Multiprograminės operacinės sistemos (MOS)

projektas

Tautvydas Januškevičius

Vincas Razma

Mindaugas Vasiliauskas

VU MIF PS-2

2010

**Turinys**

[Turinys 2](#_Toc262038589)

[Failų sistema 3](#_Toc262038590)

[Kietojo disko atminties vaizdas 3](#_Toc262038591)

[Failų sistemos aprašas 3](#_Toc262038592)

[Failo aprašas 3](#_Toc262038593)

[Failų tipai 4](#_Toc262038594)

[Vykdomojo failo (\*.PRG) antraštė 4](#_Toc262038595)

[Deskriptoriai 5](#_Toc262038596)

[Proceso deskriptorius – CLASS PRocessDescriptor 5](#_Toc262038597)

[Resurso deskriptorius – CLASS ResourceDescriptor 5](#_Toc262038598)

[Deskriptorių ir primityvų naudojami sąrašų elementų tipai 5](#_Toc262038599)

[Primityvai 6](#_Toc262038600)

[Sukurti procesą 6](#_Toc262038601)

[Sunaikinti procesą 6](#_Toc262038602)

[Sustabdyti procesą 7](#_Toc262038603)

[Aktyvuoti sustabdytą procesą 7](#_Toc262038604)

[Keisti proceso prioritetą 7](#_Toc262038605)

[Sukurti resursą 7](#_Toc262038606)

[Naikinti resursą 8](#_Toc262038607)

[Prašyti resurso 8](#_Toc262038608)

[Atlaisvinti resursą 9](#_Toc262038609)

[Procesoriaus paskirstytojas (planuotojas) 9](#_Toc262038610)

[Resursų paskirstytojas 10](#_Toc262038611)

[Pagalbinės funkcijos 10](#_Toc262038612)

[Procesai 11](#_Toc262038613)

[Procesų prioritetai 11](#_Toc262038614)

[Procesų lentelė 11](#_Toc262038615)

[Operacinės sistemos procesų medis 11](#_Toc262038616)

[Procesų veiksmų sekų diagramos 12](#_Toc262038617)

[Proceso būsenų diagrama 16](#_Toc262038618)

[Procesų tarpusavio ryšių diagrama 17](#_Toc262038619)

[Resursai 18](#_Toc262038620)

[Resursų naudojimą aprašanti lentelė 18](#_Toc262038621)

[Užduočių pranešimai 19](#_Toc262038622)

[Darbo pabaigos pranešimai 19](#_Toc262038623)

[Virtualios mašinos bendravimas su operacine sistema 20](#_Toc262038624)

[Sisteminiai pertraukimai 20](#_Toc262038625)

[Dinaminė atmintis 20](#_Toc262038626)

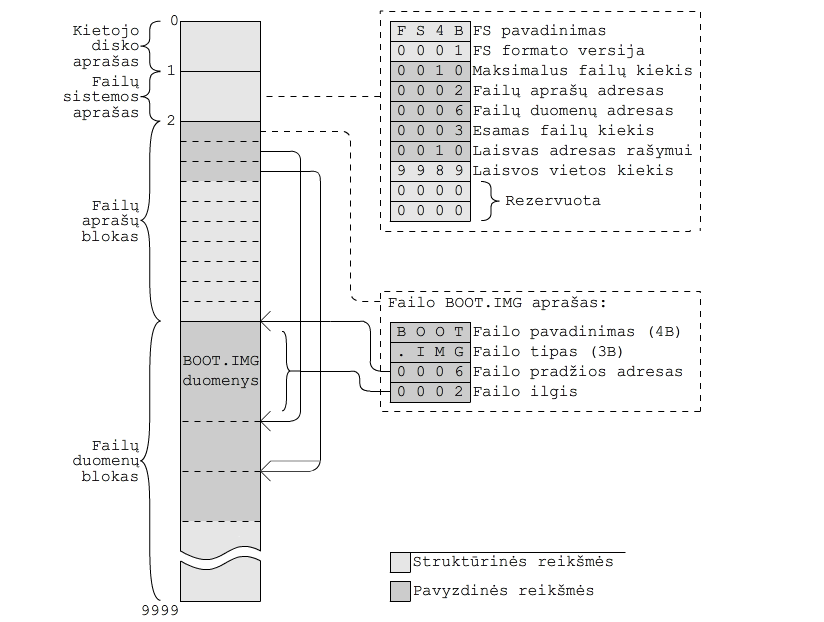
[Operacinės sistemos paleidimas 20](#_Toc262038627)

[Procesoriaus taktų paskirstymas 21](#_Toc262038628)

Failų sistema

Failų sistema projekte numatoma elementari. Pagrindiniai jos elementai –failų duomenys ir failų aprašai, nurodantys, kokiam failui kokie duomenys iš kietojo disko atminties priklauso. Dydžiai ir adresai failų sistemoje matuojami puslapiais kietajame diske.

Kietojo disko atminties vaizdas

****

Failų sistemos aprašas

|  |  |  |
| --- | --- | --- |
| **Žodis** | **Paaiškinimas** | **Statinė reikšmė** |
| 0 | Failų sistemos pavadinimas | FS4B |
| 1 | Failų sistemos formato versija. | 0001 |
| 2 | Maksimalus failų kiekis | - |
| 3 | Failų aprašų adresas (nurodo failų aprašų bloko adresą) | - |
| 4 | Failų duomenų adresas (nurodo failų duomenų bloko adresą) | - |
| 5 | Esamas failų kiekis | - |
| 6 | Laisvas adresas rašymui (nuo kurio nera failų duomenų) | - |
| 7 | Laisvas vientios atminties kiekis rašymui nuo nurodyto adreso | - |
| 8 | Rezervuota | 0000 |
| 9 | Rezervuota | 0000 |

