu būsenos

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

- **Vykdomas** - turi procesorių
- **Blokuotas** - prašo resurso (išskyrus procesorių)
- **Pasiruošęs** – vienintelis trūkstamas resursas yra procesorius. Tačiau šių būsenų gali neužtekti. Gali susiklostyti tokia situacija, kad tam tikram procesui negalima leisti gauti procesorių, nors jis ir yra pasiruošęs. Toks procesas vadinamas sustabdytu.Galime papildyti būsenų sąrašą:
- **Sustabdytas** – kito proceso sustabdytas procesas



1 pav.: Proceso būklės ir ryšiai tarp jų

Diagramoje matome 9 perėjimus:

- (1) Vykdomas procesas blokuojamas jam prašant ir negavus resurso.
- (2) Vykdomas procesas tampa pasiruošęs, jei dėl kokios nors priežasties (išskyrus resursų negavima) iš jo buvo atimtas procesorius.
- (3) Blokuotas procesas tampa pasiruošęs, jam gavus reikiamus resursus.
- (4) Procesas, gavęs procesorių tampa vykdomas.
- (5) Procesas gali tapti sustabdytu blokuotu, jei jį sustabdo dabartinis procesas, kai jis jau yra blokuotas.
- (6) Procesas blokuoto sustabdyto grįžta į blokuoto būseną, jei einamasis procesas nuima būseną sustabdytas.
- (7) Procesas gali tapti pasiruošusiu sustabdytų, jei einamasis procesas jį sustabdo kai jis yra pasiruošęs.
- (8) Procesas tampa pasiruošęs iš pasiruošusio sustabdyto, jei einamasis procesas nuima būseną sustabdytas.
- (9) Procesas tampa pasiruošęs sustabdytas iš blokuoto sustabdyto, jei jis gauna prašomus resursus.

1.2 Planuotojas

1.2.1 Planuotojo aprašymas

Planuotojo paskirtis yra atimti procesorių iš proceso, peržvelgti pasiruošusių procesų sąrašą, išrinkti pasiruošusį procesą ir perduoti procesorių jam. Multiprogramineje operacinėje sistemoje patys procesai negali perduoti valdymo, nes procesas nežino ir iš anksto negali žinoti, kuris procesas bus pasiruošęs

vykdymui, kuris blokuotas, jo veikimo metu. Jei procesai patys perdavinėtų valdymą, neišvengtume tiesiškumo. Procesų programos yra ciklinės. Procesų išlygiagretinimas ir valdymo perdavimas vyksta per resursų primityvų mechanizmą. Planuotojo tikslai:

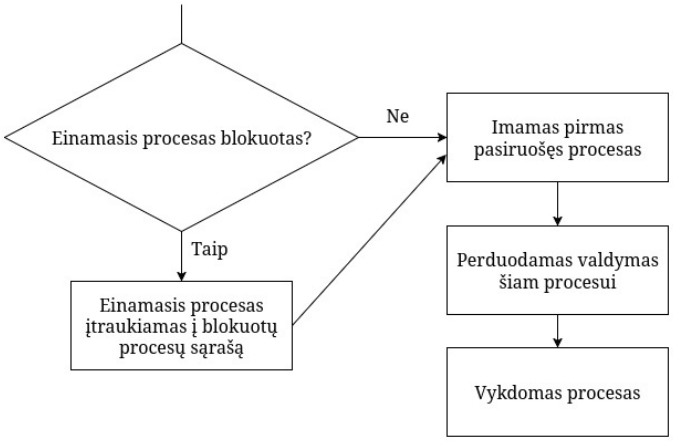
- užtikrinti, kad kiekvienas procesas pagristą laiko tarpą (nei per daug, nei per mažai) turėtų procesorių
- laikyti procesorių užimtą netoli 100%
- sumažinti iki minimumo atsakymo laiką vartotojams
- maksimizuoti darbų skaičių nuveiktą per valandą

Planuotojos iš esmės yra algoritmas, skirtas šių tikslų įgyvendinimui. Šiame OS modeyje remsimės algoritmu, grįstu prioritetais.

1.2.2 Prioritetais besiremiantis planuotojo modeis

Proceso prioritetas - tai proceso svarba, įvertiniama kokio nors skalėje (pvz., nuo 0 iki 255). 0 - tai pati žemiausia proceso svarba, skirta **Idle** procesui (aprašytas vėliau), 255 - pati aukščiausia. Procesas, turintis didesnį proritetą, atsiduria arčiau procesų sąrašo pradžios. Visi procesų sąrašai yra rūšiuojami pagal proceso prioritetą, ir arčiau pradžios esantis procesas turi didesnę galimybę tapti vykomu (pasiruošusių procesų sąrašo atveju) arba pasiruošusiu (blokuotų procesų sąrašo atveju). Sisteminių procesų paskirtis - aptarnauti varototojiškus. Aptarnaujant sisteminius procesus prieš vartotojiškus, sistemos efektyvumas auga, todėl dažniausiai vartotojiški procesai turi žemesnį prioritetą nei sisteminiai.

1.2.3 Planuotojo veiksmų seka



2 pav.: Planuotojo veiksmų sekos schema

Planuotojas kviečiamas kai norima pakeisti einamąjį procesą kitu. Pirma, planuotojas patikrina einamojo proceso būseną. Jei procesas yra blokuotas, jis įtraukiamas į blokuotų procesų sąrašą. Jei procesas nėra blokuotas, jis įtraukiamas į pasiruošusių procesų sąrašą.

Tada, iš pasiruošusių procesų sąrašo imamams pirmas pasiruošęs procesas (su aukščiausiu prioritetu) ir valdymas perduodamas šiam procesui. Proceso apraše laikoma virtualaus procesoriaus būsena priskiriama realiam procesoriui, išsaugoma einamojo proceso aplinka kintamuosiuose, užkraunama pasirinktojo proceso aplinka. Naujasis procesas pažymimas kaip einamasis. Pakeistoje proceso aplinkoje bus ir grįžimo į naująjį enimąjį procesą adresas. Taigi, valdymas netiesiogiai perduotas. Baigdama savo darbą, planuotojo funkcija grįš ne ten, iš kur atėjo, o ten, kur rodo užkrauta einamojo proceso aplinka.

1.2.4 Proceso primityvai

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

1. **Kurti** procesą. Šiam primityvui perduodama nuoroda į jo tėvą, jo pradinė būsena, prioritetas, perduodamų elementų sąrašas ir išorinis vardas. Pačio primityvo viduje vyksta proceso kuriamasis darbas. Jis yra registruojamas bendrame procesų sąraše, tėvo-sūnų sąraše, skaičiuojamas vidinis identifikacijos numeris, sukuriamas jo vaikų procesų sąrašas (tuščias), sukurtų resursų sąrašas ir t.t.

