

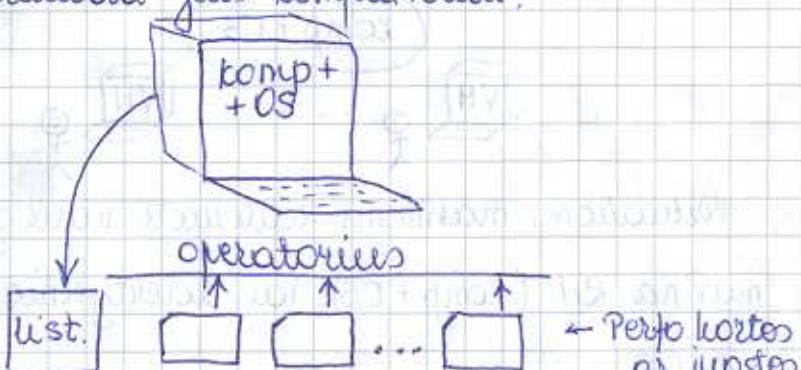
OS (kompiuteris) naidos etapai

- I Žmogus betarpiskai valdo kompiuterį. Darbas vyksta mašinine kalba. Kompiuterio panaudojimas neefektyvus (labai brangi technika). Kompiuterinių resursų nėra.



- II Sprendžiama efektyvumo problema: žmogus nutilinamas nuo kompiutero, paliekant jam kompiuterio teikiamas galimybes.

Programuotojai reformuluoją savo užduotis kaip programų paketus, turinčius perduodami operatoriui, o šis perduoda juos kompiuteriui.



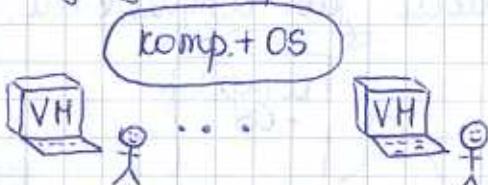
Kompiuteris buvo papildytas OS tam, kad gali-

- ma būti bendrauti su vartotoju. Tokia OS priima vieną paketo užduočių iš gretimo rezultatų (list.) programuotojui. Reikiėjo pateikti visus būtines duos -

menis.

Ši patenčio apdorojimo OS buvo neefektyvi ūmo-
gau darbo atšilgū (pvz.: kai programos buvo
užkavomos ant popieriaus, perduodamos į skai-
čiavimo centro, kur programuotojai suverdavo jas
i kompiuterį, ten kompliuojamas iš esinčių bu-
vo grąžintami ratiusiam programų asmeniui).

III Virtualių mašinų etapas. Tinklas: nurodanti
žmogui kompiuterį (I etapas) naudokant efektyvų technikos
panaudojimą (II etapas). Čia su kompiuteriu
žmogus bendrauja jau suprantama kalba.



Virtualioms mašinoms realizuoti naudojama reali
mašina RH (komp+OS). Tai uoleklyvinio naudojimo
OS.

OS iš II-o iž III-io etapų yra multiprograminės
OS. skirtumas: kai II-me etape ~~buvo~~ persijungiamas
nuo vienos užduoties rykdymo prie kitos, tai pertrau-

kimas žyklita dėl vidinių kompiuterio prietaisų, o

III-me etape pertraukiamas žykluta dėl tos pačios prie-
žasties bei dėl laiko kvanto pasibaigimo.

Jei užduotis sudetinga (pvz. transliacijos), tai
žmogui suprantamai yra pertraukimai. Tampria patogu
dirbti su dideliais resursais.

Šis etapas labai brangus (pastovios temperatū-
ros palaikymas, energijos sąnaudos ir t.t.).

IV Situacija pasikeičia atsiradus PC. Sąjungta prie
I-mo etapo: jokių multiprograminių OS.



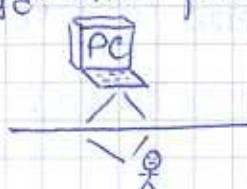
Rimti užduotiniai atliekami kaip ankstiau (III etapa).

o smulkės - kaip I-me etape.

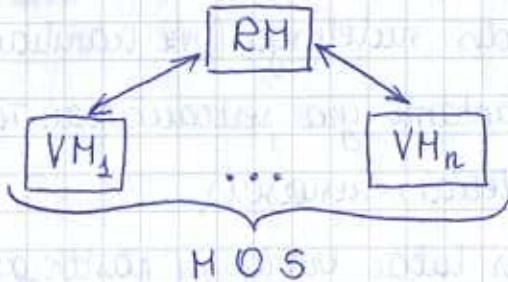
Kartu su PC atejo grafinis interfejsas (buvo alfabe-
tinis skaitmeninis).

Žmogui norėjosi vienu metu daryti daug darbų. Si-

kilo poreikiis turėti keletą virtualių mašinų vienam
žmogui. Tam tiksliui grąžtama prie multiprogrami-
nių OS. Tai sian-
dieninė būsena.



Ap 05 - tai programa, kuri modeliuoja kelių virtualiųjų matinų darbo vieneto realiųjų matinų (atvaizduoja VH į RH).



Reikia nukonstruoti:

- 1) VH apibrėžimą;
- 2) RM apibrėžimą;
- 3) HOS realizaciją, kuri $\{VM\} \rightarrow RH$.

Virtualios matinės VH bei realios

RH apibrėžimai

VH yra tili tai, kas vykdo programę. Jos načiupinėti negalima.

VH skirta naudojama tam, kad galima būtų apibrėžti matinės galinibus.

RH turi tili žemos lygio komandas, todėl joje sunkeu realizuoti HOS. Taiomis palengvinančiomis prielai-

dg: RH turi autito lygio procesorius.

RH turi panašymetį dviem režimais:

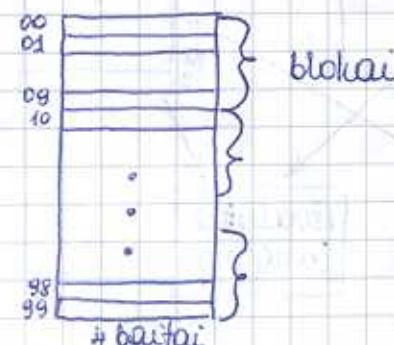
1.) vartotojo,

2.) supervisoriniu,

Jei RH dirba vartotojo režimu, tai ji faktyskai netaupia su VH dalimis.

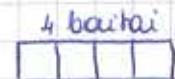
VM specifikacija

Tarkime, yra 100 žodžių atmintis ($0 \div 99$). Ši atmintis yra 4 baitų $\boxed{\quad}$. Žodžiai adreiniuojami nuo 0 iki 99. Žegul atmintis yra suskirstytas blokais po 10 žodžių.



Procesorius turi 3 registrus:

- 1) R - bendrasis registras



- 2) C - loginis trigeris, priima reikiemus true (T) arba false (F), kad būtų atliktas sly-

(5)

ginis valdymo perdavimas



3) IC - komandy suktuvas

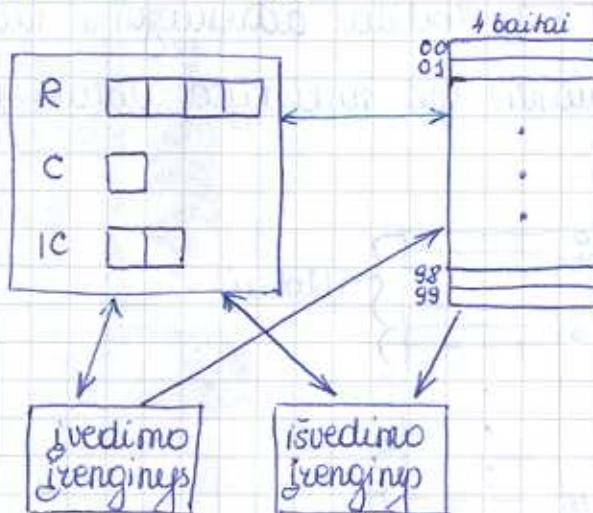
Atminties žodis interpretuojamas kaip komanda

arba duomenys. Operacijos kodas užima 2 vyrinius
baitus, o adresas - 2 jaunesniuosius:

OPK | **adr.** - komandos struktūra.

VH turi nuosekliaus įvedimo bei išvedimo įrenginius.

VH schematiškai:



Y/ iš įrenginių valdomi procesoriaus.

Komandas

AD - sudėties kom. - $x_1 x_2$

$$R := R + [a] \quad a = 10x_1 + x_2 \quad \boxed{AD \ x_1 \ x_2}$$

adresas

$x_1, x_2 \in \{0, \dots, 9\}$

⑥

LR - registro pakrovimas iš atminties - $x_1 x_2 \Rightarrow R := [a]$; $\boxed{LR \ x_1 \ x_2}$

SR - įsimenama registruo reikime - $x_1 x_2 \Rightarrow a := R$; $\boxed{SR \ x_1 \ x_2}$

CR - palyginimo kom. - $x_1 x_2 \Rightarrow$ if $R = [a]$ then $c := 'T'$
else $c := 'F'$ $\boxed{CR \ x_1 \ x_2}$

BT - sąlyginis valdytojus - $x_1 x_2 \Rightarrow$ if $C = 'T'$ then $IC := a$ mo perdavimas $\boxed{BT \ x_1 \ x_2}$

GT - apsiukti mas su išorės vyrsta blokais - $x_1 x_2 \Rightarrow \text{Read}(\lfloor \beta + i \rfloor, i=0, \dots, 9)$, $\boxed{GT \ x_1 \ x_2}$
 $\beta = 10 \cdot x_1$
 x_1 - bloko numeris

PT - išvedami duomenys - $x_1 x_2 \Rightarrow \text{Print}(\lfloor \beta + i \rfloor, i=0, \dots, 9)$ $\boxed{PT \ x_1 \ x_2}$

H - sustojimo kom. \Rightarrow Halt

Programos parasydys įvedami & skaiciuojant. Jei sudedami iš rezultatas išvedamas, tiegu skaiciuai yra max 4 simbolių.

Poz.: 12+3

Atmintis

$x_1 \ x_2$	50	IVES	60	12	40	SUHA
PT	51	KITE	61	3	71	WYRA
GT	52	TAU	62		72	LYG
LR	53	SKAI	63		73	1:
	54	CIUS			74	

AD 61

SR 74

PT 70

H

- $x_1 x_2 \Rightarrow R := [a]$; $\boxed{LR \ x_1 \ x_2}$

$\boxed{SR \ x_1 \ x_2}$

⑦

VH pradedė darbą, kai registrų IC reikiame yra 00 (įgyvendinimo komandos, kuri patalpinta nuliname skaičiuje).

RH specifikacija

Procesorius gali dirbti dviem režimais.

Supervizoriaus režime komandas atliekamos automatiškai, kuris turi priejimus prie VH atminties per pustapiavimo mechanizmą.

RH registrai tokie pat kaip ir VH.

Realaus procesorius registrai daž yra šie:

P1 - programinio pertraukimo registras, signalizuojantis, kad VH-e atrastas neognistojantis OPK ar adresas.

S1 - supervisorinio pertraukimo registras - kai VH negali įgyvendinti reiksmo ratį (jov/iss, pertraukimo). Ji kreipiasi į OS serviso. VH tiki nustato pertraukimo registro. OS tures atlikti tą veilus mgo vietoj VH.

I01 - joved/issved pertraukimo registras - signalizuoj, kad veilumas baigėsi.

T1 - taumperio pertraukimo registras - nustato pertraukimą, kai panekiamas tam tikras reikiemė.

PLR - pustapiavimo registras (4 bautys).

CUST I i J - kanalo stabdymo registras - yra 3 kanalai, kodėl $i=1, 2, 3$.

HODE - langelio reikiemės yra 2:
U - user
S - supervisor

01.09.11

Pustapiavimo registras naudojamas atvaizduoti -

mui tarp VH ir RH.

VH-je queruojama simbolinių formatu, kuris nėra skirtas hiri simbolinių formatų.

RH atmintis (vartotojo atmintis VAT) sudedala į 300 žodžius, kiekvienas iš kurių turi 4 bautus. Ši naudojama VH atminėjai modeliuoti. Kiekvienai VH atminčiai atvaizduota į VAT.

VAT yra suskaidyta į blokus po 10 žodžių. Viso yra 30 blokų (nuo 0 iki 29). Bloko dydis - 40 bautų.

Laukysime, kad isorinę atmintį sudaro tie neginio jejimo isorinių įrenginių su 40-ties bautų filiuoto ilgio frašas. Nieuas frašas atlieka bloke.

Laukysime, kad nenuo bloko apskiechius tarp VAT ir II raijas lygus nenuam laiko menetui.

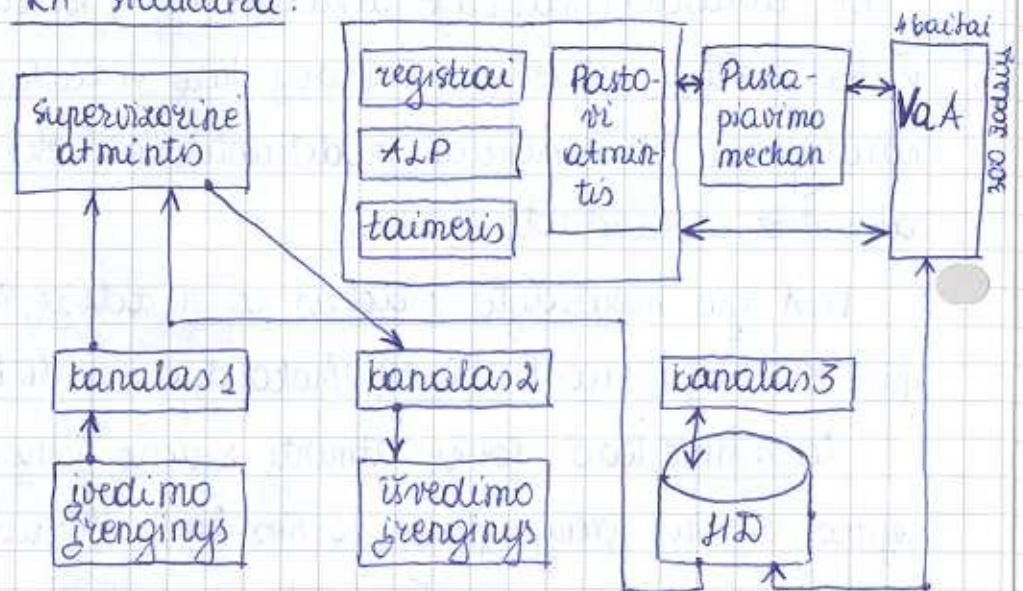
9

Laukymime, kad RM turi vieną nuoseklį gy, kur e
renio skaičymo metu nuskaitomas filiuoto ilgio (40
baitų) žradas, t.y. negaliuose nurodyti individua-
lai suvadamus žrāty dydžius. Špinketinės výksta
blokais.

Taip pat yra 40 baitų ilgio išvedimuo žrängi-
nys.

Špinketinės su jū/iso žränginiuose tinka tris
lauko menetis.

RM struktūra:



Paaškinimai :

ALP gali atlikti C arba Pascal'io programas.

(10)

Pastoviuje atmintyje yra programos, interpretuojan-
čios RM komandas.

PLR - puslapiaus registras $a_0 a_1 a_2 a_3$

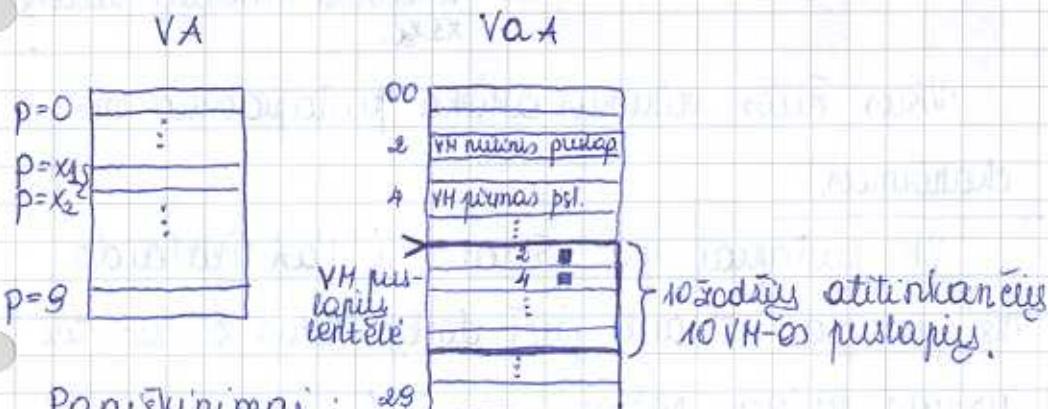
a_1 rodo puslapijų lentelės dydis - 1 (nuo 0 iki 9);
 $a_2 a_3$ rodo VAT bloko numerį, kuriamo yra duo-
tu metu dirbančių VM puslapijų lentelė. Pusta-
pijų lentelės blokų numeriai yra nuo 0 iki 29.

RM durbdama vartotojo rezimu výkdo virtualių prog-
ramų.

Ad $x_1 x_2$ ← virtualus adresas

Virhealy adresų realus atvaizduoti į realų adresų re-
giuje atmintyje. Tačiau naudojame PH'g ir PL'g

Ši situacija schematiskai atrodo taip :



p - puslapis; x_1 - puslapio n2; x_2 - žodžiai puslapuje n2.

(11)

- žodžiuose raugomi bloky numeriai, p. t. VM su pustapiu 3 bloko numeris yra 4.

Pustapiję x₃ mus domina žodis su poslinkiu x₂. VA blokai vadinami pustapais, t. y. turime x₃x₂ atvaizduoti į absolientų adresą atmintyje.

$10 \times a_2 + a_3$ - pustapijų lentelės bloko nr.
Reikia nurodys bloko nr. pereiti iš adreso:

$10 \times (10 \times a_2 + a_3)$ - pustapijų lentelės bloko adresas
Reikia psl. x₃ yra atitinkamai žodis su poslinkiu x₃, kuris yra pustapijų lentelėje

$10 \times (10 \times a_2 + a_3) + x_1$ - pustapio x₃ žodžio adresas
pustapijų lentelėje, yis duoda reikišius:
 $[10 \times (10 \times a_2 + a_3) + x_1]$ - bloko nr., iš kurio atvaizduotas pustapio x₃ Reikia bloko adreso:

$10 \times [10 \times (10 \times a_2 + a_3) + x_1]$ - pustapio x₃ bloko adresas,
jame reikia pažinti žodį x₂:

$10 \times [10 \times (10 \times a_2 + a_3) + x_1] + x_2$ - absolientes adresas, atitinkantis virtualų adresą x₃x₂.

Visus šiuos veikimus atlieka pustapavimo mechanizmas.

VM pustapiai yra užtrauktūs į VAT. Pridėta:
nisi pustapiai užtrauktūs nuo darbo. Kitas būdas: tik pirmas pustapis pabrautas į VAT.

sglyga: $x_1 \leq a_1$ (a_1 - max psl. nr). Jei ji pa-

žeidūsama, tai patiekiama adresacija, todėl reikia generuoti programinį pertraukimą.

Kiekvienas VM komanda atliekama per 3 laiko vienetus (išskyrus jo/iso - ten 3 laiko vienetai).

Jei dirbtant VM kyla pertraukimai, tai jie apdrojanti tiki pabaigus vykdymą VM komandą. Tuo metu RH persijungia iš varotojo režimo į supervisorinių režinių. Pertraukimai gali keleti, kai reikiame atlikimui reikiu os ypatumų.

Processorius darbas

supervisorinių režimių

supervisorinių programų turi praeiti per VAT ir praeiti VM atminties. Supervisorinius režime processorius nėra pertraukiama. Giliusgjani pertraukimai nuo taipmerio arba joed/isoed pertraukimai.

Yra poreikis patirinti, ar nustatyti pertraukimų registrus. Tačiau, RH turi galimybę testuoti pertraukimo registrus.

Pertraukimų registrus gali tiksinti tokios programėlės:

Test (x) { testuoja ne jis/iso pertraukimų registro: }

if x=1 then begin Test:=IOI; IOI:=0 end else

{ programinių pertraukimų registro: }

if x=2 then begin Test:=PI; PI:=0 end else

{ supervisorinių pertraukimų registro: }

if x=3 then begin Test:=SI; SI:=0 end else

{ taimerio pertraukimo registro: }

if x=4 then begin Test:=TI; TI:=0 end else

{ jei bent vienas nustatyta: }

if (IOI+PI+SI+TI)>0 then Test:=1 else Test:=0;

Atlikus eilinės komandos interpretavimo veikia pertraukinti, ar nenustatytas pertraukimas (bent vienas registras). Jei taip, tai pernijungiamą į pertraukimų apdorimo programą.

Ived/isived reikiemis atliekamas supervisorinius reikiemus. Tai skirta ived/isived reikiemo iniciavimo operacija.

start IO(ch, s, D, n), kur ch - kanalo, per kuri yra išleista ived/isived reikiemus, nr; s - iš kur paimiti; D - kur padėti; n - kiek blokų dalyvauja apnikeitimine.

S yra blokų manytas (sraitas), kurie dalyvauja.

D - blokų sraitas, kur bus rezultatas.

Jei operacija start IO užduodama užimtumai kanalui, tai turi būti laukimas, kai kanalas atsišauks. Kanalo užimtumas likrenamas taip:

$$CHST[i] = \begin{cases} 1 & \text{- užimtas,} \\ 0 & \text{- laisva.} \end{cases}$$

Tau, kad galima būty pernijungti į varotojo reikiemą, reikiu pabrести būseną, pagal kurį galime pratęsti VH darbą.

slave (ptz, c, z, rc), kur ptz - puslapio lentelės registro reikiime; c - loginio registro reikiime; z - komandy shaitluklo reikiime.

Laikome, kad supervisorius reikiemus komandos išykdamos per nulinį lauko menetį (t.y. laikas neskaičiuojamas).

Jei kanalui iškurečiamai komanda start IO, tai turi būti nustatyta $CHST[i]=1$. IO veiksmas ylus- ta sygiagrečiai su centrinio procesoriaus darbu. Atlikus šį veiksmą nustatomo $CHST[i]=0$. Tada sun- ciamas signalas, kad reikub pertraukimo.

Tai meris. Tai merio aparatura matina specialaus supervisorinės atminties šodžio turinj menetis klas 10 centrinio procesoriaus laiko menetys. Tas specialusis šodis yra TH. Kai $TH=0$, tai filmojamas pertraukimas nuo taimerio. Šodžio TH reikiame gali būti nustatoma ir patikrinama supervisorinius režime.

Pertraukimų aparatas. Pertraukimai gali būti:

- 1) programinių (parzystas OPK arba adresacija);
- 2) supervisorinius (GT, PT, H);
- 3) įvedimo / išvedimo (baigus vias operacijas);
- 4) taimerio (kai $TH=0$).

Pirmo ir antrio tipo pertraukimai lygiai dirbant tik varstotojo režimu, o trečio ir ketvirtos tipo pertraukimai gali kiti tiek dirbant varstotojo, tiek supervisorinius režimais.

Pertraukinys registru reikiemis:

$$PI = \begin{cases} 1 - \text{atminties apsauga} \\ 2 - \text{operacijos kodas} \end{cases}$$

$$SI = \begin{cases} 1 - GT \\ 2 - PT \\ 3 - H \end{cases}$$

$$IOI = \begin{cases} 1 & \text{pertraukimas I-me kanale} \\ 2 & \text{--- " --- II-me --- " ---} \\ 3 & \text{--- " --- III-me --- " ---} \end{cases}$$

Reikiemis parinkties tokius būdus taur, kad galima sukaupti pertraukimus nuo kelių įrenginių juos sumujant.

Kilus pertraukimui dirbant varstotojo režimu perduodama tokia semantika:

$$\begin{aligned} C &= C \\ Z &= R \\ IC &= IC \\ PLZ &= PLR \end{aligned}$$

Procedurių registrai jin menami supervisoriniuje atmintyje. Kilus pertraukimui nustatomas registras $H0SE = 'S'$ ir procedurių perjungiamas į supervisorinius režimus, semantiškai naudojama procedūra:

Σprograminio pertraukimo apdorojimo programos;

if $PI \neq 0$ then goto PIoint else

{ įved / išved pertr. apdor. progs; }

if $IOI \neq 0$ then goto IOint else

if $SI \neq 0$ then goto SIpoint else

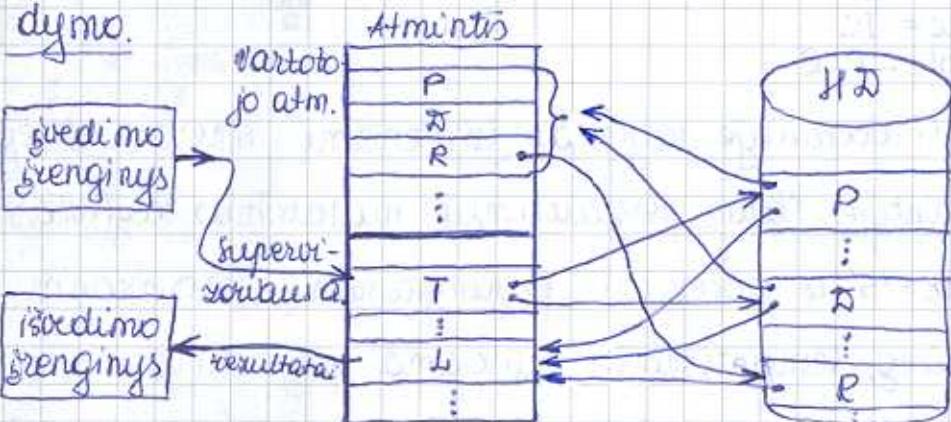
goto TIint;

Pertraukimai (kaip aparatoriinius signalus) reikiškes

transformuoti į resursus.

OS-ai turi būti pateiktas užduoties reikiškės. Tam reikalinga specifinė užduoties pateikimo klasa. Užduoties reikiškės paduodamas per renginį. Kiekviena užduotis informuojama kaip atskiras failas.

Užduoties keliai OS-je nuo jos pateikimo iki įvykdymo.



Užduotis suideda iš duomenų ir rezultatų. Kai vyksta užduoties atpažinimas, tai ji suskaudoma į dailis. T - užduotis, atliekama intelekto kontrole, sau-goma išorineje atmintyje (HDD), kai ji jau paruošta vykdymui. P - programa, D - duomenys, R - rezul-tatai. L - listingo resursas.

(18)

užduotims atiduodami ne fizinių, o virtuelius frenginius.

Kai užduotis įvykdyla, o galbūt niko pertraukimai iš procesorių vardo litaras užduotis, tai infor-macija apie užduotę kaupiama „listingo“ re-surse.

Procesai

01.09.18

Procesas **start_stop** atsakingas už korektūs darbo pradžius ir pabaigas.

Tikslos - surinkti statinius HOS proceso virš resursus

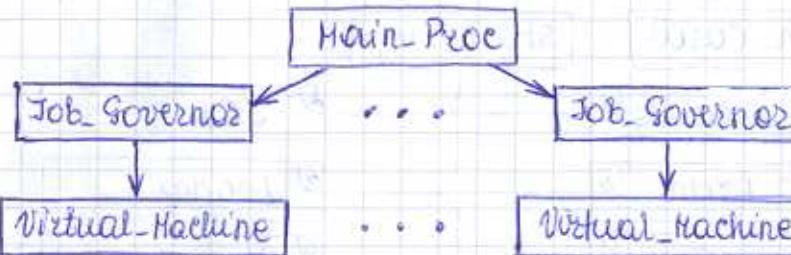
start_stop
Read_in_Cards ①
JCL ②
Job_to_Drum ③
Loader ④
Lines_from_Drum ⑤
Print_Lines ⑥
Get_Put_Data ⑦
Chan3_Device ⑧
Interrupt ⑨
Main_Proc ⑩

- ① atsakingas už betarpistko užduoties įvedimą
- ② analizuoja užduoties struktūrą - užduoties atpažinimas (sintaksė, duomenys,...)
- ③ atpažintas užduotis kaupia išorineje atmintyje
- ④ paruoštas vykdymui užduotij pakeičia $\text{L} \rightarrow \text{O}'$
- ⑤ vykdymas užduotis ir rezultatai kaupiamai išorinė - ⑯

- je atnaujinyje. Tai ju paruošimo išvediminių procesas;
- ⑥ fizinių atlikūs išvedimas;
 - ⑦ užduoties vykdymo metu gali prieikti I/O veikimų, kurių atrankos yra procesas;
 - ⑧ darbo su tiekloginio priejimo prie išorinių įrenginių procesas. Tai trečiojo kanalo aptarnavimo procesas;
 - ⑨ dirbtinių varstotojo rezilių ir vykdant programas gali būti pertraukiamai, gyvų signalai turi būti transformuojami į renesus. Tačiau tai yra procesas;
 - ⑩ užduotyų atlikimo valdymo sisteminis procesas.

Prosesas Main-Proc

Kiekvienai užduotimi užduotiniai turi būti suburtos specialus procesas. Main-Proc suburia užduoties prižiūrėjimo procesą Job Governor. Šis procesas yra tiek, tiek atliekama užduotyų:



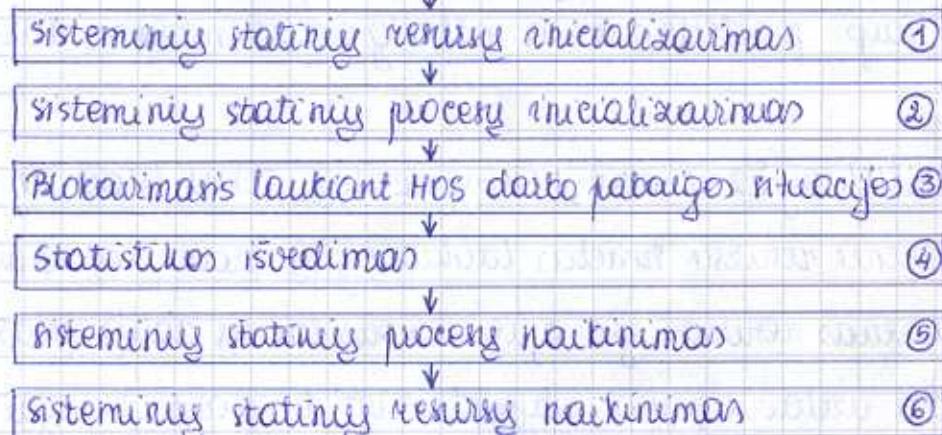
Turime pastovius sisteminius procesus, kurie įvyksta sistemos darbo laiką, t.y. jie sukūriami sistemos darbo pradžioje, o sunaikinami pabaigoje. Išimtis - Job_Governor. Jis sukūramas prieš užduoties vykdymą, o sunaikinamas jei pabaigus vykdymą.