Failo aprašas

|  |  |
| --- | --- |
| **Žodis** | **Paaiškinimas** |
| 0 | Failo pavadinimas – 4 simboliai |
| 1 | “.XXX” – pirmas simbolis visada taškas, likę 3 – failo tipas. |
| 2 | Failo duomenų pradžios adresas |
| 3 | Failo ilgis |

Failų tipai

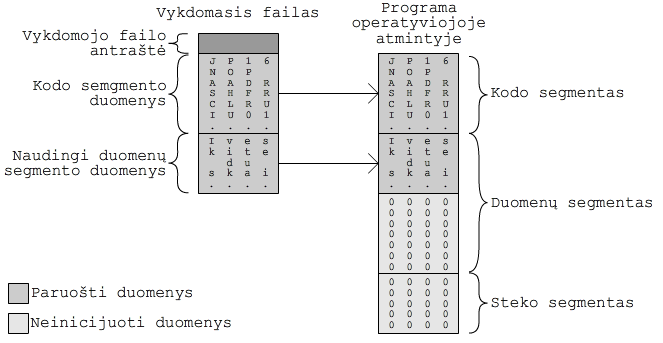
Projekte numatomi du pagrindiniai failų tipai:

1. Vykdomasis failas (\*.PRG). Talpina informacija, kuri reikalinga operacinei sistemai užkraunant ir paleidžiant atitinkamą programą.
2. Operatyvios atminties atvaizdo failas (\*.IMG). Naudojamas kaip šaltinis paleidžiant mašiną operatyvios atminties užkrovimui.

Vykdomojo failo (\*.PRG) antraštė

|  |  |  |
| --- | --- | --- |
| **Žodis** | **Paiškinimas** | **Statinė**  **reikšmė** |
| 0 | Operacinės sistemos pavadinimas | MPOS |
| 1 | Failo formato versija | 0001 |
| 2 | Kraunamas programos dydis (faile CS+esamas DS) | - |
| 3 | Realus programos dydis (puslapių lentelės dydis) | - |
| 4 | Steko ir duomenų segmento adresai viename žodyje (SS:DS) | - |
| 5 | Programos paleisties rėžimas (MODE registro reikšmė) | - |
| 6 | Kodo segmento dydis | - |
| 7 | Duomenų segmento dydis | - |
| 8 | Steko segmento dydis | - |
| 9 | Rezervuota | 0000 |

Likusią vykdomojo failo dalį sudaro kodo segmento blokas ir duomenų segmento blokas (tik naudinga ir inicijuota informacija). Programos pakrovimo i atmintį schema:



Deskriptoriai

Numatoma, kad procesų deskriptoriai, kaip atskiros duomenų struktūros, bus talpinami Processes ir Resources sąrašuose. Vidiniai vardai – sveikieji skaičiai, procesų – processId, resursų – resourceId.

Proceso deskriptorius – CLASS PRocessDescriptor

|  |  |  |
| --- | --- | --- |
| **Duomenų tipas** | **Deskriptoriaus dalis** | **Paaiškinimas** |
| String | Name | Išorinis proceso vardas |
| CPUState | CPU | Proceso procesoriaus būsena (saugo visų procesoriaus registrų reikšmes) |
| List<Integer> | RAMResources | Naudojami operatyvios atminties resursai |
| List<> | CurrentResources | Naudojami kiti resursai |
| List<Integer> | CreatedResources | Proceso sukurti resursai |
| ProcessState | State | Proceso būsena  { RUNNING, READY, BLOCKED, READYS, BLOCKEDS } |
| List<> | ListIn | Sąrašas, kuriame yra procesas  { ReadyProcessList, WaitingProcessList } |
| Integer | ParentProcess | Tėvinio proceso vidinis vardas |
| List<Integer> | ChildProcesses | Vaikinių procesų vidinių vardų sąrašas |
| Integer | Priority | Proceso prioritetas |
| *BaseProcess* | *SystemProcess* | *Sisteminio proceso objektas*  *Jei == null, tada tai VM proceso desikriptorius.* |

Resurso deskriptorius – CLASS ResourceDescriptor

|  |  |  |
| --- | --- | --- |
| **Duomenų tipas** | **Deskriptoriaus dalis** | **Paaiškinimas** |
| String | Name | Išorinis resurso vardas |
| Boolean | Reusable | Nurodo ar resursas yra pakartotinio naudojimo |
| Integer | Creator | Resurso kūrėjo (proceso) vidinis vardas |
| List<> | Elements | Resurso elementų sąrašas |
| List< > | WaitingProcessList | Resurso laukiančių procesų sąrašas |

Deskriptorių ir primityvų naudojami sąrašų elementų tipai

|  |  |  |
| --- | --- | --- |
| **Sąrašas** | **Sąrašo elemento strūktūra** | **Paaiškinimas** |
| WaitingProcessList | Integer processId | Laukiančio proceso vidinis vardas |
| Integer resourcePart | Laukiančio proceso laukiama resurso dalis |
| Pointer answerAddress | Laukiančio proceso nurodytas adresas atsakymui |
| ReadyProcessList | Integer processId | Pasiruošusio proceso vidinis vardas |
| RamResources | String resourceElement | Turimo atminties puslapio numeris |
| CurrentResources | Integer resourceId | Turimo resurso vidinis vardas |
| Integer resourcePart | Turimo resurso turima dalis |
| String resourceElement |  |
| ChildProcesses | Integer processId | Sukurto proceso vidinis vardas |
| Elements | Integer creatorId | Resurso elementą sukūrusio proceso vidinis vardas |
| Integer resourcePart | Laisva resurso dalis |
| String resourceElement | Resurso elementą apibūdinanti simboliu eilutė |

Primityvai

Sukurti procesą

PROCEDURE CREATEPD( int Name, CPUState cpuState, List<Integer> ramResources, List<ResourceDescriptor> otherResources, int priority)