2. **Naikinti** procesą. Pradedama naikinti proceso sukurtus resursus ir vaikus. Vėliau išmetamas iš tėvo sukurtų procesų sąrašo. Toliau išmetamas iš bendro procesų sąrašo ir, jei reikia, iš pasiruošusių procesų sąrašo. Galiausiai naikinami visi jam perduoti resursai ir proceso deskriptorius yra sunaikinamas.

3. **Stabdyti** procesą. Keičiama proceso būsena iš blokuotos į blokuotą sustabdytą arba iš pasiruošusios į pasiruošusią sustabdytą. Einamasis procesas stabdomas tampa pasiruošusiu sustabdytu.

4. **Aktyvuoti** procesą. Keičiama proceso būsena iš blokuotos sustabdytos į blokuotą, ar pasiruošusios sustabdytos į pasiruošusią.

Kiekvieno primityvo programos gale yra kviečiamas planuotojas.

1.2.5 Sisteminiai ir vartotojiški procesai

Jau buvo minėta, kad procesus galima suskirstyti į sisteminius ir vartotojiškus. Sisteminis procesas – tai procesas, atliekantis tam tikras sistemines funkcijas, pavyzdžiui, vartotojiško proceso palaikymą.Vartotojiškas procesas skirtas vartotojo užduočiai atlikti. Sisteminiai procesai yra kuriami paleidžiant sistemą, o naikinami – naikinant sistemą. Taigi visą sistemos gyvavimo laiką jie yra pasiruošę dirbti jiems skirtą darbą. Paprastai paleidus sistemą, visi sisteminiai

procesai anksčiau ar vėliau užsiblokuoja. Bet tai nereiškia, kad jie yra neveiklūs. Jie laukia, kol jie galės atlikti savo darbą, kuris paprastai būna susijęs su vartotojiška užduotimi.Pavyzdžiui, atėjus signalui iš vartotojo sąsajos apie naują užduotį, yra atlaisvinamas tam tikras resurso elementas. Jo dėka atsiblokuoja vienas iš sisteminių procesų, kuris nuveikia tam tikrą darbą ir atlaisvina vieną ar kelis resursus. Taigi, sisteminiai procesai vienas po kito atsiblokuoja, kol galų gale sukuriama vartotojo užduotis. Vartotojo užduočiai baigus darbą, neužsiblokavę procesai ne už ilgo užsiblokuoja, ir vėl prasideda laukimas. Vartotojiški procesai yra sukuriami sisteminių procesų jau veikiant sistemai. Kartu su vartotojišku procesu gali būti sukurti vienas ar keli sisteminiai procesai, skirti aptarnauti vartotojišką procesą. Jau buvo minėta, kad paprastai sisteminiai procesai turi turėti aukštesnį prioritetą nei vartotojiški. Centrinio procesoriaus resursas yra vienintelis, dėl kurio varžosi tiek sisteminiai tiek vartotojiški procesai.Modelyje vartotojiški procesai varžosi tarpusavyje (nes prioritetai neleidžia jiems varžytis su sisteminiais) tik dėl procesoriaus resurso.

1.2.6 Proceso ir su juo susijusių klasių aprašai

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

```
typedef struct Kernel {
    Real_machine* real_machine;
    Process_List processes;
    Process_List running_processes;
    Process_List ready_processes;
    Resource_List resources;
} Kernel;

typedef enum Process_State {
    EXECUTING,
    READY,
    BLOCKED,
    BLOCKED_STOPPED,
    READY,
    READY_STOPPED
} Process_State;

typedef struct Process_List {
    Kernel* kernel;
    Process* process_list;
    uint32_t size;
    uint32_t capacity;
}
```

```
typedef struct Saved_Registers {
    uint16_t pc;
    uint16_t pi;
    uint16_t si;
    uint16_t tr;

    uint8_t ti;
    uint8_t sf;
    uint8_t mr;
    uint8_t ss;

    uint32_t ra;
    uint32_t rb;
    uint32_t rc;
    uint16_t ptr;
} Saved_Registers;

typedef struct Process {
    Saved_Registers saved_registers;
    uint32_t unique_id;
    Process_State process_state;
    Kernel* kernel;
    CPU* cpu;
    struct Process* parent_process;
    uint8_t priority;
    Process_List children_processes;
    Process_List* friend_processes;
    Resource_List* owned_resources;
    Resource_List created_resouces;
    const char* username[255];
} Process;
```

Pertraukimų registrų struktūros:
PI_REG:

- PI_NONE - nėra pertraukimo,
- PI_INVALID_ADDRESS - neteisingas adresas,
- PI_INVALID_OPCODE - neteisingas operacijos kodas,
- PI_INVALID_ASSIGMENT - neteisingas priskyrimas,
- PI_OVERFLOW - žožio arba registro perpildymas,
- PI_DIVISION_BY_ZERO - dalyba iš nulio.

SI_REG:

- SI_NONE - nėra pertraukimo,
- SI_GEDA - žodžio įvedimo pertraukimas,
- SI_PUTA - žodžio išvedimo pertraukimas,
- SI_PSTR - simbolio eilutės išvedimo pertraukimas,
- SI_LW - žodžio užkrovimo iš atminties į RA registro pertraukimas,
- SI_SW - žodžio kopijavimo į atmintį iš RA registro pertraukimas,
- SI_BP - žodžio kopijavimo į bendrosios atminties ląstelį iš RA registro pertraukimas,
- SI_BG - žodžio kopijavimo iš bendrosios atminties ląstelės į RA registrą pertraukimas,
- SI_STOP - vartotojo programos pabaigos pertraukimas.

Proceso būsenos apibrėžimas **Process_State**:

- EXECUTING – einamasis procesas,
- READY – pasiruošęs procesas,
- BLOCKED – blokuotas procesas,
- READY_STOPPED – pasiruošęs sustabdytas procesas,
- BLOCKED_STOPPED – blokuotas sustabdytas procesas.

Proceso aprašas **Process**:

- Process_List friend_processes - procesų sąrašas, kuriam priklauso procesas. Tai gali būti pasiruošusių procesų sąrašas arba blokuotų, dėl kokio nors resurso, procesų sąrašas.
- uint32_t unique_id - vidinis proceso vardas. Kiekvienas sisteminis objektas turi savo unikalų numerį.
- Saved_Registers saved_registers - proceso išsaugota procesoriaus būsena. Naudojama perduodant valdymą procesui.
- CPU* cpu - nuoroda į procesorių.
- Resource_List created_resources - proceso sukurti resursai.
- Resource_List* owned_resources - procesui kūrimo metu perduoti resursai.
- Process_State process_state - proceso būsena.

- uint8_t priority - proceso prioritetas. Pasirinkta sistema: kuo didesnė reikšmė, tuo procesas laikomas svarbesniu.
- Process* parent_process - nuoroda į procesą–tėvą.
- Process_List children_processes - procesų–vaikų sąrašas.
- Kernel* kernel - nuoroda į branduolį.