Virtual Machine - VH vykdymo procesas.

Prosesų būsenos: pasruošęs, vykdomas, blokuotas.

Detalesnis procesų f-jų nagenėjimas

Start_Stop



Panaikinimai:

- ① tiek procesai, tiek resursai yra atstovaujami deskriptoriais. InicIALIZUOTE resursų reiskia sukurti deskriptory. Darbas su deskriptoriais galimas tik per specialias operacijas - OS bendruoliui priimtyvios: "sukurti resursą", "Nelikuoti resursą", "Prānti resursą", "Atnaujinti resursą";
- ② yra peniu "priimtyvai darbu" su procesais: "sukurti", "Nelikuoti", "Atnaujinti", "Leisti", "Stabdinti". Stabdymo veikmas dėlėja situaciją, kuriąje procesas nustabdomas del išorinių prievacijų. Blokavimas - tai susirojimas del išorinių prievacijų. Tiek jie nuo priimtyvai atliekami reikiama su deskriptoriais. ②-as procesas registruoja resursą ar procesą deskriptoriuk. Priimtyvai reikia tai apibūdinti parametrais;
- ③ toliau procesui start_stop darbas atsirodo sistemoje kaip darbas, o tai yra blokavimasis (laukimas) sistemos darbo pabaigoje bus sukurtas resursas, kurio laukė procesas start_stop. Tada jis atriblokuotas

ir baigs savo darbą;

- ④ dažas užduoties siūtės jau išvystytas ir sistema bai-
gę darbą Procesai atnabuvęja. statistikos išvedimas;
- ⑤ veiksmas atliekamus kreiplamųjų priimtyvų „finai-
kinti procesus“, t.y. sunaikinamas dekryptorius;
- ⑥ kreiplamųjų priimtyvų „Naujinti resursus“.

Taip pradedamas ir užbaigiamas sistemos dar-
bas.

Procesai laukia resursų, atnaujanties. Tokiu būdu su
liekančiu resursu susietas laukiančiųjų procesų srautai.
Liekančias resursas yra apibūdinamas laisvy dalyis srautų.
Tada uždarinys yra nelyginis tuo du srautės iš pagal
marketing strategiją suskirstytų resursų tarp procesų.

Procesoriaus resursas yra ypatingas resursas. Pa-
nurojus procesai yra tie, kurie laukia procesoriaus. Nuklo-
damas procesas – turi turi procesorių. Užsiblokavęs pro-
cessas negali prieinduti į procesorių.

Kuo met bus pasiekstyti resursai, procesai išeis
iš blokuimo būsenų. Panuorėsių procesų srautai bū-
tė, kurie laukia procesoriaus. Liekančias procesas
bet kuriamo laiko momentu yra bent nenuame iš sqa-
raus.

Laikome, kad turime paprastiausią OS atvejį, kai
užduotis paketas ateina per vieną žvedimo srautą:

Job eilutė
programa
Data eilutė
duomenys
End Job eilutė

Užduoties struktūra

Užduoties autrautes eilutes identifikacija:

\$ HPJ

|||||

uzduoties max ryk- max isvedamų
pavadin. dybos laikos eilutės skaičių

Duomenys eilutė:

STA

Pabaigos eilutė:

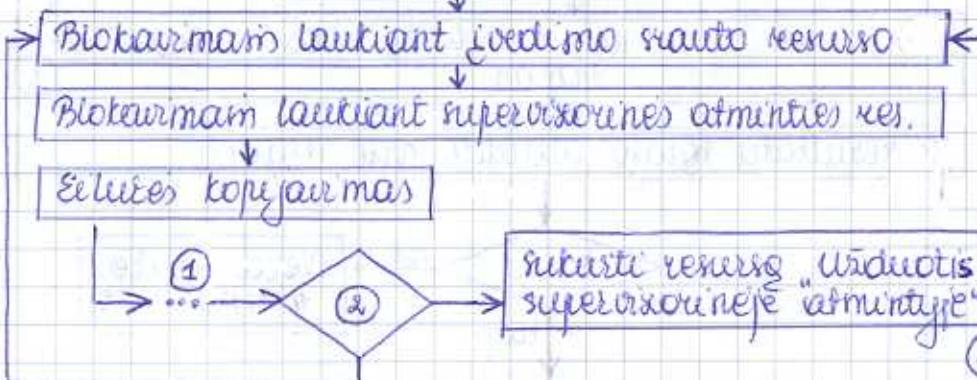
\$ ENA

|||||

pakartotas užduoties vardas

Žvedimo srauto moteras

Read_in_Cards



- ivedimo nauti reikia atpažinti užduotį. Pradedame užduoties atpažinimo proceso.
- visi proceso programos (isskaius "start_stop") yra eilės. Giličias výkdomas tai, kai yra renės.

Read_in_cards paskirtis: paimti eilutes iš įvedimo nauto, paimiti renės iš supervisorinės atminties ir perkelti eilutes į supervisorinę atmintį. Perduo nauti eilutes ir formuoti užduotis jas ius JCL ir analizuoti, ar jos teisingos.

Į įvedimo nautu sunyss tiki proceso Read_in_cards (čia apie bėdą iš 19 psl.), todėl specifiniai resursai reikalingi tiki tauri, kad nebūtų rūpesčių su įvedimo greiginiu.

Užduoties struktūros (sintakės) patikrinimas

JCL

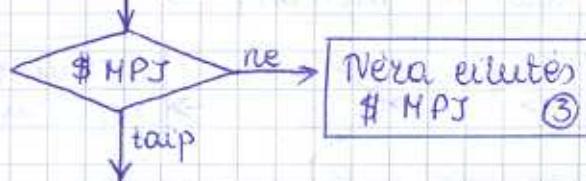
Blokavimais laukiant „Užduoties supervisorinėje atmintyje“ renės

①

Programos, duomenys, rezultatai išraišgė initializuojamas

②

Y rezultatai išraišgė šaukti viso užduotis



24



Padaiktinimai:

- procesas blokuojanči laudamas „Užduoties supervisorinėje atmintyje“ renės;
- slėnugė atiblokuoja ir analizuoja, ar užduotis teisinga;
- jei užduotis klaidinga, tai formuoja rezultatus ištraukiamas tokis problemos. Iniciuojami sėsai, kai jis pagrindu būtų nescių renės.

Resursai

Užduoties programos renės

Užduoties duomenys renės

Užduoties parametrai renės

Užduoties rezultatai renės

} renės
supervisorinėje
atmintyje

Iš renės reikalingas proceso Job_to_Drum.

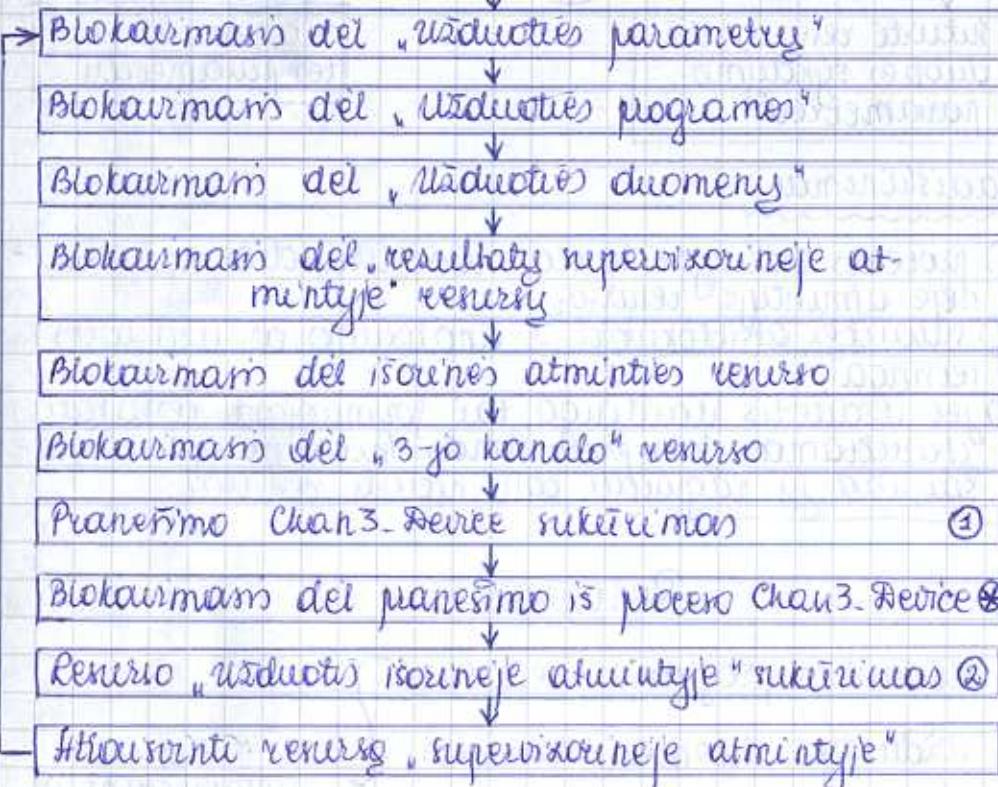
Jei užduoties parametrai klaidingi, tai naudojamas renės „dištingan supervisorinėje atmintyje“.

Iš renės JCL. Jei parametrai teisingi, tai išstinguo nukuria Lines_from_Drum. Jei ims Print_Lines

25

vi gales atlikti fizinių eilutes išvedimui.

Job-to-Drum



- ① tokios informacijos patiekime kaip planėjimą, o tai jau naujas resurs tarpame būs parakyta, iš kur paimti ir kur padetis? Šiuo laukio kol Chan3. Device atlikis perėjimo veikimo ir planės apie tai, ② tada reikia "užduotis isorineje atmintyje" sukurimui.
Resurs "isorineje atmintyje" laukio procesas loader.

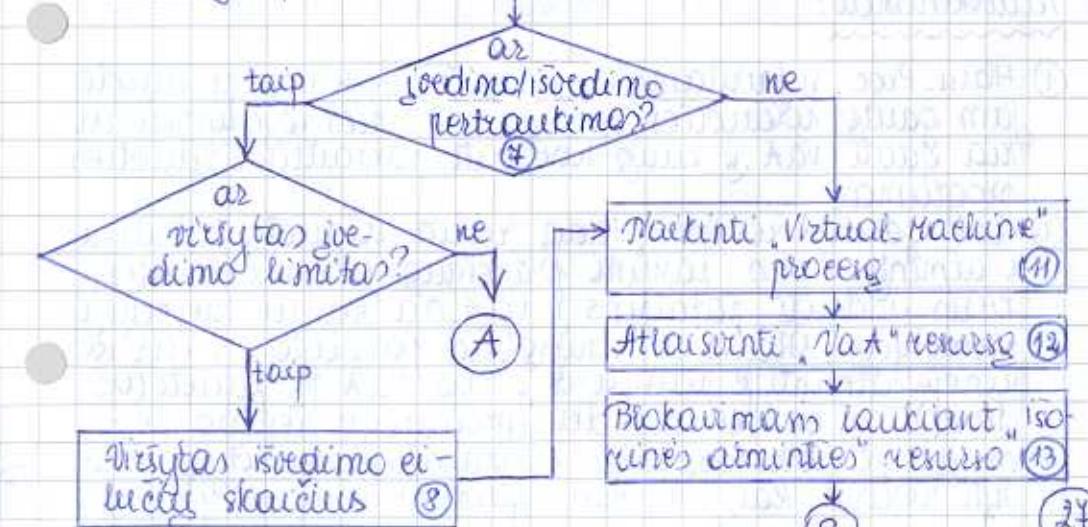
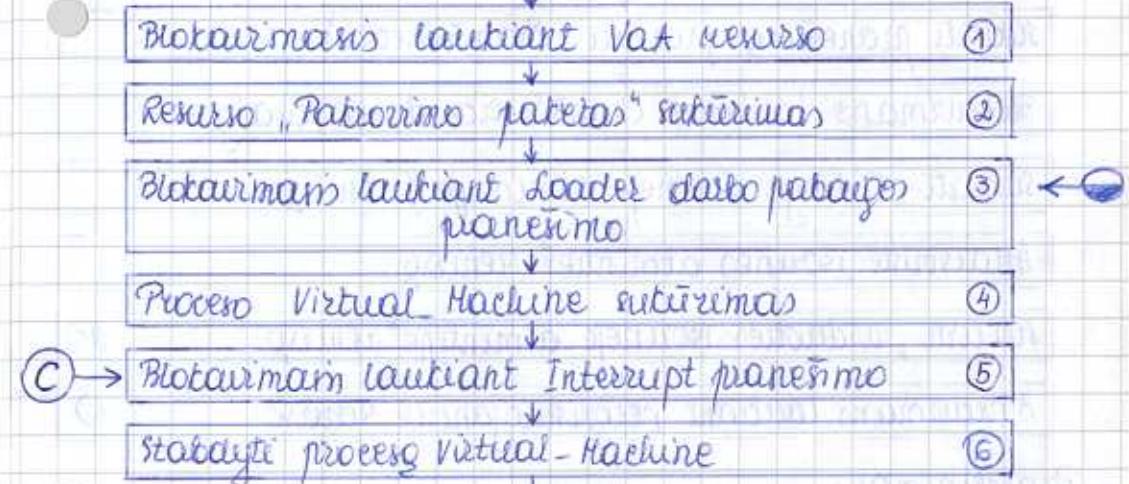
Procesas Main_Proc, negaudamas resurs "užduotis isorineje atmintyje", suburia proceso Job_Governor.

Main_Proc



2001 09 25

Job_Governor



A

•→ sukurti planėjimo proceso „Get_Put_Data“

↓
Biokarmam laukiant planėjimo iš proceso „Get_Put_Data“

↓
Sukurti proceso „Virtual_Machine“

C

B

Biokarmam laukiant 3jo kanalo "resursu"

↓
Sukurti planėjimo protokoli „Chan3_Device“

↓
Biokarmas laukiant Chan3_Device darbo pabaigos

↓
Sukurti „listingo išsineje atmintyje“ "resursu"

↓
Atlaivinti „išsineje atminties“ resursu

↓
Sukurti „išduoti išsineje atmintyje“ "resursu"

↓
Biokarmam laukiant neegzistuojančio resursu

Paauksinimai:

① Main_Proc, sukurdamas proceso Job_Governor, perduoda jam gauti išduoties resursų. Išduoties vykdymui reikia gauti VAT, ir kuriu turi būti pakrauta išduoties programa.

② kai gautas resursas, tada reikia išduoti pakrauti į atmintį. Reikia pažiūti planėjus proceso Loader, kuris išduotis pataipins į VAT. Šiai funkcijai kreipiniųsi numitys „Atlaivinti resursu“, ad pasakyta, iš kur išsineje atmintyje pažinti ir į kur VAT jis padeti (išduoti). Tai ne resurso deskriptorius, o resurso elemento pakrötmas pagal OS branduolio numitys. Pažiūti resursu. Kai jau nėra užgami resursai, tai jie greg

sunami numityre „Atlaivinti resursu“.

③ bus nėra užklausos manėjimas, kad programa pakrauta į VAT;
④ išleime kreipinių į OS branduolio numitys „Sukurti proceso“. Job_Governor toliau nebėturi jis daryti. Jis išveikis, kai yra dant virtual_Machine programą įvykis ypatingo situacijos: kai reikės programinio inkstomo. Toki aparatūrinis pertraukimo nugalas turi issausti resursu, kuriuo laukia OS, suturmingo iškurejimo pertraukimo androjimo programa. Jos darbo pasirojė turi būti sukurta, ekrinas, į kuri reaguoja OS. Job_Governor laukia planėjimo apie pertraukimo situaciją;

⑤ atsibėdavęs, Job_Governor turi išnabdyti proceso Virtual_Machine darbą, nes kai jis yra Interrupt ir kai dirba pertraukimų androjimo programa, proceso aplinka nesikeičia - toliau dirbama Virtual_Machine aplinkoje. Kai pertraukimų androjimo programa dienanti Virtual_Machine aplinkoje, kreipiančių į OS branduolio numitys su tokiu sukurti proceso, atrankanda momentas, kai jis jungia proceso resursų paskirtystojas. Procesas „renuru paskirtystojas“ skirsto resursus proceso pagal prioritetus. Procesas Virtual_Machine yra žemesnio prioritetė, nei kiti sisteminiai procesai. Kai jis yra proceso Virtual_Machine, buvo išskirtas kažkoks sisteminis proceso. Tai Job_Governor gaus proceso iš jo sustabdys proceso Virtual_Machine;

⑥ tipinė situacija: dėl ivedimo/išvedimo, jis stabdomas tol, kol bus atlikti veikimai, po kurius proceso Virtual_Machine tgs darbo;

⑦ gaujama informacija apie pertraukimą;
⑧ sukturiamas tolis priešnamas jis praromas salio rezultaty išduoties vykdymo reikia nutraukti. Einaus į saką, kur naikinamas šis proceso;

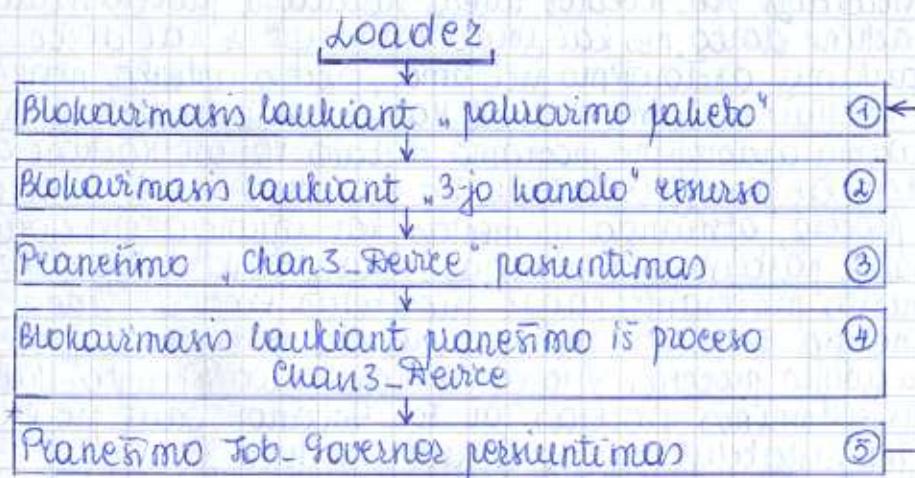
⑨ suaukus iš pradžinio, Job_Governor aktyvuoja proceso Virtual_Machine. Galima tgs kiti komandy vykdymo;

⑩ Job_Governor laukia planėjimo apie pertraukimą, todėl sunardo eiklas grįztamo į ⑤;

⑪ gali būti jau išjeldinta Virtual_Machine programa. Tai supervizorių ar sistemos pertraukimas. Išduoties vykdymo nutraukimas iš Virtual_Machine proceso naikinimas;

⑫ sunaišinamas deskriptorius, atstovaujantis į proceso;

- (13) reikiu pažinti kachopius rezultatus apie užduoties vykdymą, tam reikiu išorinės atminties rezūso.
- (14) sudėtis iš išorinės atminties skyta per 3-jį kanalo? Panaujodant primityvų sukurti planėtus apie Chan3-Derice algoritminumą;
- (15) kiekia apie daubo pabaigo pasalyti procesei Main-Proc. sukurti filtry, užduoties išorinėje atmintyje, rezūso, kuriuo laukia Main-Proc.
- (16) laukiant, kuo Main-Proc nusailins šį proceso, o nusailinti užblokuojant proceso galima.

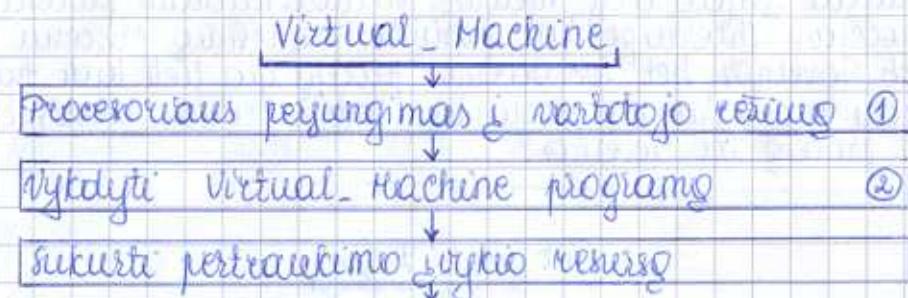


Padaukinių mai:

- ① tai apskaičiuotas su išorine atmintimi per 3-į kanalą;
- ② apskaičiuotas tarp išorės atminties ir VaL;
- ③ per primityvą „Algoritminu rezūsus“;
- ④ padaryta rėštas, ką loader turi padaryti Reikiu paustyti planėtus laukiančiujam procesui Job-Governor (24ps. ③), tur
- ⑤ planėmo rezūso pasiskirstytos gali atlikti rezūso pasiskirstymą laukiantiems procesams.

Loader darbas baigtas. Šis proceso laukia naujo pakrovimo proceso.

Job-Governor sukuria proceso Virtual-Machine.



Padaukinių mai:

- ① RH ar procesoriaus perjungimas iš varčotojo rezūmo kartu nustatoma būsenė pagal IC, registrus, nustatytus lentelės registrus.
- ② po kiekvienos gyvykdyties programos titulinamo ar nenustatytas pertraukumas. Nejamas iš nes programos gyvykta per pertraukimo apdorojimą.
- ③ eis faktiskai ydo darbo pertraukimui.

Muo aparatinis skylos pertraukia prie OS reakcijos.
Kai atliekami veiksmai su descriptoriais, gausame situaciją, kada tiltingo iš naujo iškirsti planuotojo proceso.

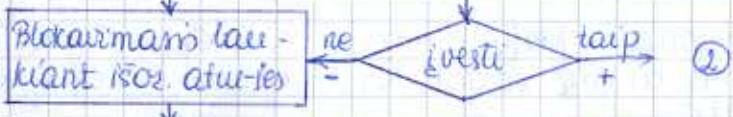


Paaiskinimai:

- 1 laukia žygio, kuri surūpina Virtual Machine procesas;
- 2 procesas Interrupt turi parasyti pranešimą procesui Job Governor. Bet Job Governor procesus yra tiek, kiek tuo metu yra naikinomų programų, todėl reikia identifikuoti procesą Job Governor.

Get_Put_Data,

Blokavimam laukiant panašimo iš Job Governor ①



Blokavimam laukiant „3-jo kanalo“ resurso

Proceso Chan3_Device sukūrimas

Blokavimam laukiant panašimo iš Chan3_Device

Panašimo procesui Job Governor sukūrimas

Paaiskinimai:

- 1 kreipinys į OS branduolio priemityg „Pražyti resurso“. Jis sukuriamas 28psi, kur ① →;
- 2 išsineje atmintyje modeliuojamas virtualus yieduino/ isvedimo įrengings. Jis sukuriama tiek, kiek yra užduočių.

Chan3_Device,

Blokavimam laukiant panašimo ①

Kopijuoti nurodyta blokų (takelių) rinkinį

Parasyti pranešimą laukiant

- 1 iš kartuolio proceso, kies jau darbo duoda yra išresurcių.

Lines_from_Device,

Blokavimam laukiant resurso „listingo išorėje“ ①

Imamas takelis (arba blokas) išsineje atmintyje

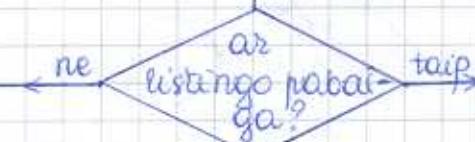
Blokavimam laukiant „superiūnės atė“ resurso ②

Blokavimam laukiant „3-jo kanalo“ resurso

Parasyti pranešimą procesui Chan3_Device

Blokavimam laukiant proceso Chan3_Device pabaiga

„Eilutė superiūnėje atmintyje“ resursas



Paaiskinimai:

- 1 užduoties galutiniuai išlydymo rezultatai, forminami kaip listingo resurso išsineje atmintyje;
- 2 tam, kad būtų kur padėti eilutę listingo.

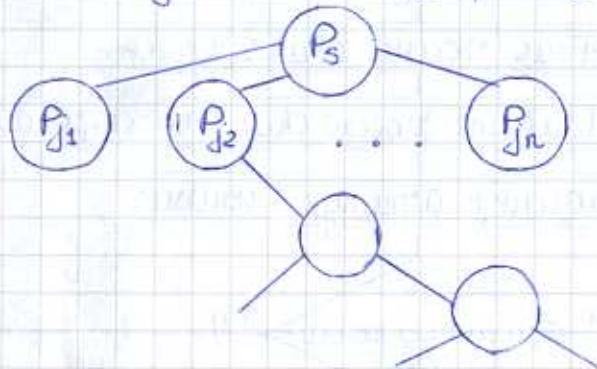
Procesų ir resursų valdymas

2001.10.09

Hos je kiekvienai naudotojui užduočiai j yra sukureti sisteminis procesas p, kuris organizuoja proceso

vykdymo. Jis taip pat užduoties resursų aprašymo. Ši resursų aprašymas nurodada iš statinės ir dinaminės informacijos. Statinė informaciją gali susideti iš prieš užduoties vykdymo žinomos parametres. Dinaminė informacija aprašo duotu metu naudojamus resursus dalis ir užduoties vykdymui reikalingus procesus.

Bendru atveju turime tolį procesų medžį:



Medžio kinta sistemos darbo eigoje.

Resursų klasei, resursų rūšiui - toliese aprašymai vadinančių proceso descriptorių. P_D sukuriamas darbo metu. Procesų ir resursų asjėje atstovauja P_D. Ta naudojami procesų ir resursų operacijose iš PMO.

A dalyvaujā:

- 1) operacijose su procesais (sistemos branduolio priimtynai);

2) PAP_E'se;

3) procesų vykdymuo planarie.

Prosesų sistemoje identifikuojama proceso ardinis vardas. Sukuriant procesą, pati sistema suteikia VI, t.y. suteikia jam vidinį numerį i.

P_D - tai tam tikra struktūra (ne masyvas). Jei kalkulome apie vieną proceso descriptorius, tai turime struktūrą masyvą, kur i - proceso VI - nurodytis struktūros numeris masyve.

P_D nurodada iš komponentų (jouis priskiriamie vardu):

[1] Id[i] - proceso išorinis vardas, reikiškintas statiniams ryšiams tarp proceso nurodyti;

[2] Matrica - turime omeny proceso vykdantį proceso rūšius apibūdinimą;

[2.1] CPU[i] - apibūdina centrinio procesoriaus būseną vykdant proceso. Kai proceso vykdymas nutraukiamas, tai eiuramoji būsena išraugoma;

[2.2] PE[i] - siame OS branduolio modelyje ga-

lumi n procesorių. Tai komponentė. P laukais identifikuos konkretus procesorius, kuriamame mylsta i-tamis procesas,

2.3. OAIJ - proceso turinys renursų aprašas. Apibūdina renursą - operatyrojo atmintį, kurio apibūdinamas kontekstas sraigtė. Disruptorius - fizinėta struktūra, jame yra nuoroda į sraigtę;

2.4. REIJ - i-tojo proceso turimi renursai. Tai informacija apie tai, kokiems renursus yra gaves procesas. Tai nuoroda į sraigtę, kuriamame išvardinti proceso renursai;

2.5. SRIZ - tai renursų kryptės. RS atsakingas už sekurito renursų prienālo sistemoje.

Sekurite renursų, reistina sekurite renursų dekriptorių, kurių gali sekurite procesas kreipdamasis į OSBP „Sekurite renursų“.

3. Proceso būsena. Ji detalizuojama dviem laukais:

3.1. STIZ - myldomo proceso statusas.

RUND - proceso turi proceso;

READY - proceso priimimo. Reikiu tik

procesoriui.

BLOCK - blokuotas proceso. Ji tampa blokuoti, kai jam reikalingas renursas. Kreipiamasi į OSBP „Prainčių renursio“;

RUN ir READY - proceso loginio myldomo būsenos. Procesas, gavęs renursą ir renursų paskirstymo, atnibokiuojasi. Procesas būdingas dėl savo ordinės priežascių. Gvenime papildomas būsenas, kai procesas pristabdomas dėl išorinių priežascių:

READYS - priimimo priestabdymas,
BLOCKS - blokuotas priestabdymas.

Procesai RUN, READY, READYS įtrauktūs į PPS.

Procesas yra įtrauktas į tam tikrą sraigtę. Sraigtai nurojti su proceso dekriptoriais.

Procesų planuotojas pagal priimimo sraigtą ir algoritmus nustato, kada išskirti proceso;

Jei proceso perejo į būseną BLOCK, tai tik dėl to, kad ji s yra paprastė renursio. Kiek yra renursų, kiek yra slėtingyų laukiančių proceso sraigtų (LPSIZ, r-renursas).

3.2. SIZ - nuoroda į sraigtę (LPS arba PPS);

4) sגרյns su kūtais procesais (nurodo meto medyje):

4.1. $T[i]$ - i-tojo proceso tevas. Tėvas - i-tojo proceso kūrejo VV;

4.2. $S[i]$ - i-tojo proceso sūnys sgrātās. Tai nuroda, kāds to proceso nākotnēs sgrātās pārādījums.

5) $P[i]$ - i-tojo proceso prioritētās, tai apliecas iinformācija:

Reursos, kuri ir išskiriami un attaisinami daug kārtu, vadinami pakartotino naudojimo resursais, tātīgāk tie ir naudojami.

