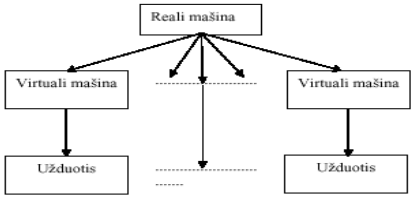
1. Operacinės sistemos sąvoka. OS – tai organizuota programų visuma, kuri veikia kaip interfeisas tarp kompiuterio ir vartotojo. OS sudaro tai kas vykdoma supervizoriaus rėžime. Ji aprūpina vartotojus priemonių rinkiniu, projektavimo ir programavimo palengvinimui, programų saugojimui ir vykdymui, ir 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 mažų mažiausiai tokia patikima, kaip aparatūra. Klaidos atveju, programiniame arba aparatūriniame 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 ryįio 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 autoriui pastovų ryšį su užduotimi. Ji leidžia vienu metu aptarnauti keletą vartotojų. Kiekvienam vartotojo procesui „kompiuteris“ suteikiamas nedideliam laiko kvantui, kuris matuojamas milisekundemis. Jei procesas neužsibaigė 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 – galimybė vienu metu vykdyti kelias užduotis.

3. Multiprogramavimo sąvoka. Multiprogramavimas atsirado kaip idėja, kuri turėjo reaguoti į skirtingus procesoriaus bei periferijos greičius. Multiprograminė operacinė sistema(MOS) – viena operacinių sistemų rūšių. Šio tipo operacinė sistema užtikrina kelių užduočių lygiagretų 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

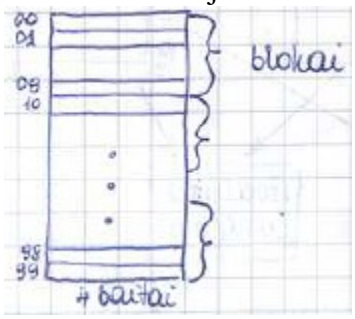
duomenis. Tuo metu vartotojas yra priverstas stebėti užduoties vykdymą ir tampa pasyviu.
Tam, kad galima būtų realizuoti MOS, kompiuterio architektūrai keliami tam tikri reikalavimai:
1.turi būti pertraukimų mechanizmas(jei jo nebūtų, liktų interpretavimo mechanizmas).
2.turi būti privilegijuotas režimas, t.y. esant privilegijuotam režimui uždrausti privilegijuotų komandų vykdyma – reikalavimas MOS realizacijai. Priešingu atvėju būtų labai ilgas darbas. MOS turi pasižymėti ta savybe, kad vienu metu dirbantys procesai neturi įtakoti vieni kitų(ar tai sisteminiai, ar vartotojo)
3. atminties apsauga.
Papildoma savybė – relokacija – 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 populiariausias 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 paslėpti 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 surenkame reikalingas realios mašinos komponentes, tokias kaip procesorius, atmintis, įvedimo/išvedimo įrenginiai, suteikiame jiems paprastesnę nei reali vartotojo sąsają ir visa tai pavadiname virtualia mašina. 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 tikrųjų, ir konkuruoja dėl realaus procesoriaus. Vienas esminių virtualios mašinos privalumų yra tas, kad užduotis, kurią vykdo virtuali mašina, elgiasi lyg būtų vienintelė užduotis visoje mašinoje. Tai yra didelė parama programuotojui. Dabar jam tenka rūpintis tik pačios programos rašymu. Pav.



Virtualios mašinos pateikimas užduotims multiprogramines operacinės sistemos atveju.
VM specifikacija. Tarkime, yra 100 žodžių atmintis (0-99). Kiekvienas žodis yra 4 baitų □□□□.

Žodžiai adresuojami nuo 0 iki 99. Tegul atmintis yra suskirstyta blokais po 10 žodžių.

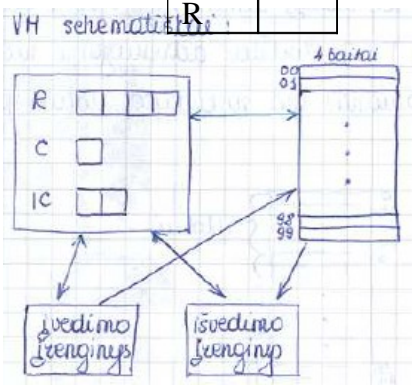


Procesorius turi 3 registrus: a) R – bendrasis registras □□□□ – 4 baitai; b) C – loginis trigeris, priima reikšmes true (T) arba false (F), kad būtų atliktas sąlyginis valdymo perdavimas □ – 1 baitas; c) IC – komandų skaitliukas □□ – 2 baitai.

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

OP	adr.

- komandos struktūra. VM turi nuoseklaus įvedimo bei išvedimo įrenginius.



Įvesties/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 є {0,..,9}; ADx_1x_2; **LR** – registro pakrovimas iš atminties - x_1,x_2 => R: = [a]; LRx_1x_2; **SR** – įsimenama registro reikšmė - x_1,x_2 => a: = R; SRx_1x_2; **CR** – palyginimo komanda - x_1,x_2 => if R = [a] then C: = 'T' else C: = 'F'; **BT** – sąlyginis valdymo perdavimas - x_1,x_2 => if C: = 'T' then IC: =a; BTx_1x_2; **GD –** apsikeitimas su išore vyksta blokais (x_1 – bloko nr) - x_1,x_2 => Read([β+i], i = 0,..,9), β= 10* x_1; GDx_1x_2; **PD** – išvedami duomenys - x_1,x_2 => Print([β+i], i = 0,..,9); PDx_1x_2; **H** – sustojimo komanda => HALT.

VM pradeda darbą, kai registro IC reikšmė yra 00 (įvykdo komandą, kuri patalpinta nuliniame žodyje).

5. Lygiagretūs procesai, notacija FORK, JOIN. Nuoseklus procesai veikia vienu metu – lygiagrečiai. Procesai neturi jokių tarpusavio sąryšių. Proceso aplinka 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(parelelizmas).