Procesų sąrašo aprašas **Process_List**:

- Kernel* kernel - nuoroda į branduolį.
- Process* process_list - dinaminis procesų sąrašas.
- uint32_t size - procesų skaičius procesų masyve.
- uint32_t capacity - dinaminio masyvo talpa;

Operacinės sistemos branduolys **Kernel**

- Process_List processes - bendras visų, esančių sistemoje, procesų sąrašas.
- Resource_List resources - bendras visų, sistemoje esančių, resursų sąrašas.
- Process_List ready_processes - pasiruošusių procesų sąrašas.
- Process_List running_processes - einami procesai. Jų gali būti ne vienas, jei realioje.
- Real_Machine* real_machine - nuoroda į realią mašiną. mašinoje yra vienas procesoriai.

1.3 Resursai

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

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

Iškart reiktų pastebėti, kad resursas iš tikrųjų tėra tik aprašas (deskriptorius) ir nuoroda į resurso elementus. Prašyti resurso – tai prašyti resurso elemento. Lengviausia būtų tai pavaizduoti atminties resurso atveju. Tarkime, vartotojo atmintis susidedanti iš 51 takelio. Resursas "atmintis" turėtų kintamuosius, tokius kaip resurso elementų kiekis, laisvų elementų kiekis, galiausiai nuorodą į pačius resurso elementus. Resurso elementai turėtų lauką, kuriame būtų takelio numerio reikšmė. Kiekvienas resursas turi laukiančių procesų sąrašą (jis gali būti ir tuščias). Kiekvienas procesas prašęs, ir negavęs resurso elemento, yra ne tik užblokuojamas, bet ir įdedamas į resurso laukiančių procesų sąrašą.

1.3.1 Resurso ir su juo susijusių kalsių aprašai

```
typedef enum Resource_Type {
    MOS_END,
    SYSTEM_COMMAND,
    HARD_DISK,
    USER_MEMORY,
    SUPERVISOR_MEMORY,
    TASK_IN_SUPERVISOR_MEMORY,
    STRING_IN_MEMORY,
    LOADER_PACKAGE,
    PIE_IN_THE_OVEN,
    NON_EXISTANT,
    CHANNEL_DEVICE,
    USER_INPUT,
    INTERRUPT,
    FROM_INTERRUPT,
    FROM_LOADER,
    FROM_USER_INTERFACE
} Resource_Type

typedef struct Resource_Element {
    int8_t return;
    Process* sender;
    Process* receiver;
    Element_List* friend_list;
} Resource_Element;

typedef struct Element_List {
    Resource_Element* element_list;
    uint32_t size;
    Resource* owner_resource;
} Resource_List;

typedef struct Resource {
    uint32_t unique_id;
```

```
    Process* creator;
    Process_List* waiting_list;
    Process_List* friend_list;
    uint32_t* waiting_count;
    Kernel* kernel;
    Resource_List* all_resources;
    Resource_Type resource_type;
    Element_List* elements;
    Element_List* waiting_process_pointers;
} Resource;

typedef struct Resource_List {
    Resource* resource_list;
    uint32_t size;
    Kernel* kernel;
} Resource_List;
```

Resurso aprašas **Resource**:

- uint32_t unique_id - vidinis resurso vardas. Kiekvienas sisteminis objektas turi savo unikalų numerį.
- Process* creator -Nuoroda į procesą, sukūrusį šį resursą.
- Element_List* elements - Nuoroda į resurso elementų sąrašą.
- Process_List* waiting_list -Nuoroda į šio resurso laukiančių procesų sąrašą.
- uint32_t* waiting_count -Nuoroda į šio resurso laukiančių procesų paprašytų nresurso kiekių sąrašą.
- Kernel* kernel - Nuoroda į branduolį.
- Resource_List* all_resources- nuoroda į visų resursų sąrašą
- Element_List* waiting_process_pointers - nuoroda į dabar laukiamų resurso elementų sąrašą.

Resursų sąrašo aprašas **Resource_List**:

- uint32_t size - sąraše esančių resursų kiekis.
- Kernel* kernel - nuoroda į branduolį.
- Resource* resource_list - dinaminis masyvas, kurio elementai – resursai.

Resurso elemento aprašas **Resource_Element**:

- Element_List* friend_list; - nuoroda į resurso elementų sąrašą, kuriame yra šis resurso elementas.

- Process* receiver - procesas, kuris turi gauti šį resurso elementą. Jei šio lauko reikšmė lygi nil, tai laikoma, kad šį elementą gali gauti bet kuris procesas.
- Process* sender - procesas, atlaisvinęs šį resurso elementą
- int8_t return - šio lauko reikšmė žymi, ar šio tipo resurso elementai bus gražinami kaip sąrašas ar kaip paprastas elementas. Pavyzdžiui atminties atveju šio lauko reikšmė turėtų būti lygi 1, nes grąžinti reikės elementų, su takelių numeriais, sąrašą.

Resurso elementų sąrašas **Element_List**:

- Resource* owner_resource - Nuoroda į resursą, kuriam priklauso šis elementų sąrašas.
- uint32_t size - Elementų skaičius sąraše. Šis skaičius nurodo, kiek resurso elementų yra laisvų (t.y. prieinamų procesams) iš viso.
- Resource_Element* element_list - dinaminis elementų sąrašas. Kreipiantis su tam tikru indeksu pasiekiamas atitinkamas elementas.

1.3.2 Resursu primityvai

Resursas turi keturis primityvus:

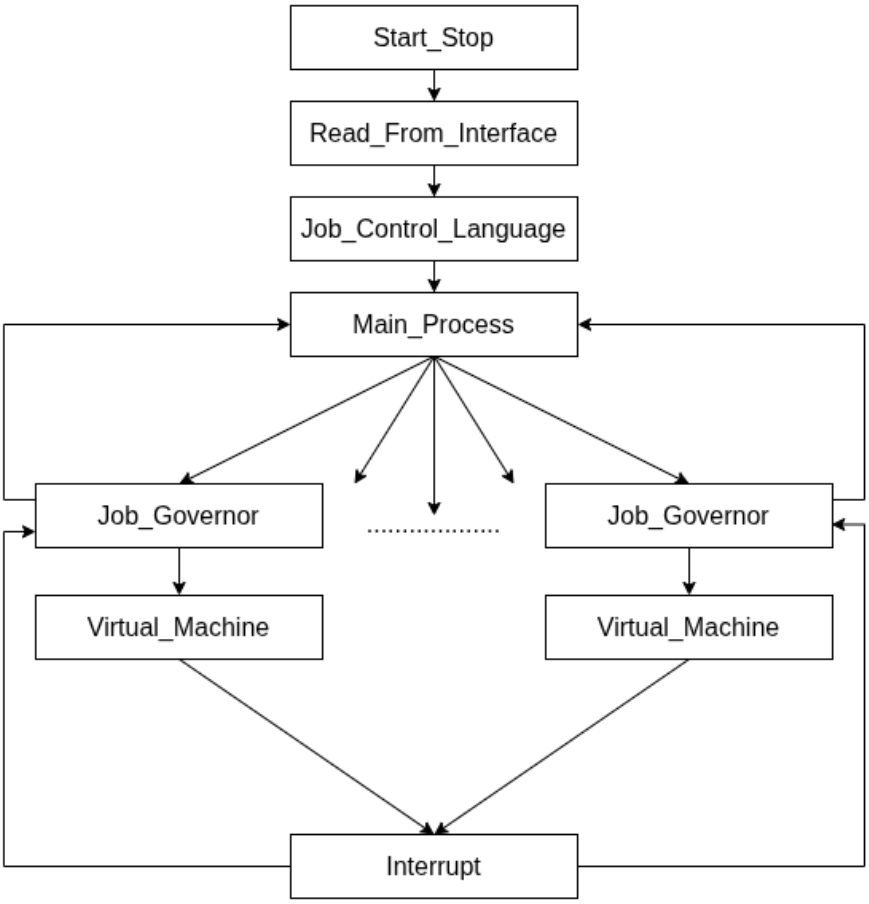
- **Kurti** resursą. Resursus kuria tik procesas. Resurso kūrimo metu perduodami kaip parametrai: nuoroda į proceso kūrėją, resurso išorinis vardas. Resursas kūrimo metu yra: pridedamas prie bendro resursų sąrašo, pridedamas prie tėvo suskurtų resursų sąrašo, jam priskiriamas unikalus vidinis vardas,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

1.3.3 MOS Resursai

Resurso pavadinimas	Kuria	Laukia	Atlaisvina
MOS pabaiga	Main_Process	Start_Stop	-
Sisteminė komanda	Start_Stop	Main_Process	Read_From_Interface
Išorinė atmintis	Start_Stop	Read_From_Interface	Job_Governor
Vartotojo atmintis	Start_Stop	Job_Governor	Job_Governor
Supervizorinė atmintis	Start_Stop	Read_From_Interface	Loader
Užduotis supervizorinėje atmintyje	Read_From_Interface	Job_Control_Language	Read_From_Interface
Eilutė atmintyje	Start_Stop	Printer	Job_Control_Language, Job_Governor
Pakrovimo paketas	Job_Control_Language	Loader	Job_Governor
Pyragas pašautas į orkaitę	Start_Stop	Main_Process	Job_Control_Language, Job_Governor
Neegzistuojantis	Start_Stop	Job_Governor	-
Kanalų įrenginys	Start_Stop	Job_Governor, Loader, Printer	Job_Governor, Loader, Printer
Vartotojo įvedimas	Start_Stop	Job_Governor	Virtual_Machine
Iš pertraukimo	Interrupt	Job_Governor	Interrupt
Pertraukimas	Start_Stop	Interrupt	Virtual_Machine
Iš užkrovėjo	Loader	Job_Governor	Loader
Iš vartotojo sąsajos	Start_Stop	Read_From_Interface	-

1.4 Proces paketas

- **Idle** - procesas, skirtas užimti procesorių kai nėra kitų aktyvių procesų.
- **Start_Stop** - procesas, atsakingas už sisteminių procesų bei resursų sukūrimą ir paleidimą.
- **Read_From_Interface** - procesas, skirtas nuskaitymui iš standartinės įvesties.
- **Job_Control_Language** - procesas, atsakingas už Supervior_Loader, Supervisor_Validator ir Loader procesų koordinavimą. Jis tampa tarpininku tarp Read_From_Interface ir Job_Governor procesų.
- **Main_Process** - procesas, valdantis Job_Governor procesus.
- **Job_Governor** - virtualios mašinos proceso tėvas, tvarkantis virtualios mašinos proceso darbą.
- **Virtual_Machine** - procesas, atsakingas už vartotojo programos vykdymą.
- **Loader** procesas, atsakingas už programos kopijavimą iš supervizorinės į vartotojo atmintį.
- **Interrupt** - procesas, atsakingas už virtualios mašinos sukeltų pertraukimų tvarkymą.
- **Printer** - procesas, atsakingas už išvedimą į standartinę išvestį.



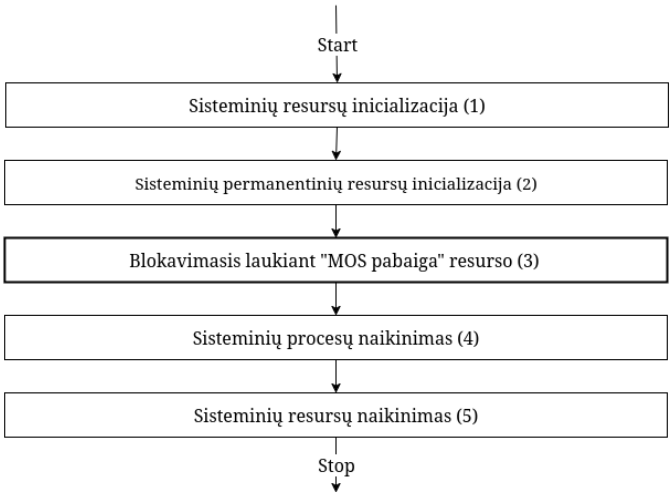
3 pav.: Procesų bendra schema

1.4.1 Idle procesas

Idle procesas skirtas ”užimti” procesoriu, kai nėra vykdomi kiti procesai. Jo egzistavimas išsprendžia klausimą ką daryti, kai nėra procesų kurie turi būtį vykdomi. Tai supaprastina (2) planuotojo schemą, nes procesorius bus užimtas. Idle procesas pats nieko nedaro, jis gali būti realizuotas amžinu ciklu.

1.4.2 Start_Stop procesas

Šis procesas atsakingas už sistemos darbo pradžios ir pabaigos darbą. Įjungus kompiuteį šis procesas pasileidžia automatiškai. Šio proceso paskirtis - sisteminių procesų ir resursų kūrimas.