Turime 2- reuso vārdi vārdo. Viņiem struktūra:

1) Identifikācija

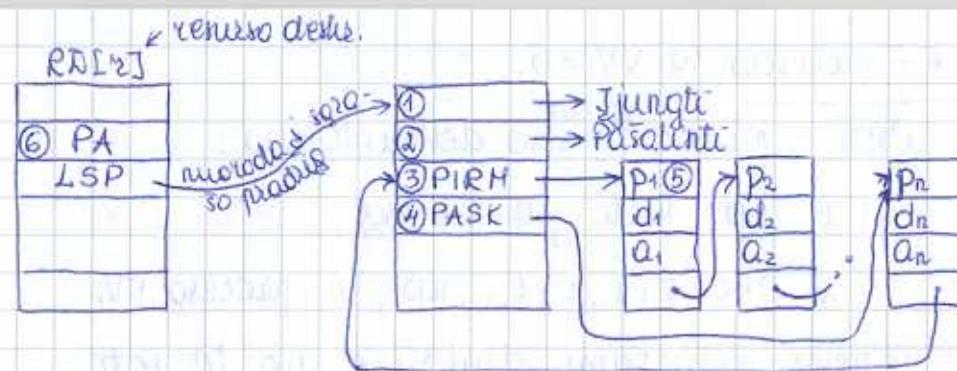
$Rid[i]$ - reuso IV;

$PNR[i]$ - ar kādiem pakartotino naudojimo, ar ne naudojamas resursas;

$K[i]$ - vienības vārds reuso kūrejo (proces), reuso prioritētām apraigumam (atvirodību);

2) $PA[i]$ - nuoroda s grātās pārādījumi. Praktiski nu reuso, descriptoriem gali būti visi kūtols.

3) $LPSI[i]$ - nuoroda s $LPSI^a$



Paaišķināmā:

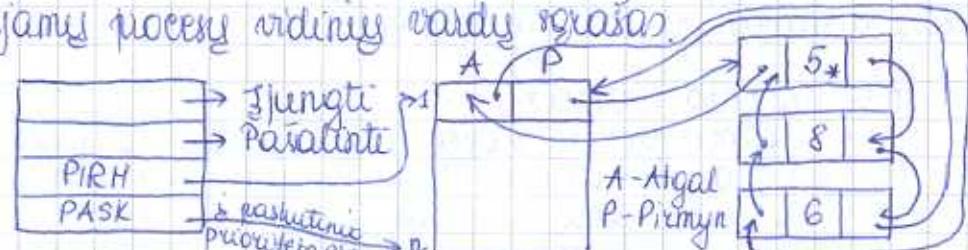
- ① programma, kuri atlieca elementu prijungību;
- ② adresas programas, kuri parādītu sgrātās elementu;
- ③ nuoroda s vienības grātās elementu;
- ④ " " - pastūtīns - " "
- ⑤ proceso VV = P_1 , jis ir viens reuso d_1 . Duoda sūti, kā turēt vārsti apie išskirtīgo P_2 , sūties adresas vārā a_1 ;
- ⑥ reuso prioritētām apraigumas.

Struktūra. Praktiski nu to, koks ir veikums noteime atlikti.

4) $PASK[i]$ - i-tojo reuso pastūstytojo programmas adresas;

5) Apliecas iinformācija. Šajā reuso descriptorijs straukāma pagalbinē apliecas iinformācijā.

$PASK(k, l, L)$ - vienu išķķetīnu galī aptarnauti kelijs procesus. Čiā k - kāds procesu aptarnauja, L - aptarnaujamus procesus vienības vārdu vārda sgrātās.



* - procesas su $VV = 5$.

PRD - proceso ^{recurso} dešriptorius:

1. np - kiek yra process;

2. PROC[i], $i=1, \dots, np$, i - proceso VV.

Ninėmis resursams struktūra yra ta pati.

[2001 10 16]

OSBP yra:

- operacijos darbui su processais

- operacijos darbui su resursais

Operacijos darbui su processais

- kurti process;
- naujinti process;
- aktyvuoti process;
- keisti proceso prioriteto;
- stabdyti process.

Naudojant naujinti proceso, reikia kreiptis į OSBP

„Kurti proceso“, faktinius parametrus nurodant tokias kuriamo proceso komponentes:

n - išorinis vardas;

s_0 - kuriamo proceso priceriuojuose madine būseną;

M_0 - OA madine būseną (kiek išskirta OA resursų);

R_0 - kurti išskiriami resursai;

k_0 - proceso prioritetas.

IV naudojanas rečiamis taip proceso nurodyti.

PROCEDURE KURTIP (n, s_0, M_0, R_0, k_0);

BEGIN

$i := NVV; \textcircled{1}$

$ID[i] := n; \textcircled{2}$

$CPU[i] := s_0; \textcircled{3}$

$OA[i] := M_0; \textcircled{4}$

$R[i] := R_0; \textcircled{5}$

$PR[i] := k_0; \textcircled{6}$

$ST[i] := READY; \textcircled{7}$

$SN[i] := PPS; \textcircled{8}$

$FE[i] := *; \textcircled{9}$

$SE[i] := 1; \textcircled{10}$

Ijungti (SE*, i); $\textcircled{11}$

Ijungti (PPS, i); $\textcircled{12}$

END;

Panaškinimai:

① reikia nustatyti ir gauti proceso VV. NVV gingesna naujų numeri;

② ID - proceso IV;

③ taip užrašoma piceriuojuose būseną dešriptoriuje,

- ④ processuvius deskriptoriuje turuius OA lieki;
- ⑤ kiti turimų proceso rendiny komponentė (irgi nuo-roda į sraugą);
- ⑥ prioritety laukcias;
- ⑦ laikynčių, kad proceso surušiamas su statusu;
- ⑧ išadangi turinėnas proceso turi priblaisyti bent vienam srautui, tai pustintiame jis PPS'ui;
- ⑨ duotu metu dirbantys proceso - einamam proceso - bus žymimas „*“. Tai einamo proceso VV;
- ⑩ proceso turi nuoroda į nėtiny sraugą, iš pradžių jo turėtai;
- ⑪ operuojame srautus su antraštėmis, o antraštės - tai programos, dirbantios su srautais. Taipas dirbantys (einamam) proceso išgyjo sūnus. Reikia papildyti jo sūnus sraugą;
- ⑫ naujas proceso veikla ištrauličių planuotinių proceso sraugą.

Primitivas. Kurti proceso informaciją apie kuriamo proceso neregistravę deskriptorius. Čia jokių informacijų nekuriamos.

OSBP „stabdyti proceso“. Čia galimas dengubas nes operacijos traktuojamas, kai proceso yra tam tikro medžio pomedžio salinis. Kyla klausimas, ar stabdyti nė vien proceso, ar visus to pomedžio proceso? Šiuo atveju stabdomas vienas proceso, kita - sūninių - taciai daubo. Parametrai nurodome:

n - išorius vardas;

a - atsakymeg susies adieras, į kurių grąžinama

informacija apie stabdomo deskriptorių būseną (jo kopija).

PROCEDURE STABDYNIP (n, a);

BEGIN

i := VV(n); ①

s := ST[i]; ②

IF s = RUN THEN STOP(i); ③

a := copydesc[i]; ④

ST[i]:= IF ⑤ s = BLOCK OR BLOCKS THEN BLOCKS

ELSE READY;

IF s = RUN THEN PLANUOTOJAS; ⑥

END;

Paciškini mai:

① pagal IV nustatomas proceso VV;

② statuso reikiame;

③ stabdomas proceso gali būti vykdomas, planuojamas, blokuotas. Pirmiausiai iš jo reikiel atimti proceso VV. STOP(i) pertraučia proceso lygio, kuris vykdo i-top proceso. Tai proceso išs P[i]. Reikia išminti pertraukto proceso būseną į CPU iš paralyti, kad proceso išs stabdyti i-top proceso yra atlaišintas: PROC[P[i]]:=1;

④ si procedūra padaro deskriptorių kopiją,

⑤ toreguojamas i-top proceso statusas;

⑥ Jei buvo pustabdytas vykdomas proceso, tai reikia iškirsti planuotojį, nes galbūt kiti proceso laukia proceso vienos. Planuotojo išlydymui naujas proceso nenušiama.

Swabu, kad iki planuimo veikimo viršas jau

būtybė padaryta, nes planuotojo darbo parinkoje procesorius atemamai iš proceso, kuriame jis pati yra vylidomas.

OSBP „Aktivuoti proceso“. Tai simetrinis OSBP. Nauja pristabdymo būteng. Galbūt ištakėčia planuotojė, jei būseną yra READY. Parametras n- IV.

PROCEDURE AKTYVUONIP(n);

BEGIN

i:=VV(n);^①

ST[i]:= IF ST[i]=READY THEN READY

ELSE BLOCK;^②

IF ST[i]=READY THEN PLANUOTOJAS;

END;

Paaistiinimai:

① nustatomas VV pagal IV;

② neįstaba buvo blocks;

OSBP „Naikinti proceso“. Procesas gali suraikinti bet kurį savo palikuonį, bet negali suraikinti savęs. Tada jis nusinešia planenius savo tevui, kuris jis ir suraikina.

Procesas Hain-Proc suturia Job-Governor, kai

yra „uzduoties išsinejė atmintyle“ reurusas.

Job-Governor negali savęs suraikinti. Tada kuriamas filijalus reurusas ir Job Governor užblokuojamas. Tada jis galima naikinti.

jei naikinti menq proceso, ar visq pomeigs? Jei suraikintime menq proceso, tai sistemoje bus nevalidomų proceso (bus chaosas). Reikiu naikinti visq pomeigs!

jei naikinti visus reurus? Jei yra is pakastiomo naudojimo reurusai. Juos reikiu atlauzinti.

Parametrai - naikinamo proceso IV.

PROCEDURE NAIKINTIP(n);

BEGIN

L:= false;^①

i:=VV(n);
P:= TE[i];
NUTRAUKTI(i);^②
Parolkti(ST[i], i);

IF L= THEN PLANUOTOJAS;
END;

Paaistiinimai:

① jei L=true, tai ištakėčias planuotojas;

② nutraukti i-togų proceso. Procedūrai perduodamas proceso VV.

PROCEDURE NUTRAUKTI (*i*);

BEGIN

IF ST[*i*] = RUN THEN

BEGIN

STOP(*i*);

b := true;

END;

Paralinti (ST[i], *i*); ①

FOR ALL SE[i] NO NUTRAUKNI(S); ②

FOR ALL ZEOAE[i] V RE[i] NO ③

✓ IF PNR THEN Ijungti (PA[Z], Paralinti(OAE[i] V RE[i]),

FOR ALL ZESRI[i] NO NRT(z); ④

NRT(*i*); ⑤

END;

Paaiškinimai:

① kadaangi proceso visada yra kaišiuame sraigtė, tai reikia jis iš ten paralinti ST saugo nuoroda į sraig-
tę, iš kurio reikia paralinti, *i*-IV.

② reikiu nutraukti *i*-tojo proceso rutinių vykdymą,

③ z - rentrai;

atlaikiname pakartotino naudojimo rentrus,

④ iku renautinti proceso rutury venusy descriptorius,

⑤ naktiname proceso descriptorių.

Priovitetas atitinko proceso padetį priamu-
rus ypač iki prioviteto, tiek sraigtų.

OSBP „Keisti proceso prioviteto“. Kalbame apie pri-
vitetos procesoriaus resursų atžvilgą. Proceso iterpi-
manas i sraigtų vytira atrizvelgiant į proceso priovitetą,
todel proceso prioviteto paketinius vytira taip: para-
liuamas iš sraigtų ir po to iterpiama pagal nau-
jį priovitetą. Parametrai: n - IV; k - priovitetas.

PROCEDURE KEISTI_PP (*n*, *k*);

BEGIN

r := VV(n);

M := PR[i]; ①

Paralinti (ST[i], *i*);

PR[i]:=k; ②

Ijungti (ST[i], *i*);

IF M < k A ST[i] = READY THEN PLANUOTOJAS

END;

- ① įsimenamas senas priovitetas;
② pusbūrimas naujas priovitetas;

Operacijos darbu su resursais

Toks OSBP ypač: kurti, nukurti, prasitti, atlaisinti.

OSBP „Prasitti resurs“.

Procesas, kuriam reikia
resursu, iškurecia iš priuityog, nurodydamas VV iš ④

adierg. Toks procesas pereina į blokavimų būseną.

Blokavimais ygyta iki priėmiant resursą. Procesas įjungiamas į laukiančių to resursu proceso sraigtą.

Parametrai: RS - resursų IV, R - kurių resursų dailes priėjoma, A - atsakymo suties adieras, y kur planesi.

PROCEDURE PRASYTI_R (RS, R, A);

BEGIN

z := RVV(RS); ①

Ijungti (LPSE[z], (*, R, A)); ②

PASC (z, k, L); ③

B := true;

FOR J:=1 STEP 1 UNTIL k DO

IF L[J] ≠ * THEN ④

BEGIN

i := L[E J];

Ijungti (PPS, i);

SN[i] := PPS;

ST[i] := IF STE[i]=BLOCK THEN READY

ELSE REAIS; ⑤

END

ELSE ⑥ B := false;

IF B THEN BEGIN

STE[*] := BLOCK;

STE[*] := LPSE[2];

PROC[P[i]] := 1; ⑦

Paralinti (PPS, *);

END;

PLANUOTOJAS;

END;

Paaiškinimai:

- ① nustatomas resursų IV pagal IV.
- ② procesas įjungiamas į laukiančių to resursu proceso sraigtą, * - tuo metu ryšdomas procesas.
- ③ resursio paslurstytijo programa. k - kiek proceso ap-tarnauja, L - aptarnaujamų proceso IV skaičius;
- ④ ar tai tuo metu nedirbantis procesas?
- ⑤ reisuo buvo BLOCKS.
- ⑥ reisuo tai yra duotie metu dirbantys procesas.
- ⑦ jei proceso ryšdiciui procesois pasielktame laisvai.

2001 10 23

OSBP „filiausinti resursų“. Tai atitinkta situacijos,

kai procesas gauja priuvestino naudojimo resursų ir kai jo jam neveikia, jis jis atlaikiuva ir įjungia į sraig-čio būtinyjį resursų. Kai kurie resursai sukuriami pro-ceso darbo mete. Parametrai: RS - resursų IV, R - OA atlaikinamų dailes aprašymas.

PROCEDURE ATLAISVINTIR (RS, R);

BEGIN

R := RVV(RS);

Ijungti (PAΣ Σ J, R); ①

PASK (R, k, L);

IF k>0 THEN FOR J:=1 STEP 1 UNTIL k DO

BEGIN

i := L[J];

Ijungti (PPS, i);

S Σ [i] := PPS;

ST[iJ] := IF ST[i] = BLOCKS THEN READY

ELSE READY

END;

IF k≠0 THEN PLANUOTOJAS;

END;

① R - atlaikinimo resurso dalių apibūdinimas.

OSBP „Kurti resurso“. Parametrai - resurso apibūdintys duomenys:

RS - resurso IV;

PNR - loginis posūmio (naujostės naudojimose);

PA - nuoroda į resurso priešnamumo aprašymą;

LPS -

PASIK - resurso pasiekimojo programa

PROCEDURE KURTIR (RS $_0$, PNR $_0$, PA $_0$, LPS $_0$, PASIK $_0$);

BEGIN

R := RVV;

// naujas resurso VV

RS Σ Σ J := RS $_0$;

// resurso dešriptoriuje IV

PNR Σ Σ J := PNR $_0$;

K[Σ J] := *; ①

PA Σ Σ J := PA $_0$;

LPS Σ Σ J := LPS $_0$;

PASK Σ Σ J := PASK $_0$;

Ijungti (SR Σ Σ J, R); ②

END;

Paašlu'nimai:

(kūrėjo)

① resurso tebė VV. Tai duotu metu vystantį procedūras, kuriamė ir panaudotan nė priimtyvas;

② i tuo metu vystydavo procedūros nukirtys venusis sėmęs įtraukiomas ir naujai nukirtas resurzas.

OSBP „Naujinti resurso“. Sunaujinti resurso galite arba jo pirmatakas. Panaujinamas resurso dešriptorius, prėi tai maitinamas jo naujantys parametrai.

PROCEDURE NAIKINTIR (RS);

BEGIN

R := RVV(RS);

R := Pasalinti (LPS[R]); ①

NHILG R \neq Δ AD

BEGIN

ST Σ R.PJ := ② IF ST Σ R.PJ = BLOCK

THEN READY

ELSE READYS;

Ijungti (PPS, R.P);

ST Σ R.PJ := PPS; ③

R.A := "atsakymas"; ④

✓ ENR;

R := Pasalinti (LPS Σ 2);

NRR(2); // naukinti resurso descriptorius

PLANUOTOJAS;

ENR;

// Pasiskuumiai: ■↓



- ① Procesas ateliai i sistemos nez OSBP "Kurti proceso";
- ② tolū signalas duodamas procesui "aktyvuoti proceso";
- ③ dėl tolū tilus išorinių priėmimų.

- ① turėsim kurodžių pasakiamą elementą iš kurodžių proceso;
- ② dirbtinė būdu suaktyvinamas procesas, fodel kuriamei statutai;
- ③ descriptoriuje koregiama komponente sū;
- ④ atraujmy laukie turėjome P.R,A (ius - A - r dalo; A - ats. rato).

P Σ 1]

IA	
CPU	
P	
OA	→ LPS
R ①	* ←
SR	* ←
ST	
SD	
T ②	→ PPS
S ③	→ ip
PR ④	

R Σ 2]

RS	
PNR	
K	
PA ⑤	
LPS	
PASK	
PRD	
np	
PRCC	

* - nuoroda į
sugrup

R Σ 2 - renurio descriptorius

① nuoroda į sugrup, kur išvardinti msi proceso gauti re-

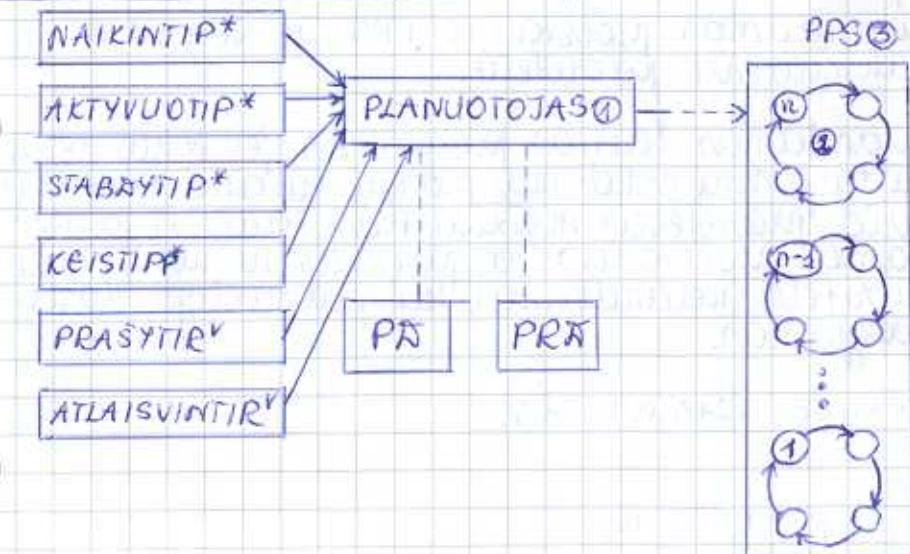
rusai;

② proceso tevo VV;

③ hñig sugrupas;

④ priekytių veilimė;

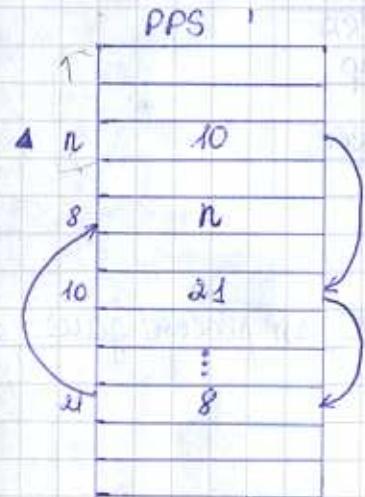
⑤ resurso prieinamumo apraýmas.



- ① planuotojas išskiriamas iš proceso* ir iš priemi-

- ① tiks v;
 ② surandoti n-tojo prioriteto procesai. Tai yra sequencas;
 ③ sequenc irgrana.

Pasiuodosinių procesų sequecijų valdymoje
naip atskirų manymo:



▲ n-tam prioritetas. Procesas nu VV=10 yra pirmasis iš n-tojų prioriteto turinčių procesų.

Etapai: 1P

- surandamas procesas iš PPS su READY ir aukščiausiu prioritetu;
- surandamas laisvas procesorius. Jei tokio nėra, tai tarp visų ylidomų procesų surandamas tas, kurio prioritetas mažiausias. Tada iš jo atimamas procesorius. Jei procesas nu mateminiu prioritetu surandamas, tai planuotojo darbas buviamas.

PROCEDURE PLANUOTOJAS;

BEGIN

p := PRT := n;
 count := 0;
 c := 1;

4: B:=true; ①

WHILE B AND PRT ≠ 0 DO ②

BEGIN

p := P[p];

IF p=PRT THEN ③

BEGIN PRT := PRT - 1;

p := PRT

ELSE B := STE[p] ≠ READY

END; ④

- 3) gali buti, kad planuotojo darbo pabaigę procesorius atimamais iš proceso, kuriame ylidomas planuotojas. Reikia, kad procesoriui atimamais būty pateigoje pabaigę.

IF PRT = 0 THEN GOTO ENDPROC;

WHILE C ≤ np DO

IF PROC[i] ≠ Λ THEN C := C + 1 ⑤

ELSE

BEGIN PROC[C]:=p; ⑥

P[p]:=C; ⑦

STE[p]:=RUN;

IF C ≠ P[*] THEN Astatytu
- būsenę (C,CPUEP);

ELSE P*:=p;

```

C := C+1;          (1) 9.01.2018 11
GOTO L
END;

PRTMIN := PRT;    (8)
FOR C:=1 STEP 1 UNTIL np DO
BEGIN
    q := PROC[C];
    IF PREQ[J] < PRTMIN THEN
        BEGIN PRTMIN := PREQ[J];
        CP := C; END;
    END;
    IF PRT ≠ PRTMIN THEN    (9)
        BEGIN q := PROC[CP]; STEQ[J] := REARY; (10)
        P[PJ] = CP;
        PROC[CP] := p; STEP[J] := RUN;
        IF q = * THEN P* := p; (10)
        ELSE Perjungti(p, q, CP);
        /atimis iš p/
        GOTO L;
    END;

```

2001 10 30

```

EINPROC: IF STEP[J] ≠ RUN THEN Perjungti(*, p*, PEX)
END;

```

Padažinimai:

- ① stebiuje kol suraname REARY proceso;
- ② PPS sraute įvedame lauką su padažinimu p
(pri. 54, kur ①);
- ③ jei perkaupine prie mūsų srauto nėra n iki n, tai per-
kaupame prie kito rečiaro, t.y. prie kito prioriteto;
- ④ išeikaukite iš eilio, kuri nurodės procesas su nauju-
čiaunu prioritetu;
- ⑤ jei procesoriaus dešriptoriuje yra informacija,
kad procesoriaus užimtas;
- ⑥ procesoriaus c atiduodamas pirmame etape sur-
baudytam procesu;
- ⑦ p - tag, proceso ryšutis procesoriaus c;
- ⑧ prioriteto patenkti gimehamo paruošinio proceso
prioriteto rečiuje;
- ⑨ jei ≠, tai užimtu' procesoriu;
- ⑩ tik išmenamas pibero V;
- ⑪ kai kurių užmamas procesoriaus, tai užmamas
statys;

I procedūra „Perjungti()“ kreipintys buvo tada,
kai 3-me etape buvo perjungiamas procesoriaus
PROCUREE PERJUNGTI(p, q, c);

BEGIN

Pertraukti(c);

Išminti_būseną (c, CPU[p]);

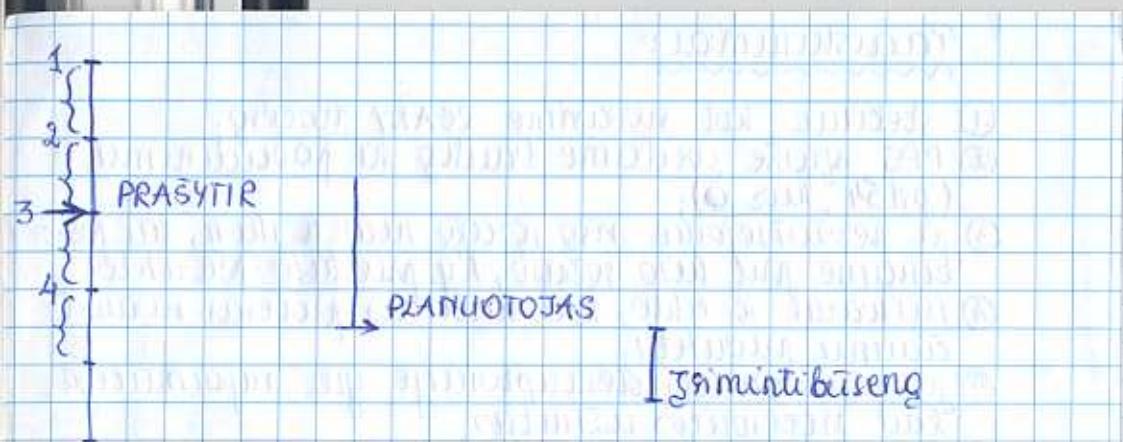
if p = * THEN Išminti_GA(GA, CPU[p]); (1)

Aištatyti_būseną (c, CPU[q]); (2)

END.

① GA - gyvimo adresas;

② aištatomos procesoriaus c būseną pagal proceso q
dešriptorioides komponentes rečiuje.



fulisto lygio procesorius (programa) operuoja
žymėnius (1, 2, ...). Tai iš bus adferai.

Lygiagrečiai veikiančių

procesų sroveikos principai

Nosektūs procesai veikia vienu metu - lygiagrečiai.

Procesai neturi jokių tarpusavio ryšių. Proceso aplinkoje nudaaro reурсai, kuriuos proceso naudoja iš kuriuos sukuria.

- Prasminis ryšys tarp procesų išreiškiamas per proceso reурсus.

OS gali būti apibūdinta kaip proceso rinkinys, kur
procesai:

- veikia beveik nepriklausomai (lygiagrečiai);
- bendrauja per pranešimus ir signalus (reурсus);
- konkuruoja dėl reурсų.

Skaičiuojamas sistemos minimas aparaturinis iš

- loginių lygiagretumų (parallelių).

Aparaturinis lygiagretumas – reiskia lygiagretę,
menalaičių aparatūros darbą (pvz., išorinių frenginių
kontrole, kur liekurių iš jų kontroliuoja kitas procesas).

Loginiame lygiagretume nenašbu lygiagretumas.

Apie jį halbama taip, kai teoriniai darbai gali būti vystomas lygiagrečiai.

figuratinis paraleizmas įvedamas efektyvumo sumetimais, kad greičiau vyktų darbas. Procesoriuje rengant aparatiniam paraleizmui visien varbu meninteli procesor. darbo laiką skirtystę keliems procesams. Todel įvedama lygiagrečiai vykdomo proceso abstrakcija.

Neformalai proceso - tai darbas, kurį atlieka procesoriai, vykdymas darbo su duomenimis.

Loginis paraleizmas panākumi tuo, kad kiekviena proceso turi savo procesorių ir savo programą. Realiai, skirtingi procesai gali turėti to patį proceso ar to pačių programą.

Prosesas yra para (procesorius, programa).

Prosesas - tai būsenų seku $s_0, s_1, s_2, \dots, s_n$, kuri kiekviena būsena saugo visų proceso kintamujų programos reikimes. Pagal proceso būseną galima prasti būtine proceso darbo. Proceso būsena turi turėti sekantiuos výkdomos programos adresą. Proseso būsena gali buti pakelta paties proceso darbo rezultate arba kiti proceso darbo rezultate.

Valdymo iš informacinių rūsių tarp proceso realizuojamas per bendrus kintamuosius. Nagrinėjant tik vienius processus, gaunami nauji pro-

cessorių nepriklausomi sprendimai.

Lygiagrečiai veikiančių proceso principus pirmas pateikė R. J. Liptua 1965 m.

s_1, s_2 - nuosekliai

s_1 and s_2 - lygiagrečiai

Prz. $(a+b) * (c+d) - (e/f)$

begin

$t_1 := a+b$ and $t_2 := c+d;$

$t_4 := t_1 + t_2$

end
and

$t_3 := e/f$; $t_5 := t_4 - t_3;$

Translatorius turėtų išskirti lygagrečius veiksmus ir sugeneruoti autėčių užrašytą programą.

Jei proceso pavyklo komanda FORK N (issisakojimas), tai išsaukiama proceso q výkdomas nuo žymės N.

Jei proceso pavyklo komanda QUIT, tai jis pabaigia.

Prosesų aplinkybės komanda JOIN T, N:

$T := T - 1;$

IF $T = 0$ THEN GOTO N;

} nedalomi (nepertraukiame)
veikimai.

Prx. $N := 2;$

FORK P3;

$M := 2;$

FORK P2; /* iš viso dabar dirba 3 procesai (pi-
mas tas, kuris nėra initializuotas*)/
 $T1 := A + B;$ JOIN M, P4; QUIT;

P2: $T2 := C + D;$ JOIN M, P4; QUIT;

P4: $T4 := T1 * T2;$ JOIN N, P5; QUIT;

P3: $T3 := E / F;$ JOIN N, P5; QUIT;

P5: $T5 := T4 - T3;$

P1 X

P2 X

jei turi bendras kintamasis X.

Tarkime, P1 atlieka veiksmą $X := X + S$, ir P2 atlie-
ka to patį veiksmą.

Prosesas P1 turi procecorius C1 u registorių R1, o
prosesas P2 turi procecorius C2 u bendrus registorių R2.
Dėl ko skaičiai:

$$\begin{matrix} x = v \\ \downarrow \\ t_0 \end{matrix}$$

$\rightarrow t$

a) P1: $R1 := X;$ $R1 := R1 + 1;$ $X := R1;$...