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, kaiteoriškai darbas gali būti vykdomas lygiagrečiai.
Aparapurinis paralelizmas įvedamas efektyvumo sumetimais, kad greičciau vyktų darbas. Procesoriuje nesant aparatūriniam 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 pasižimi tuo, kad kiekvienas procesas turi savo procesorių ir savo programą. Realiai, skirtingi procesai gali turėti tą patį procesorių ar tą pačią programą.
Procesas yra pora(procesorius, programa).
Procesas – tai būsenų seka s0,s1,...,sn, kur kiekviena būsena saugo visų proceso programos kintamųjų reikšmes. Pagal proceso būsena galima pratęsti 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 nou procesoriaus nepriklausomi sprendimai.
s1,s2 – sakinai
s1,s2 – procesai vyksta nuosekliai
s1 and s2 – 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.
Jei procesas p įvykdo komandą FORK W (išsišakojimas), tai iššakojimas proceso q vykdymas nuo žymės W.
Procesų aplinkybių komanda JOIN T,W:
T:=T-1;
IF T=0 THEN GOTO W; nepertraukiami veiksmai
Pvz:

```
N:=2;
FORK P3;
M:=2;
FORK P2;/*dirba 3 procesai,pirmas tas kuris viska inicializavo*/
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;

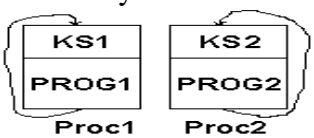
6. Kritinė sekcija. Tarkime turime du procesus p1 ir p2, kurie atlieka tą patį veiksmą x:=x+1(x-bendras kintamasis). jie asinchroniškai didina x reikšmę vienetu. p1 vykdo procesoriusc1 su registru r1, o p2 – c2 su r2. t_o = v

t_o----------->t_n laiko ašis

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]

Gavome: a)x=v+1 ir b)x=v+2, o taip būti negali, tai dviejų procesų problema. Ir pirmu atveju gali būti panasi situacija, kaip antru, dėl pertraukimų.
Programos dalis, dirbanti su bendrais procesų resursais, vadinama kritine 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. BOOLEAN PROCEDURE PP(x);
BOOLEAN x;
BEGIN
PP:=x;
X:= true;
END
Tegul turime dvejetaini semafora S, tada P(S) ekvivalenti L: IF PP(x) THEN GOTO L;
Kai S=0, tai x=true, kai S=1, tai x=false; S>0 – P(S) turetu neissaukti laukimo.
(Kai S=1 => x=false => PP(x) netenkina salygu; Kai s=0 => x=true =>PP(x) tenkina sal.,laukimas, kol x nepasidarys false)
V(x) ekvivalenti x:=false;
KS apsauga, naudojant auksciau aprasyta mechanizma, galetu atrodyti taip:
L: IF PP(x) THEN GOTO L;
KS;
x:= FALSE;

8. Dekerio algoritmas. Procesas atžymi savo norą įeiti į KS loginiu kintamuoju Ci=false. Išėjus iš kritinės sekcijos Ci=true. Įeiti į KS procesas gali tik tada, kai kitas procesas nėra

KS‘je arba nėra pareiškęs noro ją vykdyti. Sveikas kintamasis EILE naudojamas tada, kai du procesai susiduria KS‘je (pvs.: noras vykdyti KS, įėjimas į KS). Procesas, laukiantis, kol kitas procesas baigs vykdyti KS, kai eilė vykdyti KS priklauso kitam procesui.

```
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;
```

*) procesas pareiškia norą vykdyti KS.

v) tai antrasis procesas atsisakys noro vykdyti KS.

Toks sprendimas persudėtingas, kad juo remiantis toliau organizuoti darbą. Netinka, nes nėra tinkamų primityvų.

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 atvėju 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.
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;
```

Jei semaforas įgyja tik dvi reikšmes 0,1, tai jis vadinamas 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 pasiųs jam atitinkamą signalą.

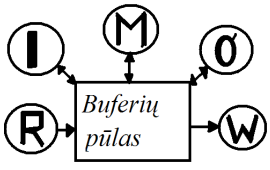
```
BEGIN SEMAPHORE S;
S:=0;
P1: BEGIN
…
P(S); // tai signalo iš proceso laukimas
...
END
and
P2: BEGIN
…
V(S); // siunčiamas signalas procesui P1
…
END
END;
```

Jei pirma bus vykdomas procesas P2, tai P1 neįeis į laukimo būseną.

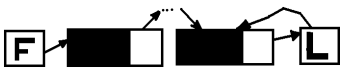
Skaičiavimo sistemose procesai charakterizuojami naudojamų ir atlaisvinamų resursų tipais.
Pvz2. Procesas gamintojas sukuria informaciją ir įrašo į buferį. Lygiagrečiai dirba proc. Naudotojas, kuris paima informacija iš buferio ir ją panaudoja(apdoroja).
Tegul buferio atmintis susideda iš N buferių.
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;

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. Naudojami tam du semaforai: T – tuščių buferių semaforas, U – užimtų buferių semaforas. Kadangi buferis bendras resursas abiem procesam, tai jis yra kritinė sekcija B, sauganti nuo kolizijų.

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 procesai, bet Read, Write..



R–Read; I–Input; M–Main; O–Out; W–Write;
Įvedimo proceso f-jos. Buferių pūlas – sąrašas(faktiškai 3 sąrašai). Iš pradžių buferių pūlas apjungtas į laisvų buferių sąrašą.



Iš pradžių 

- visas kūnas laisvi buferiai. Ir įvedimo buferių sąrašas, ir išvedimo buferių sąrašas. Adorojimo proceso užduotis paruošti informaciją išvedimui.
Veiksmų seka yra tokia: 1. Įvedimo procesas paima buferį iš laisvų buferių sarašo 2. Paleidžia jį skaitymui 3. Užpildytą buferį įjungia į įvedimo buferių sąrašą. 1)Pagrindinis apdorojimo procesas *MAIN* ima buferį iš įvedimo buferio sąrašo 2) Apdoroja informaciją 3)Ima buferį iš tuščių buferių sąrašo, ji užpildo, išveda informaciją. 4) Buferį įjungia į išvedimo buferių sąrašą.

Ivedimo procesas I gali bykti tik tada, kai yra laisvų buferių.M gali vykti, kai yra įvedimo sąraše elementai ir laisvi buferiai. Jei I ima paskutinį laisvą buferį, tai M užsiblokuoja, nes jau nėra laisvų buferių. Darbui su buferiais reikia apsirašyti tokius parametrus: laisvų buferių skaičių; įvedimo/išvedimo buferių skaičių; KS apsaugos semaforą; įvedimo/išvedimo buferių KS apsaugos semaforus.

12. Įvedimo-išvedimo SPOOLER'io pagrindinis procesas. Buferių pūlas – sąrašas (faktiškai 3 sąrašai). Iš pradžių buferių pūlas apjungtas į laisvų buferių sąrašą.

- įvedimo buferių sąrašas; - išvedimo buferių sąrašas.

Apdorojimo proceso užduotis – paruošti informaciją išvedimui. Veiksmų seka yra tokia:

1. įvedimo procesas paima buferį iš laisvų buferių sąrašo;
2. paleidžia jį skaitymui;
3. užpildytą buferį įjungia į įvedimo buferių sąrašą.
1. Pagrindinis apdorojimo procesas Main ima buferį iš įvedimo buferių sąrašo; 2. Apdoroja informaciją; 3. Ima buferį iš tuščių buferių sąrašo. Jį užpildo išvedama informacija; 4. Buferį įjungia į išvedimo buferių sąrašą.