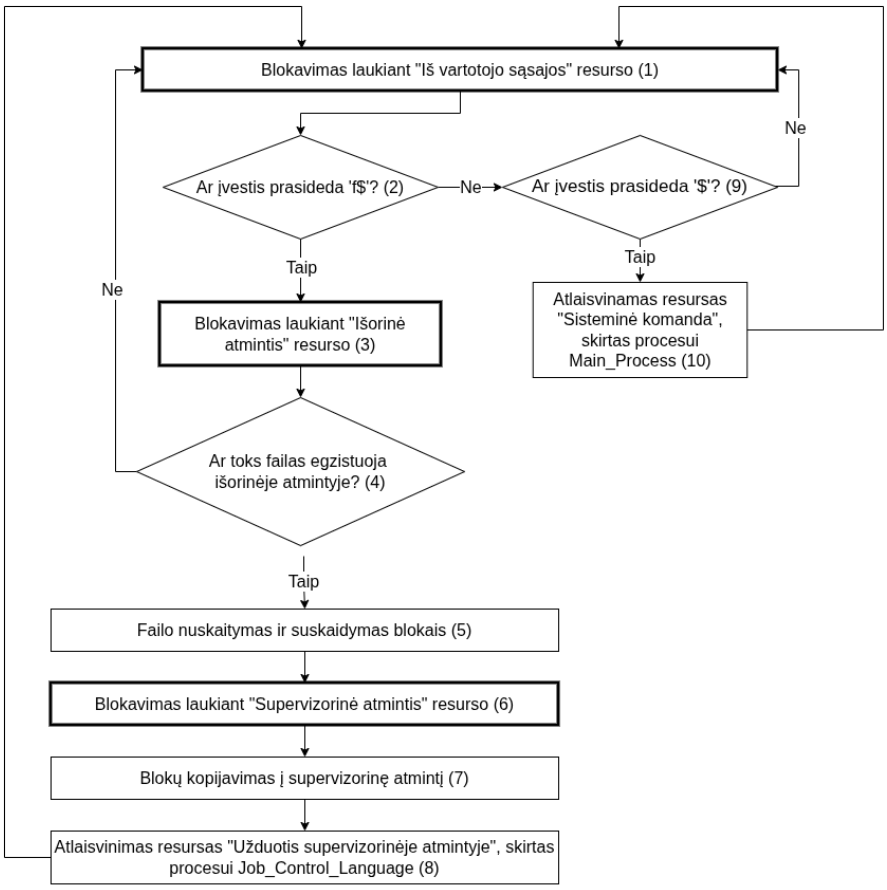
4 pav.: Start_Stop proceso diagrama

Procesas *Start_Stop*, gavęs procesorių, savo darbą pradeda sukurdamas visus siteminius resursus (1). Laikoma, kad procesoriaus resurso kurti nereikia, jis kuriamas ir naikinamas įjungiant ir išjungiant kompiuterį. Sukūręs resursus, *Start_Stop* kuria permanentinius procesus (2), kurie bus aktyvūs visą MOS gyvavimo laiką. Mūsų modelyje tokie procesai bus: *Read_From_Interface*, *Job_Control_Language*, *Main_Process*, *Interrupt*, *Idle*, *Loader* ir *Printer*. Sekantis *Start_Stop* žingsnis yra prašyti resurso "MOS pabaiga" (3). Šioje vietoje procesas blokuojasi ir laukia, kol jo negaus. Gavęs minėtą resursą, *Start_Stop* naikina sisteminius procesus (4) ir sisteminius resursus (5).

1.4.3 Read_From_Interface procesas

Šį procesą kuria ir naikina procesas *Start_Stop*. Šio proceso paskirtis - gavus informaciją iš įvedimo srauto ir atlikus pirminį jos apdorojimą, atiduoti informaciją tolesniam apdorojuimui, kurė atliks *Job_Control_Language* procesas. 5 paveikslėlyje pateikta proceso diagrama. *Read_From_Interface* pirmasis žingsnis - laukti įvedimo srauto (1). Srautu yra tikimasi vieno iš dviejų: f$*failo_pavadinimas* (2) arba $*sisteminė_komanda* (9). Jei įvesta sisteminė komanda (9), atlaisvinamas resursas "Sisteminė komanda", skirtas pocesui *Main_Process* (10). Jei buvo įvestas failo pavadinimas (2), laukiama resurso "Išorinė atmintis" (3) ir tada patikrinama ar toks failas egzistuoja kietajame diske (4). Jei ne, procesas vėl užsiblokuoja laukdamas. Jei taip, failas yra nuskaitomas ir skaidomas blokais (64 baitai)(5). Apdorojimo rezultatas - blokų sąrašas laikomas supervizorinėje amintyje, todėl mes jos prašome (6) ir kopijuojame blokus į supervizorinę atmintį (7). Kopijavimas vykstat pasinaudojant kanalų įrenginiu - nustatomi kanalų įrenginio registrai ir kviečiama komanda **XCHG**. Galiausiai

yra sukuriamas ir atlaisvinamas resursas ”Užduotis supervizorinėje atmintyje”, skirtas procesui *Job_Control_Language* (8), kuriame yra informaciją apie blokų padėtį atmintyje. Proceso programa yra ciklinė, todėl procesas *Read_From_Interface* vėl užsiblokuoja laukdamas.

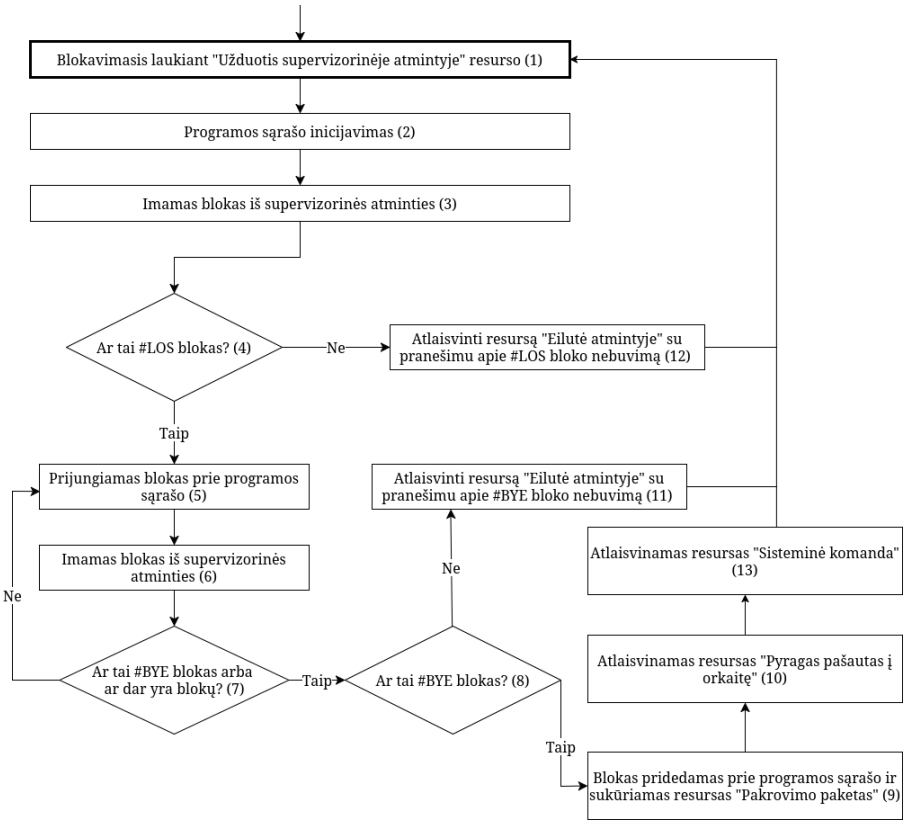


5 pav.: Read_From_Interface proceso diagrama

1.4.4 Job_Control_Language procesas

Procesą *Job_Control_Language* kuria ir naikina procesas *Start_Stop*. Proceso *Job_Control_Language* paskirtis - pravaliduoti programą supervizorinėje atmintyje gauta iš *Read_From_Interface* ir atidavus ją procesui *Loader* vėl blokuotis. Procesas pradeda savo darbą blokuodamasis, laukdamas pranešimo iš proceso *Read_From_Interface* (1). Sulaukus šio pranešimo procesas pasiruošia darbui ir inicijuoja programos blokų sąrašą (2). Imamas pirmas blokas iš supervizorinės atminties (3) ir žiūrima ar pirmi keturi baito blokai yra #LOS (4).