{

processId = GetNewProcessId();

Processes[processId].Name = Name;

Processes[processId].CPU = cpuState;

Processes[processId].RAMResources = ramResources;

Processes[processId].CurrentResources = otherResources;

Processes[processId].CreatedResources = null;

Processes[processId].Priority = priority;

Processes[processId].State = READYS;

Processes[processId].ListIn = ReadyProcessList;

Processes[processId].Parent = CurrentProcessId;

Processes[processId].ChildProcesses = null;

Include(Processes[CurrentProcessId].ChildProcesses, processId);

Include(ReadyProcessList, processId);

}

Sunaikinti procesą

PROCEDURE DESTROYPD( int processId )

{

Remove(Processes[processId].ListIn, processId);

FOR (EACH child IN Processes[processId].ChildProcesses)

{

DESTROYP(child);

}

FOR (EACH ramPage IN Processes[processId].RAMResources)

{

Include(Resources[RAMRESOURCE].Elements, ramPage);

Remove(Processes[processId].RAMResources, ramPage);

}

FOR (EACH resource IN Processes[processId].CurrentResources)

{

IF (Resources[resource.resourceId].Reusable)

{

Include(Resources[resource.resourceId].Elements,

resource.resourcePart);

}

Remove(Processes[processId].CurrentResources, resource);

}

FOR (EACH createdResource IN Processes[processId].CreatedResources)

{

DESTROYR(creadedResource);

}

}

Sustabdyti procesą

PROCEDURE STOPP( int processId, ProcessDescriptor descCopy )

{

// REIKIA TIKRINTI AR PATS SAVES NESTABDO

processOldState = Processes[processId].State;

IF (processOldState == BLOCKED or processOldState == BLOCKEDS)

{

Processes[processId].State = BLOCKEDS;

}

ELSE

{

Processes[processId].State = READYS;

}

descCopy = COPYOF(Processes[processId]);

IF (processOldState == RUNNING)

{

SCHEDULER();

}

}

Aktyvuoti sustabdytą procesą

PROCEDURE ACTIVATEP( int processId )

{

IF ( Processes[processId].State == READYS )

{

Processes[processId].State = READY;

}

ELSE

{

Processes[processId].State = BLOCKED;

}

IF ( Processes[processId].State == READY )

{

SCHEDULER();

}

}

Keisti proceso prioritetą

PROCEDURE CHANGEPP( int processId , int newPriority )

{

oldPriority = Processes[processId].Priority;

Remove(Processes[processId].ListIn ,processId);

Processes[processId].Priority = newPriority;

Include(Processes[processId].ListIn ,processId);

IF (oldPriority < newPriority and Processes[processId].State == READY)

{

SCHEDULER();

}

}

Sukurti resursą

PROCEDURE CREATERD( int Name, boolean Reusable, List<ResourceElement> Elements, List<Integer> WaitingProcessList, DISTRIBUTOR )

{

resourceId = GetNewResourceId();

Resources[resourceId].Name = Name;

Resources[resourceId].Reusable = Reusable;

Resources[resourceId].Creator = CurrentProcessId;

Resources[resourceId].Elements = Elements;

Resources[resourceId].WaitingProcessList = WaitingProcessList;

Resources[resourceId].DISTRIBUTOR = DISTRIBUTOR ;

}

Naikinti resursą

PROCEDURE DESTROYRD( int resourceId )

{

FOR(EACH waitingProcess IN Resources[resourceId].WaitingProcessList)

{

IF (waitingProcess.State == BLOCKED)

{

waitingProcess.State = READY;

}

ELSE

{

waitingProcess.State = READYS;

}

Include(ReadyProcessList, WaitingProcess);

waitingProcess.ListIn = ReadyProcessList;

waitingProcess.RequestAnswer = null;

Remove(waitingProcessList, resourceId);

}

Remove(Resources,resourceId);

SHEDULER();

}

Prašyti resurso

PROCEDURE REQUESTR( int resourceId, int resourcePart, int\* answerAddress)

{

Include(Resources[resourceId].WaitingProcessList, (currentProcess, resourcePart, answerAddress));

DISTRIBUTOR(resourceID, processesId);

currentProcessIsNotInProcessesId = TRUE;

FOR (EACH processId IN processesId)

{

IF ( processId != currentProcess )

{

Include(ReadyProcessList, processId);

Include(Processes[processId].ListIn, ReadyProcessList);

IF ( Processes[processId].State == BLOCKED )

{

Processes[processId].State = READY;

}

ELSE

{

Processes[processId].State = READYS;

}

}

ELSE

{

currentProcessIsNotInProcessesId = FALSE;

}

}

IF ( currentProcessIsNotInProcessesId )

{

Processes[currentProcess].State = BLOCKED;

Include(Processes[currentProcess].ListIn,

Resources[resourceId].WaitingProcessList);

PROCESSOR = free;

Remove(ReadyProcessList, currentProcess);

}

SHEDULER();

}

Atlaisvinti resursą

PROCEDURE FREER( int resourceId, String resourceElement, int resourcePart)

{

Include(Resources[resourceId].Elements, (resourceElement, resourcePart));

DISTRIBUTOR(resourceId, processesId);

FOR (EACH processId IN processesId)

{

Include(ReadyProcessList, processId);

Processes[processId].ListIn = ReadyProcessList;

IF ( Processes[processId].State == BLOCKEDS )

{

Processes[processId].State = READYS;

}

ELSE

{

Processes[processId].State = READY;

}

}

IF (IsNotEmpty(processesId))

{

SHEDULER();

}

}

Procesoriaus paskirstytojas (planuotojas)

PROCEDURE SCHEDULER()

{

priority = MaxPriority;

found = false;

