

## Contents

Operacinės sistemos sąvoka.....	3
Operacinių sistemų kategorijos .....	3
Multiprogramavimo sąvoka .....	4
Virtualios mašinos sąvoka.....	4
Lygiagretūs procesai. Notacija „and“ .....	7
Lygiagretūs procesai. Notacija „FORK, JOIN, QUIT“ .....	8
Kritinė sekcija .....	9
Kritinės sekcijos apsauga.....	10
Dekerio algoritmas .....	12
Semaforai.....	13
Procesų „gamintojas“ ir „naudotojas“ sąveika.....	14
Įvedimo-išvedimo spuleris. Bendra charakteristika .....	15
Įvedimo-išvedimo spuleris. Pagrindinis procesas.....	16
Įvedimo/išvedimo spuleris. Įvedimo ir skaitymo procesas. ....	17
Įvedimo-išvedimo spuleris. Išvedimo ir rašymo procesai .....	18
Dvejetainių ir bendrų semaforų sąryšis .....	19
Operacijų su semaforais realizacija .....	20
Operacijų su semaforais realizacija be užimančio laukimo .....	21
Procesų ir resursų sąvoka.....	22
Proceso deskriptorius (PD) .....	22
Resurso deskriptorius .....	23
Primityvas „kurti procesą“ (KURTIP) .....	23
Primityvas „naikinti procesą“ (NAIKINTIP).....	24
Primityvas „Stabdyti procesą“ (STABDYTIP).....	25
Primityvas „aktyvuoti procesą“ (AKTYVUOTIP).....	25
Primityvas „keisti proceso prioritetą“ (KEISTIPP).....	26
Primityvas „kurti resursą“ (KURTIR).....	26
Primityvas „naikinti resursą“ (NAIKINTIR) .....	27
Primityvas „prašyti resurso“ (PRAŠYTIR).....	28
Primityvas „atlaisvinti resursą“ (ATLAISVINTIR) .....	29
Procesų būsenų schema.....	30
Procesų planuotojas.....	31
Pasiruošusio proceso su aukščiausiu prioritetu nustatymas ir neaktyvaus procesoriaus radimas ..	32
Procesų planuotojo procesoriaus perskirstymo etapas .....	33

OS (kompiuterių) architektūros raida.....	34
Mikrobranduolio architektūra .....	36
Mikrobranduolio architektūros charakteristika.....	37
Gijos. Bendra charakteristika.....	38
Vartotojo ir branduolio gijos .....	38
Virtualios atminties sąvoka.....	39
Komandos vykdymo schema architektūroje su dinaminio adresų nustatymu .....	40
Statinis ir dinaminis atminties skirstymas .....	41
Puslapinė organizacija.....	41
Polisegmentinė virtuali atmintis.....	42
Atminties skirstymo puslapiais strategijos .....	43
Atminties išskyrimas ištisinėmis sritimis.....	43
Kintamo dydžio ištisinės atminties sričių grąžinimas .....	44
Failų sistemos sąvoka .....	45
Failinės atminties įrenginių charakteristikos.....	45
Įvedimo/išvedimo įrenginių greičių charakteristika .....	46
Įvedimo/išvedimo įrenginių procesų schema .....	48
Užklauso failinei sistemai realizavimo pavyzdys .....	49
Failų deskriptorius. Aktyvių failų katalogas.....	50
Failų sistemos hierarchinis modelis .....	50
Failų sistemos įvedimo-išvedimo posistemė .....	51
Bazinė failų valdymo sistema .....	51
Loginė failų valdymo sistema.....	52
Priėmimo metodai .....	52

# 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. MOS 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.

## Operacinių sistemų kategorijos

---

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

1. Vartotojo (užduoties autoriaus) sąveika su užduotimi jos vykdymo metu;
2. Sistemos reakcijos laikas į užklausą užduočiai vykdyti.

**Grynosios OS kategorijos:**

- Paketinio apdorojimo OS. Tai sistema, kurioje užduotys pateikiamos apdirbimui paketų pavidale įvedimo įrenginiuose. Vartotojas neturi ryšio su užduotimi jos vykdymo metu. Sistemos reakcijos laikas matuojamas valandomis. Tokios OS yra efektyviausios mašinos resursų naudojimo prasme, bet labai neefektyvios žmogaus resursų atžvilgiu.
- Laiko skirstymo OS. Užtikrina pastovų vartotojo ryšį su užduotimi. Ji leidžia vienu metu aptarnauti keletą vartotojų. Kiekvienam vartotojo procesui „kompiuteris“ suteikiamas nedideliu laiko kvantui, kuris matuojamas milisekundėmis. Jei procesas neužsibaigė tol, kol baigėsi jo kvantas, tai jis pertraukiamas ir pastatomas į laukiančiųjų eilę, užleidžiant „kompiuterį“ kitam procesui.
- Realaus laiko OS. Jos paskirtis – valdyti greitaesgius 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 – galimybe vienu metu vykdyti kelias užduotis.

# 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 lygiagrečių vykdymą, t. y. leidžia operatyvioje atmintyje būti kelioms vartotojo programoms, skirstydama procesoriaus laiką, atminties vietą ir kitus resursus aktyvioms vartotojo užduotims. MOS 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 būtų galima realizuoti MOS, kompiuterio architektūrai keliami tam tikri reikalavimai:

- *Pertraukimų mechanizmas* (jei jo nebūtų, liktų interpretavimo mechanizmas).
- *Privilegijuotas režimas*, t. y. esant privilegijuotam režimui uždrausti neprivilegijuotų komandų vykdymą. Priešingu atveju būtų labai ilgas darbas. MOS turi pasižymėti savybe, kad vienu metu dirbantys procesai neturi įtakoti vieni kitų (ar tai sisteminiai, ar vartotojo).
- *Atminties apsauga*. Jei vykdam komandą suformuojamas adresas, išeinantis iš komandoms skirtos adresų erdvės – suformuojamas pertraukimas.
- *Relokacija* (papildoma savybė). Tai programos patalpinimas į bet kokią atminties vietą, t.y. programos vykdymas gali būti pratęstas ją patalpinus į kitą atminties vietą. Tai efektyvumo klausimas.

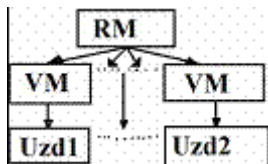
MOS yra populiariausias šio laikmečio operacinių sistemų tipas. MOS – kai vienam vartotojui suteikiama galimybė vienu metu daryti kelis darbus.

# Virtualios mašinos sąvoka

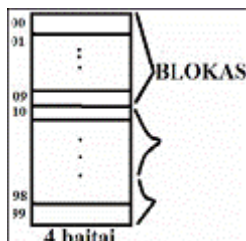
---

Realioji mašina - tai kompiuteris. Užduotis susideda iš programos, startinių duomenų ir vykdymo parametrų. Rašyti programą realiojai mašinai būtų sudėtinga ir nepatogu. Todėl vienas iš operacinės sistemos tikslų yra paslėpti realiąją mašiną ir pateikti mums virtualią. Užduoties programą vykdo ne realioji, o virtualioji mašina. Virtualioji mašina – tai tarsi virtualioji realios mašinos kopija. Virtualioji reikalauja netikra. Mes tarsi surenkame reikalingas realios mašinos komponentes, tokias kaip procesorius, atmintis, įvedimo/išvedimo įrenginiai, suteikiame jiems paprastesnę nei realiojo vartotojo sąsają ir visa tai pavadiname virtualioja mašina. Vienas iš virtualios mašinos (VM) privalumų yra programų rašymo palengvinimas, todėl realios mašinos komponentės, turinčios sudėtingą arba nepatogią vartotojo sąsają, virtualiojoje mašinoje yra supaprastintos. Virtualioji mašina dirba su operacinės sistemos pateiktais virtualiais resursais, kurie daugelį savybių perima iš savo realiųjų 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 multiprograminės operacinės sistemos atveju.