Jei ne atlaisviname resursą ”Eilutė atmintyje” su pranešimų apie #LOS nebuvimą (12). Jei taip jį prijungiame prie programos blokų sąrašo (5) ir vėl imamas blokas iš supervizorinės atminties (6). Jei blokų dar yra ir paimtas blokas nėra #BYE (7) blokas, grįžtame į (5) žingsnį. Kitu atveju tikriname ar blokas yra #BYE blokas (8), jei ne - atlaisviname resursą ”Eilutė atmintyje” su pranešimu apie #BYE trūkumą (11). Jei taip - blokas pridedamas prie programos blokų sąrašo ir sukuriamas resursas ”Pakrovimo paketas” (9). Tuomet atlaisviname resursą ”Pyragas pašautas į orkaitę” (10) ir procesas vėl blokuojamas.

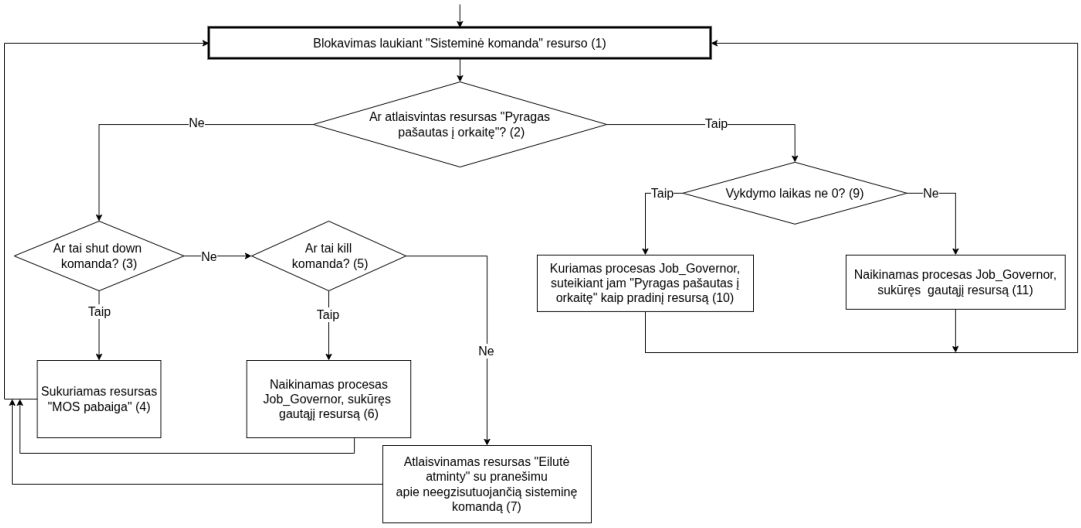


6 pav.: Job_Control_Language proceso diagrama

1.4.5 Main_Process procesas

Procesą *Main_Process* kuria ir naikina procesas *Start_Stop*. Šis procesas turi 2 paskirtis: kurti ir naikinti procesą *Job_Governor* ir vykdyti sistemines komandas. Šis procesas pradeda savo veiklą prašydamas ”Sisteminė komanda” arba ”Pyragas pašautas į orkaitę” (1). Jei atlaisvinamas ”Sisteminė komanda resursas” (2), patikrinama ar tai egzistuojanti sisteminė komanda (3). Jei taip, ji yra įvykdoma (4), jei ne - procesas vėl užsiblokuoja. Jei buvo atlaisvintas

”Pyragas pašautas į orkaitę” (5), *Main_Process* tikrina ar jo vykdymo laikas nėra 0 (6). Nenulinė reikšmė - naujo *Job_Governor* proceso kūrimo ženklas (7), kai tuo tarpu nulinė reikšmė yra *Job_Governor* proceso naikinimo ženklas (8). Kurdamas *Job_Governor* resursą, *Main_Process* suteikia jam ”Pyragas pašautas į orkaitę” kaip pradinį resursą. Naikinamas būtent tas *Job_Governor*, kuris atsiuntė ”Pyragas pašautas į orkaitę” resursą su nuliniu vykdymo laiku. Atlikes savo darbą *Main_Process* cėl blokuojasi laukdamas ”Sisteminė komanda” arba ”Pyragas pašautas į orkaitę” resurso.



7 pav.: Main_Process proceso diagrama

1.4.6 Job_Governor procesas

Procesą (galima sakyti ir procesus - jų vienu metu gali būti keli) *Job_Governor* kuria procesas *Main_Process*. Proceso paskirtis - kurti, naikinti ir padėti procesui *Virtual_Machine* atlikti savo darbą, t.y. atlikti veiksus, kurių *Virtual_Machine* procesas, dirbant vartotojo režimu, nesugeba atlikti. Vienas *Job_Governor* aptarnauja vieną virtualią mašiną. Procesas *Job_Governor* pradeda darbą laukdamas vartotojo atminties (1), kur bus perkelta vartotojo užduoties programa. Prašoma tiek takelių, kiek yra blokų. Gavęs resursą ”Vartotojo atmintis”, procesas atlaisvina resursą ”Pakrovimo paketas” (2), skirta procesui *Loader* ir blokuojasi, laukdamas pranešimo apie *Loader* darbo pabaigą (3). Perkėlus vartotojo užduoties programą į vartotojo atmintį, reikia atlaisvinti išorinę atmintį, kur ji buvo iki šio (4). Tada procesas paprašo dar vieno takelio vartotojo atminties, kur bus laikoma virtualios mašinos puslapių lentelė (5). Ši lentelė užpildoma išskirtosios vartotojo atminties adresais. Tada, kuriamas procesas *Virtual_Machine* (6). *Job_Governor* procesas blokuosis iki tol, kol negaus pranešimo iš proceso *Interrupt* apie įvykusį pertraukimą *Virtual_Machine* proceso