Įvedimo procesas I gali vykti tik tada, kai yra tuščių buferių. Atitinkamai H gali vykti, kai yra įvedimo sąraše elementai ir laisvi buferiai. Jei I ima paskutinį laisvą buferį, tai H užsiblokuoja, nes jam nėra laisvų buferių.
nl – laisvų buferių skaičius; nin – įvedimo buferių skaičius; nout – išvedimo buferių skaičius; ml – KS apsaugos semaforas; min – įvedimo buferių KS apsaugos semaforas; mout – išvedimo buferių KS apsaugos semaforas.
Pagrindinio proceso H programa:
M: BEGIN LH:
P(nin); //H turi imti įved. buf. Jei tokio nėra, tai H blokuojasi.
P(min); // jei kitas procesas vykdo įvedimą, tai H blokuojasi.
Paimti pirmą buf. iš įvedimo buf. sąrašo;
v(min); // nuimama įvedimo KS apsauga;
Apdoroti buferio turinį;
P(nl);
P(ml);
Paimti pirmą buf. iš laisvų buf. sąrašo;
v(ml);
Uždaryti paimtą buf. išvedama info;
P(mout); // išved. buf. KS apsauga
Įjungti buf. į išved. buf. sąrašo galą; // imama iš sąrašo pradžios, bet rašoma į galą
v(mout);
P(ml); // buvo paimtas įved. buf. Jį reikia atlaisvinti. Tai darbas su KS
Įjungti buferį į laisvų buf. sąr. galą;
v(ml);
v(nl);
GOTO LH;
END;

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. Procesas R vykdo P(SR), o I – v(SR). Jei SR = 0, tai v(SR) = 1 ir procesas R galės baigti darbą. I blokuojasi nuo P(FR), o procesą R paleidžia v(FR). I ir R galima sinchronizuoti bendram darbui. I turi aprūpinti skait. įrenginį tuščiais buferiais, inicijuoti R ir prijungti į pūlą užpildytus per įvedimą buferius, kai R baigia darbą.

13. Įvedimo/išvedimo SPOOLER'io įvedimo ir skaitymo procesas. Atliekant įvedimo veiksmą, darbas turi būti sinchroniuojamas su įvedimo įrenginiu. Įvedimo įrenginio darbo pradžios ir pabaigos situacija(realiai pertraukimo situacija) modeliuosime semoforais.
SR – skaitymo įrenginio startas
FR – skaitimo įrenginio finišas
Procesas R vykdo P(SR), o I – V(SR). Jei SR=0, tai V(SR)=ą ir proceasas R galės baigti darbą. I blokuojasi nuo P(FR), o procesą R paleidžia V(FR).
I ir R galima sinchronizuoti benfram darbui. I turi aprūpinti skait.įrenginį tuščiais buferiais,inicijuoti R ir prijungti į pūtą užpildytus per įvedimą buferius, kai R baigia darbą.
<u>Įvedimo proceso valdymas</u>

```
I: BEGIN αI:    P(nl);
                 P(ml);
                IF liko paskutinis laisvas buferis THEN
                BEGIN V(nl); V(ml); GOTO αI 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 αI
END;
```

<u>Skaitimo proceso valdymas</u>

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

14. Dvejetainių ir bendrų semaforų ryšys. Semaforinių primityvų relizacijai, reikia bendruosius semaforus išreikšti dvejetainiais. Jei semaforas gali įgyti tik dvi reikšmes – jis dvejetainis. Bet kuris bendras semaforas gali būti išreikštas 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)~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)~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:= true;
END
S – dvejetainis semaforas
P(S); Uždrausti pertraukinus
L: if PP(x) then goto L;
S:=S-1;
If S<=1 then
Begin (užblokuoti iššaukiantį procesą)
Paimti proc.iš sąrašo A;
X:=false;
Perduoti proc.iš A; lesti pertrauk.;
End
ELSE Begin x:=false; lesti pertr.;
End
V(S); Uždrausti pertr.;
L: if PP(x) then goto L;
S:=S+1;
If S<=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.

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. Procesų aprašai dinaminiai objektai – jie gali būti sukurti/sunaikinti sistemos veikimu metu. Realiai procesą, kaip ir resursą OS-je atstovauja D. PD – tai tam tikra struktūra(ne masyvas) – jei kalbame apie visų P‘u D‘us, tai turime struktūrų masyvą, kur *i* – proceso VV – nurodytų struktūros numerį masyve.
PD – susideda iš komponenčių, kurioms priskiriame vardus:
1)*Id[i]* – proceso išorinis vardas, reikalingas statiniams ryšiams tarp rocesų nurotyti.
2) Mašina – čia turime omeny procesą vykdančio procesoriaus apibūdinimą.
2.1)*CPU[i]* – apibūdina centrinio procesoriaus būseną vykdant procesą. Kai proceso vykdymas nutraukiamas, proceso būsena išsaugoma.
2.2)*P[i]* – identifikuoja procesorių ir i-ąjį procesą(mūsų nagrinėtas OS modelis galejo turėti n procesorių).
2.3)*OA[i]* – proceso turimų resursų aprašas.Apibūdina resursą – operatyvią atminti, kuris apibūdinamas konkrečiai sąraše. D – fiksuota struktūra, jame yra nuoroda į sąrašą.
2.4)*R[i] – i-jo* proceso turimi resursai – informacija, kokius resursus yra gavęs procesas. Tai nuoroda į sąrašą, kuriame yra išvardinta resursai.
2.5)*SR[i]* – tai resurso kūrėjas. Jis atsakingas už sukurto resurso priežiūrą sistemoje.

18. Resurso deskriptorius. Resurso deskriptorius (Resurso valdymo blokas) yra fiksuoto formato duomenų struktūra, sauganti informaciją apie resurso einamąjį 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.