P2: ... $R2 := X;$ $R2 := R2 + 1;$ $X := R2;$... $X = V + 1$

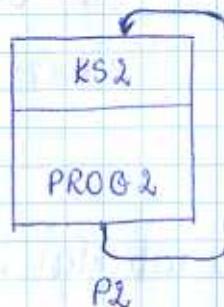
b) P1: $R1 := X;$ $R1 := R1 + 1;$ $X := R1;$

P2: ... $R2 := X;$ $R2 := R2 + 1;$ $X := R2;$ $X = V + 2$

Gausame $X = V + 3$ ir $X = V + 2$, o tai p negali būti.

Tai dėl jų procesų problemos.

Programos dalis, dirbanti su bendrais proceso re-
sursais, vadina kritinė sekeja. Negalima leisti,
kad du procesai vienu metu įsitytų į kritinę sekejiją.
a) ir b) naikina kritinės sekejijas, o taskai - neliri-
ting sekejijas.



Šiu lygiagrečiai dirbantys procesai P1 ir P2.

BEGIN

P1: BEGIN L1: KS1; PROG1; GOTO L3 END

ANT

P2: BEGIN L2: KS2; PROG2; GOTO L4 END

ANT

LN: BEGIN LN: KSN; PROGN; GOTO LN END

END;

Jei minties blokavimas - tai j vieną atminties
loštelę gali kreiptis vienu metu tik vienas procesas.
Pielaida. Procesų cirkulymo greičiai bet kokie. Prog-
rama gali būti nutraukta tik išejuo iš KS.

BEGIN INTEGER EILĖ; EILĖ:=2;

P1: BEGIN L1: IF EILĖ=2 THEN GOTO L1;

KS1; EILĖ:=2;

PROG1; GOTO L1

END

AND

P2: BEGIN L2: IF EILĖ=1 THEN GOTO L2;

KS2; EILĖ:=1;

PROG2; GOTO L2

END

END;

Kuo autūciamu pateiktas kodas blogas? Tarkime, proceso vylidymo laikas yra P1 >> P2. Turėsime laukti tol, kol bus vylidoma PROG2. Rekiučiai tai greitinti.

2001 11 06

Atvejus proceso bendro darbo programa:

BEGIN

BOOLEAN C1, C2; C1:=C2:=true;

P1: BEGIN L1: IF 7C2 THEN GOTO L1;

C1:=false; KS1; C1:=true;

PROG1;

GOTO L1

END

AND

P2: BEGIN L2: IF 7C1 THEN GOTO L2;

C2:=false; KS2; C1:=true;

PROG2;

GOTO L2

END

END; /* L1 yra proceso eilės programos pabaiga */

Vykdyta kita problema. Nykdymo greičiai yra bet

kokius. Gali būti, kad procesai vienu metu vylido KS.

Taigi problema išsprendžiā gana sudetingas tekėjimo algoritmas. Procesas atžymi savo kryptį į KS logi-

nii kontamuoju Ci=false. Tiesėjus iš kurių reikiėj

Ci=true. Žeiti į KS procesas gali tik tada, kai kitas

procesas nėra KSje arba nėra pareiškės noro jė vylidyti.

Siekiama išvystyti EILĖ naudojamas tada, kai du

procesai suvaidina KSje (pvz.: noras vylidyti KS, išjimas į KS).

Procesas, laukiantis, kol kitas procesas baigs

vylidyti KS, išeina iš laukimo pats atrisakydamas

vylidyti KS, tai eile vylidyti KS pukilauso kitam pro-

cesui.

BEGIN INTEGER EILĖ;

BOOLEAN C1, C2;

C1:=C2:=true; EILĖ:=1;

P1: BEGIN A1: C1:= false; ①

L1: IF E1E THEN

BEGIN

IF E1E=1 THEN GOTO L1;

C1:= true;

B1: IF E1E=2 THEN GOTO B1;

GOTO A1

END;

KS1; E1E:=2;

C1:= true;

PROC1:

GOTO A1.

END

AND

P2: BEGIN A2: C2:= false;

L2: IF E2E THEN

BEGIN

IF E2E=1 THEN GOTO L2;

C2:= true;

B2: IF E2E=1 THEN GOTO B2;

GOTO A2;

END;

KS2; E2E:=1;

C2:= true;

PROC2;

GOTO A2

END

END;

② procesas pareiskia karo vykdyti KS.

③ tai autams procesas atnaujins noro vykdyti KS.

Toks sprendimas yra nedidelis, kad juo remiantis būtina organizuoti darbu netinkamai nes tada bus labiausiai sudėtingas.

Postuliuoti pirmyn yra maniai varbu - realizacija.

Ajubresime aparato, tinkaantį lygiagrečiem vykdymui.

SE M A F O R A S

Semaforas S - tai sveikas skaičius (≥ 0) kintamasis, su kuriuo atliekamos operacijos: P(s), V(s). Jos panūjimi saubėmis:

- 1) P(s), V(s) - nedalomas operacijos, t.y. jis vykdymo negalima pertraukti vi jis vykdymo metu negalima kreiptis į semaforo S,
- 2) V(s): $s := s+1$; (didinama semaforo reikime)
- 3) P(s): $s := s-1$; (sumatinama, jei $s > 0$)
- 4) jei semaforo reikime lygi nuliui ($s=0$), tai procesas p, kuris vykdo operacijos P(s), laukia, kol sumatinimas menetis bus galimas. Šiuo atveju P(s) yra pertraukiamas;
- 5) jei keletas procesų reikėtų metu iškviesti V(s), P(s) ne menetis semaforu, tai užklausimai vykdome nesekiliai.
- 6) jei keletas procesų laukia operacijos 'P(s)', S - tas pats,

tai s reikiamei tapus teigiamai (kai karkuris procesas įvykdė operaciję $V(s)$), kai kuris iš laukiančių proceso bus pradėtas vykdyti.

Primitivas P aktiūlia kita proceso aktyvaujį rezgul turime tolj semaforą:

Jei yra kritinės sekcijos apraugsiantis semaforas.

n - proceso skaičius;

BEGIN SEMAPHORE H;

H := 1; // pradinė reikimė.

P1: BEGIN ... END

END

P2: BEGIN d_i: P(H); KSI; V(H); PROGI END

END

Pn: BEGIN ... END

END.

Jei semaforas igyja tok dienreikimes 0, 1, tai jis vadintamas dvejetainiu, jei bet kokią, tai bendriiu.

Semaforius galima naudoti proceso sinchronizacijai. Turime du proceso. Norime, kad antras pr-

detų vykdyti savo programą, tuomet, kai pirmas pasisi jau atlikus signalą,

BEGIN SEMAPHORES;

S := 0;

P1: BEGIN :

→ P(S); // tai signalo iš proceso laukimas

END

AND

P2: BEGIN

:

→ V(S); // iš nėšomai signalas proceso P1

:

END;

jei pirmas bus vykdomas proceso P2, tai P1 neveis į laukimo būseną.

Panaudotekime proceso, kuris paimo informaciją iš bufferio atmintis atlikties iš to informacijos apdorojo.

Rezul bufferio atmintis sudedota iš N bufferių.

semaforas T - turėti bufferių skaičius;

semaforas U - užimti bufferių skaičius;

Rez B - semaforas, saugantis KS, atliekančią veikimus su bufferio sekcijomis.

BEGIN SEMAPHORE T,U,B;

T:=N; U:=0; B:=1;

^① GAN: BEGIN LG: grašo-gaminimas;

P(T); P(B), užrašymas į bufferį; V(B);
V(U);

GOTO LG;

END

AN

NAUD: BEGIN LN: P(U);

P(B); paemimas iš bufferio; V(B);

V(T); grašo-apdorojimas; V(B);

GOTO LN;

END

EM;

^① proceso gamintojo programa.

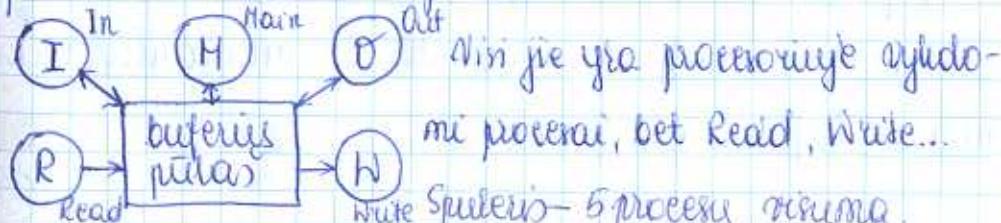
^② darbas su bufferiais yra domas kaip darbas su semaforais.

^③ jei $T=0$, tai $P(T)$ pereis į laukimą.

Ivedimo / išvedimo SPOOL'eriai

SPOOL yra OS dalis, atliekanti jv/iso virtualizaciją.

Kuomet yra suhaupiama ivedama informacija, ji apdorojama ir išvedama kaip rezultatas.



Ivedimo proceso funkcijos

Buferių pūtas – srautinis (faktiskai tys srautai), kuri pradinių buferių pūtų apjungta į laisvų buferių sraigtą.



Ys pradinių $F \rightarrow L$ – miras kūnas yra laisvi buferiai.

- iivedimo buferių srautis;

- išvedimo buferių srautis;

Apdorojimo proceso užduotis – paruošti informaciją išvedimui.

Veiksmus seka yra tokia:

^① iivedimo proceso paima bufferį iš laisvų buferių sraigt;

^② paleidžia į skaičiavimų;

^③ užpildyti bufferį į jungia į iivedimo buferių srautę.

1. Pagrindinis apdorojimo proceso Main išna buferį iš iivedimo buferių srauto;

2. Apdoroja informaciją;

3. Išna buferį iš tūčtų buferių srauto. Jei užpilda išvedama informacija;

4. Bufferį į jungia į išvedimo buferių srautę.

Ivedimo proceso T gali ykti tik tada, kai yra
tuččius buferius. skirtintamai H gali ykti, kai yra įvedimo
sąraše elementai iš laisvo buferiai. Jei T nėra
parinktių laisvo buferio, tai H užblokuoja, nes jau
nėra laisvybė buferių.

nl - laisvybė buferių skaičius

nin - įvedimo

nout - išvedimo

ml - KS apraugs, semaforas

min - įvedimo buferių KS apraugs semaforas

mout - išvedimo

Pagrindinio proceso H programa:

M: BEGIN LH:

P(nin); // H turi nuti įvedimo buferį. Jei
// tolko nėro, tai H blokuojan.

P(min); // jei kitas proceso yktido įvedi-
// mo, tai H blokuojasi

Paimti pirmą buf. iš įvedimo buf. sąrašo;

V(min); // nuimama įvedimo KS aprauga;

Apldototi buferio turinį;

P(nl);

P(ml);

Paimti pirmą buf. iš laisvo buf. sąrašo;

V(mi);

Užpildyti paimtis buf. išvedama info;

P(mout); // išved. buf. KS aprauga.

I jungti buf. į išved. buf. sąrašo galę; // nėra
// ma. iš sąrašo pradžios, bet raišma į galą;
V(maut); !!!

P(ml); // buvo parinta įvedimo buferis. Jei rei-
// kia atlaikinti, tai darbas nu KS;

I jungti buferių į laisvybė buf. sąz. galę;

V(ml);

V(nl);

GOTO LH;

END;

filtrekant įvedimo reikimą, darbas turi būti mch-
ronizuojamas su įvedimo įrenginiu. Įvedimo įrengi-
nio darbo pradžios ir pabaigos situacijos (realiai - per-
traukimo situacijos) modeliuose siame semaforais

SR - skaitymo įrenginio startas;

FR - skaitymo įrenginio finišas;

Procesas R yktido P(SR), o I - V(SR). Jei SR=0,
tai V(SR)=1 iš procesas R gales baigtis darbą. I blo-
kuojasi nuo P(FR), o proceso R paleidžiame V(FR).

I iš R galima synchronizuoti bendram darbui.

I turi apieipinti proceso R tuščias buferius, nucijuoti
R iš įrengutų į pūlo užpildytus per įvedimus buferius, kai
R baigta darbą.

2001 11 13

Išvedimo proceso valdymas

I: BEGIN LR: P(nl);
P(ml);

IF iško paskutinių laisvų buf. THEN

BEGIN V(nl); V(uil), GOTO LI ENR
Paukšt. buf. iš laisvų buf. pag.
V(ml);

V(sr);

P(FR);

P(min);

Prijungti išvedimo buf. prie išvedimo buf. pag;

V(min);

V(nin);

GOTO LI

ENR;

Skaidymo proceso valdymas

R: BEGIN LR: P(sr);

Perkaikyti į nurodytus buf.

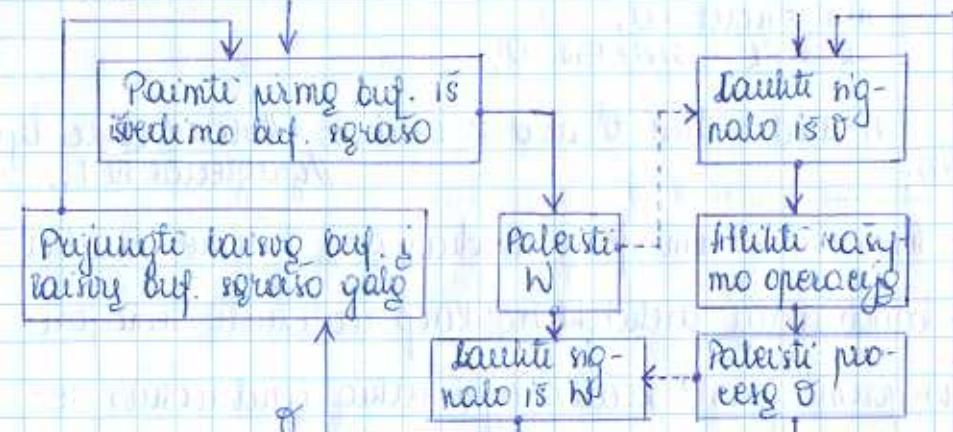
V(FR);

GOTO LR

ENR;

Panašiai reguliuojant išvedimo ir kaiymo protokolai.

Schematiskai:



schema programos pavidaile:

O: BEGIN LD: P(naut),
P(maut);
Paimti išvedimo bufey;
V(maut);
V(SN);
P(FN); //laikoma signalo iš kaiymo
//proceso N - raigimo proc. siūlės;
P(ml);
Prijungti laisvo bufey;
V(ml);
V(nl);
GOTO LD

ENR;

W: BEGIN LW: P(sn);

uzraigt. iš buferio; // jis nurodomas;
V(FN);
GOTO LN

ENR;

// visų perkius proceso bendro darbo programa)

BEGIN SEMAPHORE ml, min, maut, nl, nin, naut, SR, FR,
SN, FN;
ml := min := maut := 1;

$n1 := n;$
 $nin := nout := 0;$
 $SR := PR := SN := FN := 0;$

M and I and O and R and W . // turi duobė ly-
ENR; // galite išleisti n I, ...

Apibūdintime semaforų mechanizmo. Turime tradici-
ne kompiuterių architektūrą. Kaip realizuoti semafori-
nius primitiveus? Realizuojame reikiu bendruosius se-
maforus išreikšti dvejetainiais, o mes operavome kaip
tik bendraintais semaforiais. Bet kuri bendras sema-
foras gali būti išreikštinas dvejetainiu semaforu.

Tegu S - bendras semaforas. Tada jo gali būti
pažeistas kontaminuoju NS ir dviem dvejetainiais sema-
forais H ir R .

$P(S)$ - nedaloma operacija, H - kurtinės sekejės apraugs.

$P(H);$ // KS apraugs nuo dvejetainio semaforo H ,

$NS := NS - 1;$

IF $NS \leq -1$ THEN // turi žoytii blokavimais

BEGIN $V(H)^*$; $P(R);$ ENR // $V(H)$ kui mama apraugs;
// $P(R)$ -turi išraukti blokavimą;
ELSE $V(H);$

// jei (*) nemumtume KS apraugs, tai kuras protetas
// kreipdamasis negalečių atlikti $P(S)$ veiksmą.

$M := 1;$

$NS := S;$

$R := 0;$

$P(S)$ atvejai:
1) $S > 0$ - kui mama KS apraugs ir neįraukiamas
laukimas;

2) $S = 0$ - turi žoytii perejmas į laukimą. Tada bus
panekiamas protetas? $V(S).$

$NS \geq 0$ parodo laukiančių perejų skaičių.

$V(S) \sim P(H), NS := NS + 1;$

IF $NS \leq 0$ THEN $V(R);$ // yra laukiančių perejų,
// kurie užblokuoja semaforu $R;$
 $V(H);$

$RS,$

PRAŠYTI R (RS, R, A);

ATLAISVINTI R (RS, R);

Semaforo mechanizmas išreikšiamas taip:

$P(S) \sim PRAŠYTI R (S, \Omega, \Omega),$

$V(S) \sim ATLAISVINTI R (S, \Omega),$

Operacijų su semaforiais
realizaciją

Paprastai komandų sistema neturi operacijų P, V .

Atlikurais atvejais - turi. Tačiau, kai galima būti reali-
zuoti semaforinius primitiveus, reikiu, kad architek-

ture nedalome rečiu patikintys žodžių: $PP(x)$;

BOOLEAN $PP(x)$;

BOOLEAN x ;

BEGIN

$PP := x$;

$x := \text{TRUE}$;

END;

Tegul turime dvejetainį semaforgą S. Tada $P(S)$ ekvivalenti:

$P(S) \sim L: \text{IF } PP(x) \text{ THEN GOTO } L,$

/* kai $s=0$, tai $x=\text{TRUE}$; $s=1 - x=\text{FALSE}$; $s>0 - P(S)$
turetyje neįsėjanti laukimo.
kai $s=1 \Rightarrow x=\text{FALSE} \Rightarrow PP(x)$ neterkiua, rolypu;
kai $s=0 \Rightarrow x=\text{TRUE} \Rightarrow PP(x)$ terkiua sgi, laukimas,
kai x neįspardaujus FALSE */

$V(S) \sim x := \text{FALSE};$

Ks apsauga, naudojantie autoriai apima įtis
mechanizmą, galėtę atrodyti taip:

$L: \text{IF } PP(x) \text{ THEN GOTO } L;$

KS;

$x := \text{FALSE};$

Nabas panaudonime bendrą ir dvejetainių semaforų išryškiai. Turedamis bendrą semaforgą S, $P(S)$ ir $V(S)$ norėtume išreiškti komandomis: pirsturti ir patikinti.

$P(S) \sim L1: \text{IF } PP(NS) \text{ THEN GOTO } L1;$

$NS := NS + 1;$

$\text{IF } NS \leq 1 \text{ THEN}$

BEGIN $HS := \text{FALSE};$

$L2: \text{IF } PP(NS) \text{ THEN GOTO } L2;$

END

$\text{ELSE } HS := \text{FALSE};$

/* HS skirtas KS apranga, */

$V(S) \sim L3: \text{IF } PP(NS) \text{ THEN GOTO } L3;$

$NS := NS + 1;$

$\text{IF } NS \leq 0 \text{ THEN } HS := \text{FALSE},$

$HS := \text{FALSE};$

/* pabrailtose eilutėse būtų laukimas. */

Kad būtų padidintas sistemos efektyumas rečiuk,

kad P proceso blokuoty, o V ji atsigrebtų.

Blokuoti proceso rečiukia – tuo jo naudymo atnaujinamas procesorius ir proceso P būrena įjungiamas.
1 5 6 2 3 4
blokuoty proceso išraiž pagal semaforgą S.

su $V(S)$ atnaujinciai.

$P(s)$ ir $V(s)$ programinės realizacijos

operacijoje $P(s)$, kai s bendras semaforas, ekvivalentū tokiai programinėi konstrukcijai:

$P(s) \sim$ uždrauti petraukimus;

2: IF $PP(x)$ THEN GOTO L;

$s := s - 1;$ // s - kintamasis

IF $s \leq 1$ THEN

BEGIN užblokuoto išverčianys proceso; ①

paimti proceso is sąrašo $List_A$;

$x := FALSE$,

periuko valdymo leidžiant petraukimus;

END

ELSE BEGIN $x := FALSE$;

leisti petraukimus END;

$V(s) \sim$ uždrauti petraukimus,

2: IF $PP(x)$ THEN GOTO L;

$s := s + 1$;

IF $s \leq 0$ THEN ③

BEGIN

paimti proceso is $List_B$ → $List_A$;

IF proceso išverčiamas tada vykdyti proceso

END,

$x := FALSE$;

leisti petraukimus;

- ④ tai reiškia, jog į blokuičio proceso sąrašą $List_B$, kuris s reikia nuo semaforo s, jaučiasi;
- ⑤ faltiskai tai yra planuotojo dietus.
- ⑥ naudytiname rengiųjų blokuičio semaforo s proceso.

Kad būtų galima realizuoti HOS, architektūrai reikia tam išsi reikalavimai:

- 1) turi būti petraukimų mechanizmas;
- 2) privilegiuotas režimas.

Einant privilegiuotam režimui uždrauti privilegiuotų komandų vykdymą – tai reikalavimas HOS režizavimui. HOS turi pasigyneti ta savybe, kad vienu metu dirbantys proceso (ar tai vartotojo, ar sisteminai) neturi jutakoti vienas kita;

- 3) atminties apranga.

Atminties relokacija – savybė patalpinti programingą į bet kurį atminties vietą. Tai programos nepriklausomumas nuo vietos. Tai tik efektyvumo klausimas.

Trys glynorios OS kategorijos:

- ① pakelinio apdorojimo sistemos;
- ② laiko skirstymo sistemos;

③ realaus laiko sistemos.

OS skirtumas į kategorijas remiančiu būvai
tinkamais kriterijus. Požymiai:

- 1) užduoties autorius reiškia su jo užduotimi
užduoties vykdymo metu,
- 2) sistemos reakcijos laikas į užklausimą užduo-
čią įvykdytų.

Pakelinių apdorijimo OS'je autorius neturi jokiio
ryžio su užduotimi jos vykdymo metu. Reakcijos lai-
kas matuojamas valandomis. Tokios OS efektyviau-
nės mažinos rensyų naudojimo prasmę, bet labai
neefektyviai žmogaus rensyų naudojimo prasmę.

Laiko skirtumo sistemos užtikrina pastaro rūsį
su užduotimi. Reakcijos laikas skaičiuojamas milise-
kundėmis, todėl sistemos naudojui lieka mažinines
galiomybės, lyg jis pats disponuotų ...

Realaus laiko sistemoje, skirtose greitaeigies pro-
cessų valdymui, laikas skaičiuojamas mikrosekun-
dėmis. Reikiu daug rensyų. RLS daromos su fo-
miniu rezūmu, kai sistema atlieka kiten veiksmus, kai
neturi kitių rensyų.

Operatyvių atminties valdymas

2001 11 20

OA - tai ta, į kuris procesorius gali kreiptis tieno-
giai imant komandas ar duomenis programos vykdymo
metu. Tokia schema turi daug nepatogumų, kai
programoje nurodomi absolютūs adresai.

Translatoriai buvo perdantys laip, kai gamintų ob-
jektinę programą, nenurodyti jų pataipinimo vieta, t.y.
kilnojamus objektiūs modelius (nustatant programos
adresus pagal išskirtą atminties vietą). Nuo atminties
skirtymo pererte programos vykdymo metu perėta, prie
atminties skirtymo prieš programos vykdymą. Prieš pro-
gramos vykdymą reikia nuroti ir pataipinti jų atmintys.
Tai atliekanti programa – surūstantis paluovėjas.

Kai visi darbai atliekami prieš programos vykdymą,
kalbame apie statis adresų nustatymo – statis at-
minties skirtymo. Fixinis atminties skirtymas –
kai adresai nustatomi betarpiskai liejančiųjų at-
mintių. Statisch nustatant adresus būtina prieš prog-
ramos vykdymą žinoti išskirtos atminties vietą. Fixi-
ninis – kai fixinis adresas nustatomas tienogiai
prieš kreipimąsi į tą adresą.

uderant programas tenka abstraguctis nuo atminties žodžių ir naudoti tam tikras lokalius adresus, kurie reikiav kokių sau būdų susiejami su fiziniu adresais.

Tokia lokalius adresų vieta - virtuali atmintis. Fizinių adresų vieta - reali atmintis.

Virtuali atmintis su fizine gali būti susijama statiskai arba dinamiškai.

Schematiškai pavaižuose komandy nuklyding, kuomet nebu dinaminio atminties skirtymo:

HEQ:NJ - atmintis;

K - komandy skaitliukas; → architektūra

R - registras.

L: N := HEQ:NJ;

OK := N op kodas, // OK-operacijos kodas;

ARR := N operando adres;

K := K + 1;

ADD: IF OK=1 THEN R := R + HEQ:ARR;
ELSE

SR: IF OK=2 THEN HEQ:ARR := R;

BR: IF OK=3 THEN K := ARR;

GOTO L;

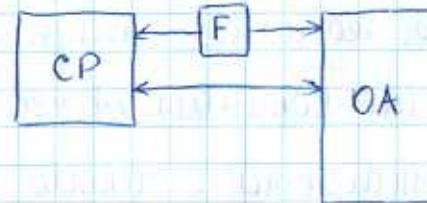
čia K, ARR vadinami efektuotais adresais

Tai yra dinaminė adresų nustatymo aparatu re, tai efektuotūs adresai yra atvaizduojami į absolūtius, (fizinius) adresus: $F(ea) = aa$.

$HEQ:NJ \sim H(F(K))$

$H[ARR] \sim H[F(ARR)]$

Y F galima žiūrėti kaip į aparaturinį bloką, jungiantį procesorių su OA. schematiškai:



Kreipimasis į OA pagal ea (efektuotus adresus):

1) aa nustatomas pagal F;

2) kreiptis į atmintį su nustatytu aa.

Nuklydant atvaizdavimus F, galimi kreipimai į OA vienig ir daugiau kartų. Nuo ištinkos fizines atminties metas priklauso tiki atvaizdavimas F, o ne pati programa.

MOS gali funkcionuoti tiek aparatu roje su statiniais, tiek su dinaminiu adresų nustatymu. Statinis efektuotinis laiko atžvilgiu, bet morūnai tankus ar

minties atvirlygi. Dinaminis adresas nustatomas yra laikinėmis.

Virtuali atmintis turi elmingą naibę – tai yra programinis interfeisas ~~matotojui~~. VA - tai tierine adresuojamų elementų seka:

$0, 1, 2, \dots, n-1$, kur $n=2^k$ – vieno segmento VA.

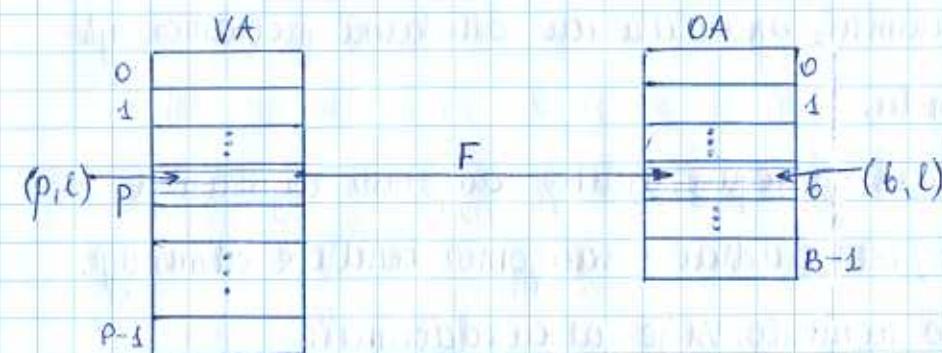
Yei mename segmente elementai adresuojami porinkiu, tai juos identifikuotume: s, w , kur s – segmentas, w – žodis segmente. Segmentas – manyras.

Puslapinių organizacijos – tai konkrečius VA realizavimo būdas. OA yra suskaidoma į menodo ilgio ištaisius blokus b_0, b_1, \dots, b_{B-1} . OA'ies adreso nurodomas kaip pora (b, l) , kur b – bloko nr., l – žodžių nr. bloko b viduje. Situacijomai, VA'ies adresine erdve suskaidoma į menodo dydžio ištaisius puslapius p_0, p_1, \dots, p_{P-1} . VA adresas nustatomas pora (p, l) .

Puslapio dydis lygus bloko dydžiui. VA adresas lygus efektiviam adresui, kurį reikiu atvaizduoti į aa.

ea \xrightarrow{F} aa. Kūsy atvejus: $(p, l) \xrightarrow{F} (b, l)$.

Schematiskai:

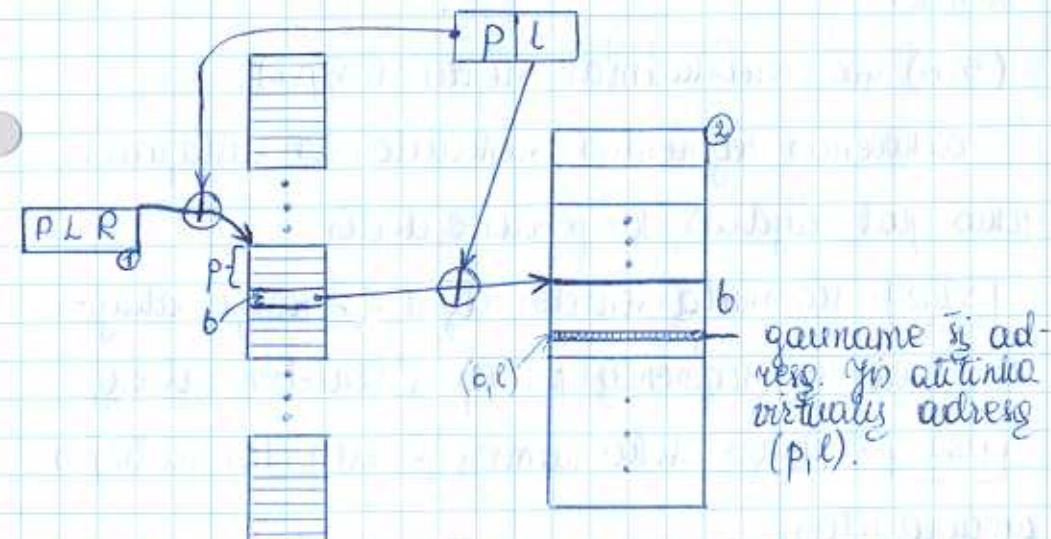


Bendra suminė VA yra daug didesnė už OA.