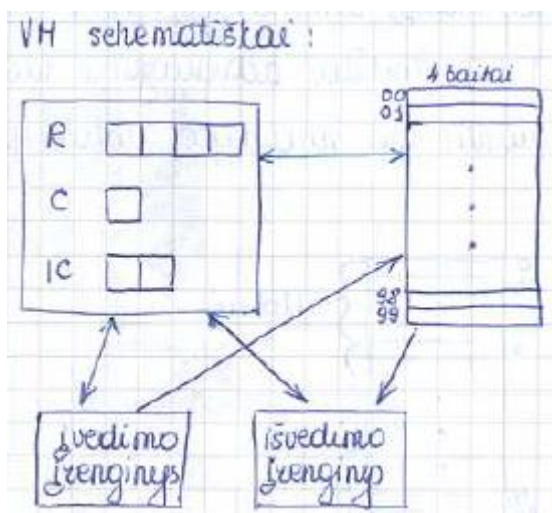
*VM specifikacija.* Tarkime, yra 100 žodžių atmintis (0-99). Kiekvienas žodis yra 4 baitų □□□□. Žodžiai adresuojami nuo 0 iki 99. Tegul atmintis yra suskirstyta blokais po 10 žodžių.



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 vyresnius baitus, o adresas – 2 jaunesnius baitus.

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



Įvesties/išvesties įrenginiai valdomi procesoriaus.

Virtualios mašinos procesoriaus komandos su paaiškinimais:

Komanda	Paaiškinimas
AD	sudėties komanda – $x_1x_2$ , $R := R + [a]$ , $a$ - adresas, $a := 10 * x_1 + x_2$ , $x_1, x_2 \in \{0, \dots, 9\}$ ; $ADx_1x_2$ ;
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.) - $x_1, x_2 \Rightarrow \text{Read}([\beta + i], i = 0, \dots, 9)$ , $\beta = 10 * x_1$ ; $GDx_1x_2$ ;
PD	išvedami duomenys - $x_1, x_2 \Rightarrow \text{Print}([\beta + i], i = 0, \dots, 9)$ ; $PDx_1x_2$ ;
H	sustojimo komanda $\Rightarrow$ HALT. VM pradeda darbą, kai registro IC reikšmė yra 00 (įvykdo komandą, kuri patalpinta nuliniame žodyje).

# Lygiagretūs procesai. Notacija „and“

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

Prasminis ryšys tarp procesų išreiškiamas per proceso resursus.

OS gali būti apibūdinta kaip procesų rinkinys, kur procesai:

- veikia beveik nepriklausomai (lygiagrečiai),
- bendrauja per pranešimus ir signalus (resursus),
- konkuruoja dėl resursų.

Skaičiavimo sistemose minimas aparatinis ir loginis lygiagretumas (paralelizmas).

Aparatinis lygiagretumas – reiškia lygiagretų, viena laiką 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.

*Aparatinis paralelizmas* įvedamas efektyvumo sumetimais, kad greičiau vyktų darbas. Procesoriuje nesant aparatiniam paralelizmui vis vien svarbu vienintelį proc. darbo laiką skirstyti keliems procesams. Todėl įvedama lygiagrečiai vykdomo proceso abstrakcija.

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

*Loginis paralelizmas* pasižymi tuo, kad kiekvienas procesas turi savo procesorių ir savo programą. Realiai, skirtingi procesai negali turėti tą patį procesorių ar tą pačią programą.

Procesas yra pora (procesorius, programa).

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

Valdymo ir informacinis ryšys tarp procesų realizuojamas per bendrus kintamuosius. Nagrinėjant tik pačius procesus, gaunami nuo procesoriaus nepriklausomi sprendimai.

( $s_1, s_2$  – sakiniai)

$s_1, s_2$  – 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.

## Lygiagretūs procesai. Notacija „FORK, JOIN, QUIT“

---

Jei procesas p įvykdo komandą FORK W (išsišakojimas), tai sukuriamą p proceso kopiją q, vykdomą nuo žymės W.

Jei procesas p įvykdo komandą QUIT, tai jis pasibaigia.

Procesų aplinkybių komanda JOIN T, W:

```
T := T - 1;  
IF T = 0 THEN GOTO W;
```

*(Svarbu! Šios komandos yra nedalomos/nepertraukiamos!)*

---

Pvz.

```
N := 2;  
  FORK P3;  
  M := 2;  
  FORK P2; /* iš viso dabar dirba 3 procesai (pirmas tas, kuris viską  
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;
```



## 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 bendru registru r2.  $t_0$  momentu  $x = v$

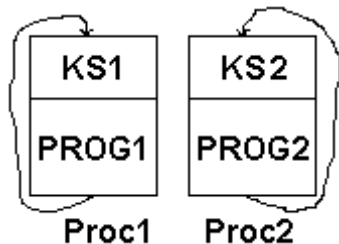
$t_0$ -----> $t_n$  laiko ašis

```
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 panaši 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:



N lygiagrečiai dirbantys procesai Proc1, Proc2, ... ProcN:

```
Begin  
  Proc1: begin L1: KS1; PROG1; GOTO L1; end;  
  and  
  Proc2: begin L2: KS2; PROG2; GOTO L2; end;  
  and  
  ...  
  and  
  ProcN: begin LN: KSN; PROGN; GOTO LN; end;  
End;
```

# Kritinės sekcijos apsauga

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

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

Tannenbaumo knygoje "Operating systems design and implementation" kritinės sekcijos apsaugai pateikti reikalavimai truputi kitokie: 1. Du procesai vienu metu negali būti kritinėje sekcijoje 2. Sprendžiant problemą negalima daryti prielaidų apie procesų vykdymo greičius 3. Procesai vykdydami komandas nekritinėje sekcijoje neturėtų blokuoti kitų procesų 4. Procesai tam, kad patektų į kritinę sekciją, neturėtų laukti be galo ilgai.

Tegu 2 procesai turi bendrą kintamąjį (EILĖ), kuris nurodo, kurio proc. eilė kreiptis į atm. P1: jei EILĖ =2 tada blokuokis, kitu atveju: KS1; EILĖ = 2; PROG1; į pradžią. P2 – atvirkščiai. Sprendimas negeras, nes blokuojamas kritinės sekcijos vykdymas, kai kitas procesas nėra kritinėje sekcijoje (Tannenbaumo knygos 3 reikalavimas). Jei vykdymo laikas  $P1 \gg P2$  tai P2 ilgai lauks, kol P1 įvykdys savo PROG1, kad galėtų vykdyti savo KS2:

```
BEGIN INTEGER 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;
```

Įveskime du bendrus kintamuosius:  $C_i = \text{FALSE}$ , kai vykdoma  $KS_i$ .  $P_1$ : jei vykdoma  $KS_2$ , tai blokuokis, kitu atveju:  $C_1 = \text{FALSE}$ ;  $KS_1$ ;  $C_1 = \text{TRUE}$ ;  $PROG_1$ , į pradžią.  $P_2$  – atvirkščiai. Bet šiuo atveju egzistuoja tikimybė, kad abu procesai pradės vykdyti  $KS$  vienu metu:

```
BEGIN BOOLEAN C1:= TRUE; C2:= TRUE;
  P1: BEGIN L1: IF NOT C2 THEN GOTO L1;
        C1:=FALSE; KS1; C1:=TRUE;
        PROG1;
        GOTO L1;
      END

  and

  P2: BEGIN L2: IF NOT C1 THEN GOTO L2;
        C2:=FALSE; KS2; C2:=TRUE;
        PROG2;
        GOTO L2;
      END

END;
```

Problemą išsprendžia Dekerio algoritmas.

Paprastiau problemą išspręsti naudojant Semaforus:

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

```
BEGIN SEMAPHORE 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;
```

# Dekerio algoritmas

Procesas atžymi savo norą įeiti į KS loginiu kintamuoju  $C_i = \text{false}$ . Išėjus iš kritinės sekcijos  $C_i = \text{true}$ . 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 (pvz.: noras vykdyti KS, įėjimas į 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.
```

Toks sprendimas per sudėtingas, kad juo remiantis būtų galima toliau organizuoti darbą. Netinka, nes nėra tinkamų primityvų.

Postuluoti primityvus yra maža. Svarbu – realizacija.

Reikia apibrėžti aparatą, tinkantį lygiagrečiam vykdymui – semaforus.

# 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 išskviečia  $V(S)$  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 tam pačiam semaforui, tai reikšmei tapus teigiamai (kai kažkuris kitas procesas įvykdė  $V(S)$  operaciją), kažkuris iš laukiančių procesų bus pradėtas vykdyti.

Pagal prasmę operacija  $P$  atitinka perėjimo iškvietimą, o  $V$  – kito proceso aktyvaciją.

Jei semaforas įgyja tik dvi reikšmes 0 ir 1, tai jis vadinamas dvejetainiu, jei bet kokias - bendriniu.