Remove(ReadyProcessList,currentProcessId);

Include(ReadyProcessList,currentProcessId);

IF (Processes[currentProcessId].State == RUNNING)

{

Processes[currentProcessId].State = READY;

}

WHILE( NOT found and priority >= 0 )

{

FOR(EACH processId IN ReadyProcessList[priority])

{

IF( ReadyProcessList[priority][processId].State == READY )

{

chosenProcess = processId;

found = TRUE;

BREAK;

}

}

priority = priority-1;

}

Processes[chosenProcess].State = RUNNING;

Processes[currentProcessId].State = READY;

Switch( Processes[chosenProcess] , Processes[currentProcessId] );

}

Resursų paskirstytojas

PROCEDURE DISTRIBUTOR ( resourceId , WaitingProcessList )

{

priority = MaxPriority;

WHILE( priority >= 0 and Resources[resourceId].Elements IS NOT EMPTY)

{

IF(WaitingProcessList [priority] IS NOT EMPTY )

FOR( EACH waitingProcess IN WaitingProcessList[priority] )

{

Atvejis **A** arba **B**;

IF(Resources[resourceId].Elements IS EMPTY)

{

BREAK;

}

}

}

priority = priority - 1;

}

}

**A: (resourcePart nurodo konkretų resurso elementą)**

FOR ( EACH element IN Resources[resourceId].Elements )

{

IF ( element.resourcePart == waitingProcess.resourcePart )

{

Include(Processes[waitingProcess.processId].CurrentResources,

(resourceId, element.resourcePart));

Include(waitingProcess.answerAddress, element.resourcePart);

Remove(Resources[resourceId].Elements, element);

BREAK;

}

}

**B: (resourcePart nurodo resurso elementų kiekį)**

IF( Count(Resources[resourceId].Elements) >= waitingProcess.resourcePart )

{

FOR( EACH element IN Resources[resourceId].Elements FROM 1 TO waitingProcess.resourcePart )

{

Include(Processes[waitingProcess.processId].CurrentResources,

(resourceId, element.resourcePart));

Include(waitingProcess.answerAddress, element.resourcePart);

Remove(Resources[resourceId].Elements, element);

}

}

Pagalbinės funkcijos

Ši funkcija naudojama perjungti procesorių nuo vieno proceso prie kito.

PROCEDURE Switch(processId,chosenProcess)

{

SaveCPUState( Processes[processId].CPU );

LoadCPUState( Processes[chosenProcess].CPU );

}

Šios dvi funkcijos pseudokoduose naudojamos elementams įtraukti į sąrašus arba pašalinti iš jų.

Include( <sąrašas> , <elementas> );

Remove(<sąrašas> , <elementas> );

Procesai

Procesų prioritetai

Prioritetai reikalingi operacinės sistemos darbo optimizavimui, kai greitai atliekamiems procesams duodamas didesnis prioritetas negu lėtesniems. Šioje OS išskiriame dvi statinių prioritetų grupes:

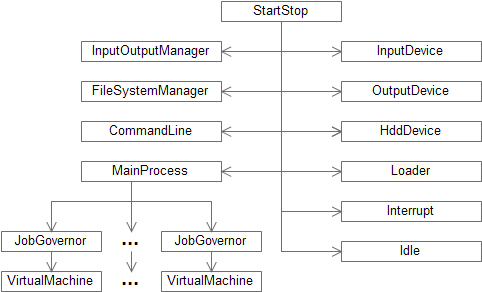
1. Vartotojo procesų prioritetai. Pasirenkami vartotojo paleidžiant programą (pagal nutylėjimą prioritetas yra 2). Skalė: 0-4.
2. Sisteminių procesų prioritetai. Nustatomi kuriant sisteminius procesus StartStop proceso.

Skalė: 5-9.

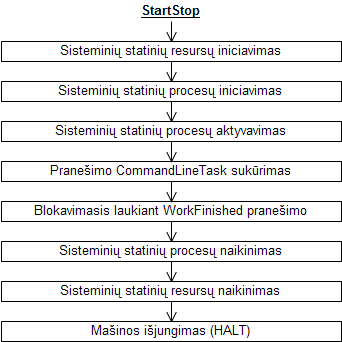
Procesų lentelė

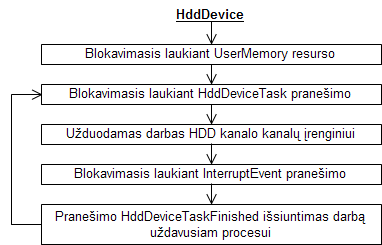
|  |  |  |
| --- | --- | --- |
| **Prioritetas** | **Procesas** | **Aprašymas** |
| 9 | StartStop | Pradinis OS procesas, kuriantis resursus ir kitus procesus |
| 5 | HddDevice | Kietajį diską virtualizuojantis procesas |
| 5 | FileSystemManager | Procesas skirtas darbui su failų sistema |
| 7 | InputDevice | Įvedimo įrenginį virtualizuojantis procesas |
| 8 | OutputDevice | Išvedimo įrenginį virtualizuojantis procesas |
| 7 | InputOutputManager | Procesas skirtas darbui su įvedimo/įšvedimo srautu |
| 7 | CommandLine | Procesas skirtas vartotojo darbui su OS |
| 7 | Loader | Skirtas programos užkrovimui į atmintį |
| 8 | Interrupt | Priima pertraukimų situacijas virtualiojoje mašinoje ir perduoda jas JobGoverner procesui |
| 7 | MainProcess | Procesas kuriantis JobGoverner procesus |
| 7 | JobGovernor | Virtualią mašiną aptarnaujantis procesas |
| 2 | VirtualMachine | Virtualios mašinos procesas (išorinis vardas atitinka programos vykdomojo failo pavadinimą) |
| 0 | VirtualMachine (Idle) | Sistemos paleistas procesas reikalingas kanalų įrenginio pertraukimų (išvedimo/įvedimo/išorinės atminties) fiksavimui, kai nėra kitų VirtualMachine procesų su būsena READY. Proceso programa tieisog vykdo nesibaigiantį ciklą. |