19. Primityvas „kurti procesą“. Norint sukurti procesa reikia kreiptis į OSBP „kurti procesa“, faktiniais parametrais nurodant tokiais kuriamo proceso komponentes:
N – išorinis vardas
S0 – kurimo proceso procesoriaus pradine būsena;
M0 – OA pradinė būsena (kiek išskirta OA resursu);
R0 – kiti išskiriami resursai;
K0 – proceso prioritetas
IV naudojamas ryšiams tarp procesų nurodyti
PROCEDURE KURTIP(n,s0,M0,R0,k0);
BEGIN
I:=NVV;//reikia nustatyti ir gauti proceso VV.NVV gražina naują numerį
Id[i]:=n;//Id proceso IV
CPU[i]:=s0;//taip užrašoma procesoriaus būsena deskriptoriuje
OA[i]:=M0;//procesoriaus deskriptoriuje turimas OA kiekis
R[i]:=R0;//kitų turimų proceso resursų komponente(nouroda į sąrašą)
PR[i]:=k0;//prioritetu laukas

ST[i]:=READYS;//laikysime, kad procesas sukuriamas su statusu(pvz. pasiruošes)
SD[i]:=PPS;//kadangi kiekvienas procesas turi priklausyti bent vienam sarašui, tai priskiriame ji PPSui
T[i]:=*;//duotu metu dirbantis procesas bus žymimas "*". Tai einamojo proceso VV
S[i]:=Λ;//procesas turi nuoroda į sūnų sąrašą iš pradžių jis tuščias
Įjungti(s[*],i);//operuojame sarašais suantraštemis, o antraštes – tai programos, dirbančios su sąrašais. Dabar dirbantis(einamasis) procesas įgijo sūnų. Reikia papildyti jo sūnų sąrašą.
Įjungti(PPS,i);//naują procesą reikia įtraukti į pasiruošusiu procesų sąrašą.
END;
Primityvas "kurti procesą" informaciją apie kuriamą procesą suregistravo į deskriptorių. Čia jokia informacija nekuriama.

20. Primityvas „naikinti procesą". OSBP „Naikinti procesą". Procesas gali sunaikinti bet kurį savo palikuonį, bet negali sunaikinti savęs. Tada jis nusiunčia pranešimą savo tėvui, kuris jį ir sunaikina.
Procesas Main_Proc sukuria Job_Governer, kai yra „Užduoties išorinėje atmintyje" resursas Job_Governor negali savęs sunaikinti. Tada kuriamas fiktyvus resursas ir Job_Governor užsiblokuoja. Tada jį galima naikinti.
Ar naikinti vieną procesą ar visą pomedį? Jei sunaikinsime vieną procesą, tai sistemoje bus nevaldomų procesų (bus chaosas). Reikia naikinti visą pomedį. Ar naikinti visus resursus? Juk yra ir pakartotino naudojimo resursai. Juos reikia atlaisvinti.
Parametrai – naikinimo proceso IV.
PROCEDURE NAIKINTIP(n);
BEGIN
L:=false;[1]
i:=VV(n);
P:=T[i];
Pašalinti(S[p], i);
NUTRAUKTI(i);[2]
IF L = ? THEN PLANUOTOJAS;
END;
Paaiškinimai:
1. jei L = true, tai iškviečiamas planuotojas; 2. nutraukti i-tąjį procesą. Procedūrai perduodamas proceso VV.
PROCEDURE NUTRAUKTI(i);
BEGIN
IF ST[i] = RUN THEN
BEGIN
STOP(i);
L:=true;
END;
Pašalinti(SD[i], i);[1]
FOR ALL S є S[i] DO NUTRAUKTI(s);[2]

FOR ALL r є OA[i] VR[i] DO[3]
IF PNR THEN Įjungti(PA[r], Pašalinti(OA[i] VR[i]));
FOR ALL r є SR[i] DO NDR(r);[4]
NPD(i);[5]
END;
Paaiškinimai:
1. kadangi procesas visada yra kažkokiame sąraše, tai reikia jį iš ten pašalinti. SD saugo nuorodą į sąrašą, iš kurio reikia pašalinti, i – VV; 2. reikia nutraukti i – tojo proceso vykdymą; 3. r – resursai; atlaisviname pakartotinio naudojimo resursus; 4. liko sunaikinti proceso sukurtų resursų deskriptorius; 5. naikiname proceso deskriptorių.
Prioritetas atitinka proceso padėtį pasiruošusių procesų sąraše kiek prioritetų, tiek sąrašų.

21. Primityvas „Stabdyti procesą“. OSBP.“Stabdyti procesą“ Galimas dvigumas šios operacijos traktavimas, nes procesas yra tam tikro medžio pomedžio šaknis. Kyla klausimas: ar stabdyti vieną procesą, ar visus to pomedžio procesus? Tarkime, jog stabdomas vienas procesas, o likę sūninei procesai tęsia darbą. Parametrais nurodome: n – išorinis vardas, a – atsakymų srities adresas į kurį gražinama informacija apie stabdomą deskriptoriaus būseną(jo kopija).
Procedure OSBP(n,a); Begin i:=vv(n);[1)] s:=st[i];[2)] if s=RUN then STOP(i);[3)] a:=copydest[i];[4)]if [5)] s=BLOCK or BLOCKS then ST[i]:=BLOCKS else ST[i]READYS; if s=RUN then PLANUOTOJAS;[6)] End
1)Pagal IV nustatomas proceso VV 2)Statuso reikšmė 3)Stabdomas procesas gali būti vykdomas, pasiruošęs, blokuotas. Pirma, atemame procesorių.STOP[i] pertraukia procesorių, kuris vykdo i-ąjį procesą.Tai procesorius P[i].Reikia įsimini pertraukio procesoriaus būseną į CPU[i] ir pasakyti, kad procesorius vykdęs i-ąjį procesą yra atsilaisvinęs PROC[P[i]]:=1 4)Procedūra padaranti deskriptoriaus kopiją 5)Koreguojamas i-ojo proceso statusas 6)Jei buvo pristabdytas vykdomas procesas, tai reiškia iškviesti planuotoją, nes kiti procesai laukia procesoriaus. Planuotojo vykdymui naujas procesas nesukuriamas. Svarbu, kad iki planavimo veiksmo viskas jau būtų padaryta, nes planuotojo darbo pasekoje, procesorius atimamas iš proceso, kuriame jis pats yra vykdomas.

22. Primityvas „aktyvuoti procesą“. OSBP „Aktyvuoti procesą“. Tai simetrinės OSBP. Nuima pristabdymo buseną. Galbūt iškviečia planuotoją, jei būsena yra READY.
Parametras n – IV.
PROCEDURE AKTYVUOTI(n);
BEGIN
i:=VV(n);[1]
ST[i]:=IF ST[i]=READYS THEN REDY ELSE[2] BLOCK;
IF ST[i]=READY THEN PLANUOTOJAS;
END;
Paaiškinimai:
1. nustatomas VV pagal IV; 2. reiškia buvo BLOCKS.