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

# 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.

Problema atsiranda, kai gamintojas nori patalpinti informaciją į buferį, bet jis jau yra pilnas. Sprendimas yra pasiųsti gamintoją miegoti ir pažadinti tada, kai naudotojas paims bent vieną informacijos vienetą iš buferio. Analogiška problema iškyla, kai vartotojas nori paimti informaciją iš buferio, kuris yra tuščias.

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.

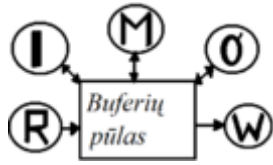
```
BEGIN SEMAPHORE T,U,B;
      T:=N; U:=0; B:=1; // B=1 - kritinė sekcija nevykdoma, B=0 - vykdoma
GAM:  BEGIN
      LG:  įrašo gaminimas;
          P(T);
          P(B);
          užrašymas į buferį;
          V(B);
          V(U);
          GOTO LG;
      END

      AND

NAUD:  BEGIN
      LN:  P(U);
          P(B);
          paėmimas iš buferio;
          V(B);
          V(T);
          įrašo apdorojimas;
          GOTO LN;
      END
END;
```

## Išvedimo-išvedimo spuleris. Bendra charakteristika

Spuleris (angl. *spooler*) – OS dalis atliekanti I/O virtualizaciją, kuomet yra sukaupiama įvedama informacija, ji apdorojama ir išvedama kaip rezultatas. Spuleris 5 procesų visuma. Visi jie yra procesoriuje vykdomi procesai.



*R–Read; I–Input; M–Main; O–Out; W–Write;*

Buferių pūlas – sąrašas (faktiškai 3 sąrašai – laisvų, įvedimo ir išvedimo buferių). Iš pradžių buferių pūlas apjungtas į laisvų buferių sąrašą, o įvedimo ir išvedimo sąrašai tušti. Apdorojimo proceso užduotis paruošti informaciją išvedimui.

Darbai 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 – laisvų, min – įvedimo, mout – išvedimo buferių KS apsaugos semaforas.

### Spūlerio veikimas:

```
BEGIN
  nl := n;
  nin := nout := 0;
  SR := FR := SW := FW := 0;
  M and I and O and R and W; // Main dirba lygiagrečiai su I,O,R,W
END;
```

## Ivedimo-išvedimo spuleris. Pagrindinis procesas

---

1. Ima buferį iš įvedimo buferių sąrašo;
2. Apdoroja jame esančią informaciją;
3. Ima buferį iš laisvų buferių sąrašo.
4. Jį užpildo išvedama informacija;
5. Buferį įjungia į išvedimo buferių sąrašą.

Pagrindinio proceso M programa:

```
M: BEGIN
LM:  P(nin); // M turi imti įvedimo buferį: Jei tokio nėra, tai M blokuojasi
      P(min); // jei kitas procesas vykdo įvedimą, tai M 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);
      Užpildyti paimtą buf. išvedama info;
      P(mout); // išvedimo buferio KS apsauga
      Įjungti buf. į išved. buf. sąrašą; // imama iš sąrašo pradžios, bet rašoma
      į galą.
      V(mout); !!!
      V(nout);
      P(ml); // buvo paimtas įved. buferis. Jį reikia atlaisvinti. Tai darbas su
      KS;
      Įjungti buferį į laisvų buf. sąr;
      V(ml);
      V(nl);
      GOTO LM;
      END;
```



## Įvedimo/išvedimo spuleris. Įvedimo ir skaitymo procesas.

Įvedimo procesas I gali vykti tik tada, kai yra tuščių buferių. Atitinkamai M gali vykti, kai yra įvedimo sąraše elementai ir laisvi buferiai. Jei I ima paskutinį laisvą buferį, tai M užsiblokuoja, nes jam nėra laisvų buferių.

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 skaitymo įrenginį tuščiais buferiais, inicijuoti R ir prijungti į pūlą užpildytus per įvedimą buferius, kai R baigia darbą.

### Įvedimo proceso valdymas

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

### Skaitymo proceso valdymas

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

Proceso I veiksmų seka yra tokia:

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

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

## Įvedimo-išvedimo spuleris. Išvedimo ir rašymo procesai

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; FW – išvedimo įreng. finišas.

### Išvedimo proceso valdymas

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

### Rašymo proceso valdymas

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

## Dvejetainių ir bendrų semaforų sąryšis

Semaforinių primitivų realizacijai, reikia bendruosius semaforus išreikšti dvejetainiais. Jei semaforas gali įgyti tik dvi reikšmes – jis dvejetainis. Bet kuris bendras semaforas gali būti išreikštas dvejetainiu semaforu.

Tegu  $S$  – bendrasis semaforas. Tada jis gali būti pakeistas kintamuoju  $NS$  ir dviem dvejetainiais semaforais  $M$  ir  $D$  ( $M$  – kritinę sekciją apsaugantis semaforas).

$P(S)$  – nedaloma operacija,  $M$  – kritinę sekciją apsaugantis dvejetainis semaforas.

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

```
P(S) ~ BEGIN
    P(M); // KS apsauga nuo dvejetainio semaforo M
    NS := NS-1;
    IF NS <= -1 THEN // turi įvykti blokavimasis
    BEGIN
        V(M); // nuimama apsauga, nes kitu atveju, kitas procesas
        kreipdamasis negalėtų atlikti P(S) veiksmo.
        P(D); // turi iššaukti blokavimą;
    END
    ELSE V(M);
END.
```

$P(S)$  atvejai:

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

$NS < 0$  – parodo laukiančių procesų skaičių.

```
V(S) ~ BEGIN
    P(M);
    NS := NS+1;
    IF NS <= 0 THEN V(D); // yra laukiančių procesų, kurie užsikodavę
    semaforu D
    V(M);
END.
```

$RS$ ; PRAŠYTIR( $RS, D, A$ ) ATLAISVINTI( $RS, D$ );

Semaforo mechanizmas reiškiamas taip:

```
P(S) ~ PRAŠYTI(S,  $\Omega$ ,  $\Omega$ );
V(S) ~ ATLAISVINTIR(S,  $\Omega$ );
```

# Operacijų su semaforais realizacija

Paprastai komandų sistema neturi operacijų P ir V. Norint realizuoti semaforinius primityvus, reikia kad kompiuterio architektūra leistų nedalomu veiksmu *patikrinti ir pakeisti* žodžio reikšmę  $x$  - PP(x).

```
BOOLEAN PROCEDURE PP (BOOLEAN var x);  
BEGIN  
    PP:=x;  
    x:= TRUE;  
END;
```

Dabar pasinaudosime bendrų ir dvejetainių semaforų ryšiais. Turėdami bendrą semaforą S, P(S) ir V(S) norėtumėme išreikšti komandomis: priskirti ir patikrinti.

- NS - semaforo skaitinė reikšmė
- MS - KS apsauga
- DS - semaforo blokavimas

```
P(S) ~ BEGIN  
    L1: IF PP(MS) THEN GOTO L1; // vyksta laukimas  
        NS:= NS - 1;  
        IF NS <= -1 THEN  
            BEGIN  
                MS:= FALSE  
                L2: IF PP(DS) THEN GOTO L2;  
            END  
        ELSE MS:= FALSE;  
    END;  
V(S) ~ BEGIN  
    L3: IF PP(MS) THEN GOTO L3; // vyksta laukimas  
        NS:=NS+1;  
        IF NS<=0 THEN DS:= FALSE;  
        MS:= FALSE;  
    END;
```

# Operacijų su semaforais realizacija be užimančio laukimo

Kad būtų padidintas sistemos efektyvumas reikia, kad P procesą blokuotų, o V jį aktyvuotų.

Proceso blokavimo metu nuo jo vykdymo atlaisvinamas procesorius ir proceso P būseną įjungiamą į blokuotų pagal semaforą S procesų sąrašą.

Su V(S) atvirkščiai.

## P(S) ir V(S) programinės realizacijos

Operacija P(S), kai S – bendrasis semaforas, ekvivalenti tokiai programinei konstrukcijai:

```
P(S) ~ BEGIN
    Uždrausti pertraukimus;
    L: IF PP(x) THEN GOTO L;
    S:=S-1;
    IF S<=-1 THEN
        BEGIN
            Užblokuoti iškviečiantį procesą; // (2), (1)
            Paimti procesą iš sąrašo LIST_A; // (2)
            x:= FALSE; // (2)
            Perduoti valdymą; // (2)
        END
    ELSE x:=FALSE;
    Leisti pertraukimus;
END;

V(S) ~ BEGIN
    Uždrausti pertraukimus;
    L: IF PP(x) THEN GOTO L;
    S:=S+1;
    IF S<=0 THEN // (3)
        BEGIN
            Paimti procesą iš LIST_BS -> LIST_A;
            IF procesorius laisvas THEN vykdyti procesą iš LIST_A;
        END;
    x:= FALSE;
    Leisti pertraukimus;
END.
```

(1) Tai reiškia įjungimą į blokuotų procesų sąrašą LIST<sub>BS</sub>, kur s reiškia nuo semaforo S.