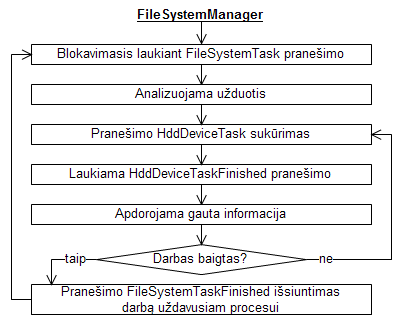
Operacinės sistemos procesų medis

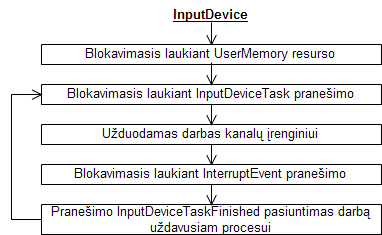


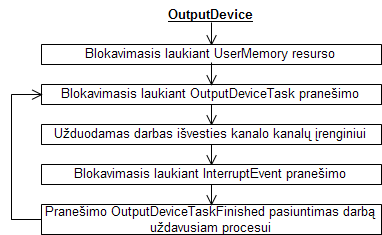
Procesų veiksmų sekų diagramos

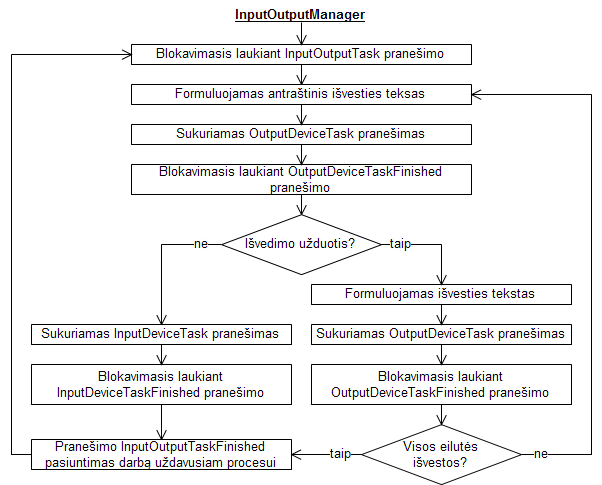


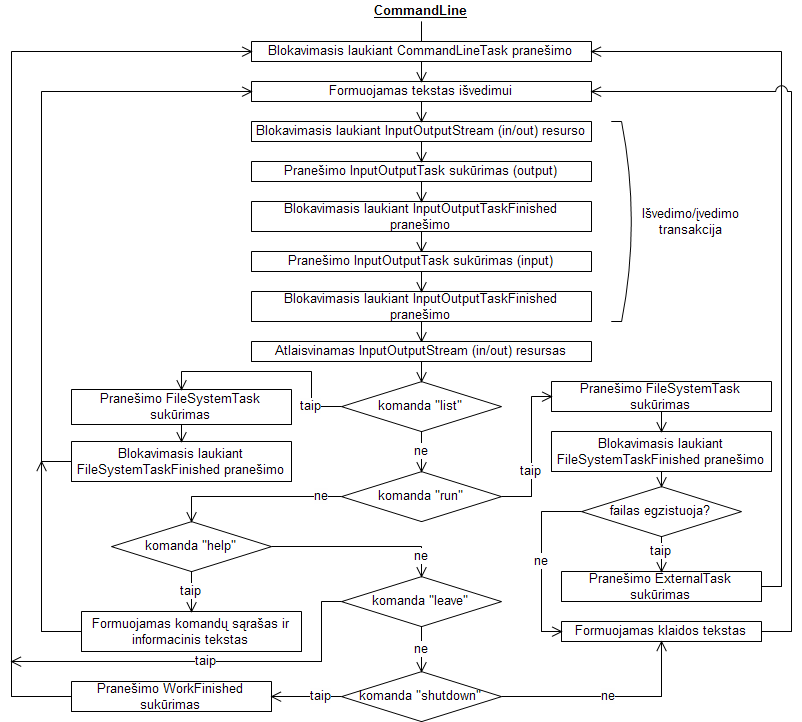


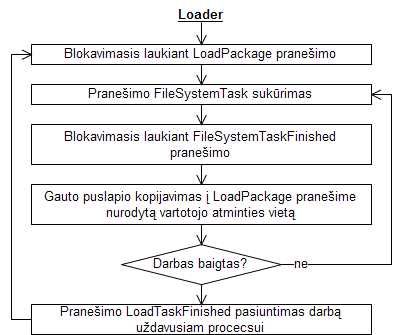


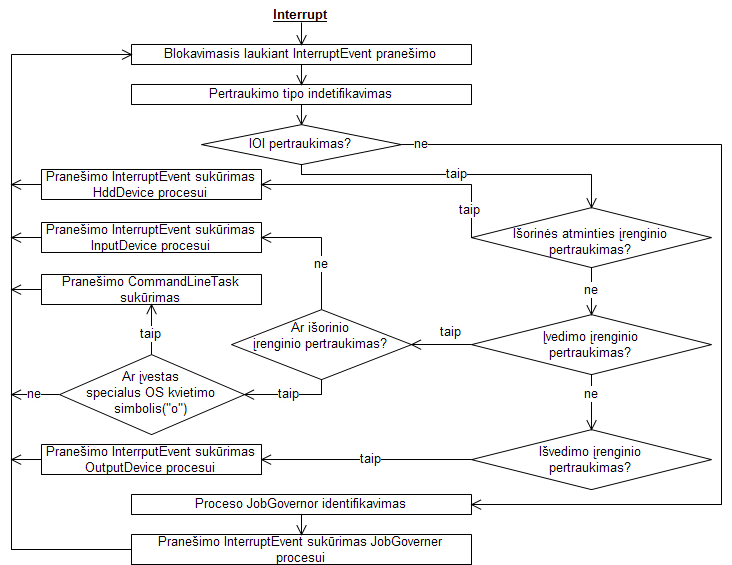