Puslapinių organizacijos schema:

- 1) VA'je turime $p | l$;
- 2) apvalterioje turi būti nurodytas puslapiaus lentelės registras PLR ;
- 3) puslapiaus lentelės saugomas atmintyje.

Puslapiaus lentelės ^{vežimio} dydis atitinka VA puslapiaus storias.



- ① PLR rodė, kurie yra tuo metu dirbanti pusl. lentele
② žodkai, skirti puslapiaus saugojimui.

Šie reikimai aparatūriskai atliekami programos vykdymo metu.

Kartais puslapio lentelės saugomos registrinėje atmintyje, bet bendrai - saugomos bendroje atmintyje.

Atieso segmento VA'je atvaizduojamas:

$$F(p,l) = \frac{\text{bloko nr}}{H \cdot \Sigma PLR + P3} \cdot 2^k + l.$$

Polisegmentinė VA

Ps adresų erdvė realizuojame duem būdais:

1) atmintis skirtoma segmentams kintamo ilgio blokais (dinamiskai),

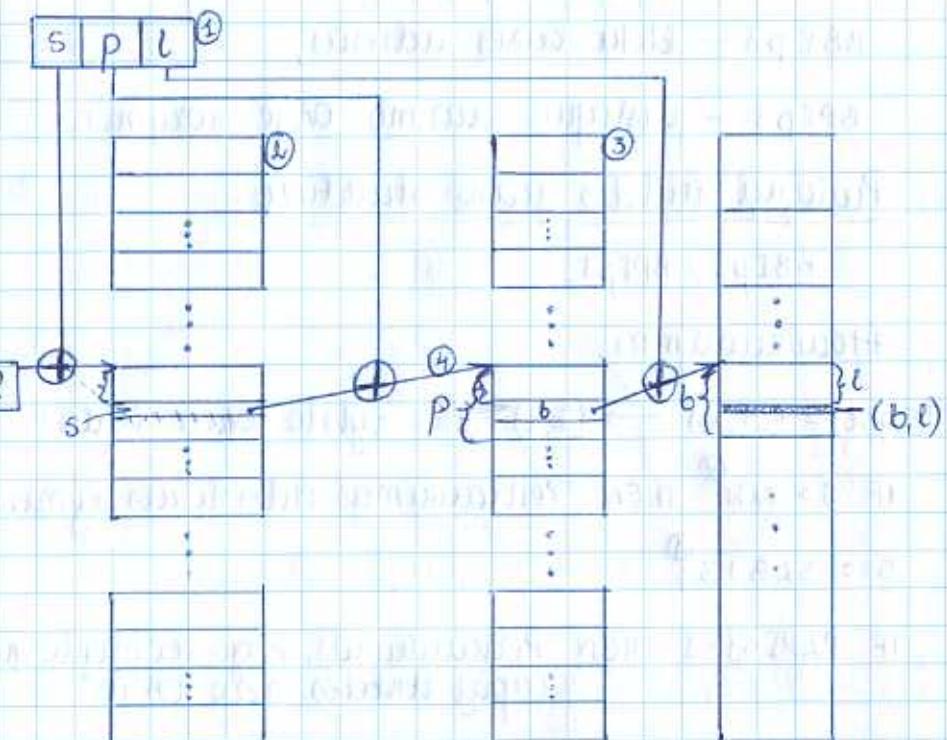
2) puslapine organizacijoje su segmentacija. Turėtume virtualy adresą, kuris išreiskiamas kaip sritis segmente.

(s, n) yra išreiskiamas frejus (s, p, l).

Kiekvienas segmentas suskaidomas puslapiais tokio pat dydžio, kaip bloko dydis.

SLR - segmentų lentelės registras saugo atlyvius proceso segmentų puslapio lentelės adresus.

PSL - saugo bloko numerį, ž kurį tas puslapis atvaizduotas.



① virtualios adresas,

② OA'je segmentų lentelės;

③ OA'je puslapio lentelės;

④ s-tojo segmento puslapio lentelės adresas;

S2B komponente - segmentų lentelės base - murodo adresą OA'je.

S2D - liek segmentų VA'je (segm lentelės dydis).

S2B ir S2D - segmentų lentelės registro struktūra.

segmentų lentelės žrāto struktūra:

PLB { SJ , PLNE SJ }.

Gali būti, kad puslapio lentelės nei OA bloke, todėl yra:

PLP { SJ - puslapio lentelės bufero OA'je pozymis },

BBE_{PJ} - bloko bares adresas;

BPE_{PJ} - puslapio buvimo OA'je pozīcija.

Puslapijas lenteles jārīo struktūra:

BBEPJ, BPEPJ.

Atloai \rightarrow adaumas;

(s,p,t) = (s,n) \rightarrow (b,l). Šis atloks tokiu būdu:

IF $s > \text{start}^{\oplus}$ THEN "Pertraukumas neleistības segmentas"

$s := \text{start} + s$ \ominus

IF $\text{PLPS} \neq 1$ THEN "Pertraukumas s-tojo segmento puslapijas lenteles nēre OA'je"

IF $p > \text{PLPS}$ THEN "Pertraukumas Neleistības puslapi"

$p := \text{PLPS} + p$;

IF $\text{BPIPJ} \neq 1$ THEN "Pertraukumas Nēre p-tojo puslapio OA'je"

$F := \text{BBEPJ} + l$; // F yib da

\oplus jei $nr > sl dydis \Rightarrow$ kļaudīga, virtuālus adresas \Rightarrow jo negalīma atbilstoši \rightarrow ad.

\ominus adresas segmentos lenteles basejē, atlikumam s.

Ne veikumi gali būt atliekami aparātu ietekmē.

Pārbaudīt tāvajā blokās srajāj.

Yei yra statīvus atminties skirstymas, tai VĀ dydis negali virsīgti OA dydi. Dinamisko atminties skirstymo atozījums izklydīmo metu puslapjai yra sunejami su-

blīvais, tādēļ pamatīgi yra pastāvīgā vīra atminīšu liekās jāsāk, bet vis vīren jās novērtās, išķēlēta defektu, tādēļ reikiās atminīšu pārskaitītē.

Statīvo lauku, kāds procesai atlaidīs atminīšu, albo pārveidītai jās atlaidītē. Pamatīgi vāndojamās pārveidības atminīšu atlaidīšanas, nes diārīšanāsā procesai atminīšu nebāndoja, bet jās ir neatlaidītē.

Programas izklydīmo reiba tāpēc, kaip jās vāndoja puslapju $z_0, z_1, z_2, \dots, z_r$ - puslapijas trase.

Tegu programas izklydmēi skurstei vi OA'hei blokē: b_1, b_2, \dots, b_m . n - skurstei ypačīgs puslapijas skurciens.

Uzdarījums netrūkstēs, kai $m \leq n$. Reikiās iusprastī, vietoj hokio puslapio pārstāsti netā manjām puslapjiem.

Kuo daudzajiem puslapjiem reikiēs leisti, kuo blagēmē puslapjiem skurstymo stratēģija. Puslapijas skurstymo uzdarījums turi atsaikytī \rightarrow klausīties :

- hokiem momentus salieisti puslapjiem;
- hokys puslapji pātelpinti jā atminē (iz ierīnes jā OA);
- vietojē hokio puslapio.

Puslapijas skurstymo stratēģija, pagal kurīs vīre - 91

ta pusiapčių surama betarpustai juočių s. j. laipiantis, vadinama strategija pagal pareikalavimą (SPP). Ji užfiksuoję atsalymus į pirmuoju dvi klausimus.

Greedyje, kad bet kokiu pusiapčiu surystymo strategijai egzistuoja neblogesnė strategija pagal pareikalavimą. Nadinasi, optimalius strategijos pareikoje galima apsiūboti SPP klasė.

SPP, kurioje išstumiami tie pusiapčiai, kuriuose bus nėra nėra pusiapčių trasa, vadinama Bellady strategija.

Prz. Tarkime, turime du blokus b_1, b_2 ir pusiapčius $p_1, p_2, p_3, p_4, p_1, p_3, p_2$. Sprendimas.

$b_1 \quad b_2$ Šioliu strategijoje yra optimali.

$p_1 \quad p_2$ Tačiau išskyla taikymo problemos, nes juočių programos naudymo pusiapčius trasa nėra žinoma.
 $p_1 \quad p_3$
 $p_1 \quad p_4$
 $p_1 \quad p_3$
 $p_1 \quad p_2$

Praktikoje naudojamos tam tikros strategijos, kai paliestiamas: atnaujinti, ilgiausiai buvantis OA'je, į kurį buvo kreipti seniausiai.

Nidutinėkai du kurtus makios pusiapčius pa-

keitimus, jei naudojame hētigj strategiję: naujas puslapis g. grąžomas metoj seniausiai naudoto.

2001 11 27

Segmentų lentelės registras saugo aktyvius procedūrų segmentų lentelės priėmės adresus. fitociaudomas ea \xrightarrow{F} aa. galelys atrodyti taip:

$$F(S, W) = H \underbrace{[STR + S]}_{\text{pos.}} + W \quad // \text{gauname aa regm. lentelės priėmės adresas atmenyje}$$

skirstant atmenys kiemo ilgio adrenes erdvės ištinkinius blokus išskyla problema. Gali buti, kad suminės laisvos atminties yra, o ištinkinio bloko - ne. Tai vadinama fragmentacijos problema.

Dinaminio atminties surystymo mechanizmas, kai yra ištinkinis adresas marycas, skirtas atminties surystmui, ir reikalingos procedūros:

GET AREA (ADDRESS, size), kur size nurodo, koks minimalus atminties kelias yra užpratomas.

Atminties atlaisinimui naudojamos dvi schemas:

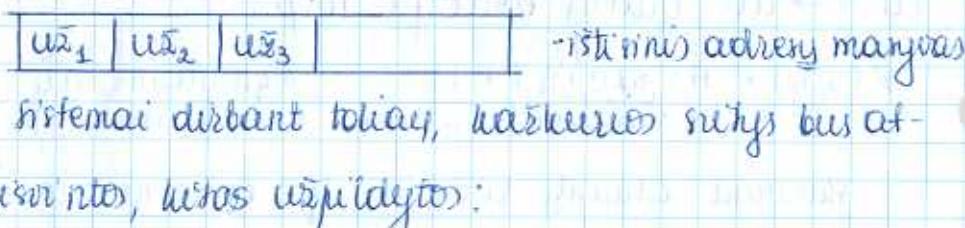
- 1) kai atmintis atlaisinama iš karto, kai j. yra nebenaujojama. Procedure:

FREE AREA (ADDRESS, size);

- 2) iš elgo atmintij neatlaisinama, bet kai prie-

reikita atminties eiliniam užklausimui patenkinti,
tuomet paleidžiama speciali procedūra

FREE GARBAGE, kuri naudojamas atminties
suto puzentuojant vieną vietoje iš dalies defragmentacijo
sistemos darbo pradžioje turėtume:

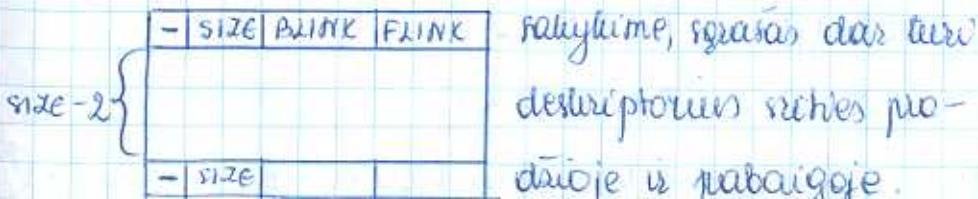


Reikia rekti, kad kebūtyg greta esančių direkinių
baimų sračius. Jei taip atsitinka, jas herime apjungti.
Tai atlieka **FREEAREA**.

Reikia turėti baimy sračius sraigt. Kai gaujamos
užklausimas laisvalaicių, reikia eiti per sraigtą iš
surašiu bent tokio, kuri bus didesnė už **SIZE**.

Ar atiduoti pirmojo pačiai kuris, ar masinaus
iš tikraneity? Būdys ypač daug.

Reikalinga struktūra - direkinių baimų sraigtai.

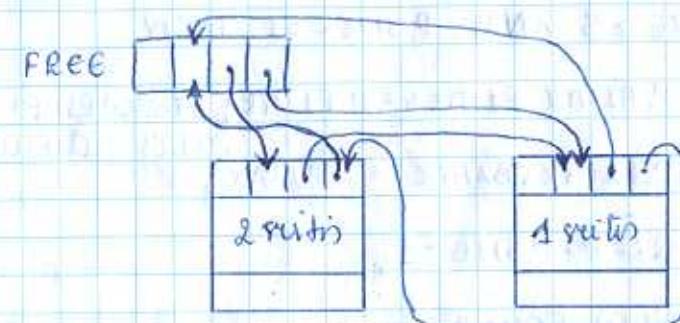


salyklime, sraigtas dar turė-
descriptoriųs sračių pro-
duoje už pabaigę.

- laisvo, + užimta, BLINK - nuoroda atgal, FLINK -
nuoroda priėmyn.

Šieji baimų sračiai užaro laivinimui:

FREE [- SIZE BLINK FLINK]



PROCEDURE GETAREA (ADDRESS, size);

BEGIN POINTER P, Q, Q1;

 INTEGER S, S1;
 // S-sukties dydis, S1 - max dydis,
 P := FREE.FLINK;

 S := S1.E + 2, S1 := S + C;

 WHILE P ≠ ADDRESS(FREE) DO

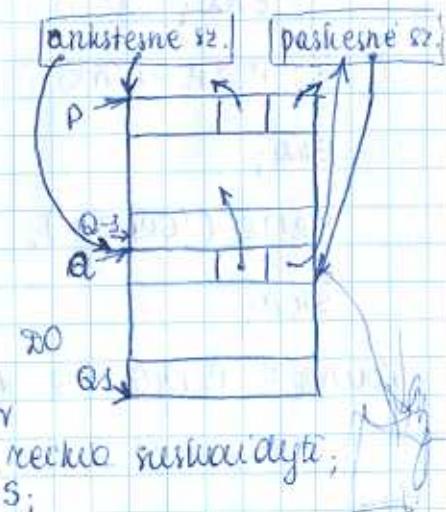
 BEGIN IF P.SIZE ≥ S1 THEN
 // jei ≥ max dydžiu S1, tai vytį reikia susluožinti;
 BEGIN S1 := P.SIZE - S;

 Q := P + S; Q1 := Q + S1 - 1;

 CONTENTS(Q) := CONTENTS(P);

 Q.SIZE := Q1.SIZE := S1;

 Q.TAG := '-';



P.FLINK.BLINK := P.BLINK.FLINK; Q = Q;

Q := Q - 1; P.SIZE := S;

GOTO FOUND;

END;

IF P.SIZE ≥ S AND P.SIZE < SI THEN

BEGIN P.BLINK.FLINK := P.FLINK; // anlitternes
P.FLINK.BLINK := P.BLINK; // rutines adheras,

Q := P + P.SIZE - 1;

GOTO FOUND;

END;

P := P.FLINK // reikšė uversta laisvo nėra; ja-
// da eina į laisvų sūčių sąraš;

END;

WRITR ('ERRR');

STOP;

FOUND: P.TAG := Q.TAG := '+';

ADDRESS := P + 1;

END;

PROCEDURE FREEAREA (ADDRESS, SIZE);

BEGIN POINTER H, U, L, Q; // - H j einaugis, U - j re-
// kantis, L - j priei, Q - j v. pabaigas;
H := ADDRESS - 1; // H := nuo rodos į atlaivinamas ruties
// tamibing žodys;
U := H + H.SIZE; // rodos iškaičiuotas ruties pabaiga;

L := H - 1; // rodos į virš atlaivinamas ruties
// tamibing žodys;

IF U.TAG = '-' THEN

BEGIN H.SIZE := H.SIZE + U.SIZE; // rutis x pajuogima;

U.BLINK.FLINK := U.FLINK; // naktinamas nuo ro-
// do;

U.FLINK.BLINK := U.BLINK;

END;

Q := H + H.SIZE - 1;

H →

IF L.TAG = '-' THEN
// jei viršutine rutis laisva
BEGIN

L := H - L.SIZE; // nuo eina mosios
// H.SIZE atimam anlitternes size;
L.SIZE := L.SIZE + H.SIZE;

Q.SIZE := L.SIZE; Q.TAG := '-';

END

ELSE // jei viršutine rutis užimta

BEGIN // tamis reformuojus laisvą sut reikia ypač
// būtiny sūčių sąraš;

Q.SIZE := H.SIZE;

H.TAG := '-';

H.FLINK := FREE.FLINK;

H.BLINK := ADDRESS(FREE),

FREE.FLINK.BLINK := H;

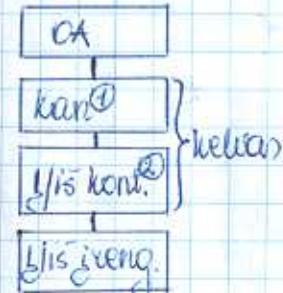
FREE.FLINK := H;

ENR

ENR:

2001 12 04

Ivedimo / išvedimo valdymui kelionėnai ju/is
įrenginių iš surūpiamųjs aptarnaujantys procesas
Pi. Jis atstovauja įrenginį sistemoje. Jis pasistisi –
inicijuoti ju/is operacijas, perduoti planenimus apie
pertraukimus i planuoti apie ju/is veikimo paba-
gos.



- ① specializuotas procesorius su savo komandos sistema (registrai, valdymo įrenginiai) perduoda duomenis tarp OA ir ju/is kontrolierio;
- ② kontrolieris – specializuotas procesorius, užtireiktas j tam tikrų tipo.

Ju/is įrenginį atstovaujančios procesas pi:

BEGIN

L: PRASYTIR (TTI^①, ŽL^②, (P^③ II prog)^④);

PRASYTIR (KK^⑤, TTI^⑥, KELIAS);

Komutacijos iš įrenginių parinkimo komando;

ju/is inicijavimas;

PRASYTIR (II P^⑦, KELIAS, PRAN);^⑧

Pertraukimo dekodavimas;

Papildomas ju/is komandas;^⑩

filtravimo informacijos;

ATLAISVINTIR (KK, KELIAS);

ATLAISVINTIR (TTI, PRAN, (P, ATS));

GOTO L

ENR;

Paauškinimai:

- ① resursas ypa. ju/is įrenginys
- ② nėra specifikuojama, kurių resursų dalies reikia,
- ③ prasidintis procesas p.
- ④ jis turėtų atlikti ju/is įrenginį aptarnaujančios procesorių (siuptyminga raišyme, failo atidaryma,...);
- ⑤ kanalo kontrolelis,
- ⑥ reikia kelių iki tolko įrenginio,
- ⑦ ju/is pertraukimas;
- ⑧ tai gali būti bet kurii iš kelių sudaromosios dalijų;
- ⑨ anksčiau ar vėliau tolko resurs bus nuldupta;
- ⑩ reikmei atveju jei nėra būtinės;

Techniškai, pertraukimo apdorojimo programa

vyksta to paties proceso aplinkoje, kuriame kito per-
traukimas. Bet abstrakčiai – PAP₂ galima traktuoti
kaip ciklinius procesus. PAP₂ laulkia pertraukimo. Kai
vyksta pertraukimas, PAP₂ iustato tarnybinius procesus,
kuri apdoroja pertraukimo iš pasiūlės atlikimamais
procesu išpranėjimą.

VH iustato ju/is komandas. Igvysta pertraukimas;^⑪

PAP_t naudojantie kaip ciklinis proceso:

I_{IP_t}: Įsiminti (CPUΣ+J);

STE_tJ := READY; // nustatome, nes tuoj maias
// processui;

PRO_tE_tJ := Ω; // nurodomė, kad processius
// gali būti atiduota kitam processui,
nustatyti P_t;

ATLAISVINTIR (PERT, (P_t, pert_t info));

BEGIN

/* Tarkime, kito pertraukimas nuo taimerio (ar bar-
gen' užduocių skirtas laikas) - užduotis nutrau-
kiamo, bet jei būgen' užduoties realizacijai skir-
tu laiko koeltas pertraukimas (vykdymas, atide-
dama). Taip I_{IP_t} veikty kaip planuotojas. */

P_t: L: PRASYTIR (PERT, LAIKAS, p),

KEISTAPP (p, PER(p)),

GOTO L

END;

Fauly sistema

FS yra OS dalis, kuri valdo jo/iš, jo/iš metu
naudojamus resursus ir operacija informacija failuo-
se. J F galima išnerti kaip į virtualy jo/iš įrenginį
toly, kuris turi patogis programuotojui struktūrą.

Programuotojui patogu operuoti failine informacija
loginiame lygyje. J failas galima išnerti kaip:

(F, e), kur F - failo vardas, e - elemento identifika-
torius failo.

Galima kaičiė apie virtualis failing atminti. FS
virtualios failines atminties paslėptis apimtinių failoje, tarpom
pagrindinos funkcijos: erode jis failas nabalansuoti.

- 1) užklaunus VFA'iai transformuojamas į RFA;
- 2) informacijos perdavimas tarp RFA ir OA.

Failines failines atminties įrenginiai apibūdinami

tam tikromis charakteristikomis:

1) talpumas - maksimalus informacijos kiekis, kuris ga-
li būti saugomas;

2) gražo dydis - minimalus informacijos kiekis, kuris
gali būti saugomas;

3) Įraiai įrenginiuose gali būti kintamo arba pasto-
vaus algis.

3) Nuėjimo būdai:

- tieroginis - operuojama aparatura;
- nuoseklus - kai priejimui prie įrašo reikalingas vien tarpinių išraiškų peržiūrėjimas,

4) FA informacijos nerejas - tomas. Tomo charakteristika - jo pakeliučiamumas - iugalina iš emės padidinti naudotojo naudojamas VA apimtis. Pvz., diskas.

5) Nuomenų perdavimo greičiai

Jis matuojamas bėtais ^{arba būtais} per sekundę perduodant informaciją tarp OA ir įrenginio.

KB - K bėtais

KB - K būtais

6) Charakteristika - užtakymas

Įrenginii gavus eilinę žvėris komandą (jei tai priestatinis įrenginys), jam reikia įnageti nuo praeito skaitymo prie naujo pradžios. Jei diskas arba bugnas - užtakymas - tai apsisekimas nuo pradžios iki šeštų metų, kur yra informacija.

7) Charakteristika - nustatymo laikas.

Tai galutėjus perstumimo laikas (diskas). Bugnai yra įrenginiai su filiuotomis galutėmis.

Priklausomai nuo įrenginio charakteristikų ir nuo jo parauðymo sijos, vien ar keli įrenginiai yra priimtinė falių tikras atvejis.

Naujaujinimas falių. Tegu yra nuoseklus falias su filiuoto išigio (80 bėty) įrašais. Čia nė atlikti skaitymo veiksmų iš falo. Naujotojas užduoda tolį veiksmų nurodydama išorinį falo vardo:

REAN1 (FN, A), kur FN - falo IV, A - OA sietis, iš kuris skaičytinė.

Skirtingi naujotojai gali nurodyti menodus fai.
Taip, kai objektai būtų identifikuojami, tenko pereiti prie VV.

REAN2 (FN, A, Z, 80); - šiuo atveju:

Z := E + 80;

Kur E - nodykile, o 80 - per keli padidinti rodymų eilės skaičymo metu.

(17) Ką prienareikiu miskai iugalina identifikuoti falo descriptories. Fai, kaip ir patys f., yra saugomi išorineje atmintyje, o aktyvių falių descriptorių saugomi tieje pačioje direktoriuje OA je.

Fai struktūra:

F varda : FN;

F padeto : įrenginio adresas, failo pradžios ad-
resas įrenginyje;

F organizacijoje : nuosekli;

F ilgis : L;

F tipas : ... (pastovus, laikinas);

F savininkas : U;

F naudotojai : {U};

F aprašymas : READ (siurtais tek skaičiymui);

Kad galima būtų atlikti veiksmą READ, reikiu
nurodoti failo su IV = FN descriptoriu. Toks FJ nura-
dimas žinomas kaip failo atidarymo veikmas. FJ
nurodymas gali būti užduotas tokiu fragmentu:

open := false;

if yra AFK(FN, d) THEN open := true

// Jei F yra AFK' e, tai gerokai TRUE iš FJ geruoja d;
// AFK - atlygys failys kataloge, d - FJ;

else // neikioti sisteminiame kataloge (FN, fn);

// kai nuroda, FIV geruoja fn;

if !open then

begin gauti descriptorius (fn, d);

įjungti į AFK(d);

end;

(104)

FJ naudojamas patikrinimui:

if ! (U ∈ d(nauj) ∧ REAR ∈ d(aprauga) ∧ 2+79=d(ilgis))

/* jei vartotojas, kuris dare užklausimo, priklauso
descriptoriuje nurodytis vartotojo grupei, tai ge-
rai */

then KLAIDA;

REAR 3 (d, 1, 2, 80);

/* datuai yra iš veiksmuose naudojama buferizacija,
tai, kai ne bu buferiuose tokie užklausimai atlieki
funkcijai yra iš operacijo. Buferizacija - sumazinti
kreipimyni į išvystę atminty slenčius. */

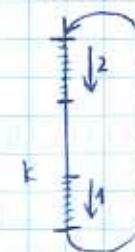
GET BUF (k); // k - buferio indeksas;
// gerokai sudėtinga rehanejo užpildyto buferio;
MOVE(BUF[k], A); // persiųsti gautus buters į CA lauką A;
RELEASE (BUF[k]); // atlaivinti paraudotą bufer;

su failo buferizacija gali būti sinchronizuotas pro-
cessas, kuris užpildo buferius išsinehe informacija kai tik
atlaivinamas palankiamas buferis slenčia.

Atkreipimui nurodoma 800 byte. Kai atlaivini-
ama 10 buferių, užduodamas veiksmą:

READ4 (įrenginys, geruoja adresas įrenginyje; BUF[0], 800),

← buferiu kūnas (užklaus). Jei atlaivinami
buferiai nuo k-tojo, tai iuvame buferius
pagal rodylites 1, 2.



(105)

grāto adieras := grāto adieras + 800; } norint pereiti
 $k := k + 10 \text{ (mod } n\text{)};$
 pie kitō buferio.

//n - buferius po 80 baičių skaičius;
 FS funkcijos pagal apibūdinti pagal jų tipus;
 pradedant nuo aparaturinio iki vartotojo serviso klaušinių, FS suskirstymas į loginius tipus:

DBVS ⑤

priejmo ④ metodai

virtuali ③ (loginė) FS

reali (ba- ② zine) FS

žv. iš sis ① tema

2001 12 11

Teigul yra failas A. sumdedo. iš 11 žrąs, kurios
 kiekvienas 250 baičių.

Saugoma blokai po 1000 baičių kiekviename
 bloke, t.y. iš vienos bloko telpa 4 loginiai grātai.

Paaistiškūmai:

- ① koordinuoti fizinių įrenginių darbą.
 Šis tipio procedūrai atlieka informacijos blokų apibrėžimą kaip OA ir rezines atminties pagal išduoto adresą;
- ② transformuoti failo vartų unikalių identifikatorius į FA;
- ③ pagal vartotojo nuteikto IV nustato jo vartų unikalių vardo. Naudojamas vartotojo failų katalogas. Virtuelius tipus nepriklauso nuo fizinių įrenginių;
- ④ realiuosiai dėmų, pagal kurius apibrėžiami failo grātai. Toks dėmų išduoda vartotojas pagal pripačius parames.
 Pvz. 1 nurodulės priejimas arba kai grātai apibrėžami pagal lauko (rauto) rezimės didejimo ar mažejimo tvarka;
- ⑤ realiuosiai failo logines struktūres vartodami į fizinių struktūrus.

A											
1000B											
0	1	2	3	4	5	6	7	8	9	10	11
5	6	7	8								
10	11	12	13	14							

Grāto A grātai pradedo autraumė bloke žarchi-
 me, programoje yra užklausimas nuskaityti failo A

grāto 6 :

READ FILE (A) RECORD (6) SIZE (250) LOCATION (3UF),

transliatoriui, nėra kito operatorių, pervez prie keli-
 jinį :

- 1) nustatyti fizinių failo A adresą;
- 2) nustatyti blokų, g kurių ženklas grātas 6;
- 3) nuskaityti blokų į OA;
- 4) persiųsti iš OA nuskaityto bloko 6-jis grātas į sutų BUF.

Kad būtų nustatytas failo A pradišios fizinių adre-
 sas, yra naudojama tomo turinio lentelė VTOL. Tomo
 turinys, kaip failas, yra saugomas tame pačiame
 bloke. Schematiskai :

vardas	adresas
A	2950
B	900
C	2000

← pagal mūsų pažadą (1),
 ← nėra pavyzdžiai;

Kad tolis užtvarimai failinei sistemoi būty pa-
tenkintos:

- 1) VTOL'je nustatyti graž \downarrow ,
- 2) nustatyti santykinį adresą: $(j_2 \cdot n_2 - 1) * j_2 \cdot dydis = 1250$,
- 3) pagal santykinį adresą nustatyti, kuris failo blokas
reikalingas (mūsų atveju 1),
- 4) nustatyti fizinių blokų, kurieks priklauso pirmas
(nuo atveju) santykinių blokų (mūsų atveju 3)
- 5) trečias fizinius blokus nuskaitomos iš OA pagal
skaičymo reikmę,
- 6) iš nuskaitytos bloko į bufferį priimami burtai nuo
200 iki 489, o jei persunkiuoti į sutį su vardu
buf.

Norbo nu failais metu liekureus karto reikiuti
bloko nu failo pradžia - neefektyu. Todėl tas gražas
perkeliamas į failų katalogą iš tomo trankio leidelių
iš vadiniamas atidarymo reikmė (desriptorių pes-
keriuos iš OA).

Einant nuosekliai gražų apdorojimui, galime
numaiinti juos iš reikmų skaičių. Mūsų atveju mes
maiintome 3 kartus.