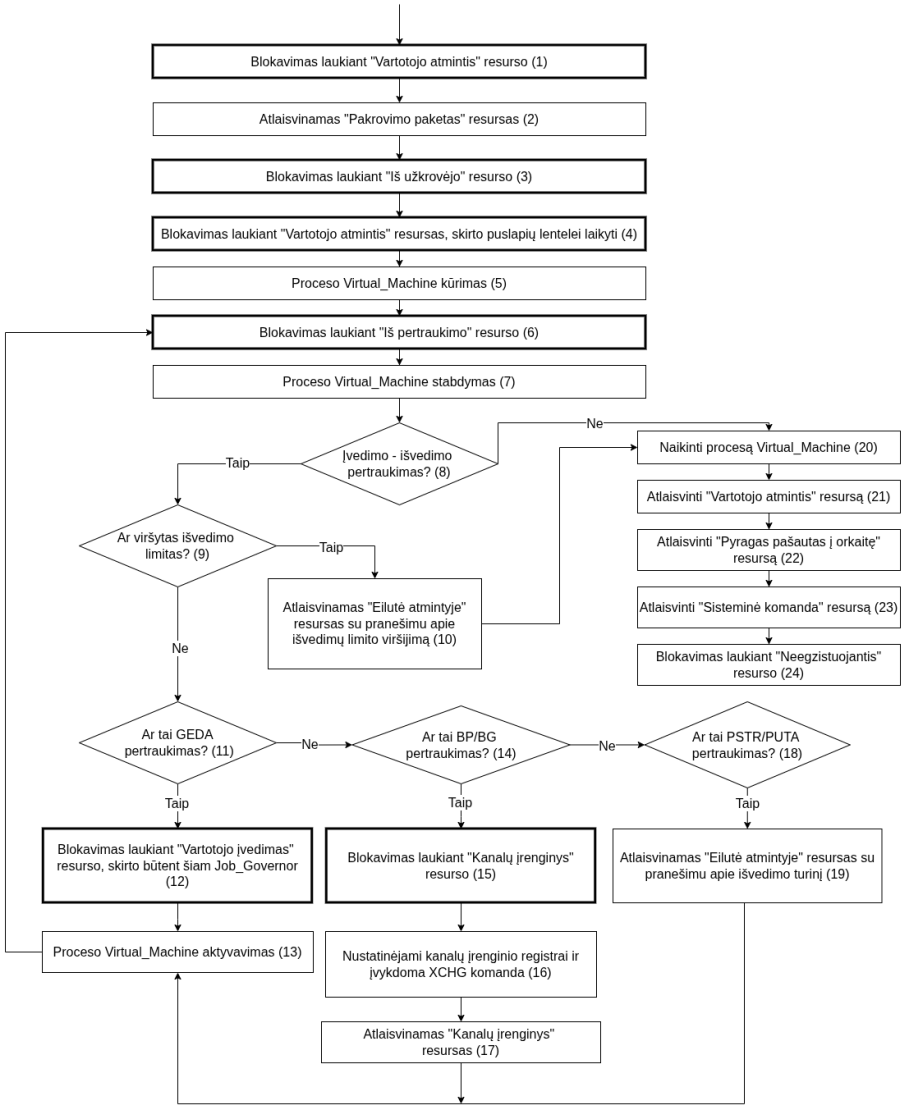
darbo metu (7).

Gavęs pranešimą apie pertraukimą, *Job_Governor* stabdo procesą *Virtual_Machine* (8). Tuomet, yra tikrinama ar tai yra įvedimo - išvedimo pertraukimas (9). Jei tai nėra įvedimo - išvedimo pertraukimas, naikinamas procesas *Virtual_Machine* (20), atlaisvinama jos užimta vartotojo atmintis (21) ir duodamas ženklas procesui *Main_Process* apie tai, kad šis *Job_Governor* jau baigė darbą ir turi būti sunaikintas. Šis ženklas - sukurtas ir atlaisvintas fiktyvus procesas "Pyragas pašautas į orkaitę" su nuliniu vykdymo laiku (22). Kadangi šis procesas savo darbą jau baigė, tačiau dar nėra sunaikintas, jis blokuojasi laukdamas "Neegzistuojantis" resurso (23). Šio resurso *Job_Governor* niekada negaus, tad jis blokuosis tol, kol nebus sunaikintas. Kai procesas *Main_Process* gaus *Job_Governor* atlaisvintą fiktyvų "Pyragas pašautas į orkaitę" resursą ir sunaikins šį *Job_Governor* procesą.

Jei tai buvo įvedimo - išvedimo pertraukimas (9), yra tikrinama ar neviršytas išvedimo limitas (10). Limitą viršijus, yra atlaisvinamas "Eilutėje atmintyje" resursas su pranešimu apie išvedimų limito viršijimą (11). Toliau atliekami auksčiau aptarti veiksmai: (20), (21), (22) ir (23). Jei limitas nebuvo viršytas, yra tikrinama ar tai GEDA pertraukimas (12), procesas blokuojasi laukdamas "Vartotojo įvedimas" resurso (14). Šioje vietoje *Job_Gorvernor* prašo resurso, kurį įves "anapus" modelio esantis vartotojas į virtualios mašinos langą. Taigi vartotojas mato, kaip jo programa laukia jo įvedimo, tuo tarpu vykdydama kitas užduotis. Gavęs reikiamą resursą, procesas *Virtual_Machine* yra aktyvuojamas (14) ir procesas *Job_Governor* ciklišKai grįžta toliau blokuotis, laukdamas "Iš pertraukimo" resurso (7).

Jei tai nebuvo GEDA pertraukiamas, tikrinama ar tai buvo BP arba BG pertraukimas (15). Jei taip, tada procesas blokuojasi laukdamas resurso "Kanalų įrenginys" (16). Sulaukęs šio resurso, *Job_Governor* procesas nustato kanalų įrenginio registrus ir įvykdoma XCHG komanda (17). Tada, aktyvuojamas *Virtual_Machine* procesas (14) ir procesas vėl blokuojasi (7).

Jei tai nebuvo BP ar BG pertraukimas, procesas tikrina ar tai PSTR ar PUTA pertraukimas (18). Tokiu atveju, atlaisvinamas "Eilutė atmintyje" resursas su pranešimu apie išvedimo turinį (19). oliau, lygiai taip pat kaip ankstesniais atvejais, aktyvuojamas *Virtual_Machine* procesas (14) ir procesas vėl blokuojasi (7).