(2) Faktiškai tai yra planuotojo darbas.

(3) Suaktyviname vieną iš blokuotų semaforo S procesų.

# Procesų ir resursų sąvoka

---

Procesas – tai vykdoma programa, kartu su esamomis registrų reikšmėmis ir savo kintamaisiais. Kiekvienas procesas turi savo virtualų procesorių. Nors skirtumas tarp programos ir proceso nėra didelis, bet jis svarbus. Procesas – tai kokioje nors veiklumo stadijoje esanti programa. Tuo tarpu programa – tai tik tam tikras baitų rinkinys. Veiklumo stadiją apibūdina proceso aprašas – deskriptorius. Apraše yra laikomi visi procesui reikalingi parametrai, tokie kaip virtualaus procesoriaus registrų reikšmės, ar jam reikalingi kintamieji. Paprastai procesus galima suskirstyti į vartotojiškus ir sisteminius. Sisteminių procesų paskirti – aptarnauti vartotojiškus. Tuo tarpu vartotojiško proceso paskirtis yra vykdyti vartotojo programą. Procesai gali kurti kitus procesus. Formuojamas procesų medis.

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

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

## 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 deskriptorius. PD – tai tam tikra struktūra (ne masyvas). Jei kalbame apie visų procesų deskriptorius, tai turime struktūrų masyvą, kur  $i$  proceso vidinis vardas – nurodytų struktūros numerį masyve. PD – susideda iš komponentų, kurioms priskiriame vardus:

1.  $Id[i]$  – proceso išorinis vardas, reikalingas statiniams ryšiams tarp procesų nurodyti.
2. Mašina (procesą vykdančio procesoriaus apibūdinimas):
  1.  $CPU[i]$  – apibūdina centrinio procesoriaus būseną ir teises vykdam procesą. Kai proceso vykdymas nutraukiamas, proceso būsena išsaugoma;
  2.  $P[i]$  – identifikuoja procesorių;
  3.  $OA[i]$  – i-tojo proceso turima operatyvioji atmintis (nuoroda į sąrašą);
  4.  $R[i]$  – i-tojo proceso turimi resursai – informacija, kokius resursus yra gavęs procesas. Nuoroda į sąrašą, kuriame išvardinti resursai;
  5.  $SR[i]$  – proceso sukurtų resursų sąrašas (resursų deskriptorių);
3. Informacija apie proceso būseną:
  1.  $ST[i]$  – proceso statusas (RUN, READY, READYS, BLOCK, BLOCKS);
  2.  $SD[i]$  – nuoroda į sąrašą kuriame yra procesas (PPS – pasiruošusių procesų sąrašas, LPS – laukiančių procesų sąrašas);
4. Proceso sąryšis su kitais procesais:
  1.  $T[i]$  – i-tojo proceso tėvas (tėvinio proceso vidinis vardas);
  2.  $S[i]$  – nuoroda į i-tojo proceso vaikinių procesų vidinių vardų sąrašą;
5.  $PR[i]$  – prioritetas.

# 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.

r – resurso vidinis vardas (indeksas resursų deskriptorių masyve)

1. identifikacija.
  1. Rid[r] – išorinis vardas
  2. PNR[r] – ar tai pakartotinio panaudojimo resursas (jei taip - jį reikia grąžinti)
  3. K[r] – resursą sukūrusio 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.

## Primityvas „kurti procesą“ (KURTIP)

Sukuriamas naujas deskriptorius masyve. Procedūra įvykdoma iššaukiančio proceso aplinkoje.

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

- n – išorinis vardas;
- S0 – procesoriaus pradinė būsena;
- M0 – OA pradinė būsena (kiek išskirta OA resursų);
- R0 – kiti išskiriami resursai (nuoroda į sąrašą);
- K0 – proceso prioritetas;
- \* – duotu metu dirbantis 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; // Kitų proceso turimų resursų komponentė
  PR[i]:=K0;
  ST[i]:=READY; // laikysime, kad procesas sukuriamas su statusu (pvz.
  pasiruošęs)

  SD[i]:=PPS;
  T[i]:=*;
  S[i]:=Λ; // Sūnų sąrašas iš pradžių tuščias
  Įrašyti(S[*],i); // * įgavo naują sūnų i
  Įjungti(PPS,i);
END;
```

# Primityvas „naikinti procesą“ (NAIKINTIP)

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
sar.
  FOR ALL s e S[i] DO NUTRAUKTI(s); // reikia nutraukti i-tojo proceso sūnų
vykdymą
  FOR ALL r e OA[i] v R[i] DO // r - resursai. Atlaisviname pakartotinio
panaudojimo resursus
    IF PNR THEN Įrašyti(PA[r], Pašalinti(OA[i] v R[i])); //Jeigu buvo
pakartotinai panaudojamas resursas, jį įjungia į Prieinamumo sąrašą
  FOR ALL r e SR[i] DO NRD(r); //naikinami resursų deskriptoriai
  NPD(i); //naikiname proceso deskriptorių
END;
```



## Primityvas „Stabdyti procesą“ (STABDYTIP)

Tėvinis procesas gali stabdyti vaikinį nestabdant vaikinio palikuonių.

- $n$  – išorinis vardas
- $a$  – adresas į proceso deskriptoriaus būseną (jo kopiją)

```
Procedure STABDYTIP ( $n, a$ );  
begin  
   $i := VV(n)$ ; // pagal IV nustatomas proceso VV  
   $S := ST[i]$ ; // einamasis proceso statusas  
  if  $S = RUN$  then  $STOP(i)$ ; // (1)  
   $a := copydesc[i]$ ;  
  if  $S = BLOCK$  or  $BLOCKS$  then  $ST[i] := BLOCKS$   
    else  $ST[i] := READYS$ ;  
  if  $S = RUN$  then  $PLANUOTOJAS$ ; // (2)  
End;
```

(1)  $STOP(i)$  įvykdo veiksmus:

1. Pertraukia procesorių  $P[i]$
2. Įsimena pertraukto procesoriaus  $P[i]$  būseną proceso deskriptoriaus komponentėje  $CPU[i]$
3.  $PROC[P[i]] := \Lambda$  - pažymima, kad procesorius  $P[i]$  yra laisvas

(2) Jei buvo pristabdytas vykdomas procesas, tai reikia iškviešti planuotoją, nes galbūt kiti procesai laukia procesoriaus. Planuotojo vykdymui naujas procesas nesukuriamas.

Svarbu, kad iki planavimo veiksmo viskas jau būtų padaryta, nes planuotojas atims procesorių iš šito paties proceso, kuriame planuotojas buvo iškvieštas.

## Primityvas „aktyvuoti procesą“ (AKTYVUOTIP)

Nuima pristabdymo būseną. Galbūt iškviečia planuotoją, jei būseną yra READY.

```
PROCEDURE AKTYVUOTI ( $n$ );  
BEGIN  
   $i := VV(n)$ ;  
  IF  $ST[i] = READYS$  THEN  $ST[i] := READY$   
    ELSE  $ST[i] := BLOCK$ ; // buvo BLOCKS  
  IF  $ST[i] = READY$  THEN  $PLANUOTOJAS$ ;  
END;
```

## Primityvas „keisti proceso prioritetą“ (KEISTIPP)

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.

- $n$  – proceso IV (Išorinis Vardas);
- $k$  – naujas prioritetas.

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

## Primityvas „kurti resursą“ (KURTIR)

Resurso deskriptoriaus sukūrimas. Resursus kuria tik procesas.