Panaudoto būreti poāymys, kuriis blokas yra bufferiale,
kad daugiau nebūtų skaitomi iš failo. Metodas
bufferizacijos. Kiekvienam failui gali būti ravers bufe-
ris arba visiems failams vienas bufferis, arba minkš-
tai failui - keli bufferiai.

5.1) Autostartuojančio FS sukuriant failą reikėjo nuro-
dyti jo dydį. Tai nepatogu, todėl svarkerai turėtė gali-
mybę failings atnaujinti skirtysti dinamiskai. Tam
realizuoti gali būti naudotas metodas su faili-
nemis leideliemis.

Nauzdinjamie failius nuteinę, kuri dirba su faili-
nemis leideliumis. Nulinis blokas yra stūktos tomo
turinė. Jame turiame nurodyta failo vardo, o
failo pradžia - tai ne f santykiniuo nulinio blo-
ko adresas, o failo leidelės (desriptorių) adresas.

F je išvardinti failo santykinių blokų vs
paraiškė, į kurius fizinius blokus jie yra atvaiz-
duoti.

VTOZ	1	1	0	2	0	14	3	ALPHA MULTIUS BLOKAS	4	0	3
	2	1	-	1	13				5	1	7
	5	2	-	2	12				6	2	8
	4	3	-	3	11				7	3	-
				4	9				8	4	-
5	0	6	6	7		8			9		
1	10	GAMMA		ALPHA		ALPHA			10		
2	-	MULTIUS		PYRMAS		autras			11		
3	-	BLOKAS		BLOKAS		BLOKAS			12		
4	-								13		
10			11		12		13		14		
GAMMA		haisbos		haisbos		haisbos					
prima)		atominis		atominis		atominis					
blokas		3 blokas		2 blokas		4 blokas					
15			16		17		18		19		

- ① haisbos atominis f2 yra autraus bloke (xiūniam).
- ② VTOZ yra 1-me xiūniam bloke.
- ③ VTOZ multius sautylusis blokas yra multiusis hainiaus bloke.

Failo niciavimo metu sukuriamais grafas turiu leideliu ir blokas faily leideliu. Naudojant failo TTL nepakanka vertikaliu slėtingumu varotojams failys su neuodais varda issaugojim. Turi reikiu is TTL išbrauktis failo vardo, kieturenuu varotoju sekuris specialys failas, kuriamis-varotojo sekurias failas su turiu leideliu reikiame. TTLje failas yra unikalios failo varda sistemoje.

Failo varda susijauja su failys leideliu atgalis descriptoriu:

FS realizacijos schema, kuri lothakuoje

FV kieturenuu varotoju atskirai
F je nodelis f-ly leideliu yse f-to identifikac.



- ① f-to descriptoris naudomas 2-ame bloke
- ② failo turinio leid. duode nurodo f failys leideliu
- ③ pagrudiuniam zinyma naudoma fido apie user
- ④ 8-je failo fes pozic. yra nurodo 3 JONO ZINYMUS
- ⑤ F-je ALFA unikalus varda yra 7
- ⑥ kasnokiuose blokuose yra failas
- ⑦ 3-me bloke yra ALFA descriptoris
- ⑧ 10 unikalus varda yra 12 (4-aje eilut.)



File system sudaro procesorai naudantys S/15 darba f-ly - gidas
procesorai krypti failp moce
si atobulos zini failp
zineptiniu kryptis failp
zineptiniu failsporadic
failas krypti failp (no
ceny) fail spalv
failius zinepti failp proce
darbas naudomas kai (11)

- 8-je faili privatumas.
- 1) asmenininis zinepti
darbas naudomas kai (11)

2) Nutitauromybe meo cengaisis specifitas, koto-
liuojame remontuame, hypyc. k. kinti legas
merkei uupi tuo ha rufo.

3) ypatius si teurys o bauhauens pertuev,
apdozopales aktiuuus?

so/18 cintuus mukedele iF poništeunis
+ tipo graipi arbus.

Basius NS Aphato so/18 mukuo redintys
meo os chines dalius so/18 apakainos
specifig. Puer pantucenti rapi olois
derile nissabyn OA adies 12 valus ads
mukuanie sruo Toly aurore 12 informuys
baprie F1S natai F1 (fai. de) 80% BF1S
valdo 1502 atuunies tounus (2 apodroja tony
faile descriptonis, b6n)

7. junes BHVs tiglueus komaudes

Suburto descripton (sutis) (is hau blokis)
je grybduus ilukatireigant roeis apas-
nauant, tony keu - hau lauus afuerbes
failes standouas y daliuus failes.

Dalunti (faile descript, sutis), k. hau marki-
nuuas failes, jo atuunti atlai smuuvia.

17 grybduus s lauus atu. faile;
Isplesti (faile descript, sutis);

Lopine FVS. Atvaroduoje lokalius
USER odarus i mukuanis faile jateisti hal
LFVs realistis, darbo su t. mukaudes.
Genuei jaun atuuliunas paruus fvs
mukaudes, mukose jaun mukodouas muk-
les fvs id.

LFVs susieje USER i id V su U. F id.

Varktojo mukas mukaudo jamas kai
mukelkum

Temos

1. OS sąvoka
2. OS kategorijos
3. Multiprogramavimo sąvokos
4. Virtualios mašinos sąvoka
5. Lygiagretūs procesai
6. Kritinė sekcija
7. Kritinės sekcijos apsauga
8. Deterio algoritmas
9. Semaforai
10. Procesų gamintojas ir naudotojas, sąveika
11. Įvedimo/išvedimo spoolerio bendra charakteristika
12. I/O spoolerio pagrindinis procesas
13. I/O spoolerio įvedimo ir skaitymo procesai
14. Dvejetainių ir bendrų semaforų sąryšis
15. Operacijų su semaforais realizacija
16. Procesų ir resursų sąvokos
17. Proceso deskriptorius
18. Resurso deskriptorius
19. Primitivas kurti procesą
20. Primitivas naikinti procesą
21. Primitivas stabdyti procesą
22. Primitivas kurti procesą
23. Primitivas aktyvuoti procesą
24. Primitivas keisti proceso prioriteta
25. Primitivas kurti resursą
26. Primitivas naikinti resursą
27. Primitivas prašyti resurso
28. Primitivas atlaisvinti resursą
29. Primitivas proceso planuotojas
30. Primitivas pasiruošusio proceso su aukščiausiu prioritetu nustatymas ir neaktyvaus procesoriaus radimas
31. Procesų būsenų schema.
32. Virtualios atminties sąvoka
33. Komandos vykdymo schema
34. Puslapinė organizacija
35. Polisegmentinė virtuali atmintis
36. Atminties skirtumo puslapiais strategijos
37. Atminties išskyrimas ištisinėmis sritimis
38. Kintamo dydžio ištisinėjimų atminties sričių grąžinimas
39. Gijų samprata, vartotojo ir branduolio gijos
40. Mikrobranduolio architektūra
41. Įvedimo/Išvedimo procesai
42. Failų sistemos sąvoka
43. Failinės atminties įrenginių charakteristika
44. Failų deskriptorius, aktyvių failų katalogas
45. Užklausimo failinei sistemai pavyzdžiai
46. Failų sistemos hierarchinis modelis
47. Failų sistemos Įvedimo/Išvedimo posistemė
48. Bazinė failų valdymo sistema
49. Loginė failų valdymo sistema
50. Priėjimo metodai

1. OS – tai organizuota programų sistema, kuri veikia kaip interfeisas tarp kompiuterio ir vartotojo.

OS sudaro visa tai, kas vykdoma supervizoriaus režime.

Funkcijos: 1) suteikia OS vartotojams tam tikrą servisą 2) valdo resursų skirtymą efektyvū resursų naudojimą.

2. Nusistovėjusios grynosios OS kategorijos: 1) paketinio apdorojimo sistemos 2) laiko skirstymo (kolektyvinio naudojimo) sistemos 3) realaus laiko sistemos
3. Multiprogramavimas – savybė vienu metu vykdyti kelias užduotis.

Tam reikia: 1) pertraukimo mechanizmo, 2) privilegiuoto (supervizoriaus) režimo 3) atminties apsaugos.

4. Virtuali mašina – virtuali realios mašinos kopija. Ji paslepia realios mašinos realizacija po virtualiais komponentais, kurie reikalingi kokioms nors užduotims atlikti Supaprastina nepatogią ir sudėtingą vartotojo sąsaja.
5. Lygiagretūs procesai – procesai vykstantys vienu metu ir nepriklausomai vienas nuo kito.
6. **Kritinė sekcija (KS)** - proceso kodo dalis, reikalaujanti bendrų resursų panaudojimo vienu metu. Kad išvengtų užblokovimo, du susiję procesai negali būti savo kritinėse sekcijose tuo pačiu metu. Visos kritinės sekcijos lygiavertės. Procesų vykdymo laikas bet koks. Programa gali būti užbaigta tik už kritinės sekcijos ribų.
7. Deterio algoritmas – $C_i = \text{false}$. Įvykdomas noras vykdyti kritinę sekciją, jei kiti neturi ketinimų vykdyti kritinę sekciją. Bet gali būti, kad vienu metu norės vykdyti keletą procesų KS. Tada įvedamas bendras kintamasis EILĖ. Jei procesas neturi eilės vykdyti kritinę sekciją, jis atsistako šito savo noro.
8. Semaforai – sveikas neneigiamas kintamasis, su kuriuo galima atlikti dvi operacijas: wait (dar vadinama P) ir signal (dar vadinama V). Įėjimas į KS kontroliuojamas **wait** operacija; apie išėjimą iš KS signalizuojama **signal**. Šios operacijos yra nedalomos ir vykdymo negalima nutrauktii ar vykdymo metu kreiptis į tą semaforą. Kiekvienam semaforui sistema sudaro jo laukiančių procesų eilę. Kitas privalumas - OS perveda laukiantį procesą į, pvz., būseną "sustabdytas", t.y. kol vyksta laukimas procesoriaus laikas nenaudojamas.

wait(s):

```
if s > 0  
then s := s - 1  
else {užblokuoti kviečiantį procesą};  
wait (s);{ KS } ; signal (s)
```

signal(s):

```
if {yra užblokuotas procesas (-ai)}  
then {pradėti kurį nors procesą}  
else s := s + 1;
```

9. Procesas gamintojas sukuria informaciją ir patalpina ją į buferį, o lygiagrečiai veikia kitas procesas naudotojas paima informaciją iš buferio ir apdoroja. Naudojami tam du semaforai T – tuščių buferių semaforas U – užimtų buferių semaforas. Kadangi buferis bendras resursas abiem procesams, tai jis yra kritinė sekcija B, sauganti nuo kolizijų.
10. OS tai pat turi vadinamą SPOOL sistemą periferinių sistemų on-line'e. Jis skirtas I/O įrenginių virtualizavimui.

SPOOL – lygiagrečiai veikiančių procesų visuma. Buferio spoolė organizuojami trys sąrašai: a) laisvi buferiai b) įvedimo buferiai c) išvedimo buferiai

Proceso I f-ja: Paima buferį iš laisvų buferių sąrašo, pradeda skaitymo procesą ir duoda ja laisvą buferi.

Proceso O f-ja: imai buferį iš išvedimo buferio sąrašo, paleidžiaW ir jam perduoda paimtą išvedimo buferi

11. Bet kuris semaforas gali būti išreikštas dvejetainiu semaforu.

NS,M,D NS-kintamasis, M,D-dvejetainiai semaforai. P(S) – kritinę sekciją saugantis semaforas, kurio reikšmę modeliuoja NS.

12. Procesas – operacinės sistemos primityvas, apibrėžiamas labai skirtingai – nuo „savarankiškai vykdoma programa“ iki „tai, kas sukuria fork(2) arba clone(2)“.

Resursai yra tai, dėl ko procesas yra blokuojamas.

Proceso savyoka su dvimi pagrindinėmis charakteristikom:

1) visų resursų rinkinio valdymas. Tai sudaro sudėtingą procesų kontekstą. Toks pakeitimas – ilgai trunkanti operacija. Iš kitos pusės net ir esant tam pačiam kontekstui, reikia vykdyti skirtinges kontekstus, todėl reikalinga skirtingu instrukcijų sekė. Tokia instrukcijų valdymo sekė vadinama gija.

13. Virtuali atmintis yra realios atminties modelis, kuris supaprastina adreso transliaciją. Jis turi savyje patogumo elementą. Jeigu anksčiau turėjo pats programuotojas rūpintis kaip skirstyt atmintį, o tam reikia aukštos kvalifikacijos. OS perėmus šį procesą, ji pateikia patogų interfeisą, kiekvienas vartotojas turi didelę atskirą virtualią atmintį. VR gali būti saugoma su RAM arba statiskai, arba dinamiškai. Yra svarbu, kad programos adresų nustatymas pagal fizinę vietą būtų atliekamas architektūriškai.
14. Puslapinė organizacija – konkretus VR įgyvendinimo būdas. Vienas VR puslapis atvaizduojamas į vieną atminties bloką. Čia fragmentacijos problema nėra aktuali. Antras būdas puslapinė organizacija su segmentacija. Architektūroje turi būti numatyta segmentų lentelės registras, kuris rodo į aktyvaus proceso segmentą, o jau jis turi savo PLR.
15. Galima apibrėžti daugelio segmentų virtualią atmintį. Tada jis būtų identifikuojamas nurodant segmentą ir žodžiame.
16. Visų resursų rinkinio valdymas sudaro gana sudėtingą procesų kontekstą. Toks pakeitimas – ilgai trunkantis operacija. Iš kitos pusės net ir esant tam pačiam kontekstui reikia vykdyti skirtinges kontekstus, todėl reikia skirtingu instrukcijų sekė. Tokia instrukcijų valdymo sekė vadinama gijos (thread). Sukurti, sunaikinti, pereiti, komunuoti tarp gijų yra daug greičiau nei tarp procesų. Gijos turi būsenas ir jų vykdymas gali būti, o kartais ir turi būti sinchronizuotas. Neturi pristabdymo būsenos. Su gijomis asocijuojamos operacijos. Operacijos: Create – kai sukuriamas procesas, automatiškai sukuriama ir gija. Gija gali sukurti naują giją proceso viduje. Finish – gijos darbo užbaigimas. Block. Unblock – patalpinama į pasiruošusių gijų sąrašą. Gijų blokavimas gali užblokuoti ir procesą.

Vartotojo gijos. Šių gijų perjungimui nereikia branduolio įsikišimo. Gijų biblioteka nepriklauso nuo OS įsikišimo, tai taikomojo lygmens programa. O vartotojo gijų trūkumai – jei gija blokuoja procesą, blokuojamos ir kitos proceso gijos. Vartotojo gijos negali išnaudoti daugiau procesorinę sistemą.

Branduolio gijos. Jų valdymas atliekamas branduolyje. Jei taikomojoje programoje sukurtos gijos priklauso vienam procesui ir turi tą patį kontekstą, tai branduolys valdo visas gijas ir įveikia visus minėtus trūkumus, iššauktus vartotojo gijų. Branduolio gijų skaičius ribotas.

Jei persijungimai vyksta vartotojų gijų ribose, tai užtenka vartotojo gijų, jei kviečiamas sisteminis procesas, tai naudojamos branduolio gijos.

17. Mikrobranduolys yra OS nedidelė atminties dalis, įgalinanti OS modulinį plėtimą. Nėra vieningos mikrobranduolio sandaros. Problema yra driver'iai, juos reikia padaryt efektyvius. I driver'į galima žiūrėti kaip į virtualų įrenginį, kuris palengvina įrenginio valdymą, pateikdamas patogesnį interfeisą. Kitas klausimas, kur vyksta procesai, ar branduolio erdvėj, ar už jo ribų. Pirmos OS buvo monolitinės, kur viena procedūra galėjo iškvesti bet kokią kitą procedūrą. Tai tapo kliūtimi augant OS. Buvo įvesta OS sluoksninė architektūra. Sudėtingumas nuo to nedingo. Kiekvienas sluoksnis gana didelis. Pakeitimai vienam sluoksnje iššaukia pakeitimus ir gretimuose sluoksniuose. Sunku kurti versijas pagal konkrečią konfigūraciją. Sunku spresti saugumo problemas dėl gretimų sluoksnų sąveikos.

Mikrobranduolio sistemos atveju visi servisai perkelti į vartotojo sritį. Jie sąveikauja tarpusavyje ir su branduoliu. Tai horizontali architektūra. Jie sąveikauja per pranešimus, perduodamus per branduoli. Branduolio funkcija tampa pranešimo perdavimas ir priėjimas prie aparatūros.

Mikrobranduolio architektūros pranšumai: 1) vieningas interfeisas 2) Išplečiamumas 3) Lankstumas 4) Pernešamumas(Portability) 5) Patikimumas 6) Tinkamumas realizuoti paskirytas (išskirstytas) sistemas. Neigiamo savybė – nepakankamas našumas, kalta pranešimų sistema, Ji reikia pernešti, perduoti, gavus atkoduoti. Atsiranda daug perjungimų tarp vartotojo ir supervizoriaus režimų.

18. I/O ir perdavimo valdymo sistema. Architektūra įgalina pertraukimus realizuoti kaip pranešimus. Branduolys atpažista pertraukimus, bet jų neapdoroja, o perduoda pranešimą vartotojo procesui, kuris atsako už interrupt'ą. Pačius įrenginius galima traktuoti kaip gijas, turinčias identifikatorius ir siunčiantiems pranešimus vartotojo erdvėje. I/O valdymui kiekvienam I/O įrenginiui sukuriamas jų aptarnaujantys procesai.
19. **Failų visuma** su visais jų tarpusavio ryšiais ir nusako failų sistemą. Talpumas – informacijos kiekis, kurį gali saugoti įrenginys. **Irašo dydis** – informacijos kiekis, kuris gali būti įrašytas, atpažintas. **Failas** – turintivardą sutvarkyta elementų seka. Failų sistemos funkcijos: 1) Užklausimų virtualiai failinei atminčiai konvertavimas į realią atmintį. 2) Failo info perdavimas tarp realios ir virtualios failinės atminties.
20. /*Tomai – diskeliai, cd-rom. Užlaikymas – timeout. Nustatymo laikas – galvučių perstūmimo laikas. Diskiniai įrenginiai : 1) diskinės atminties talpa 2) perdavimo, apsikeitimo greitis.

RAID – Redundant Array of Inexpensive (Independant) Disk. RAID0-RAID6 – RAID'ų standartai. */ ?

21. Failų deskriptorius FN – failo vardas, FA – failo padėties įrenginyje. Nuosekli failų organizacija. L – ilgis, laikinas tipas, U – savininkas, {U} – naudotojai, READ – apsauga. Failų deskriptoriai saugomi diske kaip ir failai Aktyvių failų deskriptorius įtraukiama į aktyvių failų katalogą (AFK) operatyvioje atmintyje. Vartotojo procesai – DBVS (duombazė)->priėjimo metodas->virtuali/loginė sistema <->OS procesai->bazinė/reali sistema->I/O sistema.
22. Hierarchinį modelį galima įsivaizduot kaip medį sudarytą iš failų, kai kurie iš jų yra direktoriujos. Yra viena išimtis, kiekvienas failas ar direktorija gali save rasti priskirtą tik vienoje šakoje. Išimtis tik root direktorija. Ji nors nėra tiesioginiai sujungta su jokia šaka, tačiau yra fiktyviai prijungta prie failų sistemas. Kiekvienas žemiau hierarchijoje esantis failas yra žemesnio rango nei aukščiau stovintis.
23. I/O sistemą sudaro procesai valdantys I/O, kiekvienam įrenginiui atskiras procesas. Ryšys tarp procesų atitinka ryšį tarp įrenginių. Privalumai: 1) asynchroninis darbas, valdomas kaip nepriklausomas procesas ir įrenginių greičio skirtumai nereikalauja specialių OS veiksmų. 2) Ypatingos situacijos įrenginio darbe apdorojamos artimiausiai įr. lygyje.
24. Bazinė failų sistema transformuoja vidinį virtualaus failo adresą į išorinį (hardware) adresą. Darbui su failais turi funkcijas: SUKURTI (failo vardas, sritis), DALINTI(-||-,||-) - sukuriamas dalinis failas, IŠPLĒSTI (-||-,||-) - padidinti failo atmintį ATLAISVINTI (-||-,||-)
25. Loginė failų valdymo sistema atvaizduoja failų vardus į identifikatorius. Loginė sistema reikalauja pranešimo apie darbo su failu pradžią, tada atidaro failą. Kiekvienas vartotojas sistemoje identifikuojamas vartotojo deskriptoriumi, kuriame yra informacija apie vartotojo failus. Loginė failų sistema pateikia komandas, kurioms realizuoti kreipiasi į bazinės failų sistemos komandas: REGISTRUOTI(f), SUKURTI

(fv, [tomo vardas]), SUNAIKINTI (fv), ATIDARYTI (fv), UŽDARYTI (fv), SKAITYTI (fv, bl.nr, OA, bl.nr), RAŠYTI(-||-, -||-, -||-, -||-, -||-, -||-)

26. Priejimo būdai: 1) tiesioginis – kai operuojama aparatūriniu įrenginio adresu. 2) nuoseklus – kai reikalingas visų kitų praeinamų įrenginių įrašų peržiūrėjimas. Ne tai/

1) baziniai arba su eilėmis 2) tiesioginiai arba nuoseklūs.

Bazinis – vartotojas pats turi dirbti su buferiu.

Su eilėmis – visi įrašai siunčiami į programą iš eilės.

Nuoseklūs – visi įrašai skaitomi iš eilės.

Tiesioginis – galima kreiptis į bet kurią failo dalį.

http://vejas.pit.ktu.lt/~os/Atminties_valdymas.htm atmintis

1. Operacinės sistemos sąvoka. OS – tai organizuota programų visuma, kuri veikia kaip interfeisas tarp kompiuterio aparatūros ir vartotojo. OS sudaro tai kas vykdoma supervizoriaus režime. Ji aprūpina vartotojus priemonių rinkiniu, projektavimo ir programavimo palengvinimui, programų saugojimui ir vykdymui, ir tuo pat metu valdo resursų pasiskirstymą, kad būtų užtikrintas efektyvus darbas. OS – tai programa kuri modeliuoja kelių virtualių mašinų darbą, vienoje realioje mašinoje(projektuoja VM į RM). OS branduolyje yra priemonės, kurių pagalba realizuojamas sinchronizuotas procesorių, OA ir periferinių įrenginių darbas. OS turi tenkinti tokius reikalavimus: 1) patikimumas – sistema turėtų būti mažu mažiausiai tokia patikima, kaip aparatūra. Klaidos atveju, programiniame arba aparatūriname lygmenyje, sistema turi rasti klaidą ir pabandyti ją ištaisyti arba minimizuoti nuostolius. 2) apsauga – apsaugoti vartotoją kitų vartotojų atžvilgiu.

2. OS kategorijos. Yra trys grynosios OS kategorijos. Skirstymas į jas remiasi šiais kriterijais:

- 1.užduoties autoriaus sąveika su jo užduotimi pastorosios vykdymo metu;
- 2.sistemos reakcijos laikas į užklausą užduočiai vykdyti.

Kategorijos:

1. Paketinio apdorojimo OS. tai sistema, kurioje užduotys pateikamos apdirbimui paketų pavidale įvedimo įrenginiuose. Užduotis autorius neturi rygio su užduotimi jos vykdymo metu. Sistemos reakcijos laikas matuojamas valandomis. Tokios OS yra efektyviausios mašinos resursų naudojimo prasme, bet labai neefektyvios žmogaus resursų atžvilgiu.

2.Laiko skirstymo OS. Užtikrina užduotis autorui pastovų ryšį su užduotimi. Ji leidžia vienu metu aptarnauti keletą vartotojų. Kiekvienam vartotojo procesui „kompiuteris“ suteikiamas nedideliam laiko kvantui, kuris matuojamas milisekundemis. Jei procesas neužsibaigę tol, kol baigėsi jo kvantas, tai jis pertraukiamas ir pastatomas į laukiančiųjų eilę, užleidžiant „kompiuterį“ kitam procesui.

3.Realaus laiko OS. Jos paskirtis – valdyti greitaeigius procesorius (pvz: skrydžio valdymas). Sistema turi pastovų ryšį su užduotimi užduoties vykdymo metu. Jos reikalauja papildomų resursų(prioritetinių). Čia labai griežti reikalavimai procesų trukmei. Būtina spėti sureaguoti į visus pakitimus, kad nei vieno proceso nei vienas signalas nebūtų praleistas. Reakcijos laikas matuojamas mikrosekundėmis.

Visos šios sistemos pasižymi multiprogramavimu – galimybę vienu metu vykdyti kelias užduotis.

3. Multiprogramavimo sąvoka. Multiprogramavimas atsirado kaip idėja, kuri turėjo reaguoti į skirtinges procesoriaus bei periferijos greičius. Multiprograminė operacinė sistema(MOS) – viena operacinių sistemų rūšių. Šio tipo operacinė sistema užtikrina kelių užduočių lygiagretę vykdymą, t.y. leidžia operatyvioje atmintyje būti kelioms vartotojo programoms, skirstydama procesoriaus laiką, atminties vietą ir kitus resursus aktyvioms vartotojo užduotims. Multiprograminės operacinės sistemos privalumai yra akivaizdūs. Vartotojui vienu metu paprastai neužtenka vienos aktyvios programos. Tai ypač akivaizdu,kai programa vykdo ilgus skaičiavimus ir tik kartais prašo įvesti duomenis. Tuo metu vartotojas yra priverstas stebėti užduoties vykdymą ir tampa pasyviu.

Tam, kad galima būtų realizuoti MOS, kompiuterio architektūrai keliami tam tikri reikalavimai:

1. turi būti pertraukimų mechanizmas(jei jo nebūtų, liktų interpretavimo mechanizmas).
2. turi būti privilegijuotas režimas, t.y. esant privilegijuotam režimui uždrausti neprivilegijuotų komandų vykdymą – reikalavimas MOS realizacijai. Priešingu atveju būtų labai ilgas darbas. MOS turi pasižymeti ta savybe, kad vienu metu dirbantys procesai neturi įtakoti vieni kitų(ar tai sisteminiai, ar vartotojo)
3. atminties apsauga. Jei vykdant komandą suformuojamas adresas, išeinantis iš komandoms skirtos adresų erdvės – suformuojamas pertraukimas.
4. Papildoma savybė – relocacija – tai programos patalpinimas į bet kokią atminties vietą, t.y. programos vykdymas gali būti pratęstas ją patalpinus į kitą atminties vietą. Tai efektyvumo klausimas.

MOS yra populiausias šio laikmečio operacinių sistemų tipas. MOS – kai vienam vartotojui suteikiama galimybė vienu metu daryti kelis darbus.

4. Virtualios mašinos sąvoka. Reali mašina - tai kompiuteris. Užduotis susideda iš programos, startinių duomenų ir vykdymo parametrų. Rašyti programą realiai mašinai būtų sudėtinga ir nepatogu. Todėl vienas iš operacinės sistemos tikslų yra paslepsti realią mašiną ir pateikti mums virtualią. Užduoties programą vykdo ne reali, o virtuali mašina. Virtuali mašina – tai tarsi virtuali realios mašinos kopija. Virtuali reiškia netikra. Mes tarsi surenkaime reikalingas realios mašinos komponentes, tokias kaip procesorius, atmintis, įvedimo/išvedimo įrenginiai, suteikiame jiems paprastesnę nei reali vartotojo sąsają ir visa tai pavadiname virtualiai mašina. Vienas iš virtualios mašinos (VM) privalumų yra programų rašymo palengvinimas, todėl realios mašinos komponentės, turinčios sudėtingą arba nepatogią vartotojo sąsają, virtualioje mašinoje yra supaprastintos. Virtuali mašina dirba su operacinės sistemos pateiktais virtualiais resursais, kurie daugelį savybių perima iš savo realių analogų ir pateikia kur kas paprastesnę vartotojo sąsają. Tai lengvina programavimą.

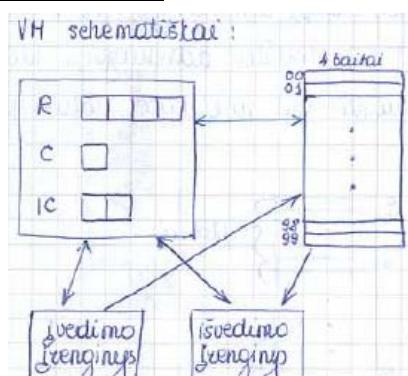
Kiekviena užduotis turi savo virtualią mašiną, kurios, iš tikrujų, ir konkuruoja dėl realaus procesoriaus. Vienas esminiu virtualios mašinos privalumų yra tas, kad užduotis, kurią vykdo virtuali mašina, elgiasi lyg būtų vienintelė užduotis visoje mašinoje. Tai yra didelė parama programuotojui. Dabar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

Virtualios mašinos pateikimas užduotims multiprograminės operacinės sistemos atveju.

VM specifikacija. Tarkime, yra 100 žodžių atmintis (0-99). Kiekvienas žodis yra 4 baitų $\square\square\square\square$. Žodžiai adresuojami nuo 0 iki 99. Tegul atmintis yra suskirstyta blokais po 10 žodžių. Procesorius turi 3 registrus: a) R – bendrasis registratorius $\square\square\square\square$ – 4 baitai; b) C – loginis trigeris, priima reikšmes true (T) arba false (F), kad būtų atliktas sąlyginis valdymo perdavimas \square – 1 baitas; c) IC – komandų skaitliukas $\square\square$ – 2 baitai.

Atminties žodis interpretuojamas kaip komanda arba duomenys. Operacijos kodas užima 2 vyresniuosius baitus, o adresas – 2 jaunesniuosius.

OPR | adr. - komandos struktūra. VM turi nuoseklaus įvedimo bei išvedimo įrenginius.



Ivesties/išvesties įrenginiai valdomi procesoriaus.