23. Primityvas „keisti proceso prioritetą“. Prioritetas atitinka proceso padėtį pasiruošusių procesų sąraše. Kiek prioritetų, tiek sąrašų.
OSBP „Keisti proceso prioritetą“. Kalbame apie prioritetą procesoriaus resursų atžvilgiu. 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. Parametrai n-IV; i – prioritetas.

```
PROCEDURE KEISTIPP(n,k);
BEGIN
        I:=VV(n);
        M:=PR[i];[1]
        Pašalinti(SD[i],i);
        PR[i]:=k;[2]
        Įjungti(SD[i],i);
        If M<k and ST[i]=READY THEN PLANUOTOJAS
END;
```

1. įsimenamas senas prioritetas;
2. priskiriamas naujas prioriteyas;

24. Primityvas „kurti resursą“. Resursus kuria tik procesas. Resurso kūrimo metu perduodami kaip parametrai: resurso išorinis vardas, pakartotinio naudijo požymis, nuoroda į resurso prieinamumo aprašymą, nuoroda į resurso laukiantį procesą, nuoroda į paskirstytojo programą. Resursas kūrimo metu yra: pridedamas prie bendro resursų sąrašo, prie pakartotinio naudojimo resursų sąrašo, jam priskiriamas unikalus vidinis vardas, sukuriamas laukiančių procesų sąrašas ir pan.:

Procedure KURTIR (RS, PN, PA_o, LPS_o, $PASK_o$);
Begin
r:=NRVV; {naujas resurso vidinis vardas}
Rid[r]:=RS; {išorinis resurso vardas, resursas prijungiamas prie bendro resursų sąrašo}
PNR[r]:=PN; {loginis požymis, ar pakartotinio naudojimo}
k[r]:=; {resurso kūrėjo tėvo vidinis vardas, duotu metu vykstantis procesas, kuriame ir panaudojamas šis primityvas}*
PA[r]:=PA_o; {nuoroda į resurso prieinamumo aprašymą}
LPS[r]:=LPS_o; {šio resurso laukiančių procesų sąrašas}
PASK[r]:=$PASK_o$; {nuoroda į resurso paskirstytojo programą}
Įrašyti (SR[*], r); {į šiuo metu vykdomo proceso sukurtų resursų sąrašą įtraukiamas ir šis resursas}
End;

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

```
PROCEDURE NAIKINTIR(RS);
Begin
r:=RVV(RS);
R:=Pasalinti(LPS[r]);
While R<>Ω do
```

```
Begin
ST[R.P]:=IF ST[R.P]=BLOCK then READ
else READS;
Irasyti (PPS,R.P);
SD[R.P]:=PPS;
R.A:='PRAN';
R:=Pasalinti(LPS[r]);
End
NRD(r); //Naikinri resurso deskriptoriu
Planuotojas;
End
```

26. Primityvas "prašyti resurso". OSBP "Prašyti resurso". Procesas, kuriam reikia resurso, iškviečia šį primityvą, nurodydamas VV ir adresą. Toks procesas pereina į blokavimosi būseną. Blokavimasis įvyksta tik prašant resusrso. Procesas įjungiamas į laukiančių to resusro procesų sąrašą. Parametrai: RS – resusrso VV; D – kokios resusrso dalies prašoma: A – atsakymo srities adresas, į kur pranešti.

```
PROCEDURE PRASYTIR(RS, D, A);
BEGIN
  R := RVV(RS); //(1)
  IJUNGTI(LPS[r], (*, D, A)); //(2)
  PASK(r, K, L); //(3)
  B := true;
  FOR J :=1 STEP 1 UNTIL K DO
  IF L[J] <> * THEN //(4)
  BEGIN
    I:=L[J];
    IJUNGTI(PPS, i);
    SD[i]:=PPS;
    SD[i]:=PPS;
    ST[i]:= OF ST[i]=BLOCK THEN READY
                                    ELSE READYS; // (5)
  END
  ELSE B:=false; //(6)
  IF B THEN
  BEGIN
    ST[*]:=BLOCK;
    SD[*]:=LPS[r];
    PROC[P[i]]:=1; //(7)
    PASALINTI(PPS, *)
  END
  PLANUOTOJAS
END
```

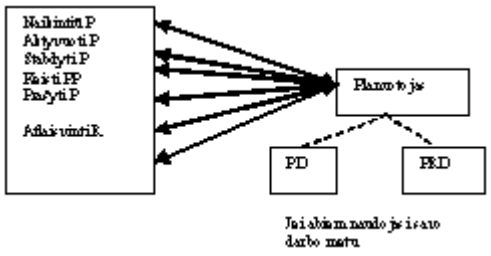
Paaiškinimai:

1) nnustatomas resurso VV pagal IV;
2) procesas įjungiamas į lauiančių šio resurso procesou sąrašą, * - šiuo metu vykdomas procesas;
3) resurso paskirstytojo programa. K – kiek procesų aptarnauja, L – aptarnautų procesų VV masyvas;
4) ar tai šiuo metu nedirbantis procesas?
5) reiškia buvo BLOCKS;
6) reiškia tai yra duotu metu dirbantis procesas;
7) šį procesą vykdžiusį procesorių paskelbiame laisvu.

27. Primityvas „atlaisvinti resursą“. OSBP “Atlaisvinti resursą”. Tai atitinka situaciją, kai procesas gauna pakartotino naudojimo resursą ir kai jo jam nereikia, jis jį atlaisvina ir įjungia į sąrašą laisvųjų redursų. Kaikurie resursai sujuriami proceso darbo metu.
Parametrai: RS – resurso IV, D – OA atlaisvinamos dalies aprašymas.

```
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. Procesų planuotojas. Planuotojas užtikrina, kad visi PP būtų vykdomi maksimaliu prioritetu (Jei prioritetai vienodi, tai vykdomas tas kuris sąraše pirmesnis).
Planuotojo veikimo schema: (Planuotojas iškviečiamas iš branduolio primityvų)



Planuotojo darbas 3 etapais: 1) Surasti pasirenkamą procesą su aukščiausiu prioritetu. 2) Surasti neaktyvų procesorių ir jį išskirti.(skirti pasiruošusiam procesui) 3) Jei procesorių nėra, peržiūrėti vykdomus procesus, ir, jei jų prioritetai mažesni, atiduot procesorių pasiruošusiems (Perskirstymas)

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

```
PROCEDURE Planuotojas;
BEGIN
p := PTR  = N;
```

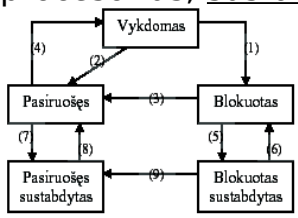
```
c := 1;
L: B := TRUE;
WHILE B AND PTR <> (!=, ne lygu) 0 DO
//  čia iš principo ir prasideda 29 klausimas
BEGIN
p = P[p];
IF p = PTR THEN
BEGIN // pereinama prie kito prioriteto
PTR := PTR -1;
p := PTR;
END
ELSE B := ST[p] <> (!=, ne lygu) READY;
END
// p saugo point. proc. su aukščiausiu prioritetu vidinį vardą.
WHILE c <= 4p DO // 2 fazė
IF PROC[c] <> (!=, ne lygu) Ω THEN c := c + 1;
ELSE BEGIN
PROC[c] := p;
P[p] := c;
ST[p] := RUN;
IF c <> (!=, ne lygu) P[x] THEN Atstatyti būseną (c, PU[p]);
ELSE p * := p; // jam reikš atiduot procesorių
c := c + 1;
GOTO L;
END
```

30. Procesų planuotojo perskirstymo etapas. *// 3 fazė*