**Parametrai:**

- $RS_0$  – resurso IV (Išorinis Vardas);
- $PNR_0$  – ar res. yra iš naujo panaudojamas;
- $PA_0$  – res.prieinamumo aprašymas;
- $LPS_0$  – to resurso laukiančių proc. sąr.;
- $PASK_0$  – res. paskirstytojo programos adresas;
- \* – resurso kūrėjo VV. Tai tuo metu vykstantis procesas, kuriame ir panaudotas šis primityvas.

```
Procedure KURTIR (RSo, PNRo, PAo, LPSo, PASKo);  
Begin  
    r:=NRVV; // naujas resurso vidinis vardas  
    Rid[r]:=RSo;  
    PNR[r]:=PNRo;  
    k[r]:=*;  
    PA[r]:=PAo;  
    LPS[r]:=LPSo;  
    PASK[r]:=PASKo;  
    Įrašyti (SR[*], r); // (1)  
End;
```

(1) Į šiuo metu vykdomo proceso sukurtų resursų sąrašą įtraukiamas naujai sukurtas resursas.

# Primityvas „naikinti resursą“ (NAIKINTIR)

Sunaikinti resursą gali jo tėvas arba pirmtakas. Sunaikinamas resurso deskriptorius, prieš tai suaktyvinami jo laukiantys procesai.

- RS – resurso IV (išorinis vardas);
- R.P – proceso vidinis vardas.

```
PROCEDURE NAIKINTIR(RS);
BEGIN
  r:=RVV(RS);
  //atblokuojami resurso laukiantys proc.jiems pasiunčiant fiktyvų res.
  R:=Pašalinti(LPS[r]); //turėsime nuorodą į pašalinamą elementą ir nuorodą į
  procesą
  WHILE R<>1 DO
    BEGIN
      IF ST[R.P]=BLOCK THEN ST[R.P]:=READY //Procesas dirbtinai atblokuojamas
      ELSE ST[R.P]:=READY;
      Įrašyti(PPS,R.P);
      SD[R.P]:=PPS;
      R.A:='PRANEŠIMAS';
      R:=Pašalinti(LPS[r]);
    END;
  NRD(r); //Naikinti resurso deskriptorių
  PLANUOTOJAS;
END;
```

# Primityvas „prašyti resurso“ (PRAŠYTIR)

Procesas, kuriam reikia resurso, iškviečia šį primityvą, nurodydamas išorinį vardą ir adresą. Toks procesas pereina į blokavimosi būseną. Blokavimasis įvyksta tik prašant resurso. Procesas įjungiamas į laukiančių to resurso procesų sąrašą.

- RS – resurso IV (išorinis vardas);
- D – kokios resurso dalies prašoma;
- A – atsakymo srities adresas, į kur pranešti;
- \* – šiuo metu veikiantis procesas.

```
PROCEDURE PRASYTIR(RS, D, A);
BEGIN
  r := RVV(RS);
  Įrašyti(LPS[r], ( *, D, A)); // procesas įjungiamas į laukiančių šio resurso
  procesų sąrašą
  PASK(r, K, L); // Resurso paskirstytojo programa
                  // K – kiek procesų aptarnauti
                  // L – aptarnautų procesų vidinių vardų masyvas
  B := TRUE; // Ar šiuo metu veikiančiam procesui nebuvo suteiktas resursas?
  FOR J := 1 STEP 1 UNTIL K DO
    IF L[J] <> * THEN
      BEGIN
        i := L[J];
        Įrašyti(PPS, i);
        SD[i] := PPS;
        IF ST[i] = BLOCK THEN ST[i] := READY
          ELSE ST[i] := READYS;
      END ELSE
        B := FALSE; // Procesas * iš karto gavo resursą ir nebuvo įrašytas į
        laukiančių procesų sąr.

    IF B THEN
      BEGIN
        ST[*] := BLOCK;
        SD[*] := LPS[r];
        PROC[P[*]] := Λ; // Iš šiuo metu veikiančio proceso atimamas procesorius
        Pašalinti(PPS, *);
      END;
  PLANUOTOJAS;
END;
```

## Primityvas „atlaisvinti resursą“ (ATLAISVINTIR)

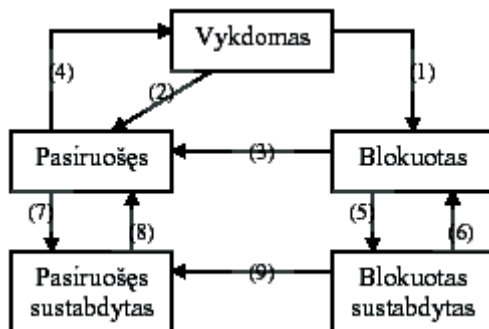
Tai atitinka situaciją, kai procesas gauna pakartotino naudojimo resursą ir kai jam nereikia, jis jį atlaisvina ir įjungia į laisvųjų resursų sąrašą bei papildo resurso prieinamumo aprašymo sąr.

- RS – resurso IV (išorinis vardas);
- D – atlaisvinamos resurso dalies apibūdinimas.

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

# Procesų būsenų schema

Procesas gali gauti procesorių tik tada, kai jam netrūksta jokie resurso. Procesas gavęs procesorių tampa vykdomu. Procesas, esantis šioje būsenoje, turi procesorių, kol sistemoje neįvyksta pertraukimas arba einamasis procesas nepaprašo kokio nors resurso (pavyzdžiui, prašydamas įvedimo iš klaviatūros). Procesas blokuojasi priverstinai (nes jis vis tiek negali tęsti savo darbo be reikiamo resurso). Tačiau, jei procesas nereikalauja jokio resurso, iš jo vis tiek gali būti atimamas procesorius, pavyzdžiui, vien tik dėl to, kad pernelyg ilgai dirbo. Tai visiškai skirtinga būsenai nei blokavimasis dėl resurso (neįskaitant procesoriaus resurso). Taigi, galime išskirti jau žinomas procesų būsenas: vykdomas – jau turi procesorių; blokuotas – prašo resurso (bet ne procesoriaus); pasiruošęs – vienintelis trūkstamas resursas yra procesorius; sustabdytas – kito proceso sustabdytas procesas.

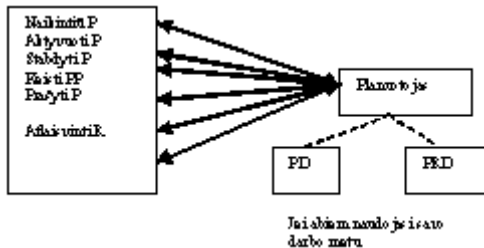


- (1) Vykdomas procesas blokuojasi jam prašant ir negavus resurso.
- (2) Vykdomas procesas tampa pasiruošusiu atėmus iš jo procesorių dėl kokios nors priežasties (išskyrus resurso negavimą).
- (3) Blokuotas procesas tampa pasiruošusiu, kai yra suteikiamas reikalingas resursas.
- (4) Pasiruošę procesai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

# Procesų planuotojas

Planuotojas užtikrina, kad visi procesai būtų vykdomi maksimaliu prioritetu (jei prioritetai vienodi, tai vykdomas tas kuris sąraše pirmesnis).

Planuotojas iškviečiamas iš branduolio primitivų. Planuotojo veikimo schema:



Planuotojo darbas 3 etapais:

1. Surasti procesą su aukščiausiu prioritetu;
2. Surasti neaktyvų procesorių ir jį skirti pasiruošusiam procesui;
3. Jei procesorių nėra, peržiūrėti vykdomus procesus, ir, jei jų prioritetai mažesni, atiduoti procesorių pasiruošusiems (perskirstymas).

# Pasiruošusio proceso su aukščiausiu prioritetu nustatymas ir neaktyvaus procesoriaus radimas

Pasiruošęs procesas su aukščiausiu prioritetu nustatomas prioritetų mažėjimo tvarka peržiūrint visus procesus, ieškant turinčio pasiruošimo (READY) būseną. To paties prioriteto procesai yra sujungti cikliniame sąraše, tad turint  $n$  prioritetų, egzistuoja  $n$  ciklinių sąrašų. Visi sąrašų elementai saugomi vienoje struktūroje  $P$ . Pirmasis kurio nors prioriteto sąrašo procesas yra patalpintas prioriteto reikšmę atitinkančioje pozicijoje (pavyzdžiui, pirmasis procesų sąrašo, turinčių prioritetą 7, procesas bus patalpintas 7 pozicijoje). Tam tikro prioriteto procesų sąraše neradus pasiruošusio proceso, prioritetas mažinamas, taip pradedant nagrinėti kitą sąrašą.