Virtualios mašinos procesoriaus komandos su paaiškinimais:

AD – sudėties komanda – x_1x_2 , $R:=R+[a]$, a - adresas, $a:=10^* x_1 + x_2$, $x_1,x_2 \in \{0,..,9\}$; **ADx₁x₂**; **LR** – registro

pakrovimas iš atminties - $x_1, x_2 \Rightarrow R := [a]; LRx_1x_2$; **SR** – įsimenama registro reikšmė - $x_1, x_2 \Rightarrow a := R; SRx_1x_2$; **CR** – palyginimo komanda - $x_1, x_2 \Rightarrow$ if $R = [a]$ then $C := 'T'$ else $C := 'F'$; **BT** – sąlyginis valdymo perdavimas - $x_1, x_2 \Rightarrow$ if $C := 'T'$ then $IC := a$; BTx_1x_2 ; **GD** – apsikeitimas su išore vyksta blokais (x_1 – bloko nr) - $x_1, x_2 \Rightarrow$ $Read([\beta+i], i = 0,..,9)$; $\beta = 10^* x_1$; GDx_1x_2 ; **PD** – išvedami duomenys - $x_1, x_2 \Rightarrow Print([\beta+i], i = 0,..,9)$; PDx_1x_2 ; **H** – sustojimo komanda $\Rightarrow HALT$. VM pradeda darbą, kai registro IC reikšmė yra 00 (ivykdo komandą, kuri patalpinta nuliniai žodyje).

5. Lygiagretūs procesai. Nuoseklus procesai veikia vienu metu – lygiagrečiai. Procesai neturi jokių tarpusavio sąryšių. Proceso aplinką sudaro resursai, kurios procesas naudoja, ir kurios sukuria.

Prasminis ryšys tarp procesų išriaškiamas perproceso resursus.

OS gali būti apibudinta kaip procesų rinkinys, kur procesai:

1. veikia beveik nepriklausomai (lygiagrečiai)
2. bendrauja per pranešimus ir signatus (resursus)
3. konkuruoja dėl resursų.

Skačiavimas sistemose minimas aparaturinis ir loginis lygiagretumas (parelizmas).

Aparaturinis lygiagretumas – reiškia lygiagretų, vienalakį aparatūros darbą (pvz. Išorinių įrenginių kontrolė, kur kiekvieną iš jų kontroliuoja kitas procesas).

Loginiame lygiagretume nesvarbu lygiagretumas. Apie jį kalbama tada, kai teoriškai darbas gali būti vykdomas lygiagrečiai.

Aparaturinis paralelizmas įvedamas efektyvumo sumetimais, kad greičiau vyktų darbas. Procesoriuje nesant aparaturiniams paralelizmui visvien svarbu vienintelį proc. darbo laiką skirstyti keliems procesams. Todėl įvedama lygegrečiai vykdomo proceso abstrakcija.

Neformaliai procesas – tai darbas, kurį atlieka procesorius, vykdamas darbą su duomenimis.

Loginis paralelizmas pasižymi tuo, kad kiekvienas procesas turi savo procesorių ir savo programą. Realiai, skirtinti procesai gali turėti tą patį procesorių ar tą pačią programą. Prosesas yra pora (procesorius, programa).

Prosesas – tai būsenų seka s_0, s_1, \dots, s_n , kur kiekviena būsena saugo visų proceso programos kintamųjų reikšmes. Pagal proceso būseną galima prateesti proceso darbą. Proceso būsena turi turėti sekančios vykdomos programos adresą. Proceso būsena gali būti pakeista paties proceso darbo rezultate arba kitų procesų darbo rezultate.

Valdymo ir informacinis ryšys tarp procesų realizuojamas per bendrus kintamuosius.

Nagrinėjant tik pačius procesus, gaunami nauji procesoriaus nepriklausomi sprendimai.
 s_1, s_2 – sakinai

s_1, s_2 – procesai vyksta nuosekliai

s_1 and s_2 – lygiagrečiai

pvz. $(a+b)*(c+d)-(e/f)$

begin

$t1 := a+b$ and $t2 := c+d;$

$t4 := t1 * t2$

end

and

$t3 := e/f$; $t5 := t4 - t3;$

Transliatorius turėtų išskirti lygiagrečius veiksmus ir sugeneruoti aukščiau užrašytą programą.

6. Kritinė sekcija. Tarkime turime du procesus p1 ir p2, kurie atlieka tą patį veiksmą $x:=x+1$ (x-bendras kintamasis). jie asinchroniškai didina x reikšmę vienetu. p1 vykdo procesorius c1 su registru r1, o p2 – c2 su r2. $t_o = v$

$t_o \dots \rightarrow t_n$ laiko ašis

a)p1: $r1:=x; r1:=r1+1; x:=r1; \dots$

p2: $\dots r2:=x; r2:=r2+1; x:=r2; \dots [x=v+1]$

b)p1: $r1:=x; r1:=r1+1; x:=r1; \dots$

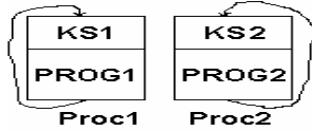
p2: $\dots r2:=x; r2:=r2+1; x:=r2; \dots [x=v+2]$

Gavome: a)x=v+1 ir b)x=v+2, o taip būti negali, tai dvių procesų problema. Ir pirmu atveju gali būti panasi situacija, kaip antru, dėl pertraukimų.

Programos dalis, dirbanti su bendrais procesų resursais, vadina kritinė sekcija.

Negalima leisti kad du procesai vienu metu įeitų į kritinę sekciją. Todėl reikia užtikrinti kad kritinėje sekcijoje tuo pačiu metu būtų tik vienas procesas. Geras būdas kritinei sekcijai tvarkyti – semaforų naudojimas.

Tarkime yra keletas ciklinių procesų:



Du lygiagrečiai dirbantys procesai Proc1 ir Proc2:

Begin

Proc1: begin L1: KS1; PROG1; GOTO L1; end;
and

Proc2: begin L2: KS2; PROG2; GOTO L2; end;

and

ProcN: begin LN: KSN; PROGN; GOTO LN; end;

End;

7. Kritinės sekcijos apsauga.

Bet kuriuo laiko momentu tik vieno proceso kritinė sekcija. Reikalavimai:

- 1) atminties blokavimas (vienu laiko momentu kreipiasi į tą pačią atmintį tik vienas procesas); 2) visos kritinės sekcijos yra lygiareikšmės; 3) vykdymo greičiai bet kokie; 4) programa gali būti nutraukta tik už kritinės sekcijos ribų.

Tegu 2 procesai turi bendrą kintamąjį (EILĖ), kuris nurodo, kurio proc. eilė kreiptis į atm. P1: jei EILĖ = 2 tada blokuokis, kitu atv.: KS1; EILĖ = 2; PROG1; į pradžią. P2 – atvirkšciai. Sprendimas negeras, nes blokuojamas kritinės sekcijos vykdymas. Jei vykdymo laikas P1>>P2 tai P1 lauks, kol P2 ilgai lauks, kol P1 įvykdys savo PROG1, kad galėtų vykdyti savo KS2.

Iveskime du bendrus kintamuosius: Ci = FALSE, kai vykdoma KSi. P1: jei vykdoma KS2, tai blokuokis, kitu atv.: C1=FALSE; KS1; C1=TRUE; PROG1, į pradžią. P2 – atvirkšciai. Šiuo atveju abu procesai gali pradėti vykdyti KS vienu metu.

Problema išsprendžia Dekerio algoritmas.

Paprasčiau problemą išspręsti naudojant Semaforus:

Tegul M – kritinę sekciją apsaugantis semaforas, n – procesų skaičius.

BEGIN SEMAPFORE M;

M:=1 //pradinė reikšmė

P1: BEGIN...END

and

...

P_i: BEGIN L_i: P(M); KS_i; V(M); PROG_i; GOTO L_i; END

...

and

P_n: BEGIN...END

END;

8. Dekerio algoritmas. Procesas atžymi savo norą jei i KS loginiu kintamuoju Ci=false. Išėjus iš kritinės sekcijos Ci=true. Jei i KS procesas gali tik tada, kai kitas procesas nėra KS'je arba nėra pareiškės noro ją vykdyti. Sveikas kintamasis EILE naudojamas tada, kai du procesai susiduria KS'je (pvz.: noras vykdyti KS, įejimas i KS). Šis kintamasis parodo, kurio proceso eilė vykdyti KS. Proc. kuris neturi eilės vykdyti KS atsisako savo noro.

BEGIN

INTEGER EILE;

BOOLEAN C1, C2;

C1:=C2:=true; EILE:=1;

P1: BEGIN

A1: C1:=false; //(*)

L1: IF not C2 THEN

BEGIN IF EILE=1 THEN GOTO L1;

C1:=true;

B1: IF EILE =2 THEN GOTO B1;

GOTO A1;

END;

KS1;

EILE:=2; C1:=true;

PROG1:

GOTo A1;

END

AND

P2: BEGIN

A2: C2:=false;

L2: IF not C1 THEN

BEGIN

IF EILE=2 THEN GOTO L2;

C2:=true;

B2: IF EILE=1 THEN GOTO B2;

GOTO A2;

END;

KS2;

EILE:=1;

C2:=true;

PROG2;

GOTO A2

END
END;

- 9.Semaforai.** Semaforas S tai sveikas neneigiamas skaičius, su kuriuo atliekamos operacijos P(S) ir V(S), kur P ir V nauji primityvai. Operacijos pasižymi savybėmis:
- 1)P(S), V(S) – nedalomos operacijos, t.y. jų valdymo negalima pertraukti ir jų vykdymo metu negalima kreiptis į semaforą S;
 - 2)V(S): S:S+1; (didinama semaforo reikšmė)
 - 3)P(S): S:S-1; (sumažinama jei S>0)
 - 4)Jei S=0, tai procesas P, kuris vykdo operaciją P(S), laukia, kol sumažinimas vienetu bus galimas. Šiuo atveju P(S) yra pertraukiamas
 - 5)Jei keletas procesų vienu metu iškviečia V(S) ir/ar P(S) su vienu semaforu, tai užklausimai vykdomi nuosekliai, kokia nors iš anksto nežinoma tvarka.
 - 6)Jei keletas procesų laukia operacijos P(S) įvykdymo, S – ta pats, tai reikšmei tapus teigiamai(kai kažkuris procesas įvykdė operaciją V(S)), kažkuris iš laukiančių procesų bus pradėtas vykdyti.

Pagal prasmę operacija P atitinka perėjimo išškvietimą, o V – kito proceso aktyvaciją.
Jei semaforas įgyja tik dvi reikšmes 0,1, tai jis vadinas dvejetainiu, jei bet kokias, tai bendriniu.

Pvz1. Semaforus galima naudoti procesų sinchronizacijai. Turime du procesus, norime, kad antras pradėtų vykdyti savo programą tuomet, kai pirmas pasiūs jam atitinkamą signalą.

10. Procesų „gamintojas“ ir „naudotojas“ sąveika. Procesas gamintojas sukuria informaciją ir patalpina ją į buferį, o lygiagrečiai veikiantis kitas procesas naudotojas paima informaciją iš buferio ir apdoroja.

Tegul buferio atmintis susideda iš N buferių.

Semaforas T – tuščių buferių skaičius.

Semaforas U – užimtų buferių skaičius.

B – semaforas, saugantis kritinę sekciją, atitinkančią veiksmus su buferio sekcijomis.

BEGIN SEMAPHORE T,U,B;

T:=N; U:=0; B:=1; // nes kritinė sekcija nevykdoma, kai vykdoma – B:=1

GAM: BEGIN LG: įrašo gaminimas;

P(T); P(B); užrašimas į buferį; V(B);

V(U); GOTO LG;

END

And

NAUD: BEGIN LN: P(U);

P(B); paėmimas iš buferio; V(B);

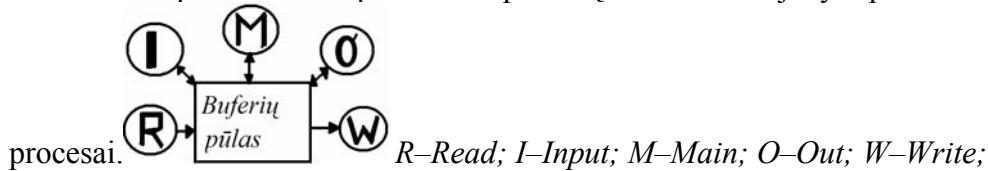
V(T); įrašo apdorojimas;

GOTO LN;

END

END;

11. Įvedimo-išvedimo spulerio bendra charakteristika. Spool'eris – OS dalis atliekanti I/O virtualizaciją, kuomet yra sukaumpiama įvedama informacija, ji apdorojama ir išvedama kaip rezultatas. Spool'eris 5 procesų visuma. Visi jie yra procesoriuje vykdomi



Buferių pūlas – sąrašas(faktiškai 3 sąrašai – laisvų, įvedimo ir išvedimo buferių). Iš pradžių buferių pūlas apjungtas į laisvų buferių sąrašą, o įved. Ir išved. Sąrašai tušti. Adorojimo proceso užduotis paruošti informaciją išvedimui. Darbui su buferiais reikia apsirašyti tokius parametrus: laisvų buferių skaičių; įved/išved buferių skaičių; KS apsaugos semaforą; įved/išved buferių KS apsaugos semaforus. nl – laisvų, nin – įvedimo, nout – išvedimo buferių skaičius; ml – lasivų, min – įvedimo, mout – išvedimo buferių KS apsaugos semaforas.

12. Įvedimo-išvedimo SPOOLER'io pagrindinis procesas.

1. ima buferį iš įvedimo buferių sąrašo; 2. Apdoroja Jame esančią informaciją; 3. Ima buferį iš laisvų buferių sąrašo. 4. Ji užpilda išvedama informacija; 5. Buferį įjungia į išvedimo buferių sąrašą.

Pagrindinio proceso M programa:

M: BEGIN LM:P(nin); P(min); Paimti pirmą buf. iš įvedimo buf. sąrašo; v(min); Apdrooti buferio turinį; P(nl); P(ml); Paimti pirmą buf. iš laisvų buf. sąrašo; v(ml); Uždaryti buf. išvedama info; P(mout); Ijungti buf. į išved. buf. sąrašą; v(mout); P(ml); Ijungti buferį į laisvų buf. sąr; v(ml); v(nl); GOTO LM;END;

13. Įvedimo/išvedimo SPOOLER'io įvedimo ir skaitymo procesas.

Atliekant įvedimo veiksmą, darbas turi būti sinchronizuojamas su įvedimo įrenginiu. Įvedimo įrenginio darbo pradžios ir pabaigos situaciją (realiai – pertraukimo situacija) modeliuosime semaforais. SR – skaitymo įrenginio startas; FR – skait. įreng. Finišas.

Proc I veiksmų seką yra tokia:

1. įvedimo procesas paima buferį iš laisvų buferių sąrašo;
2. paleidžia skaitymo procesą R ir perduoda jam laisvą buferį;
3. užpildytą buferį įjungia į įvedimo buferių sąrašą.

Įvedimo procesas I gali vykti tik tada, kai yra tuščių buferių ir lieka bent vieną laisvą, nes jo reikia procesui M.

I: BEGIN LI: P(nl); P(ml); IF liko paskutinis laisvas buferis THEN BEGIN V(nl); (ml); GOTO LI; END paimti buf. Iš laisvų buf. V(ml); V(SR); P(FR); P(min); Prijungti įvedimo buf. prie įvedimo buf. V(min); V(nin); GOTO LI; END;

R: BEGIN LR: P(SR); Perskaityti į nurodytą buf.; V(FR); GOTO LR; END;

Atliekant išvedimo veiksmą, darbas turi būti sinchronizuojamas su išvedimo įrenginiu.

Išvedimo įrenginio darbo pradžios ir pabaigos situaciją (realiai – pertraukimo situacija) modeliuosime semaforais. SW – išvedimo įrenginio startas; FR – išvedimo. įreng. finišas.

O:BEGIN LO: P(out); P(mout); Paimti pirmą buferį iš išvedamų buf. sąr.; V(mout); V(SW); P(WF); P(ml); Prijungti atlaisvintą buferį prie laisvų buf.sąr.; V(ml); V(nl); GOTO LO; END;

W: BEGIN LW: P(SW); Užrašyti iš nurodyto buf.; V(FW); GOTO LW; END;

14. Dvejetainių ir bendrų semaforų ryšys. Semaforinių primityvų relizacijai, reikia bendruosius semaforus išreikšti dvejetainiais. Jei semaforas gali igytį tik dvi reikšmes – jis dvejetainis. Bet kuris bendras semaforas gali būti išreikštęs dvejetainiu semaforu. Jei S – bendrasis semaforas, tai jį galima pakeisti kintamuoju NS ir dviem dvejetainiais semaforais M, D. M – kritinę sekciją apsaugantis semaforas.

$P(S) \sim P(M);$

$NS = NS - 1;$

If $NS <= -1$ Then Begin $V(M)$; $P(D)$ End

Else $V(M)$;

Pradiniai $M := 1$; $D := 0$;

P(S) atvejai: a) $S > 0$ – nuimama KS apsauga ir neiššaukiamas laukimas; b) $S = 0$ – turi įvykti perėjimas į laukimą, tada bus pasiekiamas procesas $V(S)$.

$NS < 0$ – parodo laukiančių procesų skaičių.

$V(S) \sim P(M);$

$NS = NS + 1;$

If $NS <= 0$ Then $V(D)$; {yra laukiančių procesų, kurie užsikodavę semaforu D}

$V(M);$

15. Operacijų su semaforais realizacija. Reikia kad komp. architektūra leistų patikrinti žodžio tūrį ir pakeisti jo reikšmę.

BOOLEAN PROCEDURE PP(x);

BOOLEAN x;

BEGIN

$PP := x;$

$X := \text{true};$

END

S – dvejetainis semaforas

P(S); Uždrausti pertraukinus

L: if PP(x) then goto L;

$S := S - 1;$

If $S \leq 1$ then

Begin (užblokuoti iššaukiantį procesą)

Paimti proc.iš sąrašo A;

$X := \text{false};$

Perduoti proc.iš A; lesti pertrauk.;

End

ELSE Begin $x := \text{false};$ lesti pertr.;

End

V(S); Uždrausti pertr.;

L: if PP(x) then goto L;

$S := S + 1;$

If $S \leq 0$ then

Begin

Paimti proc.iš sąrašo B ir perkelti į A;

```
If proc.laisvas then vykdyti proc.iš A;  
End  
X:=false;  
Lesti pertr.
```

16. Procesų ir resursų sąvoka. Kiekvienai užduočiai j susuriamas procesas P_j, kuris tvarko užduoties naudojamų resursų aprašymą. Aprašymas susideda iš statinės ir dinaminės informacijos. Statinė – max laikas. Dinaminė – duotu laiko momentu naudojami resursai. Kiekvienas procesas gali kurti kitus procesus. Gauname procesų medį.

17. Proceso deskriptorius(PD). PD – (proceso) veiklumo stadiją apibūdinantis proceso aprašas. Apraše ir yra laikomi visi procesui reikalingi parametrai: virtualus procesorius, registrų reikšmės ir jam reikalingi kintamieji. Prosesų aprašai dinaminiai objektais – jie gali būti sukurti/sunaikinti sistemos veikimu metu. Realiai procesą, kaip ir resursą OS-je atstovauja Deskriptorius. PD – tai tam tikra struktūra(ne masyvas) – jei kalbame apie visų procesų deskriptorius, tai turime struktūrą masyvą, kur *i* proceso vidinis vardas – nurodytų struktūros numerį masyve. PD – susideda iš komponenčių, kurioms priskiriame vardus:

- 1)*Id[i]* – proceso išorinis vardas, reikalingas statiniams ryšiams tarp procesų nurotyti.
- 2) Mašina – čia turime omeny procesą vykdančio procesoriaus apibūdinimą.
- 2.1)*CPU[i]* – apibūdina centrinio procesoriaus būseną ir teises vykdant procesą. Kai proceso vykdymas nutraukiamas, proceso būsena išsaugoma.
- 2.2)*P[i]* – identifikuojas procesorių
- 2.3)*OA[i]* – operatyvi atmintis turima i-tojo proceso (nuoroda į sąrašą)
- 2.4)*R[i]* – i-tojo proceso turimi resursai – informacija, kokius resursus yra gavęs procesas. Nuoroda į sąrašą, kuriame yra išvardinta resursai.
- 2.5)*SR[i]* – Sukurtų resursų sąrašas (resursų deskriptorių).
- 3) informacija apie proceso būseną:
 - 3.1) ST[i] – proceso statusas (RUN, READY, READYS, BLOCK, BLOCKS)
 - 3.2) SD[i] – nuoroda į sąrašą kuriame yra procesas (PPS – pasiruošusių proc.sar., LPS – laukiančių proc.sar.)
- 4) proc.sąryšis su kitais procesais
 - 4.1) T[i] – i-tojo proceso tėvas (tėvinio proc.vidinis vardas)
 - 4.2) S[i] – nuoroda į i-tojo proceso vaikinių proc.vidinių vardų sąr.
- 5) PR[i] – prioritetas.

18. Resurso deskriptorius. Resurso deskriptorius (Resurso valdymo blokas) yra fiksuoto formato duomenų struktūra, sauganti informaciją apie resurso einamajį stovį. Remiantis informacija resurso deskriptoriuje nurodomas jo užimtumo laipsnis, laisvas kiekis, nuoroda į pačius resurso elementus ir kt. Šia informacija naudojasi duotojo resurso paskirstytojas. Resurso inicializavimas reiškia deskriptoriaus sukūrimą. Darbas su deskriptoriais galimas tik per specialias operacijas - OS branduolio primityvus.

r-resurso vidinis vardas (indeksas resursų deskriptorių masyve)

1) identifikacija.

1.1) Rid[r] – išorinis vardas

- 1.2) PNR[r] – ar tai pakartotinio panaudojimo resursas (jei taip - jį reikia gąžinti)
- 1.3) K[r] – resursą sukurusio proceso vidinis vardas
- 2) PA[r] – nuoroda į prieinamumo aprašymo sąrašo pradžią.
- 3) LPS[r] – nuoroda į resurso laukiančių procesų sąrašo prad.
- 4) PASK[r] – resurso paskirstytojo programos adresas.

19. Primityvas „kurti procesą“. Sukuriamas naujas deskriptorius masyve. Procedūra įvykdoma iššaukiančio proceso aplinkoje.

PROCEDURE KURTIP(n,s0,M0,R0,k0);

n – išorinis vardas

S0 – procesoriaus pradine būsena;

M0 – OA pradinė būsena (kiek išskirta OA resursu);

R0 – kiti išskiriami resursai;

K0 – proceso prioritetas

//duotu metu dirbanties procesas – „*”

PROCEDURE KURTIP(n,s0,M0,R0,k0);

BEGIN

I:=NVV; //NVV-naujas vidinis vardas); Id[i]:=n;CPU[i]:=s0; OA[i]:=M0; R[i]:=R0; //kitu turimų proceso resursų komponente(nuoroda į sąrašą); PR[i]:=k0;

ST[i]:=READYS; //laikysime, kad procesas sukuriamas su statusu(pvz. pasiruošes)

SD[i]:=PPS; T[i]:=*; S[i]:=Λ; //sūnų sąrašas iš pradžių jis tuščias Irašyti(s[*],i); //igavo naują sūnų i; Ijungti(PPS,i); END;

20. Primityvas „naikinti procesą“. Procesas naikina visus savo palikuonis (sūninius procesus), bet negali sunaikinti savęs. Tada jis nusiunčia pranešimą savo tėvui, kuris jį ir sunaikina.

Pakartotino naudojimo resursus reikia atlaisvinti.

PROCEDURE NAIKINTIP(n);

BEGIN L:=false; jei L = true – yra vykdomų palikuonių, tai iškviečiamas planuotojas;

i:=VV(n); P:=T[i]; NUTRAUKTI(i); //panaikina visus proceso palikuonis

Pašalinti(S[T[i]], i); IF L THEN PLANUOTOJAS;END;

PROCEDURE NUTRAUKTI(i);

BEGIN

IF ST[i] = RUN THEN BEGIN STOP(i); L:=true; END;

Pašalinti(SD[i], i); // pašalinti iš pasiruošusių arba laukiančiųjų procesų sār.

FOR ALL S ∈ S[i] DO NUTRAUKTI(S);

FOR ALL r ∈ OA[i] v R[i] DO

IF PNR THEN Irašyti(PA[r], Pašalinti(OA[i] VR[i]));

FOR ALL r ∈ SR[i] DO NDR(r); //naikinami resursų deskriptoriai

NPD(i); //naikiname proceso deskriptorių. END;

21. Primityvas „Stabdysi procesą“. Tėvinis procesas gali stabdyti vaikinį nestabdant vaikinio palikuonių. n – išorinis vardas, a – adresas į proceso deskriptoriaus būseną(jo kopiją).

```

Procedure STABDYTIP(n,AR); Begin i:=vv(n); S:=ST[i];//einamasis proceso statusas//
if S=RUN then STOP(i);
[[[STOP(i) įvykdo veiksmus: 1) pertraukia procesorių P[i]; 2) įsimena pertraukto
procesoriaus P[i] būseną proceso deskriptoriaus komponentėje CPU[i] 3) PROC[P[i]]:=
Λ //pažymima, kad procesorius P[i] yra laisvas//]]
AR:=copydest[i]; if S=BLOCK or BLOCKS then ST[i]:=BLOCKS else
ST[i]:=READY; if s=RUN then PLANUOTOJAS; End

```

22. Primityvas „aktyvuoti proceso“. Nuima pristabdymo buseną. Galbūt iškviečia planuotoją, jei būsena yra READY.

```

PROCEDURE AKTYVUOTI(n);
BEGIN i:=VV(n); ST[i]:=IF ST[i]=READY THEN REDY ELSE BLOCK;
IF ST[i]=READY THEN PLANUOTOJAS; END;

```

23. Primityvas „keisti proceso prioritetą“. Proceso įterpimas į sąrašą vyksta atsižvelgiant į proceso prioritetą, todėl proceso prioriteto pakeitimas vyksta taip: pašalinamas iš sąrašo ir po to įterpiamas pagal naują prioritetą.

```

PROCEDURE KEISTIPP(n,k);
BEGIN I:=VV(n); M:=PR[i]; Pašalinti(SD[i],i); PR[i]:=k; Irašyti(SD[i],i); If M<k and
ST[i]=READY THEN PLANUOTOJAS; END;

```

24. Primityvas „kurti resursą“. Resurso deskriptoriaus sukūrimas. Resursus kuria tik procesas.

```

Procedure KURTIR (RS, PN, PAo, LPSo, PASKo);
RS – išorinis resurso vardas
PN - ar res.yra iš naujo panaudojamas
PAo – res.prieinamumo aprašymas
LPSo – to resurso laukiančių proc.sqr.
PASKo – res.paskirstutojo programos adresas
Begin
r:=NRVV; Rid[r]:=RS; PNR[r]:=PN; k[r]:=*; PA[r]:=PAo; LPS[r]:=LPSo;
PASK[r]:=PASKo; Irašyti (SR[*], r);End;

```

25. Primityvas „naikinti resursą“. Sunaikinti resursa gali jo tevas arba pirmtakas. Sunaikinamas resurso deskriptorius.

```

PROCEDURE NAIKINTIR(RS);
Begin
r:=RVV(RS); //atblokuojami resurso laukiantys proc.jiems pasiuńčiant fiktyvų res.//
R:=Pasalinti(LPS[r]); While R<>1 do Begin ST[R.P]:=IF ST[R.P]=BLOCK then READ
else READS; Irašyti (PPS,R.P); SD[R.P]:=PPS; R.A:='PRANEŠIMAS';
R:=Pasalinti(LPS[r]); End
NRD(r); //Naikinri resurso deskriptoriu//; PLANUOTOJAS; End

```

26. Primityvas “prašyti resurso”. Prosesas, kuriam reikia resurso, iškviečia ši primityvą, nurodydamas išorinį vardą ir adresą. Toks procesas pereina į blokovimosi

būseną. Blokavimasis įvyksta tik prašant resusrso. Procesas ijjungiamas į laukiančių to resurso procesų sąrašą.

PROCEDURE PRASYTIR(RS, D, A);

D – kokios resursro dalies prašoma:

A – atsakymo srities adresas, į kur pranešti.

BEGIN

```
R := RVV(RS); IJUNGTI(LPS[r], (*, D, A)); PASK(r, K, L); // K – kiek procesų aptarnauti, L – aptarnautų procesų vidinių vardų masyvas// B := true; //ar masyve L yra einamas procesas// FOR J :=1 STEP 1 UNTIL K DO IF L[J] <> * THEN BEGIN i:=L[J]; Irašyti(PPS, i); SD[i]:=PPS; ST[i]:= IF ST[i]=BLOCK THEN READY ELSE READYS; END ELSE B:=false; //((Procesas * iš karto gavo resursą ir nebuvvo išrašytas į laukiančių procesų sąr.) IF B THEN
```

BEGIN

```
ST[*]:=BLOCK;
SD[*]:=LPS[r];
PROC[P[i]]:=Λ; //(Procesorius laisvas)
PASALINTI(PPS, *)
```

END

PLANUOTOJAS

END

27. Primityvas „atlaisvinti resursą“. Tai atitinka situaciją, kai procesas gauna pakartotino naudojimo resursą ir kai jo jam nereikia, jis ji atlaisvina ir ijjungia į sąrašą laisvųjų resursų bei papildo resurso prieinamumo aprašymo sąr.