```
PRTMIN := PRT;
FOR c := 1 STEP 1 UNTIL np DO
BEGIN
g := PROC[c]; // imama proceso vid. vardą
IF PR[g] < PRTMIN THEN
BEGIN
PRTMIN := PR[g];
cp := c;
END; // išsaugomas procesorius, kur vykdo proc. su mažesniu prior.
IF PRT != PRTMIN THEN // buvo rasta mažesnių prioritetų orocesų
g := PROC[cp];
ST[g] := READY;
P[p] := cp;
PROC[cp] := p;
ST[p] := RUN;
IF g = * THEN p* := p;
ELSE Perjungti (p, g, cp);
```

```
GO TO L;
END
ENDPROC := IF ST[*] != RUN THEN Perjungti (p*, *, p[*])
PROCEDURE Perjungti (p, g, c)
BEGIN
Pertraukti(c); // Jei p[*] != c, priešingu atv. nedaro nieko
Įsimint_būseną (c, CPU[g]);
IF  g := * THEN ĮsimintiGA(GA, CPU[g]);
Atstatyti_būseną (c, CPU[p]);
END;
```

31. Procesų 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 tęsti savo darbo be reikiamo resurso). Tačiau, jei procesas nereikalauja jokio resurso, iš jo gali būti atimamas procesorius, pavyzdžiui, vien tik dėl to, kad pernelyg ilgai dirbo. Tai visiškai skirtinga busena nei lokavimasis 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.

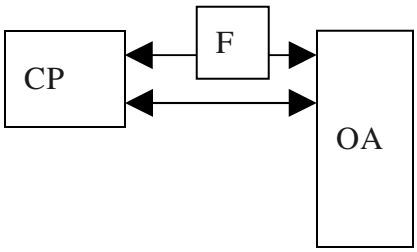


(1)vykdomas procesas blokuojasi jam prašant ir negavus resurso.(2)vykdomas procesas tampa pasiruošusiu atėmus iš jo procesorių dėl kokios nors priežasties (išskyrus resurso negavimą). (3)Blokuotas procesas tampa pasiruošusiu, kai yra suteikiamas reikalingas resursas.(4)pasiruošę procesai varžosi dėl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

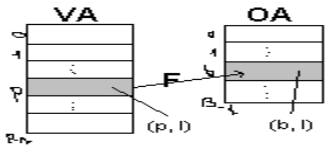
32. Virtualios atminties sąvoka. OA – tai ta, į kurią procesorius gali kreiptis tiesiogiai imant komandas ar duomenys programos vykdymo metu. Tokia schema turi daug nepatogumų, kai programoje nurodomi absoliutūs adresai.

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 adresų nustatymą – statinį atminties skirstymą. Dinaminis atminties skirstymas – kai adresai nustatomi betarpiškai kreipiantis į atmintį. Statiškai nustatant adresus būtina prieš programos vykdymą žinoti išskirtos atminties vietą. Dinaminis – kai fizinis adresas nustatomas tiesiogiai prieš kreipimąsi į tą adresą.
Sudarant programas tenka abstrahuotis nuo atminties žodžių ir nauduoti tam tikrus lokalius adresus, kurie vėliau kokiu tai būdu susiejami su fiziniais adresais. Tokia lokalių adresų visuma – virtuali atmintis. Fizinių adresų visuma – reali atmintis. Virtuali atmintis su fizine gali būti susiejama statiškai arba dinamiškai.

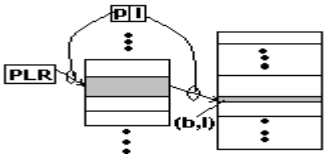
33. Komandos vykdymo schema. Schematiškai pavaizduosime komandų vykdymą, kuomet nėra dinaminio atminties skirtymo:
H[O:N] – atmintis;
K – komandų skaitliukas;
R – registras.
L: W:=H[K];
    OK:=W op kodas; // ok – operacijos kodas
    ADR:+ W operando adr;
R:=K+1;
Add: IF OK=1 THEN R:=R+H[ADR];
ELSE
SR: IF OK=2 THEN H[ADR]:=R;
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]~M[F(ADR)].
Į F galima žiūrėtu 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 $b_0b_1...b_{B-1}$. OA absoliutus 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 $p_0p_1...p_{P-1}$. 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).

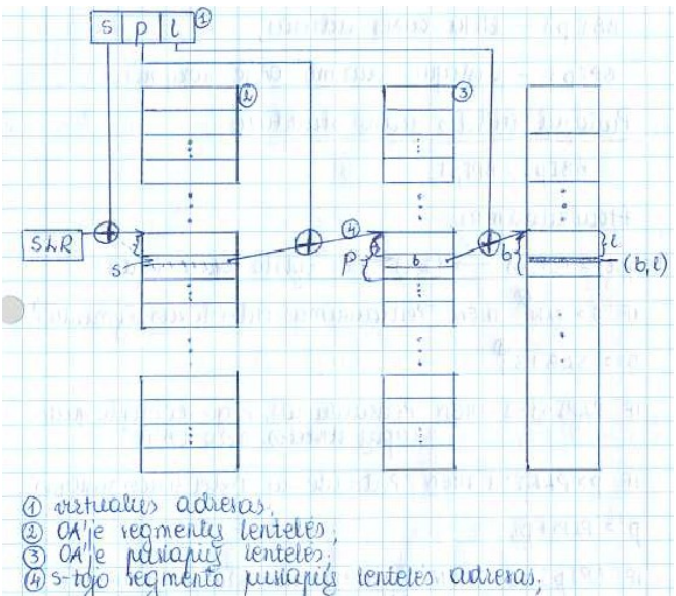


Puslapinės organizacijos schema: a) VA turime [p|l]; b) aparatūroje turi būti numatytas [PLR] – puslapių lentelės registras(rodo aktyvaus proceso puslapių lentelę); c) puslapių lentelės saugomos atmintyje.



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. 1) Turimas segmentų (toliau S) lentelės registras, rodantis į aktyvaus proceso segmentų lentelę (SL), o SL kiekvienas įrašas rodo į ištisinę srytį operatyvios atm.
2) Puslapinė organizacija su segmentacija. SLR rodo į aktyvaus proceso SL. Kiekvienas S 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.



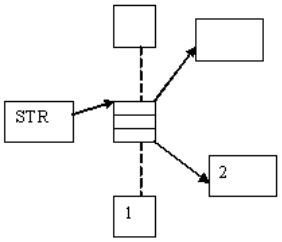
SLR – Segmentų lentelės registras, saugo aktyvaus proceso segmentų puslapių lentelės adresą.
PSL – Saugo bloko numerį į kurį tas puslapis atvaizduojamas.

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 //nera atminty
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”
(irgi del sito viso neaisku)

36. Atminties skirstymo puslapiais strategijos. Puslapių skirstymo uždavinys turi atsakyti į klausimus:
a) kokiu momentu sukeisti puslapius;
b) kokį puslapį patalpinti į atmintį (iš išorės į OA);
c) vietoje kokio puslapio.
Puslapių skirstymo strategija, pagal kurią vieta puslapiui skiriama betaroiškai prieš į jį kreipiantis, vadinama strategija pagal pareikalavimą (SPP). Ji užfiksuoja atsakymus į pirmuosius du klausimus.
Įrodyta, 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:
B_1 | p_1 p_1 p_1 p_1 p_1 p_1 p_1 p_1
B_2 | p_2 p_3 p_4 p_4 p_3 p_2
Šitokia strategija yra optimali. Tačiau iškyla taikymo problemos, nes prieš programos vykdymą puslapių trasa nėra žinoma.
Praktikoje naudojamos tam tikros strategijos, kai pakeičiamas: atsitiktinis, ilgiausiai būnantis OA’je, į kurį buvo kreiptąsi seniausiai.
Vidutiniškai du kartus mažiau puslapių pakeitimų, jei naudojama trečioji strategija: naujas puslapis įrašomas vietoj seniausiai naudoto.

37. Atminties išskyrimas ištisinėmis erdvėmis. Segmentui atmintis yra išskiriama ištisiniais kintamo dydžio atminties blokais; Kiekvienam aktyviam procesui OA yra saugoma segmentų lentelė, kurios kiekvienas i-tasis įrašas saugo i-tojo segmento adresą OA. Segmentų lentelės registras STR saugo dabar dirbančio proceso segmentų lentelės adresą. Efektyvus adresas [s, w] atvaizduojamas į realų tokiu būdu: F(s, w) = M[STR + s] + w
Iškyla išorinės ir vidinės fragmentacijos problema.



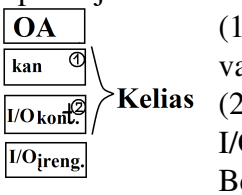
1-Segmentų lentelė
2-Segmentas

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. Į 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 iškviesti 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 sluoksiniuose. Sunku kūrti versijas pagal konkrečią konfigūraciją. Sunku spręsti saugumo problemas dėl gretimų sluoksnių 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 branduolį. 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 paskirtytas (išskirstytas) sistemas. Neigiama savybė – nepakankamas našumas, kalta pranešimų sistema, Jį 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 *Pi*. Jis atstovauja įrenginį sistemoje. Jo paskirtis inicijuoti I/O operacijas. Perduoti pranešimus apie pertraukimus ir pranešti apie I/O veiksmų pabaigą.



(1)Specializuotas procesorius su savo komandų sistema(registrais, valdymo įrenginiu). Perduoda duomenis tarp OA ir I/O kontrolerio (2)kontroleris – specializuotas kontreleris nukreiptas i tam tikrą tipą.
I/O įrenginį atstovaujantis procesas *Pi.*
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 sudedamųjų dalių (9)Sėkmės atveju jos nėra būtinos.
VM nustato I/O komandą. Įvyksta pertraukimas.
Pertraukimo apdorojimo metu nustatomas procesas, kuris turi aptarnauti konkretų pertraukimą, schematiškai tai aprašome:

```
Pi: Įsiminti(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. Į F galima žiūrėti kaip į virtualų įv/išv įrenginį tokį, kuris turi patogią programuotojui struktūrą. Programuotojui patogu operuoti failine informacija loginiame lygyje. Į failą galima žiūrėti kaip į: (F, e), kur F - failo vardas, e – elemento identifikatorius faile.
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. Programuotojam nesunku naudotis failų informacija, jai ji yra loginiame lygmenyje.
Failinė struktūra- sutvarkyta elementų struktūra, kurios elementai identifikuojami taip: (F,e) čia F- failo vardas, e- e-tasis failo f elementas.
Žinant failinės atminties realizaciją, aparatūros specifiką, galima optimizuoti jos apdorojimą. Pagrindinės operacijos kurias galima atlikti failų sistemoje:
1. Užklausimo virtualioje failinėje atmintyje transformavimas į realią failinę atmintį.
2. Informacijos perdavimas tarp realios failinės atminties ir operacinės atminties.

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 – įgalina iš esmės padidinti vartotojo naudojamos VA apimtį. Pvz. Diskelis..
5. Duomenu 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 įsibegti nuo praeito skaitmuo prie naujo pradžios. Jei diskas – užlaikymas – tai apsisukimas nuo pradžios iki tos vietos, kur yra informacija.
7. Nustatymo laikas. Tai galvučių perstumimo laikas(diskai).

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

44. Failų deskriptorius. Aktyvių failų katalogas. Failo deskriptorius:
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 AKF) – 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:
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;
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.
Dažniausiai naudojami užklausimai darbui su failine sistema:
Create – sukuriamas failas be jokių duomenų. Jo tikslas yra užtikrinti jok toks filas yra ir nustatyti jo atributus.
Delete – kai failas daugiau yra nebereikalingas jis turi buti ištrinamas kad atlaisvinti disko vietą. Visada egzistuoja toks sistemos užklausimas.
Open – prieš failo naudojimą procesas turi jį atidaryti. Open užklausimo tikslas yra leisti sistemai perduoti failo atributus bei disko adresų sąrašus į pagrindinę atmintį, kad butų galimą greitai prieiti busimoms užklausoms.
Close – kai priejimas nebereikalingas, atributai bei disko adresai – nebereikalingi, failas turi būti uždaromas kad atlaisvinti atmintį. Daugelis sistemų skatina tai nustatydamos maksimalų atidarytų failų skaičių.
Read – duomenys nuskaitomi iš failo. Užklausėjas turi nurodyti kiek duomenų yra reikalinga ir turi suteikti buferi jiems patalpinti.
Write – duomenys įrašomi į failą. Jei esama pozicija yra failo pabaiga, failo dydis yra padidinamas. Jei esama pozicija yra failo viduryje tokiu atveju esami duomenys yra perrašomi ir prarandami visiems laikams.
Rename – leidžiama vartotojui pakeisti failo pavadinimą. Ši užklausa nera butina, kadangi galima kopijuojant failą suteikti jam naują vardą.
Set attributes – galima pakeisti kai kuriuos atributus jau sukurtiems failams.