Neaktyvus procesorius ieškomas paeiliui peržiūrint visus procesorius ir ieškant nesusieto su jokia procesu. Aptikus laisvą procesorių, jam priskiriamas pasiruošęs procesas su didžiausiu prioritetu. Gali iškilti situacija, kai aptiktasis laisvas procesorius yra procesorius, vykdamas planuotoją iškvietusį procesą (pvz.: jei planuotojas kviečiamas procesui užsiblokovus dėl negauto resurso). Tokiu atveju specialiame kintamajame įsimenamas proceso, kuriam reiks atiduoti procesorių baigus planuotojo darbą, vidinis vardas.

```
PROCEDURE PLANUOTOJAS;
BEGIN
  p := PRT := n;
  c := 1;
  L: B := TRUE;
  //--Proceso su aukščiausiu prioritetu nustatymas
  WHILE B AND PRT <> 0 DO
    BEGIN
      p := P[p]; //Priskiriame kitą procesą iš ciklinio sąrašo su prioritetu
      PTR.
      IF p = PTR THEN //p = PTR reiškia, jog pasiekta ciklinio sąrašo pradžia ir
        peržiūrėti visi elementai. Reikia mažinti prioritetą.
        BEGIN PRT := PRT - 1;
              p := PRT;
        END
      ELSE B := ST[p] <> READY;
    END
  END
  //--
  IF PTR = 0 THEN GOTO EINPROC;
  //--Neaktyvaus procesoriaus radimas
  WHILE c <= np DO
    IF PROC[c] <> Λ
      THEN c := c + 1
      ELSE BEGIN
              PROC[c] := p;
              P[p] := c;
              ST[p] := RUN;
              IF c <> P[*] THEN Atstatyti_būseną(c, CPU[p])
                ELSE p* := p; //Įsimenamas proceso, kuriam reiks
                atiduoti procesorių, vidinis vardas.
              c := c + 1;
              GOTO L;
            END;
  END;
  //--
```



# Procesų 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 įsime name to proceso vidinį vardą. Iš to proceso atimame procesorių ir jį atiduodame mūsų norimam procesui.

```
PRTMIN := PRT;
FOR c := 1 STEP 1 UNTIL np DO
  BEGIN
    q := PROC(c);
    IF PR[q] < PRTMIN THEN
      BEGIN
        PRTMIN := PR[q];
        cp := c;
      END;
    END;
  IF PRT <> PRTMIN THEN
    BEGIN
      q := PROC(cp);
      ST[q] := READY;
      P[p] := cp;
      PROC[cp] := p;
      ST[p] := RUN;
      IF q = * THEN p* = p
        ELSE Perjungti(p, q, cp);
      GOTO L;
    END;
  EINPROC: IF ST[*] <> RUN THEN Perjungti(p*, *, P[*]);
END;
```

- p – proceso, kuriam bus atiduotas procesorius, VV (vidinis vardas);
- q – proceso, iš kurio bus atimtas procesorius, VV (vidinis vardas);
- c – procesorius, kuriame vyksta perjungimas.

```
PROCEDURE PERJUNGTI(p, q, c);
BEGIN
  Pertraukti(c); //jei c <> P[*];
  Įsiminti_būsena(c, CPU[q]);
  IF q = * THEN ĮsimintiGA(GA, CPU[q]); //įsime namas grįžimo adresas
  Atstatyti_būsena(c, CPU[p]);
END;
```

# OS (kompiuterių) architektūros raida

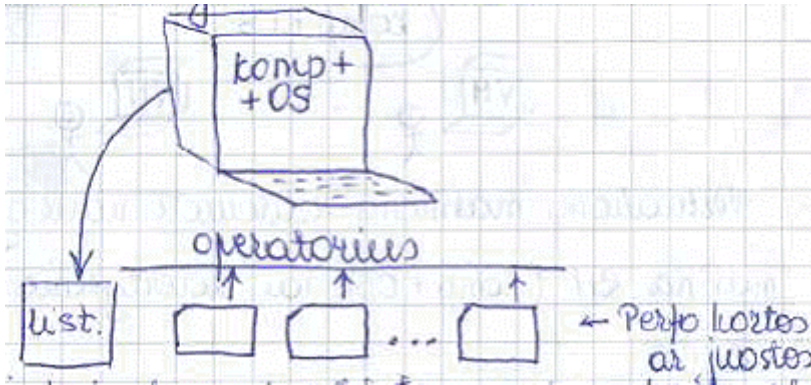
## I

Žmogus betarpiškai valdo kompiuterį. Darbas vyksta mašinine kalba. Kompiuterio panaudojimas neefektyvus (labai brangi technika). Kompiuterių resursų nėra.



## II

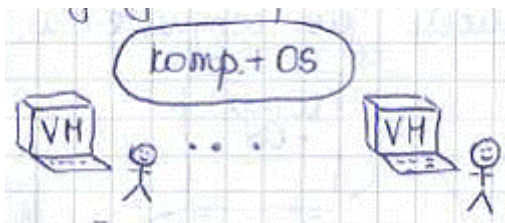
Sprendžiama efektyvumo problema: žmogus nutolinamas nuo kompiuterio, paliekant jam kompiuterio teikiamas galimybes. Programuotojai suformuluoja savo užduotis kaip programų paketus, kurie perduodami operatoriui, o šis perduoda juos kompiuteriui.



Kompiuteris buvo papildytas OS tam, kad galima būtų bendrauti su vartotoju. Tokia OS priima visą paketą užduočių ir grąžina rezultatus (listingus) programuotojui. Reikėjo pateikti visus būtinus duomenis. Ši paketinio apdorojimo OS buvo neefektyvi žmogaus darbo atžvilgiu (pvz.: kai programos buvo užrašomos ant popieriaus, perduodamos į skaičiavimo centrą, kur programuotojai suvesdavo jas į kompiuterį, ten kompiliuojamos ir listingai buvo grąžinami rašiusiam programą asmeniui).

## III

Virtualių mašinų etapas. Tikslas: sugrąžinti žmogui kompiuterį (I etapas) paliekant efektyvų technikos panaudojimą (II etapas). Čia su kompiuteriu žmogus bendrauja jau suprantama kalba.



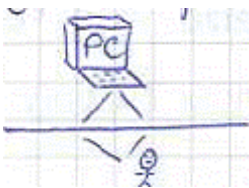
Virtualioms mašinoms realizuoti naudojama reali mašina RM (kompiuteris + OS). Tai kolektyvinio naudojimo OS. OS iš II-o ir III-io etapų yra multiprograminės OS. Skirtumas: kai II-me etape persijungiama nuo vienos užduoties vykdymo prie kitos, tai pertraukimas įvyksta dėl vidinių kompiuterio priežasčių, o III-me etape pertraukimas įvyksta dėl tos pačios priežasties bei dėl laiko kvanto pasibaigimo. Jei užduotis sudėtinga (pvz. transliavimas), tai žmogui suprantami yra pertraukimai. Tampa patogiu dirbti su dideliais resursais. Šis etapas labai brangus (pastarosios temperatūros palaikymas, energijos sąnaudos ir t. t.).

#### IV

Situacija pasikeičia atsiradus PK (personaliniams kompiuteriams). Grįžtama prie I-mo etapo: jokių multiprograminių OS.



Rimti uždaviniai atliekami kaip ankščiau (III etapas), o smulkūs – kaip I-me etape. Žmogui norėjosi vienu metu daryti daug darbų. Iš kilo poreikis turėti keletą virtualių mašinų vienam žmogui. Tam tikslui grįžtama prie multiprograminių OS. Tai šiandieninė būsena.



# 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 viename sluoksnyje iššaukia pakeitimus ir gretimuose sluoksniuose. Sunku kurti versijas pagal konkrečią konfigūraciją. Sunku spręsti saugumo problemas dėl gretimų sluoksnių sąveikos.

Mikrobranduolio sistemos atveju visi servais 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 priėmimas prie aparatūros.

Mikrobranduolio architektūros pranašumai:

1. Vieningas interfeisas. Užtikrinamas bendravimu pranešimais;
2. Išplečiamumas. Užtenka įvesti naują servisą ar pakeisti jau esamą, bet ne branduolį;
3. Lankstumas. Galima ne tik praplėsti, bet ir išjungti jau esamus servais. Mikrobranduolio sistema nėra maža sistema, bet konfigūruojama sistema;
4. Pernešamumas(portability). OS pernešimui pakanka pertvarkyti OS branduolį;
5. Patikimumas. Kuo didesnė programinio produkto apimtis, tuo sunkiau apsaugoti jį nuo klaidų;
6. Tinkamumas realizuoti paskirtas (išskirstytas) sistemas.