PROCEDURE ATLAISVINTIR(RS,D);

BEGIN

```
r:=RVV(RS);
Ijungti(PA[r],D);//
PASK(r,k,L);
IF k>0 THEN FOR J:= 1 STEP 1 UNTIL k DO
BEGIN
    i:=L(J);
    Ijugti(PPS,i);
    SD[i]:=PPS;
    ST[i]:= IF ST[i]= BLOCKS THEN READYS
                ELSE READY

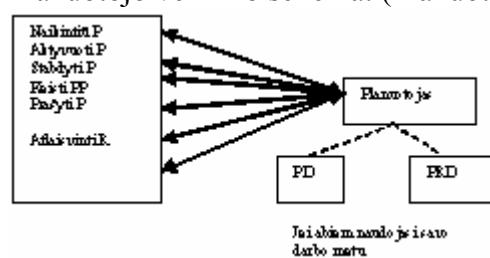
```

END;

IF k<>0 THEN PLANUOTOJAS;

END;

28. Prosesų planuotojas. Planuotojas užtikrina, kad visi procesai būtų vykdomi maksimaliu prioritetu (Jei prioritetai vienodi, tai vykdomas tas kuris sąraše pirmesnis). Planuotojo veikimo schema: (Planuotojas iškviečiamas iš branduolio primitivų)



Planuotojo darbas 3 etapais: 1) Surasti procesą su aukščiausiu prioritetu. 2) Surasti neaktyvų procesorių ir jų išskirti.(skirti pasiruošusiam

procesui) 3) Jei procesorių nėra, peržiūrēti vykdomus procesus, ir, jei jų prioritetai mažesni, atiduot procesorių pasiruošusiems (Perskirstymas)

29. Pasiruošusio proceso su aukščiausiu prioritetu nustatymas ir neaktyvaus procesoriaus radimas.

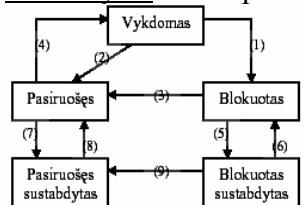
Kol neperžiūrēti visi pasiruošusių procesų sąrašai ir prioritetas nelygus nuliui darom: jei peržiūrimas sąrašas tuščias, tai prioritetas mažinamas vienetu ir imamas to prioriteto pasiruošusių proc.sar.

Pradedame nuo 1 ir sukame ciklą kol pasiekiami procesorių skaičių.tikriname ar procesoriui einamasis procesorius turi vadą. Jei turi (t.y. yra užimtas) sukame ciklo skaitliuką. Radus procesą be vardo, t.y. neaktyvų, jam priskiriamas procesas. Jei tai procesorius, kuris laisvas dėl to, kad negavęs resurso, tai įsimenama kuriam procesui reikės jį atiduoti.

30. Prosesų planuotojo procesoriaus perskirstymo etapas. Perskirstant procesorių reikia iš proceso su mažesniu prioritetu atimti procesorių. Pirmiausiai prasukame visus procesus ir surandame, kuris turi mažiausią prioritetą ir mažesnį negu mūsų proceso įsimenamą to proceso vidinį vardą. Iš to procesoriaus atimame procesorių ir jė atiduodame mūsų norimam procesui

31. Prosesų būsenų schema. Procesorius gali gauti procesorių tik tada, kai jam netrūksta jokio resurso. Procesas gavęs procesorių tampa vykdomu. Procesas, esantis šioje busenoje, turi procesoriu, kol sistemoje neįvyksta pertraukimas arba einamasis procesas nepaprašo kokio nors resurso (pavyzdžiui, prašydamas ivedimo iš klaviaturos). Procesas blokuojasi priverstinai (nes jis vis tiek negali tapti 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 busena nei blokovimasis del resurso (neturimas omeny resursas - procesorius). Taigi, galime išskirti jau žinomas procesu busenas: vykdomas – jau turi procesorių; blokuotas – prašo resurso(bet ne procesoriaus); pasiruošęs – vienintelis trūkstamas resursas yra procesorius; sustabdytas – kito proceso sustabdytas procesas.



(1)vykdomas procesas blokuojasi jam prašant ir negavus resurso.(2)vykdomas procesas tampa pasiruošusiui atėmus iš jo procesorių dėl kokios nors priežasties (išskyrus resurso negavimą). (3)Blokuotas procesas tampa pasiruošusiui, kai yra suteikiamas reikalingas resursas.(4)pasiruošę procesai varžosi dėl procesoriaus.

Gavęs procesorių procesas tampa vykdomu.(5)procesas gali tapti sustabdytu blokuotu, jei einamasis procesas ji sustabdo, kai jis jau ir taip yra blokuotas.(6) Procesas tampa blokuotu iš blokuoto sustabdyto, jei einamasis procesas nuima buseną sustabdytas.(7)Procesas gali tapti pasiruošusiui sustabdytu, jei einamasis procesas ji sustabdo, kai jis yra pasiruošęs.(8)Procesas tampa pasiruošusiui iš pasiruošusio sustabdyto, jei einamasis procesas nuima buseną sustabdytas.(9) Procesas tampa

pasiruošusiu sustabdytu iš blokuoto sustabdyto, jei procesui yra suteikiamas jam reikalingas resursas.

32. Virtualios atminties sąvoka. OA – tai ta, į kurią procesorius gali kreiptis tiesiogiai imant komandas ar duomenys programos vykdymo metu. Tokia schema turi daug nepatogumų, kai programoje nurodomi absolutūs adresai.

Transliatoriai buvo perdaryti taip, kad gamintų objektinę programą, nesusietą su patalpinimo vieta, t.y. kilnojamus objektinius modelius (nustatant programos adresus pagal išskirtą atminties vietą). Nuo atminties skirstymo programos vykdymo metu pereita prie atminties skirstymo prieš programos vykdymą. Prieš vykdymą programas reikia susieti ir patalpinti į atmintį.

Kai visi darbai atliekami prieš programos vykdymą, kalbama apie statinį adresą nustatymą – statinį atminties skirstymą. Dinaminis atminties skirstymas – kai adresai nustatomi betarpiskai kreipliantis į atmintį. Statiškai nustatant adresus būtina prieš programos vykdymą žinoti išskirtos atminties vietą. Dinaminis – kai fizinis adresas nustatomas tiesiogiai prieš kreipimasi į tą adresą.

Sudarant programas tenka abstrahuotis nuo atminties žodžių ir nauduoti tam tikrus lokalius adresus, kurie vėliau kokiui tai būdu susiejami su fiziniais adresais. Tokia lokalių adresų visuma – virtuali atmintis. Fizinių adresų visuma – reali atmintis. Virtuali atmintis su fizine būti susiejama statiškai arba dinamiškai.

33. Komandos vykdymo schema. Schematiškai pavaizduosime komandų vykdymą, kuomet nėra dinaminio atminties skirtingo:

H[O:N] – atmintis;

K – komandų skaitliukas;

R – registras.

L: W:=H[K];

OK:=W op kodas;

ADR:+ W operando adr;

K:=K+1;

Add: IF OK=1 THEN R:=R+H[ADR];

ELSE

SR: IF OK=2 THEN H[ADR]:=R;

ELSE

BR: IF OK=3 THEN K:=ADR;

GOTO L

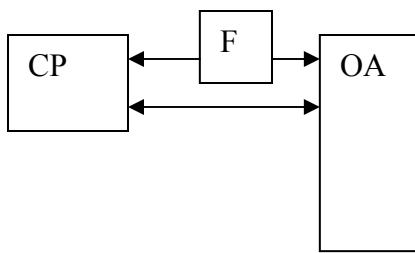
Čia K, ADR vadinami efektyviais adresais.

Jei yra dinaminė adresų nustatymo aparatūra, tai efektyvus adresai yra atvaizduojami į absoliučius(fizinius) adresus: F(ea)=aa.

H[k]~H[F(k)]

H[ADR]~H[F(ADR)].

I F galima žiūrėti kaip į aparatūrinį bloką, jungiantį procesorių su OA. Schematiškai:



34. Puslapinė organizacija. Puslapinė organizacija – tai konkretus vartotojo atminties (VA) organizavimo būdas. Operatyvi atmintis (OA) yra suskaidoma į vienodo ilgio ištisinius blokus $b_0b_1\dots b_{B-1}$. OA absolius adresas (AA) nurodomas kaip pora (b, l) , kur b – bloko numeris, l – žodžio numeris bloko b viduje. Atitinkamai VA adresinė erdvė suskaidoma į vienodo dydžio ištisinius puslapius $p_0p_1\dots p_{P-1}$.

VA adresas nustatomas pora (p, l) . Puslapio dydis lygus bloko dydžiui. Bendra suminė VA yra daug didesnė už OA. VA adresas lygus efektyviam adresui (EA), kurį reikia atvaizduoti į AA: $F(EA)=AA$, šiuo atveju $F(p, l)=(b, l)$.

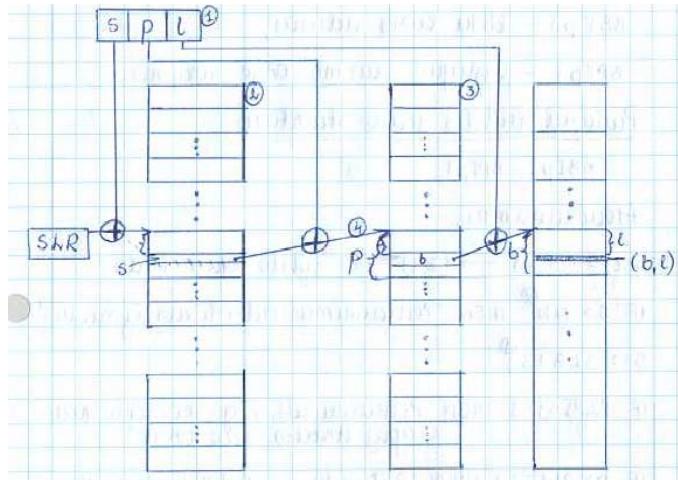
Puslapinės organizacijos schema: a) VA turime $[p|l]$; b) aparatūroje turi būti numatytas [PLR] – puslapių lentelės registras(rodo aktyvaus proceso puslapių lentelę); c) puslapių lentelės saugomos atmintyje.

Puslapių lentelės registro dydis atitinka VA puslapių skaičių. Kiekvienam procesui sudaroma puslapių lentelė. Vieno segmento VA atvaizdavimas: $F(p, l) = M[PLR+p]*2^k+l$, čia $M[PLR+p]$ – bloko numeris; 2^k – bloko dydis; i – poslinkis.

35. Polisegmentinė virtuali atmintis. Puslapinė organizacija su segmentacija. SLR rodo į aktyvaus proceso SL. Kiekvienas Segmentas yra išreiškiamas puslapiu ir turi atitinkamą puslapių lentelę.

(s,w) yra išreiškiamas trejetu (s,p,l)

Kiekvienas segmentas suskaidomas puslapiais, tokio pat dydžio kaip bloko dydis.



- ① virtualius adresas,
- ② OA'je segmentų lentelės;
- ③ OA'je puslapių lentelės;
- ④ s-čio segmento puslapių lentelės adresas;

SLR – Segmentų lentelės registras, saugo aktyvaus proceso segmentų puslapių lentelės adresą.

??PSL – Saugo bloko numerį į kuri tas puslapis atvaizduojamas.

SLR: SLB komponentė – segmentų lentelės bazė – nurodo adresą OA'je.

SLD – kiek segmentų VA'je (segmentų lentelės dydis)

SL: PLB(S) – puslapių lentelės bazė

PLD(S) – puslapių lentelės dydis

PLP(S) – puslapių lentelė buvimo OA požymis

PL: BB[p] – bloko bazė

BP[p] – buvimo OA požymis

IF $S > SLD$ THEN „Pertraukimas. Neleistinas segmentas“

$S := SLB + S;$

IF $PLP[S] = 1$ THEN „Pertraukimas. OA nera s-ojo segmento psl.lenteles“

IF $p > PLD[S]$ THEN „Pertraukimas. Neleistinas psl.segmente“

$p := PLP + p;$

IF $BP[p] = 1$ THEN „Pertraukimas. Psl.nera atminty“

$F := BB[p] + 1$

36. Atminties skirstymo puslapiai strategijos. Esant dinaminiam atm. Skirstymui VA dydis gali būti didesnis nei OA dydis. Tada reikia priverstinai atlaisvinti vietą. Puslapių skirstymo uždavinys turi atsakyti į klausimus:

- kokiu momentu sukeisti puslapius;
- kokį puslapį patalpinti į atmintį (iš išorės į OA);
- vietoje kokio puslapio.

Puslapių skirstymo strategija, pagal kurią vieta puslapiui skiriama betarpiskai prieš į jį kreipiantis, vadinama strategija pagal pareikalavimą (SPP). Ji užfiksuoja atsakymus į pirmuosius du klausimus.

Irodyta, kad bet kokiai puslapių skirstymo strategijai egzistuoja neblogesnė strategija pagal pareikalavimą. Vadinasi, optimalių strategijų paieškoje galima apsiriboti SPP klase.

SPP, kurioje išstumiami tie puslapiai, kreipiamaisi į kuriuos bus vėliausiai puslapių trasoje, vadinama Bellady strategija. Pvz.: tarkime, turime du blokus b1, b2 ir puslapių seką p1, p2, p3, p4, p1, p2, p3. Sprendimas:

B₁ | p₁ p₁ p₁ p₁ p₁ p₁

B₂ | p₂ p₃ p₄ p₂ p₃

Šitokia strategija yra optimali. Tačiau iškyla taikymo problemos, nes prieš programos vykdymą puslapių trasa nėra žinoma.

Praktikoje naudojamos tam tikros strategijos, kai pakeičiamas: 1)atsitiktinis, 2)ilgiausiai būnantis OA'je puslapis, 3)į kurį buvo kreiptasi seniausiai.

Vidutiniškai du kartus mažiau puslapių pakeitimų, jei naudojama trečioji strategija: naujas puslapis įrašomas vietoj seniausiai naudoto.

37. Atminties išskyrimas ištisinėmis sritimis. GETAREA(adress, size);
FREEAREA(address, size); FREEGARBAGE;

38. Kintamo dydžio ištisinės atminties sričių grąžinimas

Procedure FREEAREA(adress, size)

Begin pointer L;M;U;Q;

M:=address - 1;

U:=M+M.size;

L:=M-1;

IF U.tag = ‘-’ **THEN**

Begin

M.size:=M.size +U.size;

U.BLINK.FLINK:=U.FLINK;

U.FLINK.BLINK:=U.BLINK;

End;

Q:=M+M.size-1;

IF L.tag=’-’ **THEN** //Jei prieš tai yra laisva

Begin

L:=M-L.size;

L.size:=L.size+M.size;

Q.size:=L.size;

Q.tag:=’-’;

End;

ELSE Begin //Jei užimta

Q.size:=M.size;

M.tag:=Q.tag:=’-’;

M.FLINK:=FREE.FLINK; //M įjungiamas į laisvų sričių sąrašą.

M.BLINK:=address(FREE);

FREE.FLINK.BLINK:=M;

FREE.FLINK:=M;

End;

40. Mikrobranduolio architektūra. Mikrobranduolys yra OS nedidelė atminties dalis, įgalinanti OS modulinį plėtimą. Nėra vieningos mikrobranduolio sandaros. Problema yra

driver'iai, juos reikia padaryt efektyvius. I driver'į galima žiūrėti kaip į virtualų įrenginį, kuris palengvina įrenginio valdymą, pateikdamas patogesnį interfeisą. Kitas klausimas, kur vyksta procesai, ar branduolio erdvėj, ar už jo ribų. Pirmos OS buvo monolitinės, kur viena procedūra galėjo iškvesti bet kokią kitą procedūrą. Tai tapo kliūtimi augant OS. Buvo įvesta OS sluoksninė architektūra. Sudėtingumas nuo to nedingo. Kiekvienas sluoksnis gana didelis. Pakeitimai vienam sluoksnyje iššaukia pakeitimus ir gretimuose sluoksniuose. Sunku kurti versijas pagal konkrečią konfigūraciją. Sunku spręsti saugumo problemas dėl gretimų sluoksnų sąveikos.

Mikrobranduolio sistemos atveju visi servisi perkelti į vartotojo sritį. Jie sąveikauja tarpusavyje ir su branduoliu. Tai horizontali architektūra. Jie sąveikauja per pranešimus, perduodamus per branduoli. Branduolio funkcija tampa pranešimo perdavimas ir priėjimas prie aparatūros.

Mikrobranduolio architektūros pranšumai: 1) vieningas interfeisas 2) Išplečiamumas 3) Lankstumas 4) Pernešamumas(Portability) 5) Patikimumas 6) Tinkamumas realizuoti paskirytas (išskirstytas) sistemas. Neigiamo savybė – nepakankamas našumas, kalta pranešimų sistema, Ji reikia pernešti, perduoti, gavus atkoduoti. Atsiranda daug perjungimų tarp vartotojo ir supervizoriaus režimų.

41. Įvedimo-išvedimo procesai. I/O valdymui kiekvienam I/O įrenginiui i sukuriamas jį aptarnaujantis procesas P_i . Jis atstovauja įrenginį sistemoje. Jo paskirtis inicijuoti I/O operacijas. Perduoti pranešimus apie pertraukimus ir pranešti apie I/O veiksmų pabaigą.

OA (1) Specializuotas procesorius su savo komandų sistema (registrais, valdymo įrenginiu). Perduoda duomenis tarp OA ir I/O kontrolerio
kan ① \
I/O kontr. ② /
Kelias (2) kontroleris – specializuotas kontreleris nukreiptas į tam tikrą tipą.
I/O įreng.
Begin

L: PRAŠYTIR(III⁽¹⁾, Ω⁽²⁾, (P⁽³⁾, IIprogr.⁽⁴⁾));

PRAŠYTIR (KK⁽⁵⁾, III⁽⁶⁾, Kelias);

Komutavimo ir įrenginių pranešimo komandos;

I/O inicijavimas;

PRAŠYTIR (IIP⁽⁷⁾, Kelias⁽⁸⁾, Pran)⁽⁹⁾;

Pertraukimo dekodavimas;

Papildomos I/O komandos;

Atsakymo suformulavimas;

ATLAISVINTIR(KK, Kelias);

ATLAISVINTIR(III, Pran, (P, Ats));

GOTO L:

End

Paaiškinimai: (1) Resursas yra I/O įrenginys (2) Nėra specifikuojama kokios resurso dalies reikia (3) Prašantis proceso P (4) Ką turėtų atlikti I/O įrenginį aptarnaujantis procesorius (skaitymą, rašymą, failo atidarymą, ...) (5) Kanalo kontroleris (6) Reikia kelio iki tokio įrenginio (7) I/O pertraukimas (8) Bet kuri iš kelio sudedamuju dalių (9) Sėkmės atveju jos nėra būtinės.

VM nustato I/O komandą. Įvyksta pertraukimas.

Pertraukimo apdorojimo metu nustatomas procesas, kuris turi aptarnauti konkretų pertraukimą, schematiškai tai aprašome:

```

Pi: Isiminti(CPU[*]);
    ST[*]:=READY;      // pertraukimas nepervedant proceso į blokavimo būseną
    PROC[P[*]]:=Ω;     // procesoriaus atlaisvinimas
    Nustatyti Pt;
    ATLAISVINTIR (PERT, (Pt, PERTR_INF));

```

Taimerio pertraukimo apdorotojas: jei viršytas laiko limitas, tai nutraukia proceso vykdymą, o jei viršytas procesoriaus kvanto limitas, tai perkelią į to paties prioritetų procesų sąrašą.

42. Failų sistemos sąvoka. FS yra OS dalis, kuri valdo įv/išv, įv/išv metu naudojamus resursus ir operuoja informacija failuose. I F galima žiūrėti kaip į virtualų įv/išv įrenginių tokį, kuris turi patogią programuotojui struktūrą. Programuotojui patogu operuoti failine informacija loginiame lygyje. I failą galima žiūrėti kaip į: (F, e), kur F - failo vardas, e – elemento identifikatorius failė.

Galima kalbėti apie virtualią failinę atmintį. FS virtualios failinės atminties paskirtis – aprūpinti vartotoją tiesine erdve jų failų patalpinimui. Pagrindinės funkcijos: 1. užklausimų VFA'iai transformavimas į RFA; 2. informacijos perdavimas tarp RFA ir OA. **FS** - tai OS dalis, kuri yra atsakinga už failinės informacijos sukūrimą, skaitymą, naikinimą, skaitymą, rašymą, modifikavimą bei perkėlimą. Failų sistema kontroliuoja failų naudojamus resursus ir priėjimą prie jų. Programuojam nesunku naudotis failų informacija, jai ji yra loginiame lygmenyje.

43. Failinės atminties įrenginių charakteristikos. Fizines failines atminties įrenginiai apibudinami tam tikromis charakteristikomis:

1. talpumas – maksimalus informacijos kiekis, kurios gali būti saugomos;
2. įrašo dydis – minimalus informacijos kiekis, į kuri galima adresuoti įrenginyje.
Įrašai įrenginiuose gali būti kintamo arba pastovaus ilgio.
3. priėjimo būdai: a) tiesioginis – operuojama aparatūra; b) nuoseklus – kai priėjimui prie įrašo reikalingas visų tarpinių įrašų peržiurejimas.
4. FA informacijos nešejes – tomas. Tomo charakteristika – jo pakeičiamumas – īgalina iš esmės padidinti vartotojo naudojamos VA apimtį. Pvz. Diskelis..
5. Duomenų perdavimo greičiai. Jis matuojamas baitais arba bitais per sekundę perduodant informaciją tarp OA ir įrenginio. KB – kbaitai, kb – kbitai
6. Užlaikymas. Įrenginiui gaves eilinę įv/iš komandą (jei tai juostinis įrenginys), jam reikia išibegti nuo praeito skaitmuo prie naujo pradžios. Jei diskas – užlaikymas – tai apsisukimas nuo pradžios iki tos vienos, kur yra informacija.
7. Nustatymo laikas. Tai galvučių perstumimo laikas(diskai).

Priklasomai nuo įrenginių charakteristikų ir nuo jų panaudojimo sąlygų, vieni ar kiti įrenginiai yra naudojami tam tikrais atvejais.

44. Failų deskriptorių. Aktyvių failų katalogas. Failo deskriptorių:

1. Failo vardas: FN;
2. Failo padėtis : įrenginio adresas, failo pradžios adresas įrenginyje;
3. Failo organizacija: nuosekli;
4. Failo ilgis: L;
5. Failo tipas: {pastovus, laikinas};

6. Failo savininkas: U;

7. Failo naudotojai: {U};

8. Failo apsauga: READ; (skirtas tik skaitymui)

Aktyvių failų kataloge (AFK) laikomi aktyvių failų deskriptoriai (aktyvūs failai - „atidaryti failai“). Neaktyvūs („neatidaryti“) failai neturi deskriptoriaus ir jų nėra AFK. Ieškant failo deskriptoriaus, pirmiausiai ieškoma AFK, tik paskui (jeigu nėra AFK) – sisteminiam kataloge (šiuo atveju deskriptoriaus nėra, todėl ieškoma pagal išorinį vardą, o tik tada yra sukuriamas deskriptorius.). AFK yra operatyvioje atmintyje.

45. Užklausimo failinei sistemai realizavimo pavyzdys. Darbui su failine atmintimi naudojama bendros paskirties failų sistema. Užklausimas skaitymui iš failo gali atrodyti taip:

Loginiame lygmenyje vartotojo pateikta komanda: READ(A) A- nuskaitoma OA sritis.

1. READ1(FN, A) // tarkim perskaito 80B įrašą

FN – išorinis failo vardas

A – adresas

Nuskaitys į adresą A[0]....A[79]

2. Tarkim r yra nuoroda į sekanti įrašą, kurį reikia nuskaityti:

READ2(FN, A, r, 80);

r := r + 80; r – sekantis įrašas; 80 - skaitomos informacijos ilgis baitais.

Skirtingi vartotojai gali parinkti vienodus vardus, todėl reikialingas unikalus vidinis failo vardas, taip pat failų deskriptorius. Pagal failo vidinį vardą galimą nustatyti failo deskriptorių, o failų deskriptoriai kaip ir patys failai saugomi išorinėje atmintyje. Failų atidarymo metu, failų deskriptoriai iš išorinės atminties perkeliami į vidinę atmintį, aktyvių failų katalogą.

READ3(d, A, r, 80)

Tam kad optimizuoti I/O realizavimo veiksmus naudojama buferizacija, t.y. skaitom aiščiaus išorinės atminties tam tikrais blokais. Tarkim, kad buferių blokas yra 10 buferių, o mūsų buferio dydis 80.

READ4(įreginys, įrašo adr., Buf[k], 800)

Įrašo adr. := įrašo adr + 800

Turi būti sugeneruotas apsikeitimo su išoriniu įrenginiu komandos priklausomai nuo architektūros. Įvykdžius komandą grąžinamas pranešimas f-jai READ(A).

46. Failų sistemos hierarchija. Failų sistemos funkcijas patogu apibūdinti pagal jų lygi, pradedant nuo aparatūrinio iki vartotojo serviso klausimų. Failų sistemos suskirstymas į loginius lygius: DBVS(5)-Priėjimo metodai(4)-Virtuali (loginė) failų sistema (3)-Reali (bazinė) failų sistema(2)-Ivesties/išvesties sistema(1).

1) koordinuoja fizinių įrenginių darbą. Šio lygio procesai atlieka informacijos blokų apsikeitimą tarp OA ir išorinės atminties pagal užduotą adresą.

2) transformuoja failo vidinį unikalų identifikatorių į failo deskriptorių.

3) pagal vartotojo suteiktą IV nustato jo vidinį unikalų vardą. Naudojamas vidinis failų katalogas. Virtualus lygis nepriklauso nuo fizinių įrenginių.

4) realizuoja dėsnį. Paga kurį apdorojami failo įrašai. Tokį dėsnį užduoda vartotojas pagal programos prasmę. Pvz.: nuoseklus priėjimas arba kai įrašai apdorojami pagal lauko (rakto) reikšmes didėjimo ar mažėjimo tvarka.

5) realizuoja failo loginės struktūros vaizdavimą į fizinę struktūrą.

48. Bazine failų valdymo sistema. Tai įvedimo/išvedimo sistema, kurioje aparatūros specifika izoliuoja įvedimą/išvedimą nuo likusios OS dalies. Tai leidžia įvedima/išvedimą nagrinėti persiunčiamą informacinių bloku terminais.

Prieš pasiunčiant informacijos bloką reikia nustatyti operatyvios atminties adresą ir fizinį bloko adresą išoriniame įrenginyje. Tokį adresą ir suformuoja bazine failų valdymo sistema pagal failo deskriptorių. Be to bazine sistema valdo išorinės atminties failus ir apdoroja tomų failų deskriptorius.

Darbui su deskriptoriais bazine sistema turi komandas(funkcijas):

SUKURTI failo vardas, sritis), kur sritis – iš kokių blokų sukurti failą. Ji vykdoma inicializuojant procesą, aptarnaujantį tomą. Veliau laisvos atminties failas skaidomas į dalinius failus.

DALINTI(failo vardas, sritis). Kai failas yra naikinamas, jo užimama atmintis turi būti atlaisvinta ir sugražinta į laisvos atminties failą.

IŠPLĒSTI(failo vardas, sritis). Komanda skirta išorinės atminties padidinimui.

ATLAISVINTI(failo vardas, sritis). Visa arba dalis išorinės atminties grąžinama į laisvos atminties failą.

Bazine sistema turi turėti ryšį su operatoriumi arba vartotoju tam, kad pranešti apie galimybę siūsti pranešimus apie tomų pakeitimų.

Tomų deskriptoriaus sudėtis:

Tomo vardas, unikalus tomo ID, proceso arba loginio įrenginio vardas, atitinkamas požymis, nuoroda į tomo turinio lentelę, informacija apie tomo apsaugą, tomo sukurimo ir galiojimo data.

Tomų valdymo bazine sistema turi funkcijas:

REGISTRUOTI(tomo vardas) – sukuriamas naujas tomo deskriptorius,

PAŠALINTI(tomo vardas),

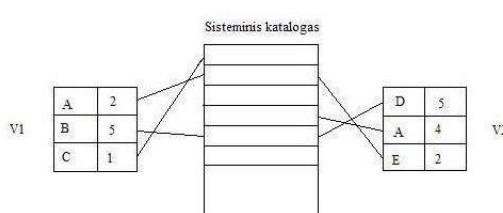
PRITVIRTINTI(tomo vardas) – tomo priskyrimas įrenginiui,

ATLAISVINTI(tomo vardas) – atjungimas nuo įrenginio.

49. Loginė failų valdymo sistema. Ji atvaizduoja lokalius vartotojo failų vardu į unikalius failų identifikatorius. Loginė failų valdymo sistema pateikia išoriniam interfeisiui komandas, kurias realizuoja bazine sistema, kuriose jau nurodomas unikalus F identifikatorius.

Loginė sistema reikalauja iš vartotojo pranešimo apie darbo su failais pradžią – komanda ATIDARYTI(funkciškai ją atlieka bazine sistema).

Vartotojo žinynas – tai informacija apie failus. Jis susieja failo išorinį vardą su jo unikaliu vardu, nurodančiu į bendrą sisteminių žinyną.



ATIDARYTI(failo vardas);

Vartotojo žinyno panaudojimas leidžia laisvai parinkti failo vardu ir užtikrina efektyvų kreipimąsi į failus.

Loginė sistema išoriniam interfeisiui pateikia komandas (jas realizuoja bazine sistema):

SUKURTI(failo vardas);

SUNAIKINTI(failo vardas);

UŽDARYTI(failo vardas);

SKAITYTI(failo vardas, bloko nr., OA, bloko sk.);

RAŠYTI(failo vardas, bloko nr., OA, bloko sk.);

Komandų rezultatai priklauso nuo konkrečios situacijos. Vartotojo procesas tas situacijas gali išskirti ir apdoroti arba naudoti kaip klaidą.

50. Priėjimo metodai. Loginės sistemos komandas naudoja bloko nr. Bet vartotojui nepatogu operuoti terminais. Tam įvedamas dar vienas failų sistemas lygmuo – priėjimo metodai.

Įrašų apdorojimui gali būti naudojami raktai. Dalinis raktas – duomenų laukas, kurio reikšmė duotu momentu gali atitikti vieną iš daugelio įrašų. Raktai gali būti naudojami įrašų identifikavimui.

Įrašai su vienodias galimas raktais apjungiami į sąrašą, todėl pakanka surasti tik pirmą sąrašo elementą.

Priėjimo prie įrašų metodai turi dvi charakteristikas: a) baziniai arba su eilutėmis; b) tiesioginiai arba nuoseklūs. Tiesioginiai – leidžia kreiptis į įrašus individualiai, o nuoseklūs – fizinė įrašų tvarka atitinka jų loginę tvarką.

Nuoseklaus ir tiesioginio metodų suderinimui naudojamas indeksacijos metodas.