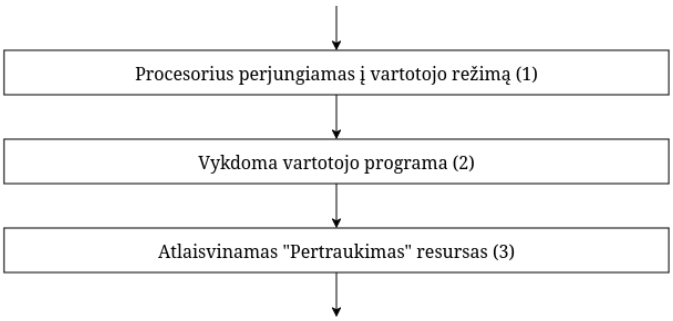


8 pav.: Job_Governor proceso diagrama

1.4.7 Virtual_Machine procesas

Procesą *Virtual_Machine* kuria ir naikina *Job_Governor*. Šio proceso paskirtis yra įvykdyti vartotojo užduotį. Procesų *Virtual_Machine* yra tiek pat kiek procesų *Job_Governor*. Virtualios mašinos darbo pradžioje procesorius yra perjungiamas į vartotojo režimą (1). Vėliau vykdoma virtualaus virtualaus procesoriaus komandų interpretavimo programa (2). Ši programa veikia tol, kol sis-

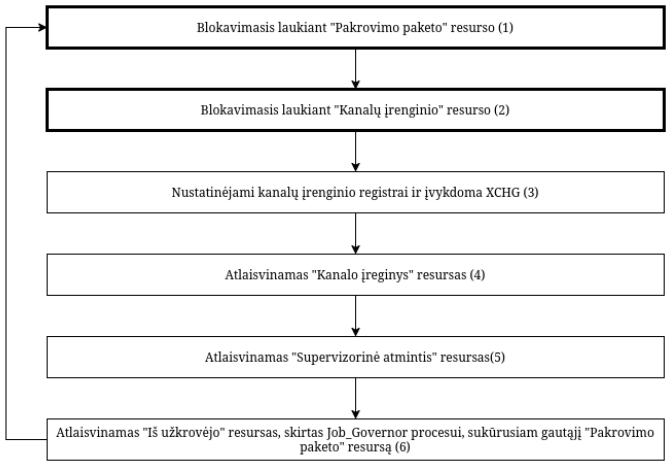
temoje neaptinkamas pertraukimas. Tada virtuali mašina išsaugo savo procesoriaus būseną, perduodamas valdymas pertraukimą apdorosiantiems procesams. Apdorojus pertraukimą, valdymui sugrįžus į virtualią mašiną, yra atlaisvinamas resursas "Pertraukimas" (3), skirtas procesui *Interrupt*. Procesas *Virtual_Machine* turi mažą prioritetą, todėl procesorių visų pirma gaus procesas *Interrupt*, kuris identifikuos pertraukimą ir perduos informaciją *Job_Governor*, kuris atlikęs veiksmus, priklausomai nuo situacijos, aptarnaus arba naikins virtualią mašiną.



9 pav.: Virtual_Machine proceso diagrama

1.4.8 Loader procesas

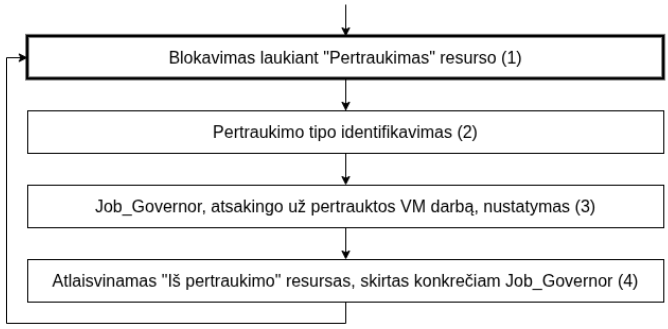
Procesą *Loader* kuria ir naikina procesas *Start_Stop*. Šio proceso paskirtis - supervizorinėje atmintyje esančius blokus perkelti į vartotojo atmintį. Procesas *Loader* pradeda savo darbą laukdamas "Pakrovimo paketo" resurso (1). Pakrovimo pakete yra laikoma informacija apie supervizorinės atminties takelius ir vartotojo atminties takelius. Pakrovimo paketą šiame modelyje siunčia *Job_Governor* procesas. Gavęs šį resursą, procesas blokuojasi laukdamas "Kanalų įrenginys" resurso (2). Toliau *Loader* dirba su kanalų įrenginiu nustatydamas atitinkamus jo registrus ir kviesdamas XCHG (3). Vėliau yra atlaisvinamas "Kanalų įrenginio" resursas(4) ir sukuriamas bei atlaisvinamas pranešimas *Job_Governor* apie darbo pabaigą (5). Įvykdęs savo darbą, procesas vėl blokuojasi ir laukia "Pakrovimo paketo" resurso.



10 pav.: Loader proceso diagrama

1.4.9 Interrupt procesas

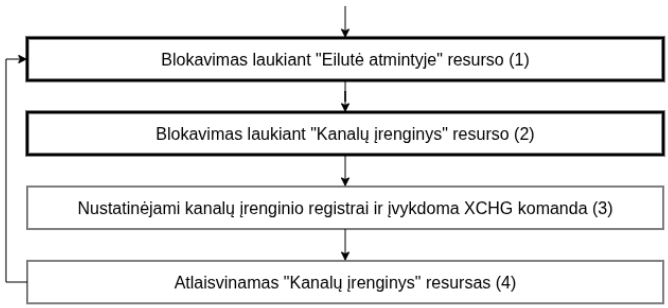
Procesą *Interrupt* kuria ir naikina procesas *Start_Stop*. Šio proceso paskirtis - reaguoti į pertraukimus, kilusius virtualios mašinos darbo metu. Proceso schema pateikiama 11 diagramoje. Procesas *Interrupt* savo darbo pradžioje laukia ”Pertraukimas” resurso, kurį siunčia procesas *Virtual_Machine* (1). Tada, procesas nustato pertraukimo tipą tikrindamas pertraukimo programų nustatytas sisteminių kintamųjų reikšmes (2). Kiekviena virtuali mašina kūrimo metu laiko savo tėvo *Job_Governor* identifikatorių. Kartu su pranešimu apie pertraukimą yra perduodamas ir tas identifikatorius, kurį *Interrupt* naudoja jam reikalingo *Job_Governer* atpažinimui (3). Galiausiai yra kuriamas ir atlaisvinamas ”Iš pertraukimo” resursas, kuris yra skirtas nustatytajam *Job_Governor* procesui (4).



11 pav.: Interrupt proceso diagrama

1.4.10 Printer procesas

Procesą *Printer* kuria ir naikina procesas *Start_Stop*. Šio proceso paskirtis - pasiųsti kokioje nors atmintyje esantį pranešimą. Šio proceso schema pateikiama 12 diagramoje. Proceso darbas prasideda blokavimusi laukiant "Eilutė atmintyje" resurso (1). Šis resursas turi parametrą, nusakantį iš kurios atminties reikės pasiųsti eilutę į išvedimo srautą bei atminties adresą, žymintį bloko numerį. Toliau procesas blokuojasi laukdamas leidimo dirbti su kanalų įrenginiu (2). Atblokuotas jis nustato kanalo duomenš perdavimo kryptį ir adresą bei įvykdo komandą XCHG (3). Įvykdęs komandą, procesas atlaisvina resursą "Kanalų įrenginys" (4). Procesas ciklišKai blokuojasi pirmame savo žingsnyje - laukdamas "Eilutė atmintyje" resurso.



12 pav.: Printer proceso diagrama