Neigiama savybė – nepakankamas našumas. Kalta pranešimų sistema - pranešimą reikia pernešti, perduoti, gavus atkoduoti. Atsiranda daug perjungimų tarp vartotojo ir supervizoriaus režimų. Sprendimo būdai didinantys našumą:

1. Padidinti branduolio funkcionalumą;
2. Dar labiau sumažinti mikrobranduolio funkcijas.

# Mikrobranduolio architektūros charakteristika

---

Mikrobranduolis turi turėti funkcijas, susijusias su tiesiogine architektūra:

1. Žemo lygio OA valdymas;
2. Procesų komunikavimas;
3. I/O ir pertraukimai.

## **OA valdymo funkcijos:**

- GRANT – vieno proceso OA atiduodama kitam;
- MAP – procesas vieną savo puslapį atvaizduoja kito proceso OA ir taip gaunamas bendras atminties bloko ruožas;
- FLUSH – procesas gali atšaukti bet kurį iš savo atiduotų ar sutapatintų blokų.

**Procesų komunikavimo funkcija.** Pagrindinė priemonė - pranešimas. Susideda iš:

- antraštės (siuntėjas ir gavėjas);
- kūno (informacija, nuoroda į blokus).

**I/O ir pertraukimai.** Mikrobranduolio architektūra įgalina pertraukimus interpretuoti kaip pranešimus. Įvedimo/išvedimo įrenginius galima interpretuoti kaip gijas.

# Gijos. Bendra charakteristika.

---

Gija (angl. *thread*) – nepriklausoma valdymo šaka procese. Gijų mechanizmas aiškina proceso vykdymo valdymą viename ir daugelyje procesų. Gijų panaudojimo privalumai:

1. Sukurti bei sunaikinti giją (proceso viduje) yra greičiau nei procesą.
2. Persijungimas tarp gijų viename procese vyksta greičiau nei tarp skirtingų procesų.
3. Komunikavimas tarp sąveikaujančių procesų, vykdomų vienos proceso gijos vyksta greičiau nei tarp programų, vykdomų nepriklausomų procesų (tokiu atveju reikalingas branduolio įsikišimas).

Jei taikomoji programa susideda iš kelių dalių, tai efektyviau ją realizuoti gijomis nei atskirais procesais.

Gijos turi būsenas, jų vykdymas turi būti sinchronizuotas. Procesui esant pristabdomo būsenoje, gijos irgi yra pristabdytos, nors jos ir neturi tokio atributo. Su gijomis atliekamos operacijos:

- CREATE – sukūrus procesą, gija taip pat sukuriamą. Sukurta gija turi atskirą registrų ir steko kontekstą.
- BLOCK – gija laukia signalo.
- UNBLOCK – gija atblokuojama, t.y. patalpinama į pasiruošusių gijų sąrašą.
- FINISH – atlaisvinamas gijos turimas registrų ir steko kontekstas.

## Vartotojo ir branduolio gijos

---

### 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 daugiaprocesorinės sistemos.

### 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.

# Virtualios atminties sąvoka

---

OA (operatyvioji atmintis) – tai atmintis, į kurią procesorius gali kreiptis tiesiogiai imant komandas ar duomenis 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 programos reikia susieti ir patalpinti į atmintį. Tai atliekanti programa – surišantis pakrovėjas.

Sudarant programas tenka abstrahuotis nuo atminties žodžių ir naudoti tam tikrus lokalius adresus, kurie vėliau koku 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.

**Statinis atminties skirstymas** – visi darbai atliekami prieš programos vykdymą. Statiškai nustatant adresus būtina prieš programos vykdymą žinoti išskirtos atminties vietą. **Dinaminis atminties skirstymas** – adresai nustatomi betarpiškai kreipiantis į atmintį. Fizinis adresas nustatomas tiesiogiai prieš kreipiantis į tą adresą.

# Komandos vykdymo schema architektūroje su dinaminio adresų nustatymu

Schematiškai pavaizduosime komandų vykdymą, kuomet nėra dinaminio atminties skirstymo:

- $M[O:N]$  – atmintis;
- $K$  – komandų skaitliukas;
- $R$  – registras.

```
L: W := M[K];  
   OK := W_operacijos_kodas;  
   ADR := W_operando_adresas;  
   K:=K+1;  
ADD: IF OK = 1 THEN R := R + M[ADR];  
      ELSE  
SR:  IF OK = 2 THEN M[ADR] := R;  
      ELSE  
BR:  IF OK = 3 THEN K := ADR;  
GOTO L;
```

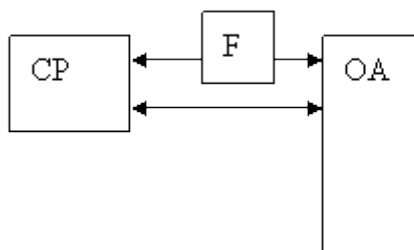
Čia  $K$ ,  $ADR$  vadinami efektyviais adresais.

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

$M[k] \sim M[F(k)]$

$M[ADR] \sim M[F(ADR)]$ .

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



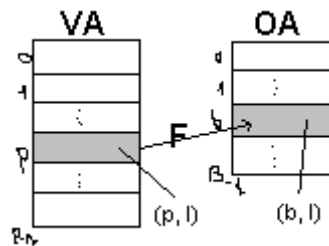


# Statinis ir dinaminis atminties skirstymas

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 naudoti tam tikrus lokalius adresus, kurie vėliau koku būdu susiejami su fiziniaisiais adresais. Tokia lokalių adresų visuma – virtuali atmintis. Fizinių adresų visuma – reali atmintis. Virtuali atmintis su fizine gali būti susiejama statiškai arba dinamiškai.

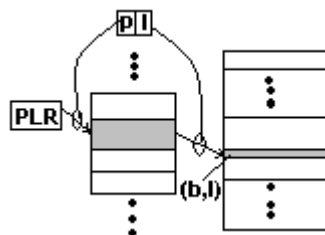
## Puslapinė organizacija



Puslapinė organizacija – tai konkretus virtualios 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:

1. VA turime  $[p|l]$ ;
2. Aparatūroje turi būti numatytas  $[PLR]$  – puslapių lentelės registras (rodo aktyvaus proceso puslapių lentelę);
3. Puslapių lentelės saugomos atmintyje.



Puslapių lentelės registro dydis atitinka VA puslapių skaičių. Kiekvienam procesui sudaroma puslapių lentelė.

Vieno segmento VA atvaizdavimas:  $F(p, l) = M [PLR + p] * 2^k + l$

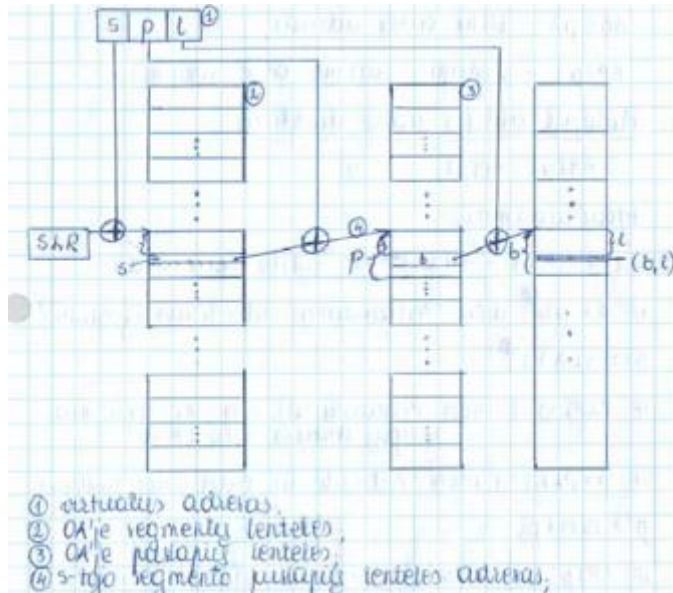
Čia  $M[PLR+p]$  – bloko numeris;  $2^k$  – bloko dydis;  $l$  – poslinkis.