Seek – užklausa kuri nustato failo žymę i tam tikrą poziciją. Po šios užklausos duomenys gali būti rašomi arba nuskaitomi nuo tos žymės.
Append – ši užklausa yra apribota write užklausa. Duomenys gali buti įtraukiama tik į failo pabaigą.

46. Failų sistemos hierarchija. Failų sistemos funkcijas patogu apibūdinti pagal jų lygį, 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)-Įvesties/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 FD.
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.
Tegul failas A susideda iš 11 įrašų, kurių kiekvienas 250 baitų.
Saugoma po 1000 baitų kiekviename bloke, t.y. į vieną bloką telpa 4 loginiai įrašai. Failo A įrašai prasideda antrame bloke. Tarkime programoje yra užklausimas nuskaityti failo A įrašą 6: READ FILE(A) RECORD(6) SIZE(250) LOCATION(BUF); Transliatorius, sutikęs tokį operatorių, perves jį į kreipinį:
1) nustatyti fizinį failo adresą;
2) nustatyti bloką, į kurį įeina įrašas 6;
3) nuskaityti bloką į OA;
4) persiųsti iš OA nuskaityto bloko 6-ąjį įrašą į sritį BUF.
Kad būtų iš OA nuskaitytas failo A pradžios fizinis adresas, yra naudojama toro turinio lentelė VTOL. Tomo turinys, kaip failas, yra saugomas tame pačiame bloke.
Schematiskai: (lentelė) [vardas | ilgis | adresas] [A | 2750 | 2] [B | 900 | 6] [C | 2000 | 7].
Kad toks užklausimas failinei sistemai būtų patenkintas:
1) VTOL‘je nustatyti failą A;
2) nustatyti santykinį adresą: (įr.nr-1)*įr.dydis=1250;
3) pagal santykinį adresą sustatyti, kuris failo blokas reikalingas (mūsų atveju 1);
4) nustatyti fizinį bloką, kuriam priklauso pirmas (šiuo atveju) santykinis blokas (mūsų atveju 3)
5) trečias fizinis blokas nuskaitomas į OA pagal skaitymo veiksmą;
6) iš nustatyto bloko į buferį paimami baitai nuo 250 iki 499, ir jie persiunčiami į sritį su vardu BUF.
Darbo su failais metu kiekvieną kartą veikti bloką su failo pradžia – neefektyvu. Todėl tas įrašas perkeliamas į failų katalogą iš tomo turinio lentelės ir vadinamas atidarymo veiksmu (deskriptorius pernešamas į OA).
Esant nuosekliam įrašų apdorojimui, galima sumažinti įv/iš veiksmų skaičių. Mūsų atveju nuskaitoma 3 kartus.

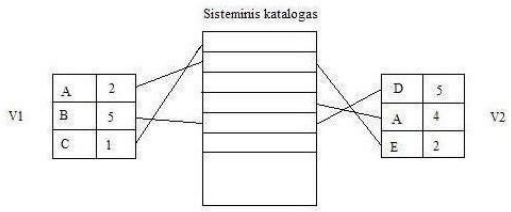
Pakanka turėti požymį, koks blokas yra buferyje, kad daugiau nebūtų skaitoma iš failo. Metodas-buferizacija. Kiekvienam failui gali būti savas buferis arba visiems failams vienas buferis, arba vienam failui keli buferiai

47. Failų sistemos įvedimo-išvedimo posistemiai. Ją sudaro procesai, valdantys įvedimo/išvedimo įrenginių darbą. Kiekvienam įrenginiui savas procesas. Ryšys tarp procesų atitinką ryšį tarp įrenginių. Įvedimo/išvedimo sistema ryšį tarp įvedimo/išvedimo įrenginių transformuoja į ryšį tarp įvedimo/išvedimo procesų.
Privalumai:
1)asinchroninis įrenginių darbas valdomas kaip nepriklausomas procesas.
2)priklausomybė nuo įrenginių specifikacijos lokalizuojama žemiausiame lygyje; į kitus lygius perduodama unifikuota informacija.
3)ypatingos situacijos apdorojamos artimniausiame lygyje. Įvedimo/išvedimo sistema susideda iš komponenčių.
Komponentės:
1)disko valdymas 2) terminalai 3) periferiniai įrenginiai
įvedimo/išvedimo įrenginių specifika leidžia jį nagrinėti persiunčiamų blokų terminais.

48. Bazinė failų valdymo sistema. Tai įvedimo/išvedimo sistema, kurioje aparatūros specifika izoliuoja įvedimą/išvedimą nuo likusios OS dalies. Tai leidžia įvedima/išvedimą nagrinėti persiunčiamų informacinių bloku terminais.
Prieš pasiunčiant informacijos bloką reikia nustatyti operatyvios atminties adresą ir fizinį bloko adresą išoriniame įrenginyje. Tokį adresą ir suformuoja bazinė failų valdymo sistema pagal failo deskriptorių. Be to bazinė sistema valdo išorinės atminties failus ir apdoroja tomų failų deskriptorius.
Darbui su deskriptoriais bazinė 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.
IŠPLĖSTI(failo vardas, sritis). Komanda skirta išorinės atminties padidinimui.
ATLAISVINTI(failo vardas, sritis). Visa arba dalis išorinės atminties grąžinama į laisvos atminties failą.
Bazinė sistema turi turėti ryšįsu operatoriumi arba vartotoju tam, kad pranešti apie galimybę siūsti pranešimus apie tomų pakeitimus.
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 bazinė 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 vardus į unikalius failų identifikatorius. Loginė failų valdymo sistema pateikia išoriniam interfeisui komandsa, kurias realizuoja bazinė sistema, kuriose jau nurodomas unikalus F identifikatorius.
Loginė sistema reikalauja iš vartotojo pranešimo apie darbo su failais pradžią – komanda ATIDARYTI(funkciškai ją atlieka bazinė sistema).
Vartotojo žinynas – tai informacija apie failus. Jis susieja failo išorinį vardą su jo unikaliu vardu, nurodančiu į bendrą sisteminį žinyną.



Vartotojo žinyno panaudojimas leidžia laisvai parinkti failo vardus ir užtikrina efektyvų kreipimąsi į failus.
Loginė sistema išoriniam interfeisui pateikia komandas (jas realizuoja bazinė sistema):
SUKURTI(failo vardas);
SUNAIKINTI(failo vardas);
ATIDARYTI(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 komandos naudoja bloko nr. Bet vartotojui nepatogu operuoti terminais. Tam įvedamas dar vienas failų sistemos lygmuo – priėjimo metodai.
Įrašų apdorojimui gali būti naudojami raktai. Dalinis raktas – duomenų laukas, kurio reikšmė duotu momentu gali atitikti vieną iš daugelio įrašą. Raktai gali būti naudojami įrašų identifikavimui.
Įrašai su vienodias galimais 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.