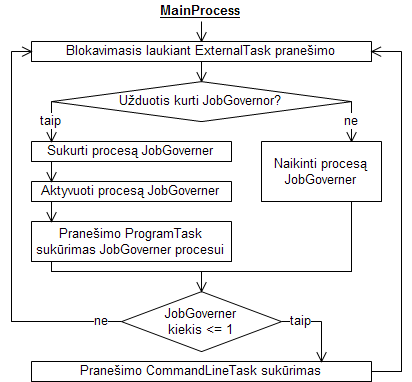


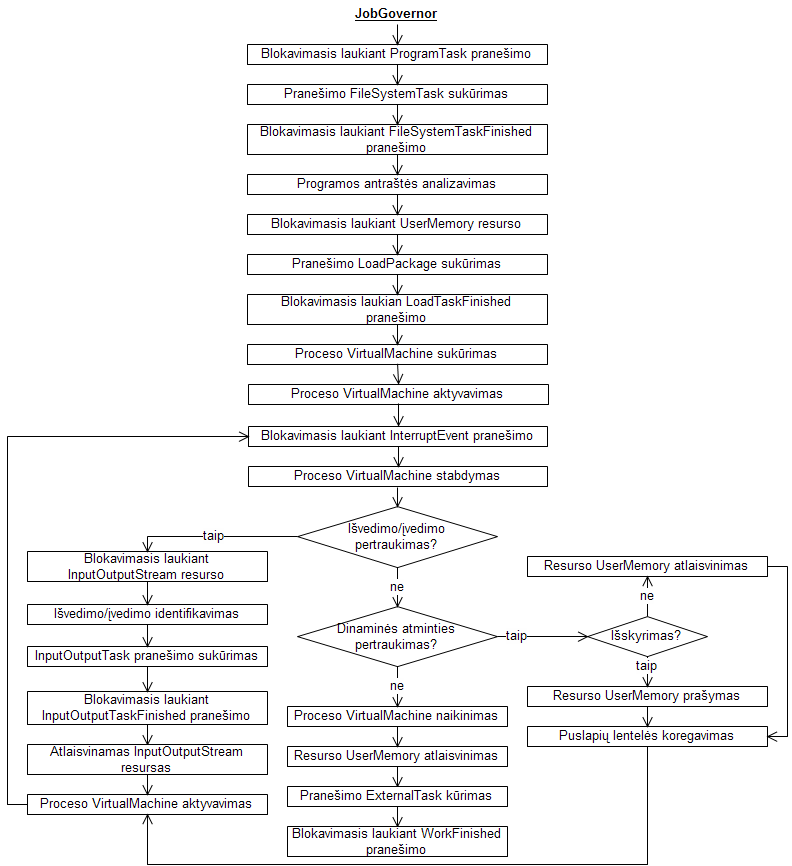




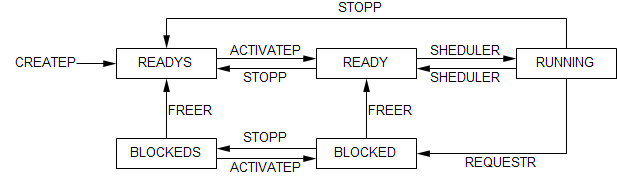




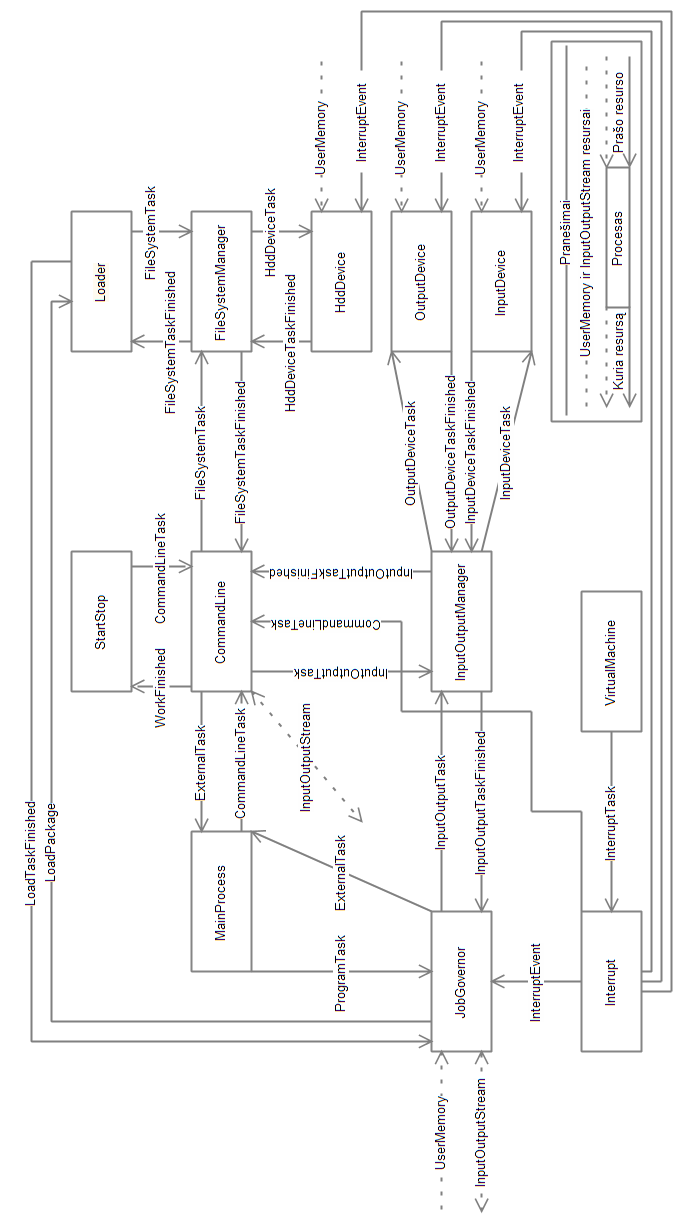




Proceso būsenų diagrama



Procesų tarpusavio ryšių diagrama



Resursai

Žemiau pateiktose lentelėse išvardinti resursai, kurie bus naudojami šioje operacinėje sistemoje.