# 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.



- 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 THEN „Pertraukimas. OA nėra s-ojo segmento psl. lentelės“
IF p > PLD[S] THEN „Pertraukimas. Neleistinas psl.segmente“
p := PLB + p;
IF BP[p] = 1 THEN „Pertraukimas. Psl.nėra atminty“
F := BB[p] + 1;
```

# Atminties skirstymo puslapiais strategijos

---

Esant dinaminiam atminties skirstymui VA dydis gali būti didesnis nei OA dydis. Tada reikia priverstinai atlaisvinti vietą. Puslapių skirstymo uždavinys turi atsakyti į klausimus:

- a) koku 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 betarpiškai 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, į kuriuos puslapių trasoje bus kreipiamasi vėliausiai, vadinama Bellady strategija. Plačiau: [http://en.wikipedia.org/wiki/Belady%27s\\_anomaly](http://en.wikipedia.org/wiki/Belady%27s_anomaly)

Pvz.: tarkime, turime du blokus b1, b2 ir puslapių seką p1, p2, p3, p4, p1, p2, p3. Sprendimas:

B<sub>1</sub> | p<sub>1</sub> p<sub>1</sub> p<sub>1</sub> p<sub>1</sub> p<sub>1</sub> p<sub>1</sub>

B<sub>2</sub> | p<sub>2</sub> p<sub>3</sub> p<sub>4</sub> p<sub>2</sub> p<sub>3</sub>

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.

## Atminties išskyrimas ištisinėmis sritimis

---

GETAREA(adress, size); FREEAREA(address, size); FREEGARBAGE;

# 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;

# 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 patogiu 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 patalpiniui. 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.

## Failinės atminties įrenginių charakteristikos

---

Fizinės failinės atminties įrenginiai apibūdinami tam tikromis charakteristikomis:

- Talpumas – maksimalus informacijos kiekis, kurios gali būti saugomos;
- Įrašo dydis – minimalus informacijos kiekis, į kuri galima adresuoti įrenginyje. Įrašai įrenginiuose gali būti kintamo arba pastovaus ilgio.
- Priėjimo būdai:
  - tiesioginis – operuojama aparatūra;
  - nuoseklus – kai priėjimui prie įrašo reikalingas visų tarpinių įrašų peržiūrėjimas (atmintis magnetinėje juostoje).
- FA informacijos nešėjas – tomas. Tomo charakteristika – jo pakeičiamumas – įgalina iš esmės padidinti vartotojo naudojamos VA apimtį. Pvz. Diskelis.
- Duomenų perdavimo greitis. Jis matuojamas baitais arba bitais per sekundę perduodant informaciją tarp OA ir įrenginio. KB – kbaitai, kb – kbitai
- Užlaikymas. Įrenginiui gavus eilinę įv/iš komandą (jei tai juostinis įrenginys), jam reikia įsibėgėti nuo praeito skaitymo iki naujo segmento pradžios. Jei diskas – užlaikymas – tai apsisukimas nuo pradžios iki tos vietos, kur yra informacija.
- Nustatymo laikas. Tai galvučių perstūmimo laikas (diskai).

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

## Įvedimo/išvedimo įrenginių greičių charakteristika

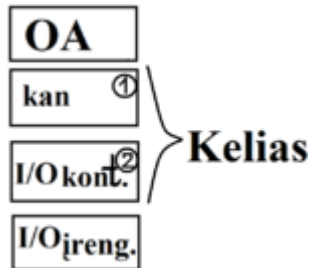
---

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec

EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

## Ivedimo/išvedimo įrenginių procesų schema

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.



(1) Specializuotas procesorius su savo komandų sistema (registrais, valdymo įrenginiu). Perduoda duomenis tarp OA ir I/O kontrolierio (2) kontrolieris – specializuotas kontrolieris nukreiptas į tam tikrą tipą.

I/O įrenginį atstovaujantis procesas  $P_i$ .

Begin

L: PRAŠYTIR(III<sup>(1)</sup>,  $\Omega$ <sup>(2)</sup>, (P<sup>(3)</sup>, Ilprogr.<sup>(4)</sup>));

PRAŠYTIR (KK<sup>(5)</sup>, III<sup>(6)</sup>, Kelias);

Komutavimo ir įrenginių pranešimo komandos;

I/O inicijavimas;

PRAŠYTIR (IIP<sup>(7)</sup>, Kelias<sup>(8)</sup>, Pran)<sup>(9)</sup>;

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 kontrolieris (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[\*]]:= $\Omega$ ; // 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.



# Užklauso failinei sistemai realizavimo pavyzdys

---

Darbai su failine atmintimi naudojama bendros paskirties failų sistema. Užklausa skaitymui iš failo gali atrodyti taip:

Loginiame lygmenyje vartotojo pateikta komanda: READ(A) A- nuskaityma 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 reikalingas unikalus vidinis failo vardas ir deskriptorius. Pagal failo vidinį vardą galima nustatyti failo deskriptorių, o failų deskriptoriai kaip ir patys failai saugomi išorinėje atmintyje. Failų atidarymo metu, failų deskriptoriai iš išorinės atminties perkeliama į vidinę atmintį, aktyvių failų katalogą.

READ3(d, A, r, 80)

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

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

Įrašo adr. := įrašo adr + 800

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

# 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) – sisteminiame kataloge (šiuo atveju deskriptoriaus nėra, todėl ieškoma pagal išorinį vardą, o tik tada yra sukuriamas deskriptorius.). AFK yra operatyvioje atmintyje.

## Failų sistemos hierarchinis modelis

---

Failų sistemos funkcijas patogiau apibūdinti pagal jų lygį, pradedant nuo aparatinio iki vartotojo serviso klausimų. Failų sistemos suskirstymas į loginius lygius:

1. Įvesties/išvesties sistema – koordinuoja fizinių įrenginių darbą. Šio lygio procesai atlieka informacijos blokų apsikeitimą tarp OA ir išorinės atminties pagal užduotą adresą.
2. Reali (bazinė) failų sistema – transformuoja failo vidinį unikalų identifikatorių į failo deskriptorių.
3. Virtuali (loginė) failų sistema – pagal vartotojo suteiktą IV nustato jo vidinį unikalų vardą. Naudojamas vidinis failų katalogas. Virtualus lygis nepriklauso nuo fizinių įrenginių.
4. Priėjimo metodai – realizuoja dėsni, pagal kurį apdorojami failo įrašai. Tokį dėsni užduoda vartotojas pagal programos prasmę. Pvz.: nuoseklus priėjimas arba kai įrašai apdorojami pagal lauko (rakto) reikšmės didėjimo ar mažėjimo tvarką.
5. DBVS – realizuoja failo loginės struktūros vaizdavimą į fizinę struktūrą.

# Failų sistemos įvedimo-išvedimo posistemė

## Bazinė failų valdymo sistema

---

Tai įvedimo/išvedimo sistema, kurioje aparatūros specifika izoliuoja įvedimą/išvedimą nuo likusios OS dalies. Tai leidžia įvedimą/išvedimą nagrinėti persiunčiamų informacinių blokų 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.

Darbai su deskriptoriais bazinė sistema turi komandas(funkcijas):

SUKURTI(failo vardas, sritis), kur sritis – iš kokių blokų sukurti failą. Ji vykdoma inicializuojant procesą, aptarnaujantį tomą. Vėliau 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 ryšį su 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 sukūrimo 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,

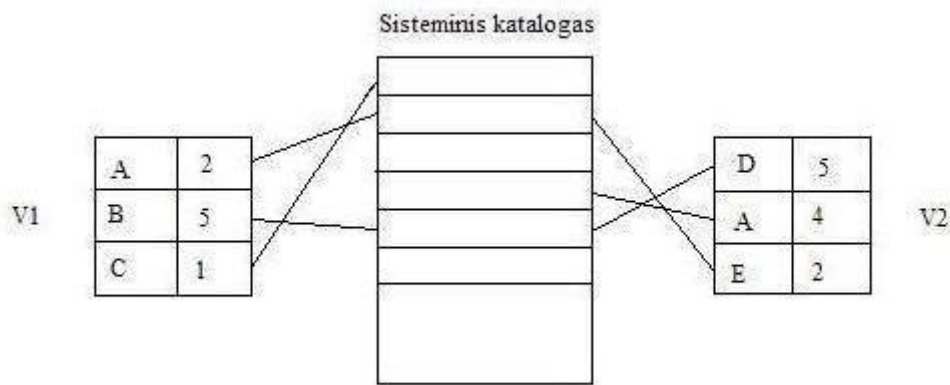
ATLAISVINTI(tomo vardas) – atjungimas nuo įrenginio.

# Loginė failų valdymo sistema

Ji atvaizduoja lokalius vartotojo failų vardus į unikalius failų identifikatorius. Loginė failų valdymo sistema pateikia išoriniam interfeisui komandas, 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 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.

## 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 įrašų. Raktai gali būti naudojami įrašų identifikavimui.

Įrašai su vienodais 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.

Nuoseklus ir tiesioginio metodų suderinimui naudojamas indeksacijos metodas.