|  |  |  |  |
| --- | --- | --- | --- |
| **Resursas** | **Aprašymas** | **Gamintojas** | **Vartotojas** |
| UserMemory | Vartotojo atmintis | StartStop | JobGovernor  InputDevice  OutputDevice |
| InputOutputStream | Įvesties/išvesties srautas | StartStop | JobGovernor  CommandLine |
| **Pranešimo resursas** | **Aprašymas** | **Siuntėjas** | **Gavėjas** |
| LoadPackage | Programos pakrovimo paketas – informacija iš kokios kietojo disko vietos, kiek puslapių pakrauti į nurodytą atminties vietą. | JobGovernor | Loader |
| FileSystemTask | Užduotis failų sistema prižiūrinčiam procesui | CommandLine  Loader | FileSystemManager |
| ExternalTask | Programos paleidimo užduotis | CommandLine  JobGovernor | MainProcess |
| CommandLineTask | Užduotis komandinės eilutės procesui | StartStop  Interrupt | CommandLine |
| InputOutputTask | Išvedimo/įvedimo užduotis | JobGovernor  CommandLine | InputOutputManager |
| ProgramTask | Programos pakrovimo užduotis – nurodo kokią programą reikia pakrauti. | MainProcess | JobGovernor |
| HddDeviceTask | Užduotis kietojo disko įrenginiui | FileSystemManager | HddDevice |
| InputDeviceTask | Įvesties įrenginio užduotis | InputOutputManager | InputDevice |
| OutputDeviceTask | Išvesties įrenginio užduotis | InputOutputManger | OutputDevice |
| InterruptTask | Užduotis apie pertraukimą | VirtualMachine | Interrupt |
| InterruptEvent | Pertraukimo įvykis | Interrupt | JobGovernor  HddDevice  InputDevice  OutputDevice |
| **Darbo pabaigos pranešimas** | **Aprašymas** | **Siuntėjas** | **Gavėjas** |
| WorkFinished | Operacinės sistemos darbo pabaigos pranešimas | CommandLine | StartStop |
| FileSystemTaskFinished | FileSystemManager darbo pabaigos pranešimas | FileSystemManager | Loader  CommandLine |
| LoadTaskFinished | Loader darbo pabaigos pranešimas | Loader | JobGovernor |
| InputOutputTaskFinished | InputOutputManager darbo pabaigos pranešimas | InputOutputManager | CommandLine  JobGovernor |
| OutputDeviceTaskFinished | OutputDevice darbo pabaigos pranešimas | OutputDevice | InputOutputManager |
| HddDeviceTaskFinished | HddDevice darbo pabaigos pranešimas | HddDevice | FileSystemManager |
| InputDeviceTaskFinished | InputDevice darbo pabaigos pranešimas | InputDevice | InputOutputManager |

Resursų naudojimą aprašanti lentelė

|  |  |  |
| --- | --- | --- |
| **Resursas** | **ResourcePart (Integer)** | **Element (String)** |
| UserMemory | xyz:  **x** – prašoma atminties dalis  {1 puslapių lentelėms,  2 – programos segmentams}  **yz** – prašomas atminties puslapių skaičius {01-99} | „n1|n2|n3|..|ni“  n -„xyzw“ : i-tojo puslapio adresas operatyviojoje atmintyje.  {0000-9999} |
| **Atlaisvinant resursą UserMemory** | |
| Atlaisvinamo puslapio adresas | “” |
| InputOutputStream | 1 - Input (prašoma įvesties srauto)  2 - Output (prašoma išvesties srauto)  3 - InputOutput (prašoma ir išvesties, ir įvesties srauto)  Procesas, gavęs norimą šio resurso dalį, privalo ją atlaisvinti baigęs darbą. | “” |

Pranešimų resursų įpatybės:

1. ResourcePart reikšmė – proceso kuriam siunčiama (kuris laukia, kuris prašo) vidinis vardas (0 – betkuriam pirmam laukiančiam procesui)
2. Kiekvienas pranešimas tipas – atskiras resurso klasės deskriptorius. Konkretūs pranešimai – to deskriptoriaus elementai.
3. Tai suvartojami resursai, atiduotas elementas sunaikinamas.

Užduočių pranešimai

|  |  |
| --- | --- |
| **Pranešimas** | **Element (String)** |
| LoadPackage | ***„x|p1 p2 p3 .. pn|a1 a2 a3 .. an“*** |
| * x – programos failo pavadinimas kietajame diske * pi – puslapiu lenteles i –tojo puslapio adresas * ai – programos atminties puslapio adresas.   (adreso formatas „xyzw“) |
| FileSystemTask | ***“užduotis|parametrai”*** |
| Užduotys:   * LIST * READFILEPAGE|p|a   {p - failo vardas, a – failo puslapio numeris (0,1,2..)} |
| ExternalTask | ***“užduotis|p|pr”*** |
| Užduotys:   * CREATEJB * DESTROYJB   p - vykdomojo failo pavadinimas (***“***FILE.PRG”)  pr – VirtualMashine proceso prioritetas |
| CommandLineTask | ***“užduotis|parametrai”*** |
| Užduotys:   * ***READTASK (be parametrų)*** * ***READTASKINITIATE(be parametrų)*** |
| InputOutputTask | ***“antraštinisTekstas|užduotis“*** |
| antraštinisTekstas – 20 simbolių tekstas aprašantis I/O operacijos inicijatorių ir/arba tikslą.  Užduotys:   * READLINE * WRITELINE|t   t – tekstas išvedimui |
| ProgramTask | ***“f|pr”*** |
| f : vykdomojo failo vardas  pr – VirtualMachine proceso prioritetas |
| HddDeviceTask | ***“READ|a“*** |
| a – HDD puslapio adresas |
| InputDeviceTask | ***„READ“*** |
| OutputDeviceTask | ***“WRITE|t”***  ***T – tekstas išvedimui*** |
| InterruptTask | ***“i|c|f”*** |
| i : {„IOI“,“PI“,“SI“,“TI“}  c : atitinkamo pertraukimo registro reikšmė  f : proceso VM, kuriame ivyko pertraukimas, tevinis procesas |
| InterruptEvent | ***“i|c”*** |
| i : { „IOI“, „PI“, „SI“ }  c : atitinkamo pertraukimo registro reikšmė |

Darbo pabaigos pranešimai

|  |  |
| --- | --- |
| **Pranešimas** | **Element (String)** |
| WorkFinished | ***“e”*** |
| e : { FAILURE, WORKFINISHED } |
| FileSystemTaskFinished | ***LIST atveju: “f1|f2|..|fn” fi : FILE.TYP – i-tojo failo vardas***  ***READFILEPAGE atveju – nuskaitytas puslapis “aaaabbbbcccc..” – a,b,c .. – eilutės*** |
| LoadTaskFinished | ***“c”***  c : { 0 – užduotis baigta be klaidos, -1 – klaidos kodas ) |
| OutputDeviceTaskFinished | ***“c”***  c : { 0 – užduotis baigta be klaidos, -1 – klaidos kodas ) |
| InputOutputTaskFinished | ***įvedimo atveju – kodas ir įvestas tekstas (t) (iki 40 simbolių)***  ***“c|t” ( įvykus klaidai grąžinamas tik “c” )***  ***Išvedimo atveju:***  ***“c”***  c : { 0 – užduotis baigta be klaidos, -1 – klaidos kodas } |
| HddDeviceTaskFinished | “-1” – klaida  “xyzw” – buferio adresas OA |
| InputDeviceTaskFinished |

Pakartotinio naudojimo resursai

Jei resursas yra pakartotinio naudojimo (reusable = true) tai procesui kvieciant primityvą FREER atlaisvinama resurso dalis tik tuo atvieju, jei procesas tokią dalį turi. Atlaisvinti neturimą dalį gali tik tėvinis resurso procesas.

Virtualios mašinos bendravimas su operacine sistema

Sisteminiai pertraukimai

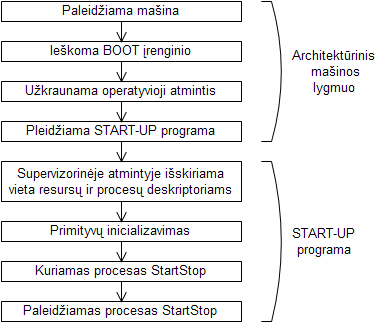
Sisteminiai pertraukimai tai supervizoriniai pertraukimai, kuriuos apdoroja operacinė sistema.

|  |  |  |  |
| --- | --- | --- | --- |
| **SI** | | **R** | **U** |
| **Kodas** | **Pertraukimo tipas** | **Parametrai MOS** | **Atsakymas iš MOS** |
| 0 | Pertraukimo nėra | - | - |
| 1 | Sustabdyti programos veikimą | - | - |
| 10 | Išvesti duomenis | Lokalus puslapio adresas | 0000 – sėkmės atvieju  0001 – klaidos atvieju, galima toliau dirbti  0002 – nesekmes atvieju |
| 11 | Įvesti duomenis | Lokalus puslapio adresas |
| 20 | Gauti puslapį duomenų segmente, kurio numeris nurodytas R registre | DS puslapio adresas (lokali adresų erdvė) |
| 21 | Atlaisvinti duomenų segmento puslapį, kurio numeris nurodytas R registre | DS puslapio adresas (lokali adresų erdvė) |

Dinaminė atmintis

Šioje operacinėje sistemoje vartotojo programos turi galimybę paprašyti papildomos duomenų segmento vietos (neinicijuotos). Nurodyto puslapio išskyrimą ar atlaisvinimą sužadina tam skirti sisteminiai pertraukimai. Įvykus pertraukimui, atminties paskirstymą toliau administruoja JobGovernor procesas.

Operacinės sistemos paleidimas



Procesoriaus taktų paskirstymas

Kiekvienos virtualios mašinos darbo pradžioje, jai išskiriama 10 laiko vienetų (TIMER =10). Kai virtuali mašina gauna procesorių ir laiko vienetai išnaudojami (TIMER=0) , įvyksta TIMER pertraukimas, virtualiai mašinai iš naujo išskiriama 10 laiko vienetų ir kviečiamas SCHEDULER(), kurius perduoda procesorių kitam pasiruošusiam procesui arba, jei tokio nerado, tam pačiam procesui.

Sisteminiai procesai dirba supervizoriaus rėžimu, todėl pertraukimai vykti negali. Dėl šios priežasties aukščiau paminėtas darbo dalinimo scenrijus netinka, todėl procesoriaus darbą sisteminiai procesai pasidalina blokuodamiesi dėl atitinkamų resursų.